

Sem vložte zadání Vaší práce.

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
KATEDRA SOFTWAREVÉHO INŽENÝRSTVÍ



Bakalářská práce

Flexibilní logování pro embedded Linuxové systémy

David Vavříčka

Vedoucí práce: Ing. Matěj Laitl

16. května 2016

Poděkování

Na tomto místě bych rád poděkoval a vyslovil uznání Mateji Laitlovi, vedoucímu mé bakalářské práce za správné směřování, cenné rady a řešení otázek, se kterými jsem se na něj obracel.

Dále děkuji firmě Nangu.TV za umožnění vzniku této bakalářské práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 16. května 2016

.....

České vysoké učení technické v Praze
Fakulta informačních technologií

© 2016 David Vavříčka. Všechna práva vyhrazena.

Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí, je nezbytný souhlas autora.

Odkaz na tuto práci

Vavříčka, David. *Flexibilní logování pro embedded Linuxové systémy*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

Abstrakt

Tato bakalářská práce se zabývá analýzou, návrhem a následnou implementací nového logovacího řešení pro set-top box EKT DID7006mTF.

Hlavním cílem je přepracovat současné logovací řešení a přidat mu podporu vzdáleně měnit jeho konfiguraci a umožnit tak snížit vytížení sítě.

Na základě analýzy byl nasazen nový logovací démon Rsyslog, pro nějž byly naprogramovány dva moduly, dále bylo naprogramováno API umožňující změnu konfigurace logování a byly napsány podpůrné skripty.

V závěru práce bylo nové řešení otestováno na reálném hardware, čímž bylo demonstrováno splnění požadavků a příznivé provozní vlastnosti.

Klíčová slova logování zpráv, logovací démoni, Rsyslog, set-top boxy, vestavěné systémy, Linux

Abstract

This Bachelor's thesis deals with analysis, design and implementation of a new logging solution for EKT DID7006mTF set-top box.

The main goal of this thesis is to redesign the current logging solution and to add functionality to change logging configuration remotely to allow to reduce amount of network load.

Based on the results of the analysis was deployed Rsyslog logging daemon and were developed two Rsyslog modules, a new API for remote configuration change and some subsidiary scripts.

The new solution was tested on real hardware where were demonstrated accomplishment of all desired goals and good running characteristics.

Keywords message logging, logging daemons, Rsyslog, set-top boxes, embedded systems, Linux

Obsah

Úvod	1
1 Technické požadavky	3
2 Logování v linuxových systémech	5
2.1 Logovací démon	5
2.2 Protokol Syslog	5
2.3 Formát zpráv	6
3 Analýza	9
3.1 Původní řešení	9
3.2 Nové řešení	11
4 Návrh a realizace	15
4.1 Sestavení a instalace Rsyslogu na STB	15
4.2 Konfigurace Rsyslogu	17
4.3 Moduly pro Rsyslog	21
4.4 Vzdálená konfigurace	24
5 Testování	27
5.1 Testovací prostředí a nástroje	27
5.2 Test využití linky zprávami za běžného provozu	27
5.3 Test rate-limitingu zpráv	29
5.4 Test vytížení systému	31
Závěr	35
Literatura	37
A Seznam použitých zkratek	39

Seznam obrázků

2.1	Formát zpráv dle RFC 3164 a RFC 5424	6
3.1	Diagram původního řešení	10
3.2	Diagram nového řešení	13
4.1	Tok zpráv v Rsyslogu	19
5.1	Využití linky v původním řešení	28
5.2	Využití linky v novém řešení	28
5.3	Využití linky v novém řešení (filtrování zpráv aplikace solid)	28
5.4	Využití linky při extrémním zatížení bez rate-limitingu	30
5.5	Využití linky v původním řešení při extrémním zatížení	30
5.6	Využití linky v novém řešení při extrémním zatížení	30

Seznam tabulek

2.1	Tabulka severit dle RFC 3164	7
2.2	Tabulka facilit dle RFC 3164	7
4.1	Převodní tabulka severit	23

Úvod

Zadavatelem práce je společnost vyvíjející set-top boxy pro sledování internetové televize, která prodává hotová řešení pro IPTV streaming jednotlivým provozovatelům.

V současné době využívají služeb provozovatelů desetitisíce zákazníků a trend ukazuje, že se jejich počet bude do budoucna nadále zvyšovat.

Každý ze set-top boxů v domácnostech uživatelů odesílá stovky zpráv za minutu o svém stavu na servery jednotlivých provozovatelů. Jejich zpracování představuje pro servery velkou zátěž, dále zprávy mohou zahltit už tak vytíženou linku ve směru od uživatele a proto byl sepsán seznam technických požadavků, jež si kladou za cíl umožnit snížit objem logů posílaných na servery.

Tato práce se ve své první části zabývá analýzou technických požadavků a výběrem vhodného postupu řešení.

Na základě analýzy je posléze provedena implementace, ve které je kladen důraz na použití pouze volně šiřitelných open-source projektů a jejich případné rozšíření.

V závěru práce je výsledné řešení podrobno testování, kde se kromě splnění technických požadavků zkoumají z důvodu omezeného výkonu set-top boxů nároky na výpočetní výkon.

Technické požadavky

Cílem je upravit logovací řešení pro set-top box EKT DID7006mTF [1] tak, aby splňovalo technické požadavky popsané v této kapitole. Řešení musí fungovat a být otestováno na zmíněném modelu set-top boxu a pokud možno by mělo být přenositelné i na jiné typy set-top boxů.

Původním záměrem bylo rozdělit požadavky na základní a rozšířené. Nakonec zadavatel rozhodl, že všechny níže zmíněné požadavky jsou základní a tedy jejich implementace je nutná pro splnění práce.

Snížení objemu logů

Je žádoucí umožnit snížit objem zasílaných logů z důvodu přílišného zatížení sítě a serverových disků. A to tak, že zadavatel bude schopen nadefinovat pro každou komponentu úroveň severity zpráv, od které mají být posílány na server. Výchozí nastavení dodá zadavatel.

Vzdálená konfigurace

Technické řešení musí být schopno za běhu pomocí SHELL-ového API měnit maximální severity zpráv pro jednotlivé komponenty a dále toto API musí umožnit nastavit výchozí severity, která se použije pro komponenty ji nemají explicitně nastavenou. Takto změněné nastavení musí být perzistentní i po restartu STB. Výchozí nastavení se obnoví až po factory resetu. API navrhne dle své libovůle sám řešitel.

Rate-limiting odesílaných zpráv

Nové logovací řešení musí být schopné provádět rate-limiting odesílaných zpráv tak, aby nepřekročilo maximální vyhrazenou šířku pásma. Naivní rate-limiting

je i v existujícím řešení, řešitel navrhne výchozí nastavení nového řešení tak, aby přibližně odpovídalo současnému chování.

Formát logů

Je nutno zachovat formát logů jako ho má původní řešení, aby se jednalo o drop-in replacement bez nutnosti jakkoli měnit konfiguraci serveru, který sbírá logy od set-top boxů.

Razítkování zpráv

Každé zprávě se musí přidat textový prefix id=N, kde N monotonicky roste s každou zprávou. To slouží pro detekci ztracených zpráv. Id přeteče po dvaceti bitech. Po rebootu STB id znovu začíná od 1.

Post-processing zpráv

Zadavatel má pouze částečnou kontrolu nad zprávami generovanými aplikacemi na set-top boxu, například nedokáže ve všech případech eliminovat dlouhé prefixy u zpráv. Je proto nutno takové prefixy rozpoznat a vhodně odfiltrovat před odesláním. Ze stejného důvodu mají některé zprávy nevhodně vyplněnou severitu a položku app-name. Jejich správné hodnoty jsou uloženy v textu zprávy, jejíž formát je pro jednotlivé skupiny zpráv konstantní. Řešení bude schopné tyto údaje z těla zprávy extrahovat a nahradit jimi původní metadata. Tato pravidla musí být možné definovat a měnit bez nutnosti nového sestavení softwaru. Řešitel vytvoří pro ukázkou 3 pravidla, která budou sloužit zadavateli jako šablony pro možná budoucí filtrovací pravidla.

Logování v linuxových systémech

Systém a aplikace na něm běžící informují o svém stavu prostřednictvím zpráv, jež jsou užitečné pro získání statistických údajů, k ladění programů nebo např. pro detekování útoků.

Je užitečné, aby zprávy měly unifikovaný formát a aby byl jednoznačně určen způsob jejich přenosu po síti. To na UNIX-ových a tedy i LINUX-ových systémech specifikuje protokol Syslog.

Dále je ku prospěchu věci zprávy centralizovaně zachytávat a spravovat, k čemuž slouží tzv. logovací démon.

2.1 Logovací démon

Démon je v UNIX-ovém světě označení pro takový proces, který oproti běžným procesům neintereaguje přímo s uživatelem, ale běží na pozadí operačního systému a funguje samostatně. Hlavním účelem logovacího démonu je sběr logů od ostatních procesů, které následně v závislosti na jeho konfiguraci dokáže filtrovat a ukládat na disk či odesílat na vzdálený server.

2.2 Protokol Syslog

Protokol Syslog [2] spatřil světlo světa již v roce 1980. Má ho na svědomí Eric Allman, který jej psal s úmyslem využít ho pro potřeby projektu Sendmail. Nicméně postupem času byl tento protokol díky skvělému návrhu a díky své jednoduchosti adaptován i jinými projekty. Rozšířil se na různé OS a později se stal standardem pro logování na většině Unixových systémů.

Protokol především specifikuje podobu zpráv a způsob jejich přenosu po síti.

2.2.1 The BSD syslog Protocol - RFC 3164

Až přibližně po dvaceti letech od vzniku protokolu Syslog (v roce 2001) sepsal Chris Lonvick dokument RFC3146 [3] popisující daný protokol. Do té doby existovala velká spousta v detailech se lišících implementací, což mělo v jistých případech za následek nekompatibilitu.

