VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

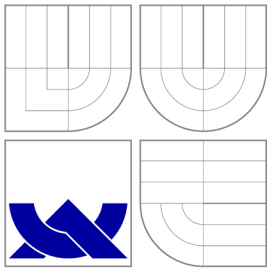
CLUSTEROVÉ ŘEŠENÍ OPENVPN PRO BEZVÝPADKOVÝ PROVOZ

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

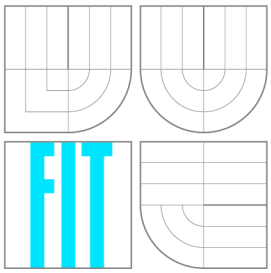
AUTOR PRÁCE
AUTHOR

JIŘÍ DOKOUPIL

BRNO 2013



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

CLUSTEROVÉ ŘEŠENÍ OPENVPN PRO BEZVÝPADKOVÝ PROVOZ

CLUSTER SOLUTION FOR HA OPENVPN

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JIŘÍ DOKOUPIL
AUTHOR

VEDOUCÍ PRÁCE — Ing. PATRIK HALFAR
SUPERVISOR

BRNO 2013

Abstrakt

Cílem této bakalářské práce je analyzovat možnosti běhu OpenVPN v clusteru a takové řešení implementovat. Nejprve se zabývá analýzou stávajících technologií, možnostmi přístupu k této problematice. Následně je pak rozvedeno navržené řešení, kde je následně popsána implementace. Nakonec jsou zhodnoceny dosažené výsledky, převážně formou testů implementovaného řešení.

Abstract

The aim of this bachelor's thesis is to analyze the possibilities for running the OpenVPN daemon in a cluster and to implement such a solution. At first, the thesis analyzes current technologies and possible approaches to this matter. Next there's one possible solution described more and implemented. In the end there are results being analyzed, mostly by describing the tests, that took place with the implemented solution.

Klíčová slova

OpenVPN, cluster, high-availability, corosync, počítačové sítě, tunel, síťový most, teleworker, OpenSSL, TLS, jazyk C, ISO/OSI

Keywords

OpenVPN, cluster, high-availability, corosync, computer networks, tunnel, network bridge, teleworker, OpenSSL, TLS, C language, ISO/OSI

Citace

Jiří Dokoupil: Clusterové řešení OpenVPN pro bezvýpadkový provoz, bakalářská práce, Brno, FIT VUT v Brně, 2013

Clusterové řešení OpenVPN pro bezvýpadkový provoz

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Patrika Halfara. Konzultace v technických záležitostech poskytl pan Ing. Vojtěch Drahoš. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal

.....
Jiří Dokoupil
10. května 2013

Obsah

Kapitola 1

Úvod

S počítačovými sítěmi, ať si to uvědomujeme, nebo ne pracujeme každý den. Zajišťování bezchybného provozu se stává stále větší prioritou. Projekt OpenVPN přináší multiplatformní řešení Virtual Private Network (dále jen VPN) s *enterprise-level* bezpečností a velkou variabilitou možných konfigurací. Ačkoliv je tento koncept schopen provozu v produkčním prostředí, není odolný vůči selhání hardware, výpadkům sítě a podobným nečekaným událostem. Tato práce se zaměřuje především na možnost běhu OpenVPN serveru jako clusterové služby, obzvláště pro bezvýpadkový (*high-availability*) provoz.

1.1 Cluster

Co to vlastně je *cluster*? Odpověď není zcela triviální. Otevřená encyklopedie Wikipedia definuje *cluster* takto [3]:

> Počítačový cluster (anglicky computer cluster) je seskupení volně vázaných počítačů, které spolu úzce spolupracují, takže navenek mohou pracovat jako jeden počítač. Obvykle jsou propojeny počítačovou sítí. Clustery jsou obvykle nasazovány pro zvýšení výpočetní rychlosti nebo spolehlivosti s větší efektivitou než by mohl poskytnout jediný počítač, přičemž jsou levnější než jediný počítač o srovnatelné rychlosti nebo spolehlivosti.

Cluster je zjednodušeně řečeno množina počítačů spojená počítačovou sítí. Tato definice není ovšem ani zdaleka blízko podstatě pravého významu.

- Sdílení dat
- Výpadek jednoho, či více počítačů
- Rozhodování „kdo bude co počítat“

Toto jsou zcela elementární problémy[6], které je napřed třeba vyřešit, než můžeme cluster začít používat. Nicméně smyslem clusteru nemusí být pouze zvýšení výpočetního výkonu, pomocí tohoto přístupu lze zvýšit odolnost proti možné poruše hardware, či zajistit zvýšenou datovou propustnost. Do kategorie poruch hardware je pro naše účely myšleno i ztráta konektivity některého uzlu, dočasné přerušení síťové komunikace apod. Právě zvýšením odolnosti proti selhání se tato práce převážně zabývá.

Existují situace, které při běhu clusteru mohou nastat, a které mohou mít fatální následky. Mezi takové situace patří tzv. split-brain situace, kdy se cluster rozdělí na dva, či

více celků, z nichž každý celek si myslí, že právě on přežil a ostatní jsou mrtví, ovšem mýlí se, pouze nemá informace o tom, zda ostatní uzly také fungují, či nikoliv. Tato situace je velmi nepříjemná už z toho důvodu, že může dojít k nekonzistenci dat, což zapříčiní to, že nebude možno rozhodnout, který cluster má aktuálnější data. Po té musí nastoupit člověk, který musí tuto situaci vyřešit, ať už jakkoliv. Tento problém ovšem u naší služby OpenVPN řešit nemusíme, protože nebude třeba ukládat žádná statická data. Framework corosync poskytuje jednoduché možnosti řešení problému *split-brain* situace. A právě toto nám vělmi usnadní práci.

1.2 Grid

Dalším uskupením, velmi podobným clusteru, je *grid*. Wikipedie definuje grid takto [3]:

> Gridový cluster (anglicky Grid cluster) je složen z nezávislých počítačů, které jsou typicky určeny pro jinou činnost (například desktop, server) a poskytování výpočetního výkonu pro využití v clusteru je jejich doplňkový úkol (i když z hlediska využití výpočetního výkonu může být situace obrácená).

V některé literatuře najdeme počítačový *grid* jako logicky stejně postavenou jednotku ke clusteru. Jsou ovšem zdroje, které uvádějí *grid* jako podmnožinu clusteru a staví jej na stejnou úroveň jako *High-availability*, *Fault-tolerance*, *Storage* a dalsí typy clusterů. V podstatě lze říci, že grid je seskupení počítačů, jejichž primárním účelem není sloužit jako uzel větší množiny. Této množině poskytuje prostředky pouze v případě, že jsou k dispozici, a neomezí primární funkci takovéhoto uzlu. Lze tedy spekulovat, zda-li se jedná o cluster v pravém slova smyslu, či nikoliv.

1.3 Corosync

Corosync je open-source projekt, který byl odvozen (přesněji řečeno je odštěpnou větví) z projektu OpenAIS. Jeho účelem je vyvíjet a podporovat komunitu v problematice clusteru. Jeho využití je velmi široké, od malého soukromého sektoru až po dnes největší clusterové hráče v komerční sféře. Tento projekt využívá dvou backendů, které mu pomáhají vykonávat svou práci.

- Backend OpenAIS
- Backend Pacemaker

Pacemaker je primárně správce zdrojů (*resource-manager*, který je schopný zjišťovat stav jednotlivých uzlů clusteru, jejich vytížení. Spravuje uživatelem definovaná pravidla pro běh služeb. Na oficiálních stránkách projektu OpenAIS je projekt definován takto [2]:

> OpenAIS is an open implementation of the Application Interface Specification (AIS) provided by the Service Availability Forum (SAForum or SA). SAForum defines a set of services for making highly available applications.

OpenAIS je v podstatě velmi podobný projekt, jako Pacemaker, ovšem je spravován jinou vývojovou skupinou.

1.4 VPN

Žádná z oblastí výpočetních technologií nemá takové množství zkratek, jako počítačové sítě. *VPN* je jedna z nich. Za těmito třemi písmeny se skrývá *Virtual Private Network*, česky soukromá virtuální síť. Tento koncept tvorby počítačových sítí vytváří spojení klient - server a vytváří logický most mezi těmito dvěma body. Běžně se používá například pro zabezpečený přístup na počítače, které nejsou možné fyzicky propojit lokální sítí, a přesto bychom takovouto síť, alespoň logickou, chtěli mezi nimi mít. VPN se zabývá několik projektů, počínaje proprietární Microsoft implementací, až po open source řešení. VPN obecně může být provozováno ve dvou základních režimech.

