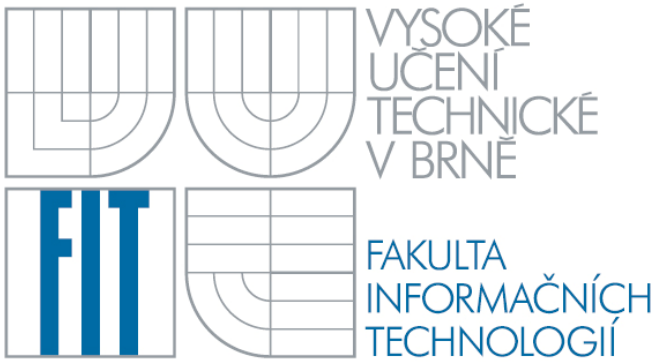


DOKUMENTÁCIA

PROJEKT ISA

2012/2013

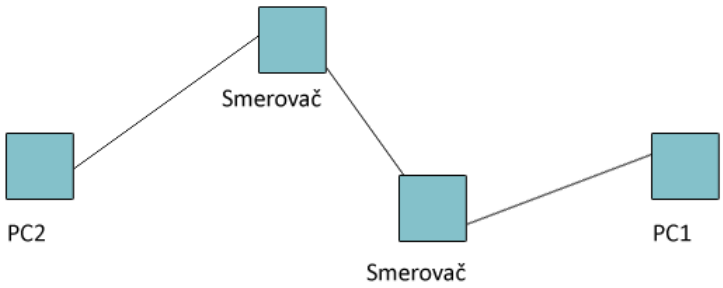
Detekcia maximálneho MTU po ceste

AUTOR: MICHAL LUKÁČ, xlukac05@stud.fit.vutbr.cz
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Obsah

Úvod

Všetko má svoje limity. Platí to aj v architektúrach počítačových sietí, ktoré sa stretávajú s mnohými problémami. Jedným z nich je problém detekcie najväčšej datovéj jednotky MTU, ktorú môže zdroj poslať svojmu cieľu bez toho aby sme tento úsek dát musel rozdeliť na viacero častí. Tomuto javu sa vraví fragmentácia paketov. Pod pojmom zdroj a cieľ si môžeme predstaviť osobný počítač, router, server, ... Prečo by sa pakety nemali fragmentovať prípadne fragmentovať čo najmenej? Dôvodom je rýchlosť. Pakety ktoré sú fragmentované je napríklad pre router časovo náročnejšie spracovať. Preto sa operačný systém snaží zistiť maximálnu "priepustnosť" siete teda aký najväčší paket môže poslať bez toho aby bol fragmentovaný pomocou techniky Path MTU Discovery.



Analýza problému a návrh riešenia

Path MTU Discovery

Aby prenos po celej ceste paketu na sieti bol čo najefektívnejší a aby router v strede cesty nemusel príliš veľké pakety fragmentovať a cieľ zase spracovávať, použíje sa technika Path MTU Discovery, ďalej už iba PMTUD. Paket prechádza cez všetky stanice a smerovače až k cieľu. Jeho maximálna veľkosť je minimálna veľkosť paketu ktorú dokážu spracovať smerovače alebo cieľ na ceste. Problematikou Path MTU Discovery sa zaoberajú RFC dokumenty 1191, 1981, 2923, 4821. Spôsob akým sa operačný systém vysporiadava s PMTUD v IPv4 a IPv6 je rozdielny. Pri IPv4 môžu pakety fragmentovať aj smerovače, narozdiel od IPv6 kde smerovače na ceste pakety nefragmentujú a jediný kto môže fragmentovať je zdrojový počítač. Obecným spôsobom ako zístiť MTU teda „Maximum transmission unit" je poslať icmp paket s requestom na server a očakávať reply[1][4]. Podľa toho či príde request alebo nie zvýšime alebo znížime veľkosť paketu a znovu odošleme paket a čakáme na príjem. ICMPv4/ICMPv6 protokol sa používa pre zasielanie chybových správ. Napríklad ak je destinácia nedostupná, príde icmp paket typu DEST_UNREACH, ak paketu vyprší TTL(zníži sa na 0), vráti sa ICMP paket s chybou TIME_EXCEEDED. Pri PMTUD využijeme icmp paket typu "fragmentation needed". RFC 3542[8] popisuje možnosti PMTUD.

PMTUD pre IPV4

Minimálna dlžka IPv4 paketu je 68 bytov, maximálna dlžka Ipv4 paketu je 65535 bytov[1]. Pri IPv4 sa pošle datagram s IPv4 a ICMP hlavičkou. Štruktúra takéhoto paketu vypadá následovne:

IPv4 Header	ICMPv4 Header	Data

Štruktúra ICMPv4 hlavičky je:

Type	Code	Checksum
Unused		
Internet Header + 64 bits of Original Data Datagram		

(zdroj: rfc 792)

Veľkosť ip hlavičky je 40 bytov. Štruktúra IPv4 hlavičky je:

Version	IHL	Options	Total Length
Identification		Flags	Fragment Offset
Time To Live	Protocol	Header Checksum	
Source IP Address			
Destination IP Address			