Ještě v dnešní době je možno nalézt systémy s logovacím řešením, které specifikuje tento devatenáctistránkový dokument. Nicméně čím dál častěji je nahrazován jeho následníkem, kterým je RFC 5424.

2.2.2 The Syslog Protocol - RFC 5424

Oficiální standard vznikl až s příchodem protokolu RFC 5424, který světu představil v roce 2009 jeho autor Rainer Gerhards. Počet stránek se rozrostl na 37 a došlo k některým zásadním a velice užitečným změnám.

Nový standard umožňuje použití libovolného transportního protokolu [4] oproti RFC 3164, kde je jako transportní protokol pevně určeno UDP [5]. Mírných změn doznal také formát zpráv, který je možno strukturovaně rozšiřovat dle specifických potřeb [6]. To má ovšem za následek jeho zpětnou nekompatibilitu s RFC 3164.

2.3 Formát zpráv



Obrázek 2.1: Formát zpráv dle RFC 3164 a RFC 5424

Zpráva se skládá ze tří částí, jak je pro lepší představu graficky vyzobrazeno na obrázku 2.1.

2.3.1 Část PRI

PRI se skládá z jedno- až tří-místného decimálního čísla PRIVAL ohraničeného symboly „<“ na začátku a „>“ na konci. PRIVAL má v sobě zakódované číselné hodnoty parametrů zprávy označovaných jako Severity a Facility.

$$Prival = 8 * Facility + Severity$$

Facility určuje zdroj logů a Severity jejich důležitost (tedy zda se jedná např. pouze o ladící zprávu nebo naopak o důležitou zprávu např. o pádu programu). Na základě těchto dvou hodnot provádí logovací démon základní filtraci zpráv.

Důležitým poznatkem je, že formát PRI je shodný podle dokumentů RFC 5424 i RFC 3164.

Tabulka 2.1: Seznam severit dle RFC 3164. [3]

Číselný kód	Severity	Význam
0	Emergency	Systém je nepoužitelný
1	Alert	Vyžadována okamžitá reakce
2	Critical	Kritický stav
3	Error	Chybový stav
4	Warning	Upozornění na hrozící chybu
5	Notice	Neobvyklý stav, ale ne chybový
6	Informational	Informační zpráva
7	Debug	Ladící zpráva

Tabulka 2.2: Seznam facilit dle RFC 3164. [3]

Číselný kód	Facility	Původ zpráv
0	kern	Zprávy jádra systému
1	user	Generované uživatelem
2	mail	Generované emailovým systémem
3	daemon	Systémoví démoni
4	auth	Zprávy autorizačního charakteru
5	syslog	Generované syslogem
6	lpr	Tiskový systém
7	news	Network news systém
8	uucp	UUCP systém
9	cron	Zprávy plánovacího systému Cron
10	security	Zprávy zabezpečovacího charakteru
11	ftp	FTP démon
12	ntp	NTP systém
13	logaudit	Log audit
14	logalert	Log alert
15	clock	Clock démon
16	local0	Lokální zprávy (0)
17	local1	Lokální zprávy (1)
...	...	
23	local7	Lokální zprávy (7)

2.3.2 Header

Tato část zprávy obsahuje časovou značku a identifikaci zdroje (podle IP adresy nebo hostname). Přesnou podobu Headeru specifikují dokumenty RFC 3164 a RFC 5424 odlišně. Zatímco starší dokument definuje přesnou podobu časové značky, tak novější specifikace dovoluje různé formáty. Díky tomu není zaručena plná zpětná kompatibilita.

2.3.3 MSG

Část Msg pokrývá zbývající část zprávy. Jejím obsahem jsou různé zbývající informace o procesu generujícího danou zprávu a pak samozřejmě samotný text zprávy.

Analýza

V úvodu kapitoly je popsán STB a na něm běžící pro tuto práci důležité aplikace. Následuje popis infrastruktury (vyjádřený formou diagramu na obr. 3.1), kde je znázorněna komunikace mezi STB a servery zadavatele. Kapitulu zakončuje analýza řešení jednotlivých technických požadavků a srovnání logovacích démonů.

3.1 Původní řešení

3.1.1 HW specifikace STB EKT DID7006

CPU: ARM 9 Ali M3733 (1GHz Dual core)

GPU: Mali 400

RAM: 512MB DDR3

Perzistentní paměť: 512MB NAND Flash

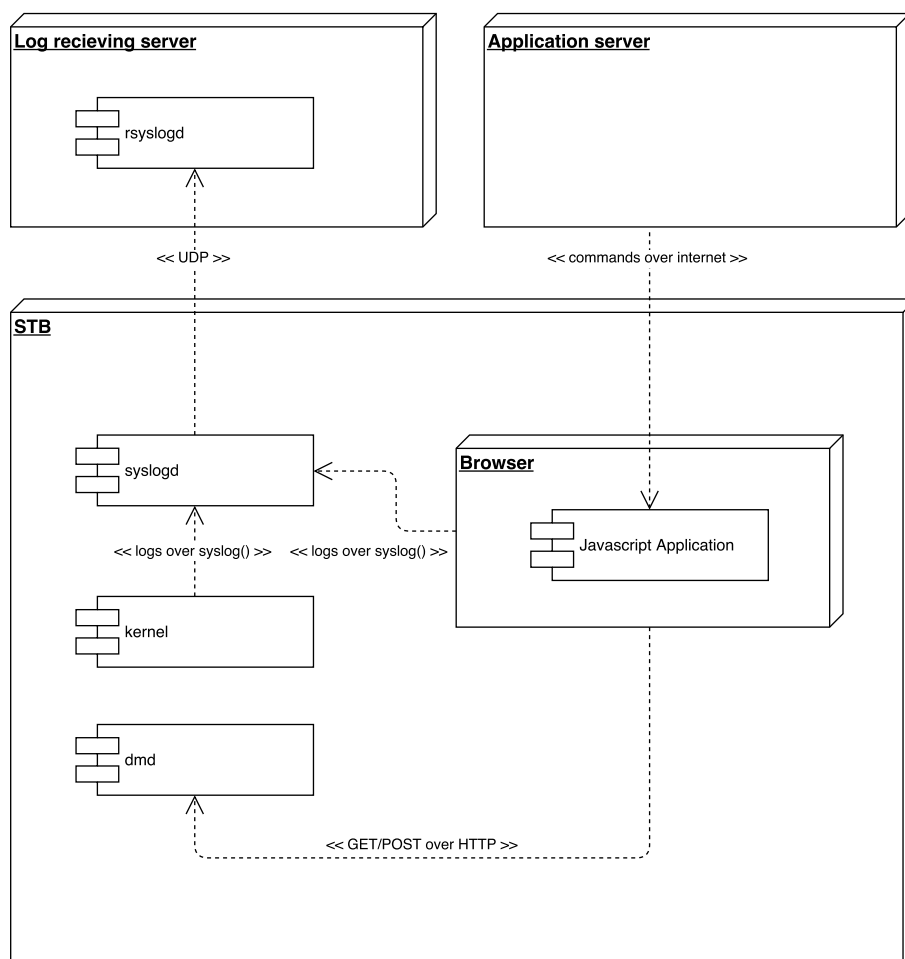
3.1.2 Runtime prostředí

Na STB běží OS GNU/Linux s BusyBox sadou aplikací, která nahrazuje standardní GNU shellové utility. BusyBox aplikace jsou díky své malé velikosti vhodnější pro embedded zařízení. Na druhou stranu neposkytují všechny funkce a možnosti nastavení plnohodnotných GNU Core Utilities.

3.1.2.1 Browser

Browser je jednoduchý internetový prohlížeč dodaný výrobcem STB. V něm běží v Javascriptu zadavatelem napsaná aplikace poskytující uživatelské rozhraní. Mimo to také naslouchá zprávám z aplikačního serveru, na jejichž základě může provádět akce, a to včetně generování POST či GET requestu pro démon dmd. Tato javascriptová aplikace spolu s dalším podpogramem nesoucím název Player generují velké množství logů. Tyto logy nepoužívají standardizovanou množinu severit dle dokumentu RFC 3164, ale svou vlastní s ní

3. ANALÝZA



Obrázek 3.1: Diagram původního řešení

nekompatibilní. Logovací démon ovšem očekává zprávy právě podle standardu zmíněného dokumentu. Implementaci Browseru ani Playeru nemůže zadavatel měnit a proto je třeba nekompatibilitu vyřešit pomocí logovacího démonu.

3.1.2.2 Servisní komponenta dmd

Dmd je v C++ napsaná servisní komponenta, která zprostředkovává komunikaci mezi Browserem a shellovým prostředím pomocí minimalistického HTTP serveru. Umožňuje JavaScript aplikaci na STB provádět operace, které jsou dostupné jen z shellového API.

3.1.2.3 BusyBox Syslogd

Jedná se o minimalistický logovací démon, který je podrobněji popsán v následující kapitole „Srovnání logovacích démonů“. Zadavatel mu pro své potřeby navíc doimplementoval číslování jednotlivých zpráv pro rozpoznání výpadku a za druhé naivní rate-limiting zpráv. Ten je naivní z důvodu, že používá statický buffer na počet zpráv, který nebere ohled na jejich velikost a hlavně bufferované zprávy se neodešlou samy, ale jen s nově příchozími zprávami. Za použití tohoto démonu není možné splnit některé stěžejní technické požadavky, jako například „Snížení objemu logů“ nebo „Post processing zpráv“. Proto bude nutné ho nahradit nějakým vyspělejším démonem. Tím se ale zabývá až další kapitola „Nové řešení“.

3.1.3 Aplikační server

Aplikační server se stará o business logiku, například poskytuje uživateli práva přehrát daný kanál nebo film, komunikuje s javascriptovou aplikací, která se vůči němu autorizuje a podobně. AS není možné v této práci jakkoli měnit. Je nutné zachovat kompatibilitu.

3.1.4 Servery pro sběr logů

Provozovatel vlastní cluster serverů pro sběr logů od desítek STB. Běží na nich Rsyslog nakonfigurovaný pro co nejvyšší propustnost.