- na druhé vrstvě (L2) ISO/OSI modelu
- na třetí vrstvě (L3) ISO/OSI modelu

Význam těchto režimů je zjevný, zde se liší rozhodovací logika a adresa, podle které se směruje/přepíná packet. Pro druhou vrstvu to je MAC adresa, zatímco pro třetí vrstvu to je adresa třetí vrstvy. Ta je zpravidla IPv4, nově i IPv6, popřípadě můžou existovat VPN implementace i pro jiné adresy třetí vrstvy. Při použití adresování podle L2 je nespornou výhodou možnost použití libovolného L3 protokolu, a fungují zde L2 broadcast zprávy. Na druhou stranu využití broadcastu nemusí být vždy žádoucí.

Existují dva základní módy, v kterých lze VPN provozovat [4]

- site-to-site
- teleworker

Mód site-to-site funguje pro propojení dvou, či více fyzických sítí do jedné logické. Tyto sítě nemusí být v jedné geografické lokaci. Ostatně tento mód byl navržen pro to, aby takovéto vzdálené sítě spojil. V tomto módu ani jedna instance nevystupuje ani vysloveně v roli serveru, ani vysloveně v roli klienta, všechny takovéto instance se chovají hybridně, na stejné úrovni. Nejčastějším případem je vytvoření takovéhoto tunelu nad druhou vrstvou ISO/OSI modelu, ovšem i varianta nad třetí vrstvou je také možná.

Na druhou stranu mód teleworker je varianta kdy se klienti připojují k serveru a tvoří takto *star* topologii. V počátcích byly takovéto mosty, či tunely používány čistě za účelem logického spojení více míst do jednoho logického. V dnešní době je ovšem velmi žádoucí alespoň základní úroveň zabezpečení, aby případný útočník po získání jakéhokoliv packetu tato data nedekódoval. V tomto ohledu zde nejčastější uplatnění nachází Secure Sockets Layer (SSL), či v jeho nástupce Transport Layer Security (TLS). Oba tyto protokoly zabezpečují šifrovaný přenos na transportní (L4) vrstvě ISO/OSI modelu.

1.5 OpenVPN

Obrázek 1.1: Oficiální logo OpenVPN

OpenVPN je jednou z mnoha implementací tunelu typu VPN. Tento open source projekt začal v roce 2001 James Yonan[4]. Od té doby získal mnohá uplatnění od osobního po korporátní využití. Je distribuován pod licencí GNU GPLv2, což je pro open source projekty typické (ačkoliv existují i jiné licence spadající do kategorie open source). OpenVPN je psán výhradně v jazyku C ve standardu ANSI C89 (*ANSI X3.159-1989*), využívá vlastní garbage collector pro alokaci a dealokaci paměti. V pravém slova smyslu se o garbage collector, tak jak jsme na něj zvyklí např. z jazyků Java, či Python, nejedná. Tento garbage collector neřeší alokace a dealokace za běhu programu, ale pouze zpříjemňuje život programátorovi, který se nemusí příliš zajímat o správu paměti, nicméně všechny alokace a dealokace jsou rozhodnuty již v době překladu. OpenVPN je ze své podstaty single-threaded a single-instance aplikace, úpravou druhé vlastnosti se zabývá tato práce.

Kapitola 2

Teoretický rozbor

2.1 Analýza clusteru

2.1.1 High-availability

High availability (dále jen HA) je takové uspořádání clusteru, která v co největší míře eliminuje možnost výpadku poskytované služby [6]. Tohoto efektu se dosahuje většinou monitoringem stavu ostatních uzlů a v případě přerušení poskytování služby je tato služba spuštěna na jiném stroji. V našem případě bychom měli zajistit, aby poskytovaná služba *OpenVPN server* byla odolná vůči selhání jednotlivých hardwarových součástí, a výpadkům sítě. V případě, že například poskytujeme virtualizační cluster, po výpadku uzlu, na kterém běžel virtuální stroj je tento stroj znovu spuštěn na jiném uzlu, ztratíme ovšem kontext a spuštěné aplikace, navíc budeme muset počkat, až virtuální stroj dokončí nový boot.

2.1.2 Fault-tolerance

Fault tolerance (dále jen FT) je velmi podobné uspořádání clusteru, poskytuje nulovou toleranci vůči výpadku [6]. Pro potřeby této práce jsme si dále budeme definovat, že výpadkem myslíme takové časové přerušení služby, při kterém se nerozváže TCP spojení. Nicméně obecně se u FT netoleruje ani takový výpadek. V případě FT clusteru by všechny uzly měly mít dostatek informací pro plnou obnovu služby v co nejkratším čase. Zpravidla je toto řešeno redundancí. Různé clustery redundantně sdílejí různé množství dat - např. při poskytování virtualizačních služeb lze v reálném čase redundantně sdílet data mezi uzly až na úrovni registrů virtualizovaného procesoru. A v případě výpadku master uzlu je dostupná kopie na některém ze slave uzlů. Nedojde k žádné ztrátě kontextu, spuštěných aplikací apod.

2.2 Analýza OpenVPN jako clusterové služby

OpenVPN je ze své podstaty single-instance daemon. Pokud chceme toto změnit, je třeba zvážit několik variant přístupu k řešení tohoto problému. Je zjevné, že když chceme minimalizovat možnost výpadku služby z důvodu selhání hardware, budeme muset mít alespoň dvě fyzické instance OpenVPN serveru spojené do jedné logické služby. Možných způsobů řešení je několik. V tento moment je ovšem třeba definovat si naše priority a požadavky.

- Požadujeme aby byl cluster odolný vůči výpadkům HW a sítě

- Požadujeme aby se celý cluster zvenku tvářil jako jedna služba, s jednou IP adresou.

Nicméně je třeba uvést, že vzhledem k tomu, že počítačové sítě jsou ze své podstaty nespolehlivým „best-effort“ přenosovým médiem, tak chceme-li zabezpečit spolehlivý přenos, musíme využít některý z mechanismů dostupných protokolů přenosové vrstvy (L4 ISO/OSI). Tudíž si pro naše potřeby můžeme nadefinovat, že výpadek služby budeme považovat za tak dlouhý downtime, že se rozváže TCP spojení, které by bylo navázáno přes VPN. Při přenosu dat nad nespolehlivými protokoly L4 jako jsou UDP, RDP apod. je nutno očekávat, že packety nemusí dojít v takovém pořadí, v jakém byly odeslány, nebo že vůbec příjemci nemusí dojít.

2.3 Definice požadavků

Zadání této práce je na jednu stranu velmi konkrétní, ovšem zamyslíme-li se více nad touto problematikou, mnoho detailů je plně v rukou autora. Proto je vhodné exaktně si definovat jak takovýto náš cluster bude fungovat.

2.3.1 Zapojení sítě

Budeme předpokládat, že jednotlivé uzly clusteru jsou spojeny sítí, která je v jedné L2 broadcastové doméně. Tuto síť budeme považovat za bezpečnou, nebude tudíž potřeba jakkoliv šifrovat, či kontrolovat autenticitu dat, které po této síti chodí. Bezpečnou ji můžeme považovat z toho prostého důvodu, že cluster by měl mít všechny uzly v jedné geografické lokaci, je tudíž možné propojit tyto uzly přímo lokální sítí. Pokud by někdo měl fyzický přístup k těmto strojům, bezpečnost BCN by byla to poslední, o co bychom se měli strachovat.

2.3.2 Definice výpadku

Co je vlastně výpadek? Je to ztráta jednoho packetu? Tyto otázky jsou relativně zrádné. Při běžné komunikaci po síti je reálné, že některé packety nedojdou na místo svého určení, ať už z důvodu špatné technické úrovně sítě, nespolehlivosti přenosového média (jako vysloveně nespolehlivé lze třeba označit bezdrátové sítě), či omezení, která se zakládají na principu zahazování packetů. Takovýmto způsobem to například řeší Frame Relay [5], které implementuje DE (discard eligible) bit, FECN (forward explicit congestion notification) a BECN (backward explicit congestion notification), kdy v případě, že je nastaven DE bit může dojít k úmyslnému zahození packetů z důvodu omezení toku dat. Při úmyslném zahození packetu při nastaveném DE bitu se přijemce o zahození packetu dozví. Z této nespolehlivé povahy sítí lze usoudit, že aplikace s ní počítají, buď požadují spolehlivý přenos a použijou k tomu patřičný L4 protokol, typickým příkladem je TCP. Anebo jim nevadí, že některý packet nepřijde, či že packety dojdou v jiném pořadí, než v jakém byly odeslány. A použijou k tomu L4 protokol, který má méně overheadu, neobsahuje však nástroje pro spolehlivý přenos. Z tohoto prostého důvodu ztráta několika packetů ještě neznamená výpadek služby. Pro naše potřeby si výpadek služby můžeme interně definovat jako čas, po který se nerozváže již navázané TCP spojení.

