DYNAMIC STATE OF OMNET++ MODEL VIA SNMP

Jakub Smejkal
Master Degree Programme (2), FIT BUT
E-mail: xsmejk00@stud.fit.vutbr.cz

Supervised by: Vladimír Veselý
E-mail: ivesely@fit.vutbr.cz

Abstract: This paper describes possible approach how to extract run-time information's from real computer network, currently focusing on topological data. As a proof of concept we have implemented experimental tool for topology reconstruction using Breadth-First search algorithm, SNMP and CDP protocols. Developed application was tested on real topology with successful results. Results will be used as input for discrete-event simulator OMNeT++. First we are going to describe used protocols and later modified version of BFS algorithm.

Keywords: OMNeT++, INET, ANSA, SNMP, LLDP, CDP, BFS

1 ÚVOD

Složitost počítačových sítí a technologií obecně je v současné době stále větší. S tím vzrůstají nároky na jejich údržbu a případné ladění chyb. Z této motivace byl stvořen nástroj OMNeT++, který představuje velmi efektivní nástroj pro simulování činnosti zkoumaného systému. Pro přiblížení jeho schopností směrem k zaměření na počítačové sítě, je možné použít rozšíření INET a na něj navazující ANSA, kde vývoj probíhá na VUT FIT. Pro korektní průběh je nutné vložit na startu co nejpodrobnější data o simulovaném systému. To zajišťuje vyvíjený nástroj. Cílem ANSA projektu je umožnit automatizovanou analýzu a verifikaci.

2 POUŽITÉ TECHNOLOGIE

Pro rekonstrukci topologie sítě se jeví jako vhodné použít protokolů CDP a LLDP, jejichž výstupy jsou dostupné prostřednictvím SNMP. Tato kapitola shrnuje nejdůležitější poznatky o těchto technologiích.

2.1 CISCO DISCOVERY PROTOCOL

Technologie provádí sběr dat o svém okolí a ukládá je do MIB, ta je přístupná přes SNMP. Protokol je proprietární a není otevřeným standardem. Pro svou činnost používá vícesměrovou adresu, která je ve tvaru 01-00-0c-cc-cc-cc. Jedná se tedy o technologii pracující na druhé vrstvě. Další charakteristikou je nezávislost na konkrétním síťovém protokolu, proto je CDP možné uplatnit na širokou škálu linkových technologií. Přenos jednotlivých informaci je realizován strukturou TLV, která obsahuje atributy. Zkratka představuje trojici typ, délka, hodnota(z anglického Type Length Value).Nástroj popisovaný v tomto článku využívá prozatím pouze položek reprezentujících textovou identifikaci zařízení a síťovou adresu. V plánu jsou dále minimálně identifikace portu a seznam schopností prvku.[2]

2.2 LINK LAYER DISCOVERY PROTOCOL

LLDP je oproti CDP průmyslovým standardem označován jako 802.1AB. Princip funkce je velmi podobný variantě od Cisca, tedy opět dochází k odesílání a kolekci informací o jednotlivých zařízeních. Tyto informace jsou uloženy v MIB podle RFC2922. Jedná se také o linkový protokol. Komunikace probíhá pomocí vícesměrové adresy, která má formát 01-80-c2-00-00-00. Zprávy zasílané pomocí

LLDP jsou složeny ze sekvence TLV položek. Otevřenost standardu tohoto protokolu pak řeší problém obtížnější správy sítě složené ze zařízení od více výrobců. Množina informací poskytovaná LLDP je mnohem obsáhlejší než u CDP. Narozdíl od CDP obsahuje i data o prvku samotném, což ulehčuje tvorbu algoritmu. Nicméně v principu nemůžeme počítat se stoprocentní podporou LLDP a proto je program uzpůsoben jak pro CDP tak LLDP.[1]