3.2 Nové řešení

Prvně je nutno zvážit, zda problém řešit na straně serveru nebo set-top boxu. Vhodnou konfigurací logovacího démonu na straně serveru, který by nepotřebné zprávy zavčas rozpoznal, zahodil a dále nezpracovával by bylo možné splnit požadavek na snížení zátěže serverových disků. Přetížení sítě se takto vyřešit ale nedá a proto bylo toto řešení zavrhnuto.

Je tedy nutno problém řešit na straně set-top boxu, kde je původní řešení postaveno na BusyBox Syslogd. Nabízí se možnost upravit fungování tím způsobem, aby se logy s nízkou severitou už na set-top boxu zahazovaly a pouze v případě potřeby bylo umožněné na dálku změnit konfiguraci démonu tak, aby se povolilo logování pro zprávy s nastavenou danou komponentou a severitou. Takovou možnost prostý Syslogd neposkytuje a je proto nutno zvážit napsání vlastního démonu, nasazení jiného, vyspělejšího logovacího démonu nebo rozšíření stávajícího démonu Syslogd.

3.2.1 Srovnání logovacích démonů

V této kapitole jsou popsány vybrané logovací demony a v jejím závěru jsou porovnány.

3.2.1.1 BusyBox Syslogd

Tato logovací utilita se skládá ze dvou démonů, jmenovitě z Klogd, který zachytává logy z kernelu a předává je ke zpracování démonu Syslogd. Syslogd pak zachytává i všechny zbylé zprávy a dále s nimi nakládá. Má však velice omezený rozsah schopností. Dokáže pouze zprávy lokálně ukládat, přeposílat je dále po síti, zahazovat duplikáty, rotovat zprávy v závislosti na velikosti a filtrovat je podle omezených kritérií. Dokáže totiž filtrovat zprávy pouze podle typu facility a už nikoliv podle názvu komponenty, která ji vygenerovala.

3.2.1.2 Syslog-ng

Syslog-ng je flexibilní logovací démon zaměřený na centralizované a zabezpečené logování. Má široké možnosti nastavení a poskytuje množství funkcí. Takže jeho vhodným nakonfigurováním se dá snadno splnit většina vytyčených technických požadavků [7], až na požadavky na vzdálenou změnu konfigurace a změnu severity zpráv.

3.2.1.3 Rsyslog

Výčet funkcí Rsyslogu je ještě obsáhlejší než u Syslog-ng [8]. Technické požadavky se s jeho použitím tedy také dají splnit všechny, kromě vzdálené změny konfigurace a post-processingu zpráv. Stejně jako Syslog-ng není totiž jen logovacím démonem, ale i analyzérem zpráv. Dokáže tak zprávy podle jejich obsahu měnit, třídit a jinak s nimi nakládat. Že je Rsyslog vyspělý a kvalitní program dokazuje fakt, že je výchozím logovacím démonem ve spoustě linuxových distribucích, jmenovitě například v Ubuntu. Jeho jedinou slabinu shledávám v nedostatečné dokumentaci.

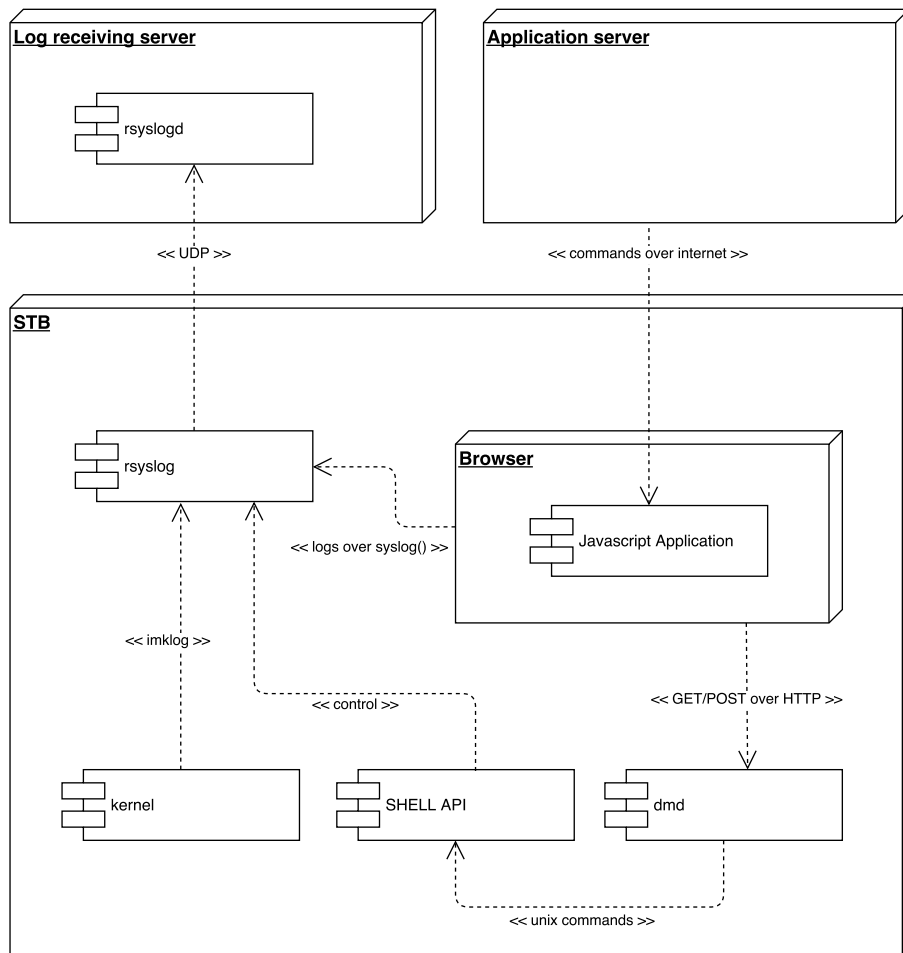
3.2.1.4 Porovnání logovacích utilit

Pouhým nasazením jakéhokoli známého logovacího démonu není možné splnit všechny vytyčené technické požadavky. V případě ponechání původního démonu BusyBox Syslogd by pro splnění technických požadavků bylo nutno doimplementovat tolik rozšíření, že by to výrazně přesahovalo rozsah bakalářské práce.

Výhodněji se jeví nasadit pokročilý logovací démon jako je Syslog-ng či Rsyslog. Oba totiž disponují funkcemi, s jejichž použitím je možné vyřešit většinu vytyčených cílů práce.

Rsyslog podporuje tzv. modulární systém [9], který poskytuje možnost napsání vlastních modulů (programů), které mohou zásadně rozšiřovat rozsah jeho funkcí a je tak možno jejich použitím doimplementovat chybějící schopnosti. Syslog-ng také poskytuje modulární systém, ale bez podpory modulů třetích stran. Modulární systém Rsyslogu tak představuje hlavní argument, proč byl nakonec vybrán jako nový logovací démon právě Rsyslog.

3.2.2 Nová podoba logovacího řešení



Obrázek 3.2: Diagram nového řešení

Na obrázku 3.2 je pro lepší představu znázorněna nová podoba řešení. Dojde k nahrazení logovacího démonu Syslogd modernějším Rsyslogem. Dále přibude rozhraní (SHELL API) pro umožnění změny konfigurace skrze internet.

To je možné díky zadavatelově komponentě DMD, jež je pro tuto práci black boxem, která dovoluje na základě dotazů GET a POST spouštět příkazy s root oprávněním v SHELLu a tedy i volat funkce nového API.

Návrh a realizace

V úvodu kapitoly je předveden buildovací systém a způsob instalace aplikací pro STB. Následuje obsáhlý popis Rsyslogu a jeho funkcí včetně ukázek jeho konfigurace. Dále je popsán vývoj modulů pro Rsyslog a jsou představeny dva vlastní naprogramované. Kapitulu zakončuje návrh API pro vzdálenou konfiguraci Rsyslogu a popis jeho implementace.

4.1 Sestavení a instalace Rsyslogu na STB

Rsyslog není součástí SDK a tak jej bylo nutné a všechny knihovny na kterých závisí (zlib, libestr, libee, liblogging a libfastjson) zkompileovat a posléze nainstalovat na STB. K tomu bude použit systém Gu.

4.1.1 Buildovací systém Gu

Gu je buildovací systém pro linuxová embedded zařízení, který si klade za cíl zjednodušit a urychlit buildovací proces. Gu využívá balíčkovací systém pacman (převzatý z Arch Linuxu) a nástroj scratchbox2 sloužící k zjednodušení cross-kompilace. Tyto 2 nástroje jsou skryty v konzolovém příkazu „gu“, kterým se celé Gu ovládá.

4.1.2 PKGBUILD

Jedná se o shellový script obsahující informace potřebné pro sestavení jednotlivých aplikací systémem Gu. Jeho syntaxe je podrobně popsána v odstavci níže.

4.1.3 Sestavení a instalace aplikací

Jako příklad zde uvádím soubor PKGBUILD pro sestavení a instalaci Rsyslogu. V podobném duchu jsou napsány i skripty pro ostatní aplikace a knihovny.

```
1 pkgname=rsyslog
2 pkgver=8.16.0-nangu-0.3
3 arch=('armv7h' 'armv7sp')
4 depends=('zlib' 'libestr' 'libee' 'liblogging' 'libfastjson')
5 source=${pkgname}-${pkgver}.tar.gz
6 md5sums=('SKIP')
7
8 build() {
9     cd ${pkgname}
10    PKG_CONFIG_PATH=/mnt/hdd_1/lib/pkgconfig
11    autoreconf -fvi
12    ./configure --prefix=/mnt/hdd_1 \
13    --disable-uuid --enable-mmsequence \
14    --enable-mmsevrewrite --enable-mmdelstr
15    make V=1
16 }
17
18 package() {
19     cd ${pkgname}
20     make install DESTDIR=${pkgdir}
21 }
```

pkgname: Název balíčku (měl by se shodovat s názvem zdrojového archivu aplikace).

pkgver: Verze balíčku (měla by se shodovat s verzí sestavované aplikace).

arch: Pole specifikující architektury cílových systémů.

depends: Pole názvů balíčků, které musí být nainstalovány před spuštěním tohoto softwaru.

source: Seznam souborů nutných pro sestavení balíčku. Soubory mohou odkazovat na lokální úložiště, nebo na vzdálený server, v tom případě se skript postará o jejich stažení. Komprimované soubory skript automaticky rozbalí.

md5sum: Pole kontrolních součtů pro každý specifikovaný „source“.

build(): Nepovinná funkce obsahující příkazy vedoucí ke konfiguraci a sestavení aplikace.

package(): Povinná funkce instalující soubory. Funkce je spouštěna až po exekuci ostatních nepovinných funkcí.

pkgdir: Proměnná obsahující umístění kořenového adresáře instalované aplikace.