2.4 Možná řešení

2.4.1 Framework Corosync a plovoucí IP adresa

S využitím mechanismu corosync a plovoucí IP adresy bychom dosáhli toho, že bude možné mít vedle sebe více konfigurací OpenVPN - více subnetů, nicméně bude třeba sdílet kontext mezi jednotlivými uzly clusteru. Kontext v tomto smyslu značí seznam klientů a k nim přiřazený uzel clusteru. Jednotlivé uzly si musí předávat efektivní data tak, aby klient při komunikaci s jiným klientem nepoznal, že data jdou přes více uzlů. Ovšem zde je třeba mít zároveň více IP adres jednoho logického celku. Toto není zrovna elegantní řešení, nicméně, když tento fakt přijmeme jako *nutné zlo*, pak lze toto obejít například pomocí více záznamů v systému DNS. V případě výpadku jednoho, nebo více uzlů přestanou klienti, kteří byli na tento uzel připojeni komunikovat se zbytkem subnetu, než systém corosync zjistí, že došlo k výpadku uzlu a tuto instanci spustí na jiném HW, na který přenese plovoucí IP adresu. Výhodou tohoto řešení je, že když přijde do systému nový klient, není třeba broadcastem zaplavovat ostatní uzly o informací této skutečnosti, každý uzel si udržuje svůj kontext. Pouze v případě, že chce klient komunikovat s jiným klientem, který je v tomto subnetu, uzel se začne dotazovat přes který uzel je schopen tento most uskutečnit. Dalším důsledkem je to, že tento mechanismus je uplatnitelný na routované (L3) VPN. V bridgeované (L2) VPN by toto teoreticky také bylo možné, nutností by zde bylo implementovat routovací protokol nad adresou druhé vrstvy. Na druhe vrstvě se neprovádí směrování (routing), ale přepínání (switching), čili bychom museli implementovat přepínací protokol nad *mac* adresou. Zde by bylo možné inspirovat se protokoly pro práci s 802.1q VLAN sítěmi, jako je VTP pruning [5].

2.4.2 Load balancer a sdílený kontext

Tento princip vzužívá předpokladu, že všechny uzly jsou si rovnocenné a sdílejí stejný kontext, stejná data. V případě, že se připojí nový klient, uzel, který tento požadavek zpracovával tuto informaci rozešle broadcastem na ostatní uzly v clusteru. Pokud chce klient komunikovat s jiným klientem ve stejném subnetu, každý uzel ví na který uzel musí packet přeposlat, aby se takovýto packet dostal k cíli. V tomto uspořádání je zcela nepodstatné s kterým uzlem klient komunikuje, protože každý uzel disponuje stejnými informecemi, jako ostatní uzly. Při odchodu jednoho uzlu pouze přestane tento uzel odpovídat, load balancer na něj přestane přeposílat data a služba funguje dále bez jakéhokoliv výpadku. Ovšem obtíž nastává v případě, že budeme mít OpenVPN s UDP jako L4 protokolem, kde nám load balancer zacne rozhazovat jednotlivé packety na různé uzly a tato část nevyhnutně vyžaduje spojení s pouze jedním uzlem. A to z toho důvodu, že SSL spojení bude komplikované navázat. Nicméně to není neřešitelná situace, inteligentní load-balancery dokáží přeposílat packety na jednu IP adresu, pokud přijdou v určitém (nastavitelném) časovém kontextu.

2.4.3 Active-passive

U tohoto řešení jsem se inspiroval Cisco protokolem HSRP (Hot Standby Router Protocol)[5]. U tohoto protokolu můžou uzly vystupovat buď jako master, nebo jako slave. Master uzel vyřizuje všechny požadavky, v případě, že ostatní uzly zjistí výpadek tohoto uzlu, ten, kdo jako první pošle packet značící *„já jsem master“* se stává novým master uzlem. Při aplikování této metodiky na náš problém bude existovat více instancí OpenVPN, budou mít své plovoucí IP adresy, které budou nastaveny na rozhraní pomocí Corosync. Master uzel bude

vždy informovat ostatní slave uzly o všech událostech, které se staly - například připojil se nový klient, odpojil se klient a podobně. Jednotliví klienti budou poslouchat co se děje a v případě výpadku master instance OpenVPN se stane master uzlem ten, který jako první pošle *„já jsem master“* packet. Při návratu bývalého master uzlu, nebo při příchodu nového uzlu do clusteru se může stát jedna z následujících věcí.

- Nastane nová volba master uzlu podle předem definovaných pravidel
- Nový uzel se automaticky stává slave uzlem

Co se v případě výpadku master uzlu stane by bylo vhodné nechat rozhodnout správce takového systému, protože každé řešení má své výhody za určitých situací.

Toto řešení by vyžadovalo součinnost jiných clusterově orientovaných frameworků, jako je například corosync, protože právě corosync umožňuje snadnou konfiguraci a definici clusterové služby, akcí, které se mají vykonat za určitých situací, řeší problém fencingu, split-brain situací a podobných záležitostí spojených s clustery.

Pokud má master uzel vyšší výpočetní výkon nebo rychlejší připojení k síti, než slave uzly, budeme chtít aby fungoval co nejvíce právě on jako master uzel, tudíž nová volba master uzlu po příchodu nového uzlu do clusteru zde má jasné výhody. Pokud ještě vhodně zvolíme jako metriku pro volbu takovéhoto uzlu právě výkon stroje nebo rychlost připojení, může nám to zajistit provoz na velmi vysoké úrovni. Změna master uzlu obnáší změnu virtuální IP adresy, odeslání gratious ARP packetu [6] a další akce, tudíž je zřejmé, že ač se budeme snažit sebevíc, nebude možné toto provést úplně bez ztráty packetů. Proto pokud mají všechny uzly stejný výpočetní výkon a rychlost připojení k síti, je zbytečné, aby se po příchodu nového uzlu znova volil master uzel a tím vyvolal možný, byť krátkodobý, výpadek.

2.5 Možnosti rekonfigurace OpenVPN za běhu

Pokud máme virtuální síť založenou na systému OpenVPN, čas od času může nastat situace, že potřebujeme na serveru změnit nastavení, uvést toto nastavení v platnost ale zároveň potřebujeme, aby byla tato síť zachována bez výpadku způsobeného restartem OpenVPN serveru. OpenVPN má velmi rozsáhlé možnosti konfigurace, nicméně jsou případy, kdy je restart služby nevyhnutelný, například měníme-li adresní rozsah ve virtuální síti.

Ovšem například změna management portu a adresy, na které server poslouchá je možné změnit bez restartu celé služby. V tomto případě je možno vytvořit nový socket, který bude poslouchat na jiné adrese, popřípadě jiném portu, než doposud a na chod služby jako celku to nebude mít žádný vliv. Problém může nastat v případě, že je někdo k management interface již připojen. V takovém případě lze zaujmout několik postojů k řešení této situace.

- Počkat až se klient odpojí, pak až provést změnu
- Interaktivně se zeptat klienta, jak se má server zachovat, zda-li se má ihned restartovat, nebo zda chce dokončit práci a až poté provést změnu
- Odpojit všechny klienty a změnit nastavení okamžitě

Poslední varianta, ač se může jevit jako *brutální*, či agresivní je ovšem podle mého mínění patrně nejvhodnější ze tří výše uvedených možností. Management interface je určen buď pro GUI, popřípadě jiné rozhraní komunikující se serverem, nebo pro administrátora. Ovšem

právě administrátor je ten, kdo by měl provádět tyto změny konfigurace na OpenVPN serveru. Tudíž můžeme spoléhat, že si je vědom toho, že mu spadne management interface konzole.

Při změně konfigurace za běhu je třeba dbát na ten fakt, že klient si ověřuje určité části nastavení serveru, oproti svým nastavením, zda-li jsou vzájemně kompatibilní. Pokud by se takovéto hodnoty měly změnit, je restart spojení nevyhnutný. Ostatní nastavení, které si klient nekontroluje, lze změnit dynamicky za běhu serveru.