3 KOMPLETACE ALGORITMU PRO PRŮCHOD GRAFEM

Jako předloha pro princip algoritmu bylo použito prohledávání do šířky. Tato varianta se nabízí z principu pojetí programu, kdy jako vstup slouží prvek v cílové topologii. Od tohoto bodu se pak nástroj dotazuje na okolní zařízení pomocí SNMP technologie. Jelikož záznamy informací o okolí dosahují pouze k nejbližším sousedům je nutné se mezi uzly sítě přesouvat a tím tedy realizovat podstatu procházení grafu. Protože vstupní bod je pouze jeden a naším cílem je analyzovat celý graf, tak se jedná o variantu vyhledávání cest z jednoho bodu do všech ostatních. Časová složitost prohledávání do šířky je $O(m+n)$ kde m značí počet hran grafu a n počet uzlů. Nyní představím stručně princip modifikované varianty prohledávání do šířky a použité datové struktury. Označení každého uzlu je realizována unikátním identifikátorem.

- **Pole** $Adj[u]$ - obsahuje všechny bezprostředně dosažitelné sousedy uzlu u.
- **Uzel** *root* - unikátní identifikace počátečního uzlu.
- **Fronta** Q - představuje hlavní prvek určující koncept algoritmu. Konkrétně pracujeme s typem FIFO. Fronta obsahuje uzly určené ke zpracování.
- **Pole** $color[u] \in \{NIL, Black\}$ - barvy reflektují aktuální stav zpracování uzlu v rámci algoritmu. Černá značí uzavřenost, tedy algoritmus dokončil zpracování uzlu a již se k němu nevrací.

Princip činnosti je totožný s prohledáváním do šířky, nicméně při nasazení do reálného prostředí je nutné provést několik změn. Například není možné provést inicializaci každého uzlu v topologii, z jednoduchého principu že ji neznáme. Z tohoto důvodu jsme v implementaci nezahrnuli bílou a šedou barvu. Tuto jejich činnost supluje unikátní identifikátor zařízení v síti. Hlavní změnou oproti prohledávání do šířky je také absence pole Adj, které se při běhu naopak vytváří. Průběh se dá popsat následovně. Algoritmu je předložen startovní uzel. Ten je identifikován pomocí IP adresy, proto je potřeba získat jednoznačnější pojmenování. Pro tento účel jsme zvolili kombinaci fyzických adres jednotlivých rozhraní na prvku plus sériové číslo saši prvku. Nyní je možné jej přiřadit do dvojdimenzionálního pole sousedností, kde do korektně asociované dimenze budou přibývat sousedé uzlu. Protože počáteční uzel není v tomto bodě ještě kompletně prozkoumaný, je nutné jej přidat do fronty pro zpracování. Po tomto úvodu vstupujeme do hlavní smyčky, kde probíhá celé jádro algoritmu. Cyklus se opakuje dokud není fronta Q prázdná. V každé iteraci dojde nejprve k vyjmutí nejstaršího prvku u ve dříve zmíněné frontě. Tento element reprezentovaný již unikátním identifikátorem je přiřazen do pole sousedností. Po této akci jsou prozkoumání sousedi tohoto prvku a jejich výčet je vložen do struktury *neighbors*. Smyčka For má za úkol projít postupně všechny sousedy prozkoumávaného uzlu u. V první řadě je každému z nich přirazen opět unikátní identifikátor. Následně dojde k testu zda tento element již byl dříve zpracován. Proběhne tedy test na černou barvu a současně se ověří zda jsme se zpětně nedostaly k počátečnímu uzlu.Pokud jsou dříve zmíněné podmínky splněny, tak přiřadíme souseda do odpovídajícího pole $Adj[u]$ a zbytek iterace smyčky přeskočíme. V opačné případě rozhodnutí struktury If dojde k dalšímu testu, kde se porovnává zda uzel není počáteční. V takovém případě je provedena operace vložení souseda u do fronty Q. Tentýž uzel je pak zpracován ve stejném kontextu, ale v rámci struktury Adj. Jakmile je obsloužen poslední element z *neighbors*, tak dojde k opuštění smyčky For a prozkoumávaný uzel u je označen černou barvou. Algoritmus samotný ke své funkci vyžaduje několik podmínek na funkcionalitu sítě. Především všechny prvky, která mají být objeveny, musí mít IP konektivitu a zřízený přístup přes protokol SNMP. Stejně důležitá je také dostupnost těchto zařízení z monitorujícího počítače a podpora alespoň CDP nebo LLDP. Pokud nebude zařízení odpovídat na SNMP dotazy, ale bude v CDP nebo LLDP databázi SNMP aktivního souseda, tak bude objeveno, ale prvky dostupné přes něj již ne. Dále zobrazíme výslednou podobu algoritmu.