Po vstoupení do adresáře obsahujícím zdrojové kódy aplikace a skript PKGBUILD stačí zadat příkaz „gu build“, čímž se uvede do chodu celý skript. Nejdříve se inicializují proměnné, posléze se vykoná funkce build() a nakonec package(), čímž by měl (pokud nedojde k žádné chybě) vzniknout v umístění „pkgdir“ balíček s přeloženou aplikací. Příkazem „gu install <název-vzniklého-balíčku.tar.xz>“ nainstalujeme balíček do SDK.

4.2 Konfigurace Rsyslogu

Rsyslog se řídí podle pravidel specifikovaných v konfiguračním souboru `/etc/rsyslog.conf`. Za použití tzv. compatibility módu [10] podporuje syntaxi známou z dříve na linuxu hojně užívaného démonu `syslogd`. Níže je uveden příklad konfiguračního souboru `syslog.conf` [11] pro `syslogd`. Na levo je možno specifikovat kritéria pro filtraci zpráv podle jejich severity a facility. Na pravo je pak umístění, kam zprávy odesílat.

```

1 *.=crit;kern.none           /var/adm/critical
2 kern.*                     /var/adm/kernel
3 kern.crit                  @finlandia
4 kern.crit                  /dev/console
5 kern.info;kern.!err        /var/adm/kernel-info

```

Díky této podpoře je možno velice snadno migrovat z `Syslogd` na `Rsyslogd`. Pro využití i jiných než základních funkcí `Rsyslogu` je nutno k nakonfigurování pravidel použít syntaxi jazyku `Rainerscript`, který je z toho důvodu v této práci použit.

4.2.1 Rainerscript

`Rainerscript` [12] je skriptovací jazyk navržený ke správě síťových událostí a dále ke konfiguraci softwaru, pro který je používán. Tento jazyk je totiž teoreticky použitelný v různých typech softwarů, nicméně v době psaní této práce byl vyvíjen a reálně používán pouze v `Rsyslogu`.

Jedná se o netypový jazyk, který poskytuje podporu regulárních výrazů [13] a má nadefinované užitečné funkce především pro práci s textovými řetězci. Podrobnosti jsou k nalezení v dokumentu s formální definicí jazyka [14].

4.2.2 rsyslog.conf

Níže je uvedena ukázka konfiguračního souboru za použití jazyku `Rainerscript`, kde jsou předvedeny základní konstrukce používané při jeho tvorbě. Tato ukázka by měla primárně sloužit zadavateli pro porozumění konfiguračního souboru a jako možný návod pro budoucí úpravy.

```

module(load="NazevModulu")

```

Na začátku souboru je možné načíst různé moduly poskytující doplňující funkce. Příkladem může být modul `imklog` čtoucí zprávy z kernelu nebo `omfwd` umožňující posílat zprávy skrze protokoly UDP nebo TCP. Je nutno poznamenat, že některé moduly nejsou ve výchozím nastavení kompilovány spolu s `Rsyslogem` a je proto nutno přidat příkazu `compile` parametr `--enable-<název_modulu>`.

```

set $!Promenna="hodnota";
unset $!Promenna;

```

4. NÁVRH A REALIZACE

Pomocí klíčových slov `set` a `unset` je možné deklarovat a rušit proměnné. Ty začínají znaky `$!` a příkaz je nutno ukončit středníkem.

```
template (name="template-name" type="string" string="<%pri%>%  
TIMESTAMP:::date-rfc3164% %$!macaddr% %syslogtag% id=%$!  
counter%%msg%\n"  
)
```

Šablony definují podobu zpráv. Do proměnné `string` lze vložit libovolné textové řetězce. Proměnná mezi dvojicí znaků procento je nahrazena jejich obsahem. Použity mohou být vlastní lokální proměnné a nebo tzv. replacement proměnné, vztahující se typicky ke zprávě, ve kterých je uložena např. severita nebo například čas odeslání. Nejdůležitější je proměnná `msg`, kde je uložen text zprávy.

```
action (template="template-name" type="omfwd" Target="192.168.1.10"  
Port="5514" Protocol="udp" )
```

Pomocí akcí se volají jednotlivé moduly. Ty mohou mít různé parametry, které upravují jejich chování. V ukázce výše se volá modul `omfwd` sloužící k posílání zpráv skrze síť. Parametry specifikují konkrétní IP a port sítě, typ protokolu a název šablony, která upravuje formát zpráv.

```
if $programname == 'solid' and $syslogseverity > 5  
then {  
    #some action  
    stop  
}
```

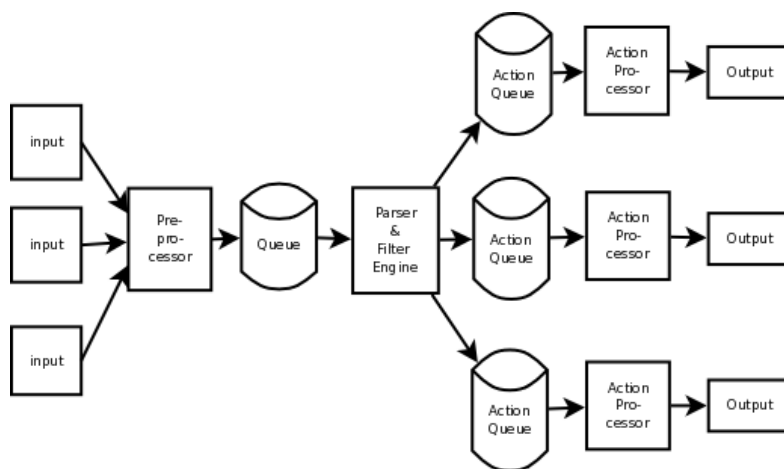
Použití podmínky `if` je zřejmé z ukázky výše. V podmínce `if` se obvykle volají různé akce.

Veškerý text za symbolem „`#`“ až do konce řádky je považován za komentář.

Nyní po obeznámení se základní syntaxí Rainerscriptu je možno popsat fungování zachytávání jednotlivých zpráv. Ty imaginárně putují skrze kód `rsyslog.conf` odshora dolů, kde prochází skrze podmínky a akce (které je eventuelně upravují nebo např. zapisují do souborů) a to až do doby, než narazí na klíčové slovo `stop` nebo na poslední řádek souboru.

Rsyslog Queues

Princip fungování front v Rsyslogu je znázorněn na obrázku 4.1. Zprávy (ať už lokální nebo ze vzdáleného zdroje putující do Rsyslogu např. skrze UDP) vstupují do Rsyslogu na obrázku zleva, kde jsou předzpracovány a putují do Main Queue. V dalším kroku jsou zprávy tříděny podle pravidel specifikovaných v `rsyslog.conf` a putují do jednotlivých Action Queues. Následně proběhne finální zpracování zpráv a opouštějí Rsyslog skrze výstupní modul ať už zápisem do lokálního souboru nebo např. předáním jinému procesu.



Obrázek 4.1: Tok zpráv v Rsyslogu [15]

Všechny zprávy tedy projdou nejprve skrze Main Queue a poté skrze libovolný počet Action Queues. Je nutno zmínit, že každá akce obsahuje vlastní frontu, ve které zprávy čekají na zpracování. V případě, že v akci žádná fronta žádoucí, stačí nastavit typ fronty jako Direct mode.

Existují totiž čtyři módy front. Speciálním (a výchozím) je Direct mode. Direct Queues ve skutečnosti nejsou frontami. Neobsahují totiž žádný buffer a zprávy rovnou přeposílají dál. Vhodné jsou například při zápisu logů do souboru na lokální úložiště.

Druhým typem jsou In-Memory Queues, které udržují zprávy v bufferu v operační paměti. Výhodou je vysoká rychlost a nevýhodou zase hrozba ztráty dat v případě neočekávaného pádu systému. Existují dva podtypy In-Memory front. Můžou být implementovány formou spojového seznamu (LinkedList queue) nebo za použití pole ukazatelů na zprávy ve frontě (FixedArray) s předem pevně určeným počtem prvků pole. FixedArray mode je nejrychlejší ze všech módů a je vhodný pro případy, kdy se očekává pouze malý počet zpráv ve frontě. Naproti tomu LinkedList Queue dynamicky alokuje místo v paměti pro každou zprávu a je výhodný například na serverech, kde se očekává velký tok zpráv.

Třetí typ front představují Disk Queues, které jak z názvu vyplývá používají disk jako vyrovnávací paměť. Využití najdou v případech kdy si žádáme vysokou soplehlivost. Jejich nevýhodou je pomalost limitovaná rychlostí perzistentních úložišť.

Disk-Assisted Memory Queues fungují stejně jako In-Memory Queues a navíc v případě potřeby (kterou může být nedostatek místa v paměti nebo očekávané náhle vypnutí systému) dokáží obsah fronty nebo její část přesunout na disk. Používají chytrý algoritmus, který při malém vytížení vůbec nepoužívá Disk Queues. V případě vyčerpání kapacity In-Memory fronty se

algoritmus postará o přesun části zpráv do diskové fronty.