Kapitola 3

Praktický rozbor

V předchozí kapitole byla nastíněna některá možná řešení při přístupu k našemu problému.

OpenVPN plugin

Služba OpenVPN nabízí poměrně široké možnosti zavádění zásuvných modulů. Tato varianta počítá s voláním předem deklarovaných funkcí, které můžeme definovat. Každá takováto funkce je volána při předem definované situaci. Tyto definice situací za nás již udělal tým vývojářů projektu a neexistuje žádný *čistý* způsob editace těchto definic a deklarací.

Modifikace OpenVPN

Druhou možností jak dosáhnout námi žádané funkčnosti je modifikace projektu. Tato varianta obsahuje mnoho úskalí, pravděpodobně obšem bude vhodné vydat se touto cestou, přestože bude nutné řešit mnohé problémy. Hlavním problémem, který se bude muset v tomto případě řešit je asynchronní zasílání zpráv pasivnímu uzlu clusteru.

Management interface

Další možností jak přistupovat k této problematice je komunikace se serverem pomocí *Management interface* OpenVPN, přes které lze serveru předat velké množstvé informací. Nelze ovšem přistupovat do všech interních struktur a nelze ani zaregistrovat backcall volání, která by nám byla schopna vracet data, se kterými bychom mohli pracovat. Velkou výhodou je ten fakt, že by nebylo třeba zasahovat do samotného OpenVPN a šlo by tudíž nasadit na různých, kompatibilních, verzích.

Corosync a plovoucí IP adresa

Toto řešení, jak bylo popsáno výše, je základní variantou clusteru. V tomto módu lze poměrně jednoduše spustit OpenVPN daemon spolu s plovoucí IP adresou. Hlavní nevýhodou je nulové zainteresování slave uzlů do clusteru. Ovšem tento koncept nám může posloužit alespoň jako výchozí bod z kterého můžeme vycházet a některé části budeme moct použít.

Load balancer a sdílený kontext

Tento návrh je zajímavý hlavně z toho důvodu, že je schopen rozkládat zátěž na všechny uzly clusteru. Výsledný výkon celého clusteru je pak reálně opravdu vyšší, než výkon jednoho uzlu. Nutno zajistit časový kontext packetů pro doručení na load-balanceru, protože je nezbytné, aby se SSL/TLS sezení ustanovilo s jedním uzlem, poté lze tento kontext sdílet mezi ostatní uzly. Nevýhodou je opravdu velmi silně vytížená Back Channel Network (BCN). Bylo by vhodné nasadit více vláknové zpracování, alespoň oddělit vlákno zpracovávající komunikaci server-client a vlákno pro komunikaci mezi uzly clusteru.

Active-passive

Velmi podobné řešení komerčně používá společnost Cisco [5], což lze považovat za známku toho, že je tento koncept funkční a ověřený. Pro praktické nasazení ovšem přímo na projektu OpenVPN nebyl vyzkoušen. Můžeme se ovšem teoreticky o tuto myšlenku opřít a využít některé principy.

Kapitola 4

Implementace

Před vlastní implementací by bylo vhodné začít odzačátku a nakonfigurovat, vyzkoušet jednotlivé nástroje, jako je cman, corosync, openais a pacemaker

4.1 Instalace jednoduchého clusteru

Zde budeme koncepčně vycházet z článku *2-node Red Hat KVM Cluster Tutorial* [1], který nám dá rámcoý přehled jak postavit cluster. Tento článek se zaměřuje na virtualizační cluster, což nám zcela nevyhovuje, proto si budeme muset postup poněkud upravit. Začneme volbou operačního systému, tento systém by měl být jednoduchý na správu a instalaci. Volím proto CentOS, protože je velmi dobře spravovatelný a je kompatibilní s balíky od společnosti RedHat. Zcela ideální řešení by bylo nasadit RHEL (Red Hat Enterprise Linux), nicméně zůstaneme v sekci neplaceného software. Instalaci provedeme pod virtualizačním nástrojem VirtualBox od společnosti Oracle. Vytvoříme dva uzly clusteru se zcela výchozím nastavením instalace. Každý uzel bude mít tři síťové karty, jednu pro přístup do inernetu, druhou jako LAN pouze pro komunikaci mezi uzly, třetí pouze pro komunikaci s hostitelským strojem. Tyto uzly připojíme k internetu buď přes NAT, nebo přes bridgeové sdílení NIC hostitelského stroje. Pro druhou síťovou kartu vytvoříme na hostitelském stroji most *br0* a zvolíme „Bridged adapter“ na toto rozhraní. Nakonfigurujeme jednotlivé uzly tak, aby spolu mohly komunikovat jak přes LAN, tak přes WAN.

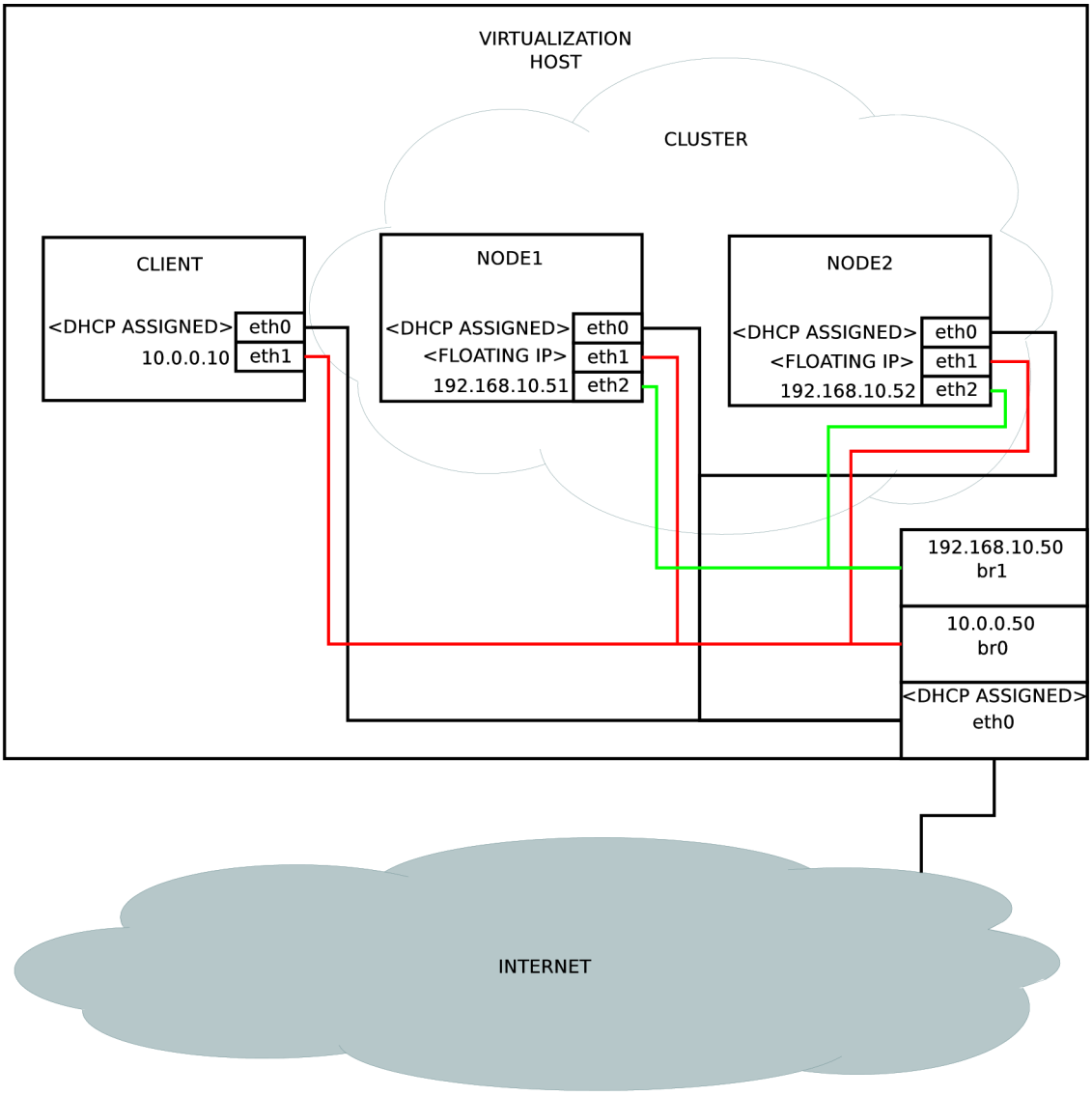
Zezačátku nám postačí pro všechny virtuální stroje základní instalace CentOS. Vše, co budeme potřebovat si nainstalujeme sami.