Níže uvedené funkce *getUID* a *getNeighbors* jsou realizovány pomocí SNMP komunikace dotazující se na konkrétní prvek v síti. Dále *Enqueue* a *Dequeue* slouží pro vkládání a vybírání z fronty.

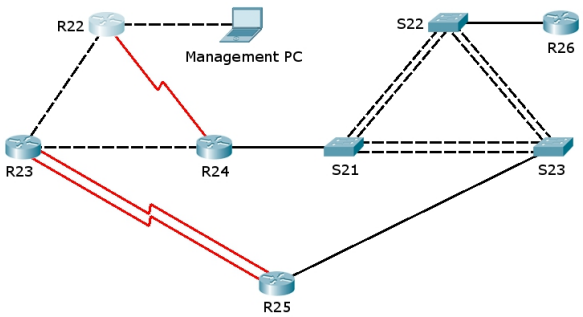
Algorithm 1 Modifikované prohledávání do šířky

```
function MODBFS(root)
    Adj ← getUID(root)
    Enqueue(Q, root)
    while Q ≠ ∅ do
        u ← Dequeue(Q)
        Adj ← u
        neighbors ← getNeighbors(u)
        for ∀v ∈ neighbors do
            neighbor ← getUID(v)
            if color[neighbor] = Black ∧ neighbor ≠ root then
                Adj[u] ← neighbor
                continue
            end if
            if neighbor ≠ root then
                Enqueue(Q, neighbor)
            end if
            Adj[u] ← neighbor
        end for
        color[u] ← Black
    end while
end function
```

4 TESTOVÁNÍ

Algoritmus byl testován na laboratorní topologii složené z několika Cisco směrovačů a přepínačů, které měli mezi sebou plnou konektivitu. Na druhé vrstvě pracoval protokol CDP. Nástroji se podařilo odhalit všechny prvky sítě. Otázkou je doba rekonstrukce topologie, která v tomto případě dosáhla zhruba čtyř sekund, v případě rozsáhlejších nebo vytíženějších sítí. Řešením by byla jednoznačně paralelizace komunikace. Níže uvádíme grafický nákres a textový výstup nástroje. 1

Tabulka 1: Testovaná topologie a výsledky.




```
'S22': ['R26', 'S23', 'S23', 'S21', 'S21']
'S23': ['R25', 'S21', 'S21', 'S22', 'S22']
'S21': ['R24', 'S23', 'S23', 'S22', 'S22']
'R26': ['S22']
'R25': ['R23', 'R23', 'S23']
'R24': ['R23', 'R22', 'S21']
'R23': ['R25', 'R25', 'R24', 'R22']
'R22': ['R23', 'R24']
```

5 ZÁVĚR

Výše zmíněný algoritmus je aktuálně stále ve vývoji a je třeba se zaměřit na implementační detail v rámci různých linkových technologií v prostředí počítačových sítí. Kromě topologie je nutné zajistit také data vztahující se k dynamickému stavu prvku a jeho konfiguraci. Sem patří například IP adresy na rozhraních, tabulky podporující přepínání rámců, nebo nastavení směrovacích protokolů. Posledním krokem bude vygenerování kompatibilní XML struktury pro simulační nástroj OMNeT++.

REFERENCE

[1] Kolektiv autorů: IEEE 802.1AB (LLDP) Specification, ISBN 0-7381-4688-9 SS95332, `http://standards.ieee.org/getieee802/download/802.1AB-2005.pdf`

[2] Cisco: Configuring Cisco Discovery Protocol, `http://www.cisco.com/en/US/docs/ios/12_1/configfun/configuration/guide/fcd301c.html`