Rate-limiting za použití Rsyslog Queues

Původní řešení trápil neduh, kdy k vyprázdnění bufferu docházelo až s nově příchozí zprávou, což mělo za následek, že v případě zaplněného bufferu a následující delší odmlky nově příchozích zpráv mohlo dojít k velkému zpoždění odeslání zpráv v ten moment uložených v bufferu. Tento problém se nasazením Rsyslogu a použitím jeho sofistikovaných front vyřeší. V této kapitole je popsána implementace fronty, která má za cíl škrtit tok odchozích zpráv podle stejných kritérií, jako tomu bylo v původním řešení.

Původní řešení bylo postavené na démonu BusyBox Syslogd doplněné o vlastní implementaci rate-limitingu zpráv a fungovalo následovně. Všechny příchozí zprávy putovaly do fronty o fixní velikosti 128 kB. V případě, kdy tok odchozích zpráv překročil 150 kbit/s nebo počet zpráv ve frontě přesáhl 800, byly nově příchozí zprávy zahozeny.

Rsyslog neumožňuje nastavení maximálního toku sítě, avšak poskytuje možnost nastavit pro frontu tyto parametry:

Queue.Size: Maximální počet zpráv ve frontě.

Queue.Dequeuebatchsize: Maximální počet zpráv vystupujících z fronty v jedné dávce.

Queue.Dequeueslowdown: Zpoždění (v μs) mezi odesíláním dávek zpráv.

Queue.Size je nově nastaven stejně na 800 zpráv.

Parametru Dequeuebatchsize byla přidělena hodnota 10. Z měření totiž bylo zjištěno, že průměrná velikost UDP paketu se zprávou ze STB má okolo 150 bytů. MTU UDP paketu je 1500 B. Deset 150-ti bytových paketů se zprávami v součtu odpovídá velikosti jednoho paketu běžně putujícího po síti a ty tak zaručeně nezpůsobí přetížení linky vlivem moc velkého objemu dat naráz. Dále je hodnota okolo deseti výhodná z důvodu, že příliš malé Dequeuebatchsize vytěžuje CPU kvůli nutnosti častého vyprazdňování fronty. Naopak vysoké hodnoty parametru Dequeuebatchsize (v řádu stovek) způsobují nežádoucí, nárazové a velké objemy zpráv na lince. Deset tedy bylo zvoleno za kompromis.

Nyní je třeba zjistit jakou hodnotu nastavit parametru Dequeueslowdown. Pomocí níže zobrazených vzorců bylo spočítáno požadované zpoždění v mikrosekundách, aby zátěž linky nepřekročila 150 kbit/s.

$$\text{počet zpráv} = \frac{\text{šířka pásma}}{\text{velikost paketu}} = \frac{150 * 1\,000 [b]}{8 * 150 [b]} = 125$$

$$\text{slowdown} = \frac{1 \text{ sekunda}}{\text{počet zpráv}} = \frac{1\,000\,000}{125} * \text{batchsize} = 8\,000 * 10 = 80\,000 [\mu s]$$

Ještě zbývá zvolit typ fronty, kde kvůli důrazu na nízké využití zdrojů byla zvolena implementace používající fixní pole ukazatelů na zprávy. Výsledný

kód pro akci s In-memory frontou s nastaveným rate-limitingem vypadá následovně:

```
action (Type="omfwd" Target="192.168.1.10" port="5514" Queue.Size="
      800" Queue.Dequeue slowdown="80000" Queue.Dequeuebatchsize="10"
      Queue.Type="FixedArray" )
```

4.2.3 Razítkování zpráv

Požadavkem zadavatele bylo značit všechny zprávy vzestupnou řadou čísel kvůli rozpoznání případného výpadku některých zpráv. Byl pro to použit modul mmsequence [16], který byl nastaven jako je ukázáno v kódu níže.

```
module(load="mmsequence")
action( type="mmsequence"
      from="1"
      to="1048576" # 2^20
      var="$!counter" )
```

Proměnné „\$!counter“ je tak přiřazeno číslo „1“ a s každou další příchozí zprávou se číslo zvýší o „1“ dokud nenarazí na maximální hodnotu specifikovanou parametrem „to“, kdy je číslo vyresetováno na původní hodnotu „1“. Tato proměnná je následně používána v šablonách a je tak připojena k tělu každé zprávy.

4.2.4 Formát logů

Požadavkem zadavatele je, aby zprávy měly identický formát, jako tomu bylo u původního řešení. Důvodem je, že zprávy ze STB sbírá vzdálený server, který očekává daný formát.

Vzor zprávy:

```
May  1 19:04:54 cc-b8-f1-04-17-89 solid: id=8853 Player Time
[00:06:56.96]
```

Formát:

```
<PRI><RFC3164 date> <STB mac address> <component>: <id=NUM> <
message>
```

Je možno si povšimnout, že vzorová zpráva začíná datem a přitom formát zprávy obsahuje na prvním místě číslo PRI a až poté následuje datum. Hodnota PRI totiž není viditelná a slouží logovacímu démonu, který z ní dokáže vypočítat severity a facility zprávy.

4.3 Moduly pro Rsyslog

Rsyslog poskytuje podporu tzv. modulů, které mohou zásadně rozšiřovat jeho množinu funkcí. Tyto moduly jsou obvykle napsány v jednom C souboru. Pro

jejich zprovoznění je třeba náležitě upravit Makefile a zkompileovat Rsyslog se zapnutou podporou pro ně.

Moduly se dělí na 6 základních kategorií, zde zmíním 3 nepoužívanější:

Output Modules: Výstupní moduly umožňují posílat zprávy na různé cíle.

Je tak možno implementovat například modul, který dokáže odesílat zprávy do nějaké exotické databáze, kterou Rsyslog v základu nezná. Jako příklad výstupního modulu zmíním ommail, což je modul sloužící k posílání zpráv skrze mail.

Input Modules: Vstupní moduly, jak název napovídá, umožňují přijímat zprávy z různých zdrojů. Hojně používaný je imklog sbírající zprávy z linuxového jádra nebo imudp přijímající zprávy skrze UDP protokol.

Message Modification Modules: Moduly, které modifikují obsah zprávy.

Jako příklad uvedu v této práci použitý modul mmsequence, který generuje podle zadaných kritérií číselné řady.

V této práci byly naprogramovány dva moduly, oba z kategorie Message Modification Modules. V následujícím textu je popsán jejich účel a způsob použití. Implementaci zde v práci neuvádím, protože se většina kódu skládá z volání Rsyslog API a kód každého z nich obsahuje stovky řádků.

4.3.1 Modul mmdelstr

Tento modul slouží ke smazání zadaného podřetězce z těla zprávy.

Použití:

```
module(load="mmdelstr")  
action(type="mmdelstr" stringtobedeleted="Some string")
```

Modul má pouze jeden parametr „stringtobedeleted“, kterým se specifikuje podřetězec, který má být smazán. V případě neexistence takového podřetězce zůstává zpráva netknutá. V případě výskytu vícero takových podřetězců je smazán pouze první.

4.3.2 Modul mmsevrewrite

Mmsevrewrite dokáže měnit severitu zadané zprávy. Rsyslog v základu funkci s takovou schopností nenabízí, protože se jedná o vzácný požadavek. Modul je použit v této práci pro konverzi zpráv pocházejících z aplikace používající s Rsyslogem jinak nekompatibilní množinu severit.

Použití:

```
module(load="mmsevrewrite")  
action(type="mmsevrewrite" severity="debug")
```

Modul obsahuje pouze jeden parametr „severity“, kterým se určuje nová severita zprávy. V případě zadání neplatné severity Rsyslog zaloguje chybovou hlášku a upravovaná zpráva zůstane v původním stavu.

4.3.3 Postprocessing zpráv

Zadavatel si přeje vytvoření ukázek pravidel pro RainerScript, které budou sloužit jako vzor, podle kterých si sám v budoucnu vytvoří sadu vlastních pravidel.

4.3.3.1 Zahazování zpráv podle typu komponenty

Pravidlo, které rozpozná zprávy podle zadané komponenty a obsahu zprávy a ty následně zahodí.

Implementace pravidla:

```
1 if $programname == 'solid' and \
2 $msg contains "Player_GetState" then
3 {
4     stop
5 }
```

Pokud zpráva pochází z komponenty „solid“ a obsahuje zmíněný podřetězec, klíčové slovo stop okamžitě zahodí v ten moment zpracovávanou zprávu.

4.3.3.2 Převod severit

Některé aplikace běžící na STB používají jinou množinu severit, než s jakými pracuje Rsyslog a navíc nejsou v hlavičce zprávy, nýbrž v jejím těle. Zadavatel proto požaduje severity zpráv extrahovat z jejich těl a nastavit je podle tabulky 4.1.

Tabulka 4.1: Převodní tabulka severit

Portal	Syslog
ERROR	ERR
WARN	WARN
INFO	NOTICE
DEBUG	INFO
TRACE	DEBUG

Implementace pravidla:

```
1 if $msg contains "mTRACE: " then
2 {
3     action(type="mmsevrewrite" severity="debug")
}
```

```
4 }  
5 else if $msg contains "mDEBUG: " then  
6 {  
7     action(type="mmsevrewrite" severity="info")  
8 }
```

Zprávy obsahující nesprávně nastavené severity je možno opravit podle výše zmíněného pravidla. Bylo totiž vyzorováno, že ony postižené zprávy obsahují podřetězec ve formátu „m<SEVERITY>: “. Stačí pak na danou zprávu zavolat akci mmsevrewrite s parametrem severity vyplněným podle převodní tabulky severit.

4.3.3.3 Smazání podřetězce z těla zprávy

Zprávám pocházejícím z komponenty solid je třeba smazat zadaný podřetězec.

Implementace pravidla:

```
1 if $programname == 'solid' and $msg contains  
2     ":[notificationFromPlayer]: INFO: " then  
3 {  
4     action(type="mmdelstr" \  
5         stringtobedeleted=":[notificationFromPlayer]: INFO: ")  
6 }
```

4.4 Vzdálená konfigurace

Požadavkem zadavatele je umožnit změnu nastavení maximální povolené severity jednotlivých komponent. Rsyslog neumožňuje změnu konfiguračního souboru za jeho běhu a proto je nutno implementovat API, které umožní přenastavení konfiguračního souboru a jeho znovunačtení (restartováním Rsyslogu). Změny tímto skriptem způsobené musí být zachovány i po restartu STB.