4.2 Konfigurace uzlů clusteru

Vzhledem k tomu, že se jedná o proof-of-concept zapojení, nebude třeba příliš řešit zabezpečení takovéhoto clusteru. Z tohoto důvodu nastavíme SELinux do permisivního módu a necháme plně otevřený firewall. Dále nainstalujeme cluster software:

```
# yum install cman corosync rgmanager ricci gfs2-utils ntp\
lvm2-cluster syslinux wget gpm rsync
```

Dále je třeba definovat názvy uzlů v souboru */etc/hosts* a zabezpečit výměnu veřejných částí rsa/dsa klíčů mezi uzly, aby mohly uzly spolu komunikovat bez nutnosti zadávat heslo.



Obrázek 4.1: Zapojení clusteru

4.3 Implementace

Nyní už víme, že je potřeba z master uzlu clusteru posílat na slave uzly informace o nových klientech. Víme také, že je třeba učinit některý uzel *master* uzlem, aby mohl posílat informace ostatním uzlům, a aby komunikoval s klienty. Potřebné informace jsou uloženy v několika strukturách, především však *multi_instance*. Je poněkud zvláštní, že tato struktura obsahuje přímo v sobě strukturu *options*, ve které jsou informace z příkazové řádky, konfiguračního souboru a všechny stavové proměnné celé aplikace. Tato data je potřeba hluboko zkopírovat pro každého klienta zvlášť, nicméně tato data nejdou přenést, jsou machine-specific. Pro naše potřeby bude potřeba přenést informace o adrese a portu, na kterém klient komunikuje. Je třeba také zajistit, aby slave uzly nekontrolovaly spojení s klientem. V důsledku toho, že slave uzly nemají přiřazenu plovoucí IP adresu, nemohou správně komunikovat s klienty a jakýkoliv pokus o komunikaci by skončil neúspěchem a zahozením

datové struktury nesoucí informaci o tomto klientovi.

4.3.1 Komunikační protokol

Pro komunikaci mezi uzly bude nejvhodnější použít UDP jako L4 protokol. A to z toho prostého důvodu, protože bude mít nejmenší overhead. A jak bylo vysvětleno výše, není nezbytné zaručit spolehlivý přenos. Pokud klient ztratí spojení se serverem, má vestavěný interval 2 sekundy, po kterém spojení zkusí znovu navázat. Pokud dojde k následující situaci:

- Klient se připojí k Node1 (aktuálně master)
- Master Node1 zašle informaci o novém klientovi Slave Node2
- Při přenosu informace přes síť se tato informace ztratí
- Master Node1 selže
- Slave Node2 se stane Master Node2
- Klient zjistí, že neumí komunikovat se serverem a znovu zahájí pokus o spojení

Pak se až tak moc nestane, klient bude moct komunikovat s výpadkem maximálně 2 sekundy, což není dost k rozvázání TCP sezení. Toto ovšem byla krajní varianta nejhorší přípustné situace. V ideálním případě selhání uzlu by to vypadalo takto:

- Klient se připojí k Node 1 (aktuálně master)
- Master Node1 zašle informaci o novém klientovi Slave Node2
- Slave Node2 tuto informaci přijme a vytvoří si novou strukturu *multi_instance*
- Master Node1 selže
- Slave Node2 se stane Master Node2
- Klient stále komunikuje a nezjistí, že komunikuje s jiným serverem

4.3.2 Komunikace mezi uzly

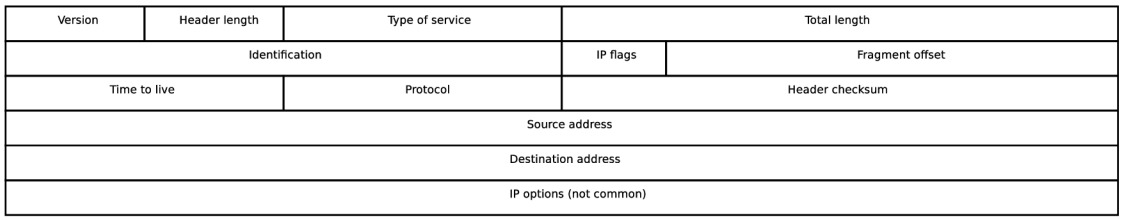
Dále je třeba položit si otázku jakým způsobem budou uzly komunikovat, zda-li přes unicast, nebo multicast, či dokonce broadcast. Při komunikaci unicastem by bylo třeba neustále udržovat databázi uzlů se kterými komunikovat. Při použití multicastu, nebo broadcastu tato potřeba neexistuje, a je v praxi výhodnější, protože informace bude vždy odesílat pouze master uzel na všechny slave uzly, nebude třeba udržovat na master uzlu seznam slave uzlů a bude méně overheadu. Nicméně tento přístup má také své stinné stránky, například nešel by nasadit do provozu v případě, že mezi master a slave uzly je pouze veřejný internet, na kterém buď jsme v jedné broadcastové doméně, a nechceme aby tyto informace poslouchali ostatní uživatelé internetu. Anebo vůbec nejsou uzly ve stejné broadcastové doméně a takováto komunikace není možná vůbec. Tato situace není, podle mého soudu, až tak vážná, protože pro každý cluster se doporučuje použití *BCN - Back Channel Network (zadní síť)*, která je oddělená od zbytku internetu. Pokud bychom přesto chtěli uzly takto rozmístit, řešením by mohlo být například nasazení další L2 VPN sítě jako BCN, nicméně tato druhá BCN-VPN by se stala single-point-of-failure. Tento případ nebudeme uvažovat a použijeme UDP broadcast.

4.3.3 Volba master uzlu

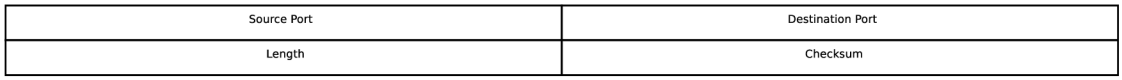
Při řešení problému volby master uzlu se můžeme inspirovat různými distribuovanými síťovými protokoly, jako je například Spanning Tree, či OSPF (Open Shortest Path First). Tyto protokoly implementují poměrně složité algoritmy pro volbu master uzlu. U našeho clusteru by bylo pěkné moci si definovat prioritu uzlu, který se kdy stane master uzlem. Bylo by tak možné definovat si priority podle výkonnosti hardware, rychlosti sítě, spolehlivosti sítě, či jakoukoliv kombinací veličin, které nám v daném konkrétním případě přijdou adekvátní. Této funkčnosti lze dosáhnout definováním takovýchto pravidel ve frameworku corosync, systém OpenVPN bude akorát poslouchat, jestli má interface s IP adresou, na kterém by měl poslouchat - je master, či nikoli - je slave. Pokud je master, nastaví si interně ve struktuře *options* bool hodnotu *i_am_master* na true a může se podle toho chovat. Což znamená, že začne broadcastem obesílat ostatní uzly o všech příchozích klientech a přestane poslouchat na broadcastové adrese, protože nemůže nastat situace, kdy jsou dva master uzly v rámci jednoho clusteru.

4.3.4 Datová struktura pro přenos

Pro návrh struktury komunikačního packetu se přímo nabízí varianta vložení celé struktury *multi_instance*. Takovéto řešení je prakticky možné, nicméně na straně příjemce je třeba některé informace vyfiltrovat a nahradit je správnými hodnotami, jako například ukazatele na pole znaků apod. Toto řešení zjevně přenáší data, která na cílovém stroji nebudou potřeba, nicméně, když jsme se zbavili TCP hlavičky a nahradili jsme ji podstatně menší UDP hlavičkou, pár kilobajtů navíc zde ušetřených si můžeme dovolit zase vyplýtvat jinde. Při dnešních rychlostech sítí by pro takovýto datový tok stačila i velmi pomalá linka. Udělejme si výpočet. Vestavěná funkce jazyka C *sizeof()*, konkrétně pak *sizeof(struct multi_instance)* nám vrátí hodnotu 3392. Což znamená 3392 bajtů informace uložené pro každého klienta. Přidejme 8 bajtů UDP hlavičku - Obrázek 4.3.4, 24 bajtů IPv4 hlavičku - Obrázek 4.3.4 (RFC 790) a řekněme, že použijeme ethernet jako L1, takže 38 (bez 802.1q) až 42 (s 802.1q) bajtů. Dostaname 3466 bajtů, které je třeba odeslat broadcastem na BCN síť. Pokud bychom měli síť s 2000 klienty, kteří by se chtěli připojit všichni v jeden okamžik, nebudeme-li brát v potaz vytížení primární sítě - spojující server s klienty, dostaneme 6932000 bajtů, což je 6769 KB, které by bylo třeba odeslat na BCN síť. Takovéto množství dat by byla schopna bez jakýchkoliv problémů přenést i 10MBit/s 10BASE5 ethernet, technologie využívaná hlavně v 80. letech minulého století. Dnes nejvíce rozšířený 1000BASE-T ethernet disponuje stonásobnou rychlostí. Což nám dává přímý důkaz, že takovýto datový tok dat není žádnou překážkou.