4.4.1 Návrh API

Skript musí umět nastavit konfiguraci souboru rsyslog.conf tak, aby v případě rozpoznání zprávy od určité komponenty spolu s nastavenou severitou vyšší, než je povolená danou zprávu zahodil. Musí tedy umět v daném skriptu nalézt část kódu, kde se zpracovává zadaná komponenta a tuto část kódu vhodně upravit.

V úvahu připadaly 2 různé přístupy. Je možné při každém zavolání skriptu generovat celý nový konfigurační soubor a nebo pouze části souboru pomocí skriptu měnit. Výhodou prvního způsobu je, že nehrozí nechtěné přepsání jiných částí skriptu, než bylo zamýšleno. Nevýhodou je složitější implementace z důvodu nutnosti při generování nového skriptu zohledňovat předchozí nastavení skriptu. Tedy by bylo nutno z původního skriptu extrahovat nastavení

jednotlivých komponent a to zkombinovat s novým nastavením. A proto byl zvolen 2. způsob řešení s tím, že zadavatel bude poučen o nutnosti dodržovat určité zásady při měnění konfiguračního souboru, aby nemohlo dojít k přepsání nesprávných částí souboru.

4.4.2 Výběr implementačních nástrojů

Jazyky jako Perl nebo Python se jeví jako ideální pro napsávání skriptu na zpracování textu. Tyto ani jiné podobné jazyky ovšem nejsou součástí SDK a zadavatel si nepřeje jejich instalaci z důvodu omezené paměti a nevalného výkonu embedded zařízení. Proto byla zvolena již na STB přítomná BusyBox implementace SHELLu nesoucí název ASH. Ta sice neposkytuje tak elegantní syntaxi a nenabízí tolik schopností jako moderní SHELL-ové jazyky jako např. BASH, ale i přesto je možné jejím použitím spolu se základními UNIX-ovými nástroji API naprogramovat.

4.4.3 Rozhraní skriptu

První možností je napsat API ve formě skriptu, který přijímá jednotlivé názvy komponent a jim příslušící maximální povolenou severitu jako parametry.

```
set_log_verbosity.sh [component1 severity1] ...
```

Jako druhá možnost připadá v úvahu skript, který čte seznam jednotlivých komponent a maximálních povolených severit ze souboru, který má následující formát:

```
component1  = DEBUG
componentXY = INFO
...
DEFAULT    = INFO
```

V implementaci byl nakonec použit první způsob pro jeho jednodušší obsluhu a navíc s jeho použitím pro zadavatele odpadá starost o další konfigurační soubor.

4.4.4 Implementace API

Skript na vstupu očekává sudý počet parametrů, kde každý sudý parametr je název komponenty a liché parametry slouží pro definování severit pomocí jejich číselných hodnot. Skript vyhledá v konfiguračním souboru rsyslog.conf řádek s danou komponentou a přenastaví maximální povolenou severitu. Skript kontroluje správnost vstupních parametrů pro zabránění neočekávaného chování programu.

Testování

V této kapitole jsou předvedeny tři typy testů. Nejprve se sleduje využití linky sítě při simulaci běžného provozu STB. Druhý test ověřuje funkčnost rate-limitingu zpráv. A v posledním testu se zkoumá vytížení prostředků STB za běžných i extrémních podmínek.

Vždy je porovnáváno nové řešení postavené na Rsyslogu oproti původnímu a následuje diskuze.

5.1 Testovací prostředí a nástroje

5.1.1 WireShark

Tento nástroj slouží jako analyzátor síťového provozu. Dokáže v reálném čase zachytávat informace o paketech putujících po síti a ty následně vyzobrazit v grafu, čehož je v oddílech 5.1 a 5.2 využito.

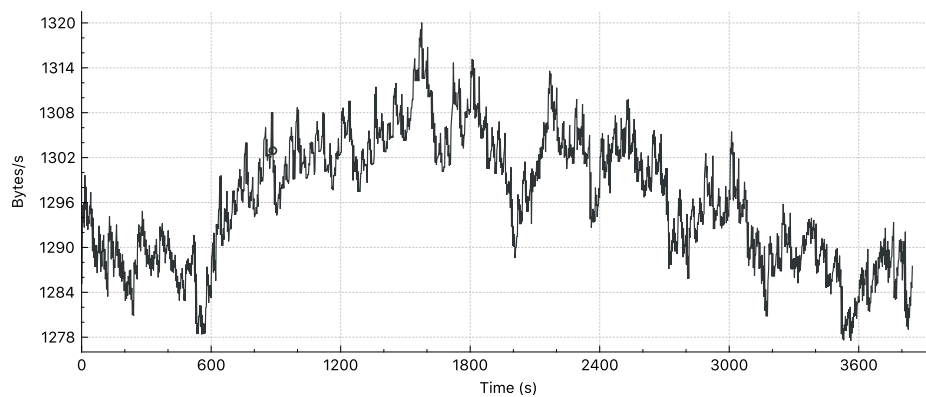
5.1.2 Testovací infrastruktura

Pro účely testování byla nakonfigurována lokální síť jakožto simulace reálných serverů provozovatele. Jako server pro zachytávání zpráv z STB byl použit notebook Apple Macbook Pro, na kterém byl nakonfigurován logovací démon Syslog-ng.

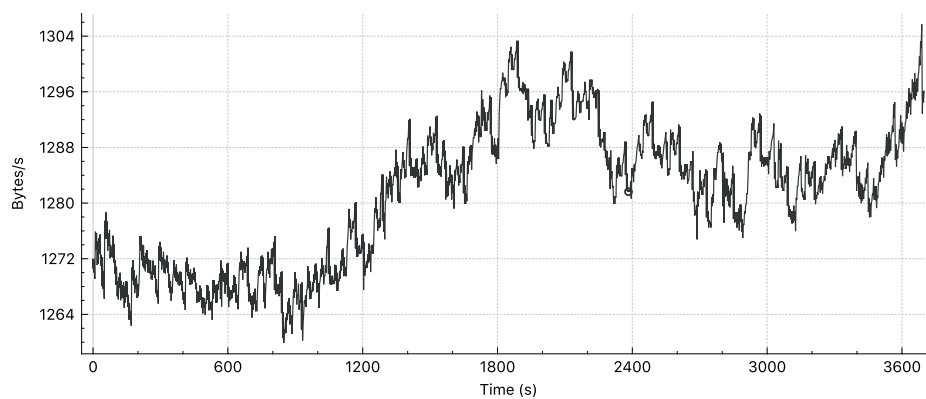
5.2 Test využití linky zprávami za běžného provozu

V tomto testu je srovnáváno vytížení linky za použití nového a starého logovacího řešení při běžném provozu STB.

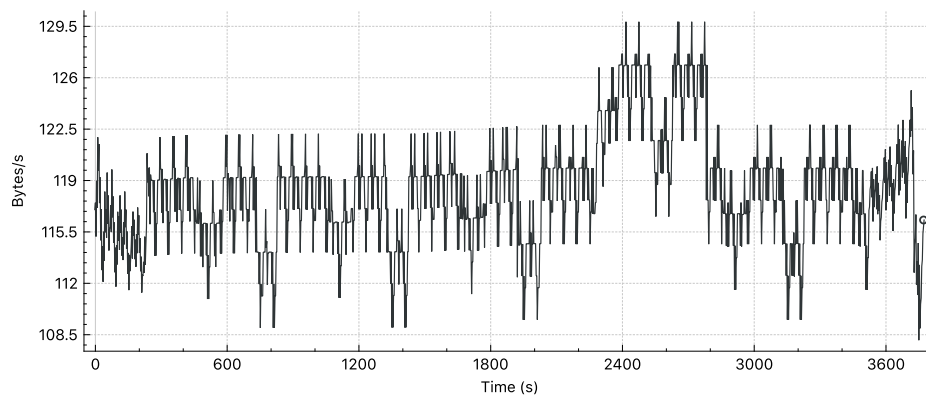
5. TESTOVÁNÍ



Obrázek 5.1: Využití linky v původním řešení



Obrázek 5.2: Využití linky v novém řešení



Obrázek 5.3: Využití linky v novém řešení (filtrování zpráv aplikace solid)

Zhodnocení

Jak je vidět z prvních dvou grafů (obr. 5.1 a 5.2), nové řešení postavené na Rsyslogd přijíma méně logů. Je to způsobeno jeho konfigurací, v níž je nastaveno pravidlo (viz kapitola 4.3.3.1), kterým se zahazují určité nepotřebné zprávy. Na STB totiž běží aplikace, již zadavatel nemá možnost upravit, která loguje mimo jiné i nepotřebné zprávy. V původním řešení podobné pravidlo nebylo možné jakkoliv nastavit. Dle Wiresharku tak nové řešení v průměru přijíma 8,6 paketů za sekundu oproti 9 paketům za sekundu v původním řešení.

Na třetím grafu (obr. 5.3) je pro zajímavost znázorněno využití linky při použití nového řešení po nastavení filtrace zpráv od komponenty solid. Chci tím demonstrovat, že zadavatel má v novém řešení možnost si upravit filtrování zpráv podle různých kritérií a může tak razantně snížit tok zpráv na síti.

5.3 Test rate-limitingu zpráv

Jak bylo zjištěno v minulém testu, zprávy generované set-top boxem tvoří tok o přibližně deseti tisících bitech za sekundu. Limit toku je nastaven na 15-ti násobek této hodnoty a tak by se v reálných podmínkách neměl nikdy aktivovat. Nicméně zadavatel rate-limiting požaduje jako pojistku pro případ, kdy by například vlivem chyby některá z komponent začala generovat nepřiměřené množství zpráv a nedošlo tak k zahlacení ať už sítě na straně uživatele STB nebo serverů provozovatele.

Pro ověření funkčnosti rate-limitingu byl proto sestrojen následující skript.

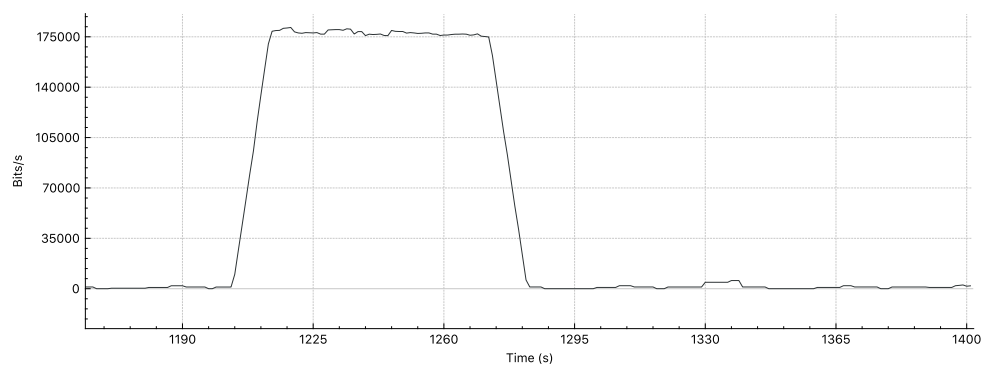
5.3.1 Skript pro otestování rate-limitingu

Účelem skriptu je generovat takové množství zpráv, aby na síti vznikl tok přesahující 150 000 b za sekundu a tedy aby logovací démon byl nucen zprávy bufferovat.

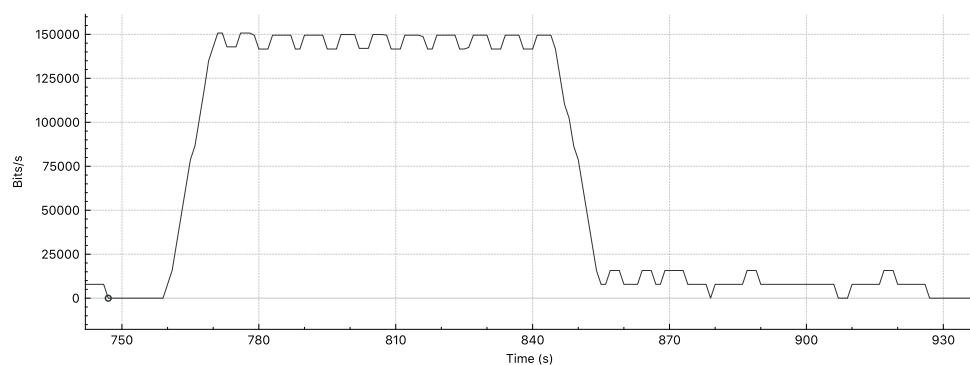
Skript je řešen tím způsobem, že je v deseti paralelních instancích zavolán for-cyklus, každý generující po tisíci zprávách. Zprávy mají konstantní velikost 60 B, což odpovídá délce průměrné zprávy generované set-top boxem.

```
1 gen_messages() {  
2   for i in $(seq 1000); do  
3     logger "Lorem ipsum dolor sit amet, consectetur adipiscing  
4       elit."  
5   done  
6 }  
7 for i in $(seq 10); do  
8   gen_messages &  
9 done
```

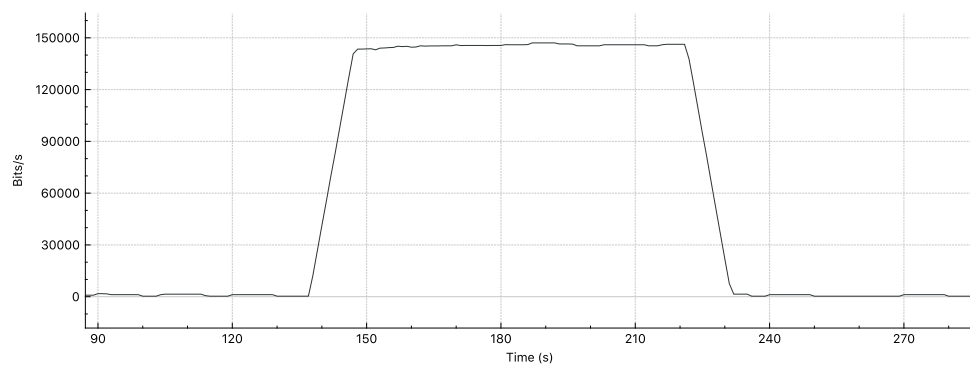
5. TESTOVÁNÍ



Obrázek 5.4: Využití linky při extrémním zatížení bez rate-limitingu



Obrázek 5.5: Využití linky v původním řešení při extrémním zatížení



Obrázek 5.6: Využití linky v novém řešení při extrémním zatížení

5.3.2 Zhodnocení

První graf (obr. 5.4) znázorňuje kolik bitů textu za sekundu daný skript ve skutečnosti generuje. Graf vznikl použitím Rsyslogu s deaktivovaným Rate-limitingem zpráv. Zbylé dva grafy (obr. 5.5 a 5.6) pouze demonstrují správnou funkčnost rate-limitingu. Tedy, že tok bitů za sekundu nepřekročí hranici 150 000.

5.4 Test vytížení systému

Protože embedded zařízení neoplývají vysokým výkonem a ani v této práci použitý STB není výjimkou, je nutno logovací řešení podrobit testům a zjistit jeho chování pod extrémním zatížením, zda nehrozí jeho přetížení a z toho vyplývající nestabilita. Dále je otestováno i vytížení systému za běžných podmínek, kde je porovnáno s původním řešením. V testech se zkoumá vytížení CPU a paměťová náročnost.

Před samotným testováním je ale nutno čtenáře seznámit se základními pojmy týkajícími se měření výkonosti CPU v Linuxových systémech [17].

5.4.1 Terminologie

Clock tick: Clock tick je jednotka pro měření CPU času.

CLK_TCK: Makro (může mít i jiný název jako např. CLOCKS_PER_SEC) definující počet clock ticků za sekundu. Na našem STB má hodnotu 100.

CPU time (CPU čas): Představuje dobu, kterou procesor strávil vykonáváním procesu vyjádřenou počtem clock ticků nebo v sekundách získaných vydělením clock ticků hodnotou makra CLK_TCK.

User time: CPU čas strávený vykonáváním procesu v „user režimu“. V tomto režimu proces nemá oprávnění provádět privilegované operace které mohou ovlivnit celý systém.

System time: CPU čas strávený vykonáváním instrukcí procesu v „kernel režimu“. V tomto režimu jádro systému provádí privilegované operace vyžádané procesem.

Real time: Čas doby běhu procesu v sekundách. Jedná se o absolutní dobu běhu programu a je v ní tedy i zahrnuta doba, kdy byl proces blokován nebo kdy se dělil o prostředky CPU s jinými procesy.

5.4.2 VFS /proc

V Linuxových systémech existuje virtuální souborový systém /proc [18], který v jednotlivých podsložkách a souborech shromažďuje informace o HW systému a procesech v něm běžících.

Informace o stavu procesu shromažďuje soubor /proc/<pid>/stat/ (pid značí číselný identifikátor zkoumaného procesu), který je ve formě textového

5. TESTOVÁNÍ

řetězce obsahujícího informace o procesu. Pro tuto práci důležité informace z tohoto souboru jsou zobrazeny a popsány v následující tabulce. Čísla v závorkách představují jejich pozice v řetězci.

utime: CPU čas strávený vykonáváním procesu v user režimu. (14)

stime: CPU čas strávený vykonáváním procesu v kernel režimu. (15)

cutime: CPU čas strávený čekáním na potomky procesu v user režimu. (16)

cstime: CPU čas strávený čekáním na potomky procesu v kernel režimu. (17)

starttime: CPU čas vyjadřující moment startu procesu. (22)

5.4.3 Vytížení CPU logovacím démonem při velké zátěži

V tomto testu je měřeno procentuální vytížení CPU logovacím démonem při extrémních podmínkách. Ty jsou simulovány skriptem z kapitoly 5.2.1, který generuje velké množství zpráv a zahlcuje tak logovacího démona.

Skript pro získání vytížení CPU

Skript měří počet clock ticků sledovaného procesu po dobu, kdy je zatížen zpracováním zpráv z paralelně spuštěného skriptu na generování zpráv z kapitoly 5.2.1. Poměr takto získaných clock ticků ku celkovému CPU času skriptu `genMessages.sh` potom představuje procentuální vytížení procesoru daným logovacím démonem.

```
1 sfile=/proc/$1/stat
2 if [ ! -r $sfile ];
3   then echo "pid $1 not found in /proc"; exit 1;
4 fi
5 start=$(cat $sfile | awk '{sum=$14+$15+$16+$17; print sum}')
6 time ./genMessages.sh 2> timeOut && \
7 stop=$(cat $sfile | awk '{sum=$14+$15+$16+$17; print sum}')
8
9 sysTime=$(cat timeOut | grep sys | awk '{print $3}' | tr -d 's')
10 userTime=$(cat timeOut | grep user | awk '{print $3}' | tr -d 's')
11 rm -rf timeOut
12 clockTicks=$((stop-start))
13
14 getMilisec()
15 {
16   sec=$(echo $1 | cut -d'.' -f1)
17   miliSec=$(echo $1 | cut -d'.' -f2)
18   echo $((miliSec*10+sec*1000))
19 }
20
21 sysTimeMiliSec=$(getMilisec "$sysTime")
22 userTimeMiliSec=$(getMilisec "$userTime")
23 timeMiliSec=$((sysTimeMiliSec+userTimeMiliSec))
24
25 cpuUsage=$((clockTicks*1000*1000/timeMiliSec))
26 echo CPU Usage: $cpuUsage
```

Výsledky měření

Vítězně z tohoto testu jednoznačně vychází Syslogd, který využívá pouhé 3,12 % CPU výkonu. Oproti tomu Rsyslog spotřebuje skoro čtyřnásobek prostředků, celkem 11,9 % CPU výkonu. Výsledek nijak nepřekvapuje, jelikož Syslogd filtruje zprávy pouze na základě severity a facility. Rsyslog zkoumá více parametrů zprávy (v některých případech dokonce analyzuje samotný text zpráv, což je výpočetně náročná operace) a navíc pro každou zprávu volá externí modul mmsequence.