Obrázek 4.2: IPv4 hlavička



Obrázek 4.3: UDP hlavička

4.3.5 L4 protokol

Zamysleme se nad rozdílem komunikace v UDP a TCP módu. V TCP módu se obě strany (klient i server) můžou spolehnout na to, že se jim nemůže ani jeden packet ztratit - tudíž se nemůže rozsynchronizovat TLS kontrolní součet. V případě, že k tomu skutečně dojde, může to signalizovat nekalé praktiky, jako např. man-in-the-middle útok. Naopak v UDP módu (který je většinou využíván) se toto stát může, je poměrně běžné, že se při přenosu přes síť packet ztratí, či packety nedojdou v takovém pořadí, v jakém byly odeslány. V UDP módu bude lépe možné spoléhat na OpenVPN, že bude schopné vypořádat se s rozsynchronizovaným TLS. Proto se touto prací omezíme pouze na UDP mód a cluster pro TCP mód nebudeme uvažovat. Nicméně využití OpenVPN nad TCP je neefektivní a nemá příliš mnoho reálných důvodů k nasazení. Příkladem takového pádného důvodu může být případ, kdy ISP (Internet Service Provider) povoluje pouze komunikaci nad portem 80 (HTTP) nebo 443 (HTTPS) TCP, což by nám umožnilo zprovoznit server nad tímto portem a protokolem a i z takovéto přípojky k síti lze komunikovat s OpenVPN. Nicméně tento mód je velmi neefektivní. Zřejmě nejhorší případ je když využijeme nad takovýmto TCP tunelem další TCP spojení, overhead se nám zdvojnásobí a budou nad takovouto komunikací reálně pracovat dva TCP stacky.

4.3.6 Modifikace

Nyní můžeme přistoupit k samotné modifikaci projektu OpenVPN. Prvně je třeba zajistit socket, který bude mít povoleno komunikovat broadcastem pro obesílání slave uzlů. Dále je třeba zajistit správné zjišťování zda-li server běží v módu master, či slave. Toto lze učinit jednoduše podle aktivního zjišťování, zda-li je dostupná adresa, na které má server poslouchat, tzn. adresa, která je uvedena v konfiguračním souboru. Je třeba také definovat některé klauzule pro konfigurační soubor, abychom mohli serveru říct na kterou adresu má posílat oznámení o příchozích klientech. A to následující:

- `cluster`
- `cluster-broadcast-address`
- `cluster-port`

Klauzule *cluster* je pouze booleovská, neočekává žádné parametry a značí pouze to, že se jedná o instanci cluster uzlu. Další klauzule nastavují adresu a port na kterých cluster bude komunikovat. Na master uzlu je třeba definovat obě dvě klauzule *cluster-broadcast-address* i *cluster-port*. Ztímco na slave uzlu teoreticky stačí pouze port, nicméně protože se očekává, že se může ze slave stát master uzel, pak je dobré tyto klauzule definovat i na slave uzlu. Tím pádem je možné sdílet jeden stejný konfigurační soubor pro všechny uzly clusteru. Stejně jako klíč a certifikát serveru, který musí být na všech uzlech shodný, aby klient nepodezíral server z nekalých praktik, což náhlá změna certifikátu může značit.

Je třeba definovat všechny tři tyto klauzule, abychom mohli využívat OpenVPN v clusteru způsobem, popsaným v této práci. Dále bylo třeba zajistit, aby slave uzel čekal na

příchozí data od master uzlu a zaindexoval si je interními funkcemi. Tohoto bylo dosaženo částečnou modifikací a voláním funkce *multi_get_create_instance_udp*. Tyto úpravy nám umožní synchronizovat data mezi jednotlivými uzly.

Kapitola 5

Testy

V následující kapitole bude rozebrán test implementovaného řešení. V zapojení budeme vycházet ze schématu z minulé kapitoly, konkrétně Obrázek 4.1.

5.1 Výbava a zapojení

Jelikož je třeba brát v potaz také hardware, na kterém testy probíhaly, je třeba jej zmínit.

Všechny testy byly spouštěny na laptopu Lenovo ThinkPad T420 s procesorem Intel i5-2410M s frekvencí 2.30GHz a 8GB operační paměti. Tento stroj díky podpoře virtualizačních instrukcí posloužil jako hostitel pro naše zapojení. Při testech bylo použito zapojení, jako je vyobrazeno na obrázku 4.1, s drobnými úpravami.

5.1.1 Ping test

Budeme testovat spojení ping a počet ztracených ICMP ECHO packetů.

Příprava

Při tomto testu vytvoříme na hostitelském virtualizačním stroji dummy rozhraní a přiřadíme mu IP adresu

```
sdoky-t420 ~ # modprobe dummy
sdoky-t420 ~ # ifconfig dummy0 inet 172.30.0.1
sdoky-t420 ~ # ifconfig dummy0 up
```

Na hostitelském stroji a na uzlech clusteru povolíme *forwarding* a plně otevřeme iptables firewall, abychom nikde neblokovali žádný provoz.

```
sdoky-t420 ~ # echo 1 > /proc/sys/net/ipv4/ip_forward
sdoky-t420 ~ # iptables -I FORWARD 1 -j ACCEPT
sdoky-t420 ~ # iptables -I OUTPUT 1 -j ACCEPT
sdoky-t420 ~ # iptables -I INPUT 1 -j ACCEPT
```

Spustíme openvpn v cluster módu a provedeme kontrolu nastavení sítě (pro přehlednost zkráceno)

```
[root@node1 ~]# ip a
2: eth0:
    inet 192.168.2.176/24
    brd 192.168.2.255 scope global eth0
3: eth1:
    inet 192.168.10.51/24
    brd 192.168.10.255 scope global eth1
4: eth2:
    inet 10.0.0.41/8
    brd 10.255.255.255 scope global eth2:1
[root@node1 ~]# ip r
172.30.0.0/24 via 10.0.0.40 dev eth2
192.168.10.0/24 dev eth1  proto kernel  scope link  src 192.168.10.51
172.21.0.0/24 dev tun9  proto kernel  scope link  src 172.21.0.1
10.0.0.0/8 dev eth2  proto kernel  scope link  src 10.0.0.41

[root@node2 ~]# ip a
2: eth0:
    inet 192.168.2.173/24
    brd 192.168.2.255 scope global eth0
3: eth1:
    inet 192.168.10.52/24
    brd 192.168.10.255 scope global eth1
4: eth2:
    inet 10.0.0.42/8
    brd 10.255.255.255 scope global eth2:1
    inet 10.0.0.51/8
    brd 10.255.255.255 scope global secondary eth2
[root@node2 ~]# ip r
172.30.0.0/24 via 10.0.0.40 dev eth2
192.168.10.0/24 dev eth1  proto kernel  scope link  src 192.168.10.52
172.21.0.0/24 dev tun9  proto kernel  scope link  src 172.21.0.1
10.0.0.0/8 dev eth2  proto kernel  scope link  src 10.0.0.42

sdoky-t420 ~ # ip a
2: enp0s25:
    inet 192.168.2.162/24 brd 192.168.2.255 scope global enp0s25
8: br0:
    inet 192.168.10.40/24 brd 192.168.10.255 scope global br0
9: br1:
    inet 10.0.0.40/8 brd 10.255.255.255 scope global br1
11: dummy0:
    inet 172.30.0.1/16 brd 172.30.255.255 scope global dummy0
sdoky-t420 ~ # ip r
10.0.0.0/8 dev br1  proto kernel  scope link  src 10.0.0.40
172.21.0.0/16 via 10.0.0.51 dev br1
172.30.0.0/16 dev dummy0  proto kernel  scope link  src 172.30.0.1
192.168.10.0/24 dev br0  proto kernel  scope link  src 192.168.10.40
```