Skoro dvanáctiprocentní vytížení výkonu CPU Rsyslogem je však v pořádku a to z důvodu, že zbylé procesy běžící na STB vytěžují CPU minimálně (dohromady v jednotkách procent) a tak STB i v této situaci běží naprosto nerušeně.

Nutno dodat, že situace, která byla tímto testem simulována, by v reálném nasazení neměla nikdy nastat.

5.4.4 Dlouhodobé vytížení CPU logovacím démonem

V tomto testu je zkoumáno využití CPU jednotlivými logovacími démony za běžného provozu STB.

Skript

Dlouhodobé procentuální vytížení CPU logovacím démonem je získáno vydělením počtu clock ticků démonu dobou v sekundách, po kterou byl spuštěn.

```

1 sfile=/proc/$1/stat
2 if [ ! -r $sfile ]; then
3     echo "pid $1 not found in /proc"; exit 1;
4 fi
5
6 totalTime=$((cat $sfile | awk '{sum=$14+$15+$16+$17; print sum}'))
7 upTime=$((cat /proc/uptime | tr '.' ' ' | awk '{print $1}'))
8 startTime=$((cat $sfile | awk '{print $22}'))
9 seconds=$((upTime-(startTime/CLK_TCK)))
10
11 cpuUsage=$((totalTime*1000/seconds))
12 echo CPU usage: $cpuUsage

```

Aby výsledky testu měly vypovídající výpovědní hodnotu, byl STB s běžícím logovacím démonem nechán zapnutý po dobu jedné hodiny a až následně byl spuštěn skript.

Výsledky měření

Z měření bylo zjištěno, že Syslogd vytěžuje CPU z 0,051 % a Rsyslogu stačí v průměru pouze 0,042 % CPU výkonu. Z tohoto zjištění tak vyplývá, že Rsyslog

je velice nenáročný co se nároků na zdroje CPU týče, jeho chod tak neovlivní ostatní aplikace běžící na STB a zadavatel ho může bez obav nasadit.

5.4.5 Paměťová náročnost

Množství procesem využívané paměti je možné v systémech postavených na linuxovém jádru zjistit např. ze souboru `/proc/<pid>/status`, kde proměnná `VmRSS` představuje velikost reálně používané paměti daným procesem.

Výsledky měření

Využití paměti za obvyklého běhu STB je konstantní na hodnotě okolo 1 300 kB v případě `Rsyslogu` a 1 100 kB u `Syslogd`.

V momentě spuštění skriptu pro generování extrémního počtu zpráv se využití paměti zvedne v obou případech na hodnotu okolo 1 600 kB a tato hodnota zůstává neměnná i nadále po skončení skriptu.

Fakt, že využití paměti po skončení skriptu v případě `Rsyslogu` neklesne je překvapující, protože by se očekávalo, že po vyprázdnění bufferu se místo v paměti uvolní. Jedná se však o navýšení o pouhých 300 kB a proto se zjišťování příčin této práce nevěnuje.

Z výsledků vyplývá, že nové řešení spotřebovává takřka identické množství paměti jako řešení původní. Využití paměti ani během testování extrémních podmínek nepřesáhlo 2 MB a vzhledem k tomu, že velikost operační paměti je 512 MB, není třeba se obávat jejího nedostatku.

Závěr

Motivací pro vznik práce byl požadavek na přepracování logovacího řešení STB, aby zadavateli bylo umožněno měnit nastavení pro filtraci zpráv za cílem snížení objemu odesílaných zpráv na servery.

Z analýzy vyplynulo jako nejvhodnější řešení nahrazení stávajícího démonu BusyBox Syslogd modernějším démonem Rsyslog spolu s implementací podpůrných skriptů a modulů pro Rsyslog.

Bylo tak naprogramováno API pro vzdálenou konfiguraci, jež dovoluje technikům měnit konfiguraci logování. V práci dále vznikly dva moduly pro Rsyslog. Jeden z nich (mmsevrewrite) slouží ke změně severit zpráv, jež je chybnější a přitom žádanou funkcí Rsyslogu [19] a proto je v plánu ho v budoucnu nabídnout k přijetí do samotného projektu Rsyslog.

Touto prací dostal zadavatel do rukou nástroj, kterým může díky širší podpoře filtrovacích pravidel než tomu bylo v původním řešení razantně snížit objem zpráv putujících na servery.

V průběhu realizace bylo odhaleno nedůsledné označování zpráv generovaných aplikacemi zadavatele nesprávnou severitou. Po přiřazení správných severit dodané řešení umožní dosáhnout ještě lepšího poměru vyfiltrovaných zpráv.

Veškerý kód napsaný v této práci byl psán s ohledem na přenositelnost a proto je možné ho nasadit na různých typech set-top boxů s OS Linux.

Díky použití Rsyslogu může zadavatel v budoucnu zvážit nahrazení stávajícího transportního protokolu UDP pro Rsyslog na míru navrženým protokolem RELP [20], který zaručuje spolehlivý přenos zpráv a umožňuje jejich šifrování. Tato práce se jím nezabývala z důvodu nutnosti změny infrastruktury.

Literatura

- [1] EKT DID7006. *ExploreDoc* [online]. 2015 [cit. 2016-04-26]. Dostupné z: <http://exploredoc.com/doc/3174828/model-did7006-high-definition-ott-stb>
- [2] Deveriya, A.: An Overview of the syslog Protocol. *Network Administrators Survival Guide*, 2005: s. 181–184, ISBN: 978-1-58705-211-8.
- [3] Lonvick, C.: The BSD syslog Protocol. *IETF Tools Pages* [online]. 2001 [cit. 2016-04-26]. Dostupné z: <https://tools.ietf.org/html/rfc3164>
- [4] Gerhards, R.: Transport Layer Protocol. *IETF Tools Pages* [online]. 2009 [cit. 2016-04-26]. Dostupné z: <https://tools.ietf.org/html/rfc5424#page-7>
- [5] Lonvick, C.: Transport Layer Protocol. *IETF Tools Pages* [online]. 2001 [cit. 2016-04-26]. Dostupné z: <https://tools.ietf.org/html/rfc3164#page-5>
- [6] Gerhards, R.: Syslog Message Format. *IETF Tools Pages* [online]. 2009 [cit. 2016-04-26]. Dostupné z: <https://tools.ietf.org/html/rfc5424#page-8>
- [7] The syslog-ng Open Source Edition 3.7 Administrator Guide. *Balabit* [online] [cit. 2016-05-14]. Dostupné z: <https://www.balabit.com/sites/default/files/documents/syslog-ng-ose-latest-guides/en/syslog-ng-ose-guide-admin/html-single/index.html#modules>
- [8] RSyslog - Features. *RSyslog* [online] [cit. 2016-04-27]. Dostupné z: <http://www.rsyslog.com/doc/features.html>
- [9] Modules. *RSyslog* [online] [cit. 2016-05-14]. Dostupné z: <http://www.rsyslog.com/doc/v8-stable/configuration/modules/index.html>

- [10] Compatibility Notes for rsyslog. *RSyslog* [online] [cit. 2016-04-30]. Dostupné z: <http://www.rsyslog.com/doc/v8-stable/compatibility/v3compatibility.html>
- [11] syslog.conf(5). *Linux man pages* [online] [cit. 2016-04-30]. Dostupné z: <http://linux.die.net/man/5/syslog.conf>
- [12] RainerScript. *RSyslog* [online] [cit. 2016-04-30]. Dostupné z: <http://www.rsyslog.com/doc/rainerscript.html>
- [13] The Property Replacer. *RSyslog* [online] [cit. 2016-04-30]. Dostupné z: http://www.rsyslog.com/doc/v8-stable/configuration/property_replacer.html
- [14] RainerScript formal definition. *RSyslog* [online] [cit. 2016-04-30]. Dostupné z: http://www.rsyslog.com/doc/rscript_abnf.html
- [15] Rsyslog Queues. *RSyslog* [online] [cit. 2016-05-01]. Dostupné z: http://www.rsyslog.com/doc/v8-stable/whitepapers/queues_analogy.html
- [16] Number generator and counter module. *RSyslog* [online] [cit. 2016-05-01]. Dostupné z: <http://www.rsyslog.com/doc/mmsequence.html>
- [17] Kaufmann, M.: Measuring Performance. *Computer Organization and Design: The Hardware/Software Interface*, 2008: s. 30–38, ISBN: 9780080922812.
- [18] The /proc filesystem. *Linux Kernel Archives* [online] [cit. 2016-05-10]. Dostupné z: <https://www.kernel.org/doc/Documentation/filesystems/proc.txt>
- [19] Mailing List Archive: Change message priority/severity. *Lists Adiscon* [online] [cit. 2016-05-11]. Dostupné z: <http://lists.adiscon.net/pipermail/rsyslog/2014-April/037273.html>
- [20] The Reliable Event Logging Protocol. *RSyslog* [online] [cit. 2016-05-14]. Dostupné z: <http://www.rsyslog.com/doc/relp.html>

Seznam použitých zkratek

API Application Programming Interface

ASH Almquist Shell

CPU Central Processing Unit

DMD Download manager daemon

GPU Graphics processing unit

IPTV Internet Protocol Television

OS Operating system

PC Personal computer

PID Process ID

MTU Maximum Transmission Unit

RAM Random Access Memory

RFC Request for Comments

RSS Resident Set Size

SDK Software Development Kit

STB Set-top Box

VFS Virtual File System

Obsah přiloženého CD

	readme.txt.....	stručný popis obsahu CD
	exe	adresář se spustitelnou formou implementace
	src	
	impl.....	zdrojové kódy implementace
	thesis	zdrojová forma práce ve formátu L ^A T _E X
	text	text práce
	thesis.pdf	text práce ve formátu PDF
	BP_Vavricka_David_2016.pdf	text práce ve formátu PDF