Na klientském stroji je možno připojit se na server a poté je třeba přidat cestu do sítě 172.31.0.0/16 reprezentovanou dummy rozhraním na hostitelském stroji. Poté lze zkontrolovat ping na hostitelský stroj.

```
[root@localhost ~]# ping 172.30.0.1
PING 172.30.0.1 (172.30.0.1) 56(84) bytes of data.
64 bytes from 172.30.0.1: icmp_seq=32 ttl=63 time=7.07 ms
64 bytes from 172.30.0.1: icmp_seq=33 ttl=63 time=2.47 ms
64 bytes from 172.30.0.1: icmp_seq=34 ttl=63 time=3.53 ms
64 bytes from 172.30.0.1: icmp_seq=35 ttl=63 time=2.77 ms
64 bytes from 172.30.0.1: icmp_seq=36 ttl=63 time=3.68 ms
64 bytes from 172.30.0.1: icmp_seq=37 ttl=63 time=3.63 ms
64 bytes from 172.30.0.1: icmp_seq=38 ttl=63 time=2.87 ms
64 bytes from 172.30.0.1: icmp_seq=39 ttl=63 time=2.41 ms
64 bytes from 172.30.0.1: icmp_seq=40 ttl=63 time=2.98 ms
^C
--- 172.30.0.1 ping statistics ---
40 packets transmitted, 9 received, 77% packet loss, time 39521ms
rtt min/avg/max/mdev = 2.418/3.494/7.073/1.344 ms
```

Zahájení testu

Všechno je v pořáku a připraveno k testu. Spustíme ping flood (což má právo udělat pouze uživatel root) a budeme sledovat kolik ICMP packetů se ztratí.

```
[root@localhost ~]# ping -f 172.30.0.1
PING 172.30.0.1 (172.30.0.1) 56(84) bytes of data.
<DOTS OMITTED>
--- 172.30.0.1 ping statistics ---
15146 packets transmitted, 14349 received, 5% packet loss, time 48347ms
rtt min/avg/max/mdev = 0.524/2.406/24.841/1.748 ms, pipe 2, ipg/ewma 3.192/4.086 ms
```

Výsledek

Z tohoto testu lze usoudit, že vypadlo 797 packetů. Podle definice parametru *-f* pingu:

```
-f      Flood ping. For every ECHO_REQUEST sent a period ``.'' is
        printed, while for ever ECHO_REPLY received a backspace is
        printed. This provides a rapid display of how many packets are
        being dropped. If interval is not given, it sets interval to
        zero and outputs packets as fast as they come back or one hun-
        dred times per second, whichever is more. Only the super-user
        may use this option with zero interval.
```

Ping se snaží posílat ICMP ECHO packety jak nejrychleji to jde, pokud nepřijde, čeká 10ms, nnež packet prohlásí za ztracený. Z čehož vyplývá, že byla služba nedostupná po dobu 7,97s. Tato doba je sice relativně krátká, neměla by rozvázat TCP spojení (viz dále), ovšem bylo by zajímavé blíže se zamyslet nad důvodem, proč tato doba není kratší. K tomu bude třeba podívat se blíže na výpisy z obou uzlů clusteru a klienta.

Rozbor výsledku

V předchozí části jsme se dozvěděli, že při testování dostupnosti služby podle ping flood byla služba nedostupná po dobu 7,97s. Pojďme se blíže podívat na důvod proč byla doba nutná ke znovu ustavení spojení zrovna taková, jaká byla. Do času pro obnovení dostupnosti služby musíme započítat následující časové údaje (dále si je blíže rozebereme a upřesníme):

- Doba, než Corosync zjistí, že je master uzel nedostupný
- Doba, než Corosync vybere nový uzel pro přiřazení IP adresy a adresu přiřadí
- Doba, než OpenVPN zjistí, že se ze slave stal master

Nutno podotknout, že uzel *node1* byl v době zahájení testu slave uzlem a uzel *node2* byl master uzlem, tudíž měl přiřazenu plovoucí IP adresu a přijímal spojení od klienta. Výpadek se stal v 17:30:00, podle času klienta

```
17:29:07 2013 us=89400 Cluster master socket created
17:29:07 2013 us=89491 Cannot bind right now, interface unavailable -> cluster slave
17:29:07 2013 us=89774 TUN/TAP device tun9 opened
17:29:07 2013 us=103970 Initialization Sequence Completed
17:29:07 2013 us=103997 waiting for clients
Listen activating.
17:29:07 2013 us=104046 no clients yet
...
17:29:11 2013 us=111630 Cluster has one new client: 10.0.0.10
17:29:11 2013 us=111778 MULTI: multi_create_instance called
17:29:11 2013 us=111911 10.0.0.10:40525 Re-using SSL/TLS context
17:29:45 2013 us=193627 10.0.0.10:40525 no clients yet
...
17:29:45 2013 us=694742 10.0.0.10:40525 no clients yet
17:29:45 2013 us=695173 10.0.0.10:40525 Interface with address 10.0.0.51 found: eth2
17:29:45 2013 us=695207 10.0.0.10:40525 I am binding the socket to interface
17:29:46 2013 us=199249 10.0.0.10:40525 TLS Error: local/remote TLS keys are out of syn
[AF_INET]10.0.0.10:40525 [0]
17:29:46 2013 us=199474 10.0.0.10:40525 [UNDEF] Inactivity timeout (--ping-restart),
restarting
17:29:55 2013 us=395762 10.0.0.10:38188 VERIFY OK: depth=1, C=CZ, ST=Morava, L=Brno,
O=Kajot, OU=Online systemy, CN=VPN Cluster CA, name=Kajot, emailAddress=admin@tech.kajo
```

Tabulka 5.1: Log uzlu node1 - pro přehlednost vypuštěny nepodstatné řádky

Z těchto výpisů se ukazuje, že po selhání master uzlu a po ustavení node1 jako nový master si začal klient se serverem nerozumět a spojení bylo zrestartováno. Poté si klient se serverem vyměnili certifikáty, tak jako to udělali již na začátku spojení. Tato funkčnost by vadila, potřebovali-li bychom FT cluster, nicméně u HA clusteru je toto chování přijatelné.

Při komunikaci nad TLS se udržuje kontext. Kontext, který se může běžně rozsynchronizovat, většinou v důsledku výpadku packetu. Což nad L4 protokolem UDP je možné, a děje se to běžně. Při takovéto ztrátě packetů se na novém kontextu umí klient se serverem domluvit tak rychle, že to nejsme schopni na kvalitě služby poznat. Předpokládalo se,

```
17:27:29 2013 us=298061 Cluster master socket created
17:27:29 2013 us=298084 Socket Buffers: R=[229376->131072] S=[229376->131072]
17:27:29 2013 us=298422 TUN/TAP device tun9 opened
17:27:29 2013 us=300943 Initialization Sequence Completed
17:27:30 2013 us=475867 MULTI: multi_create_instance called
17:27:30 2013 us=476113 10.0.0.10:40525 Re-using SSL/TLS context
17:27:30 2013 us=518825 Cluster_client/10.0.0.10:40525 port: 40525
17:27:30 2013 us=518840 Cluster_client/10.0.0.10:40525 ########
17:27:30 2013 us=518855 Cluster_client/10.0.0.10:40525 2port: 40525
17:27:30 2013 us=519193 Cluster_client/10.0.0.10:40525 message sent! 3392
```

Tabulka 5.2: Log uzlu node2 - pro přehlednost vypuštěny nepodstatné řádky

```
17:29:21 2013 us=505455 OpenVPN 2.3.0 x86_64-unknown-linux-gnu
[SSL (OpenSSL)] [LZO] [EPOLL] [eurephia] [MH] [IPv6] built on Apr 20 2013
17:29:21 2013 us=530754 VERIFY OK: depth=0, C=CZ, ST=Morava, L=Brno, O=Kajot,
OU=Online systemy, CN=Cluster_server, name=Kajot, emailAddress=admin@tech.kajot.cz
17:29:23 2013 us=911526 Initialization Sequence Completed
17:30:04 2013 us=6573 [Cluster_server] Inactivity timeout (--ping-restart), restarting
17:30:04 2013 us=7689 TCP/UDP: Closing socket
17:30:04 2013 us=7880 SIGUSR1[soft,ping-restart] received, process restarting
17:30:04 2013 us=7933 Restart pause, 2 second(s)
17:30:06 2013 us=36456 VERIFY OK: depth=0, C=CZ, ST=Morava, L=Brno, O=Kajot,
OU=Online systemy, CN=Cluster_server, name=Kajot, emailAddress=admin@tech.kajot.cz
17:30:08 2013 us=13591 Initialization Sequence Completed
```

Tabulka 5.3: Log klienta - pro přehlednost vypuštěny nepodstatné řádky

že po přepojení na nový uzel, který tento kontext nemá nastane úplně stejná situace, jako když vypadne několik packetů při UDP komunikaci s originálním (neupraveným) OpenVPN serverem.

5.1.2 TCP session test

Test při kterém si ověříme, že se nerozváže TCP sezení. Typickým příkladem služby, která využívá TCP jako L4 protokol je Secure SHell (SSH). Test uskutečníme se stejným zapojením, jako v ping testu, akorát nebudeme testovat počty ICMP packetů, ale spojíme SSH konzoli a poté uděláme výpadek uzlu, poté zkusíme zjistit, jestli se spojení rozváže, či nikoliv. V tomto případě je uzel *node1* master uzlem.

Zahájení testu

Výsledek

SSH konzole po dobu výpadku packetů nereagovala, ovšem později se všechny operace učiněné v době bez konektivity projevily. Tudíž se nerozvázalo TCP spojení a cluster poskytoval službu bez výpadku (dříve definováno).

```
06:37:40 2013 us=782791 Cluster master socket created
06:37:40 2013 us=938800 Initialization Sequence Completed
06:38:15 2013 us=812074 Cluster_client/10.0.0.10:45000 port: 45000
06:38:15 2013 us=812098 Cluster_client/10.0.0.10:45000 ########
06:38:15 2013 us=812121 Cluster_client/10.0.0.10:45000 2port: 45000
06:38:15 2013 us=812495 Cluster_client/10.0.0.10:45000 message sent! 3392
```

Tabulka 5.4: Log node1 - pro přehlednost vypuštěny nepodstatné řádky

```
06:37:44 2013 us=329356 Cluster master socket created
06:37:44 2013 us=468008 Initialization Sequence Completed
06:37:44 2013 us=468024 waiting for clients
Listen activating.
06:37:44 2013 us=468766 no clients yet
...
06:38:16 2013 us=32644 no clients yet
06:38:16 2013 us=33091 Cluster has one new client: 10.0.0.10
06:38:16 2013 us=33246 MULTI: multi_create_instance called
06:38:16 2013 us=33382 10.0.0.10:45000 Re-using SSL/TLS context
06:38:16 2013 us=536292 10.0.0.10:45000 no clients yet
...
06:43:09 2013 us=632191 10.0.0.10:45000 no clients yet
06:43:09 2013 us=632302 10.0.0.10:45000 Interface with address 10.0.0.51 found: eth2
06:43:09 2013 us=632330 10.0.0.10:45000 I am binding the socket to interface
06:43:09 2013 us=632358 10.0.0.10:45000 global_sockfd: 4
06:43:09 2013 us=632385 10.0.0.10:45000 global_addr: 0x3300000aaa040002
06:43:09 2013 us=632413 10.0.0.10:45000 global_addrlen: 16
06:43:17 2013 us=871004 10.0.0.10:46187 Re-using SSL/TLS context
06:43:17 2013 us=923219 Cluster_client/10.0.0.10:46187 port: 46187
06:43:17 2013 us=923236 Cluster_client/10.0.0.10:46187 ########
06:43:17 2013 us=923253 Cluster_client/10.0.0.10:46187 2port: 46187
06:43:17 2013 us=923526 Cluster_client/10.0.0.10:46187 message sent! 3392
```

Tabulka 5.5: Log node2 - pro přehlednost vypuštěny nepodstatné řádky

```
[root@localhost openvpn]# ssh 172.30.0.1
The authenticity of host '172.30.0.1 (172.30.0.1)' can't be established.
RSA key fingerprint is d1:a0:41:65:b9:f0:08:d8:5f:1d:1d:c2:05:18:05:ad.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '172.30.0.1' (RSA) to the list of known hosts.
Password:
Last login: Thu May  9 10:28:00 CEST 2013 from 172.29.0.2 on pts/14
sdoky-t420 ~ # date
Thu May  9 18:22:55 CEST 2013
sdoky-t420 ~ # date
Thu May  9 18:24:18 CEST 2013
sdoky-t420 ~ # ^C
sdoky-t420 ~ # logout
```

Tabulka 5.6: Log klienta - ssh spojení

Kapitola 6

Závěr

6.1 Výsledek práce

Výsledkem práce je produkt, který je schopen běhu v clusteru. Jednotlivé uzly si vyměňují informace o připojených klientech a o volbu master uzlu se stará Corosync. V úvodu práce jsme si definovali, že výpadkem služby bude pro naše potřeby myšleno přerušení poskytování služby na tak dlouhou dobu, aby se rozvázalo TCP spojení. Výsledný produkt toto kritérium plně splňuje. Očekávalo se ovšem, že se funkcionalitou více přiblíží k FT clusteru, a přerušení poskytování služby bude kratší, než se v reálných testech ukázalo. V aktuální fázi není synchronizován OpenSSL kontext, jehož synchronizace je klíčová pro FT cluster. Tato práce byla zamýšlena jako „nice-to-have“ rozšíření funkcionality, která je aktuálně nasazena ve společnosti Kajot, provozující výherní terminály po velké části Evropy. Tato společnost spojuje jednotlivé stroje do jedné velké virtuální sítě právě pomocí OpenVPN. V současné době by nebylo rentabilní přecházet na systém vyvinutý v rámci této práce. Ovšem v případě rozšíření na kvalitu poskytování služeb, kterou poskytuje FT cluster by toto nasazení bylo zřejmě přínosem.

6.2 Možnosti pokračování

Dalším směrem, kterým by se tento projekt mohl ubírat je rozšíření funkčnosti až na úroveň FT clusteru, bez jakéhokoliv znatelného přerušení poskytování služby, natož výpadku. K tomuto bude zapotřebí více pracovat s knihovnou OpenSSL. V současné době není známo, zda-li by tato funkčnost nevyžadovala zásah do samotné OpenSSL knihovny. Rovněž není známo jak rozsáhlé úpravy by tato knihovna vyžadovala. Jedna věc je v současné době jistá, rozhodně bude nutné synchronizovat kontext. Ať už knihovny OpenSSL, tak samotné aplikace OpenVPN. Toto sdílení kontextu by bylo možné realizovat až na úrovni sdílené paměti. Dále by stálo za zvážení nasazení frameworků OpenMP a OpenMPI. OpenMP by přinesl podporu vícevláknového zpracování. OpenMPI by přineslo snazší implementaci clusteru a distribuovaných výpočtů. Knihovna OpenMPI by pomohla hlavně serverům s velkým provozem, a tudíž velkým vytížením procesoru, tím, že by byla schopna rozložit tuto zátěž na větší množství strojů v rámci clusteru. Je třeba mít na paměti, jak již bylo zmíněno dříve, že nárůst výkonu není lineární.

Kapitola 7

Obsah přiloženého CD

```
.
    openvpn-2.3.0-cluster.tar.gz
    openvpn-2.3.0-original.tar.gz
    configurations
        cluster-node.conf
        client-node.conf
    ping-test
        openvpn-client.log
        openvpn-node1.log
        openvpn-node2.log
    ssh-test
        openvpn-client.log
        openvpn-node1.log
        openvpn-node2.log

3 directories, 10 files
```

Literatura

[1] 2-node Red Hat KVM Cluster Tutorial. 2013.
URL <https://alteeve.ca/w/2-Node_Red_Hat_KVM_Cluster_Tutorial>

[2] OpenAIS. 2013.
URL <https://oss.oracle.com/osswiki/OpenAIS.html>

[3] Počítačový cluster. 2013.
URL <http://cs.wikipedia.org/wiki/Po\%C4\%8D\%C3\%ADta\%C4\%8Dov\%C3\%BD_cluster>

[4] Feilner, M.: *OpenVPN: Building and Integrating Vitual Private Networks*. Packt publishing, 2006, iSBN 1-904811-85-X.

[5] Froom, R.; Sivasubramanian, B.; Frahim, E.: *Implementing Cisco Switched Networks (SWITCH)*. Cisco Press, 2012, iSBN 978-1-58705-884-4.

[6] Lucke, R. W.: *Building Clustered Linux Systems*. Prentice Hall, 2005, iSBN 0-13-144853-6.