Na danú cieľovú adresu sa pošle tento datagram s definovanou veľkosťou. Pokiaľ sa príjme icmp paket s ECHO_REPLY, môže sa program pokúsiť o zväčšenie paketu. Ak sa príjme paket s FRAGMENTATION_NEEDED, program zníži veľkosť paketu a paket znova odošle. Jednou z možností ako nastavovať veľkosť paketu je zaviesť tabulku s dopredu definovanými veľkosťami paketu ako je popísane v RFC1191[4]. Iným prístupom je nájsť MTU pomocou binárneho výhľadávania.

$$veľkosť\ paketu = \frac{min + max}{2}$$

PMTUD pre IPV6

Minimálna veľkosť IPv6 paketu respektíve veľkosť ktorú každý smerovač musí vedieť poslať je 1280 bytov. Maximálna veľkosť IPv6 paketu je 65575 bytov[1]. Štruktúra IPv6 hlavičky vypadá následovne:

Version	Traffic Class	Flow Label
Payload Length	Next Header	Hop Limit
Source Address		
Destination Address		

Štruktúra ICMPv6 hlavičky:

Type	Code	Checksum
Message body		

Podobne ako pri IPv4 pošleme paket s ECHO_REQUEST a očakávame ECHO_REPLY. Pokiaľ nám dorazí icmp paket typu PACKET TOO BIG, musíme zmenšiť veľkosť. Pokiaľ nám dorazí icmp paket typu ECHO REPLY, paket zväčšíme. Pre nájdenie MTU môžeme opäť použiť binárne vyhľadávanie. Inú možnosť nám navrhuje RFC 3542[8]. Najprv nastavíme pomocou funkcie setsockopt následovne:

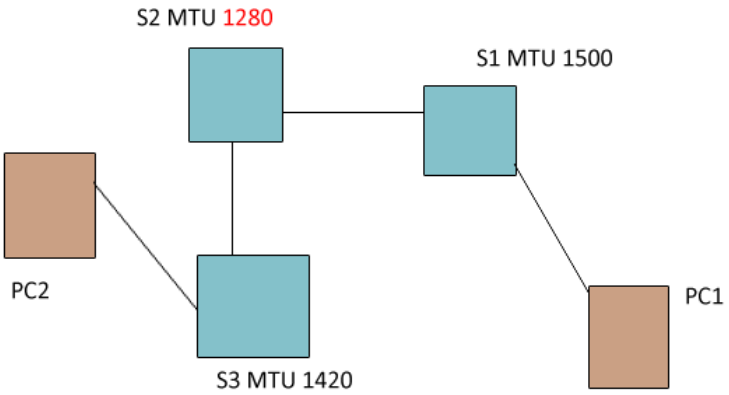
```
int on = 1;
setsockopt(fd,IPPROTO_IPV6, IPV6_RECVPATHMTU, &on, sizeof(on));
```

Defaultne je táto voľba nenastavená. Keď aplikácia príjme icmp paket typu PACKET_TOO_BIG. Potom sa zavolá funkcia recvmsg() ktorá pri takomto pakete vráti 0. Do štruktúry cmsghdr sa nastaví typ cmsg_type na IPV6_PATHMTU a cmsg_len udáva dĺžku dát v cmsg_data[8]. Prvý bit v cmsg_data[] ukazuje na štruktúru ip6_mtuinfo, ktorá je v unixe definovaná následovne:

```
struct ip6_mtuinfo {
        struct sockaddr_in6 ip6m_addr; /* dst address including zone ID */
        uint32_t ip6_mtu; /* path MTU in host byte order */
}
```

Položka ip6_mtu obsahuje hodnotu MTU. Pokiaľ chceme získať dobrú počiatočnú hodnotu veľkosti paketu, môžeme použiť na získanie podľa RFC 3542[8]:

```
struct ip6_mtuinfo mtuinfo;
socklen_t infolen = sizeof(mtuinfo);
getsockopt(fd, IPPROTO_IPV6, IPV6_PATHMTU, &mtuinfo, &infolen);
```



Nájdené MTU je 1280, teda najmenšia možná hodnota ktorú nájdeme na ceste ku zdroji.

Pri odoslaní veľkého paketu dostávame teda icmp paket typu *PACKET_TOO_BIG*, tento paket obsahuje položku MTU, ktorú dostávame od zdroja tohoto paketu. Toto MTU môžeme taktiež využiť pri hľadaní PMTU.

Implementácia

Verzia IPV4

Funkcia main zavolá processParams, ktorá spracuje parametre programu a uloží ich do štruktúry Tparams. Táto istá funkcia potom volá funkciu getAddress v ktorej využívám získanie ip adresy pomocou getaddrinfo(). Tá vráti štruktúru addrinfo, podľa ktorej následne rozhodnem o aký typ ip adresy ide. Túto štruktúru ďalej nejprehľadávame na ďalšie ip adresy, jednoduchu zoberiem prvú štruktúru, ktorá je nastavená. Vo funkci main porovnáme tieto typy adresy a zavoláme funkciu *findMTU4* pre ipv verziu 4 alebo *findMTU6* pre ipv verziu 6. Pri ipv4 využívam aj *gethostbyname* funkciu. Aby som mohol pracovať priamo s paketmi potrebujem štruktúry definované v hlavičkových súboroch netinet.

Pomocou funkcie socket(…) vytvorím vlastný socket:

socketIP = socket(AF_INET, SOCK_RAW,IPPROTO_ICMP);

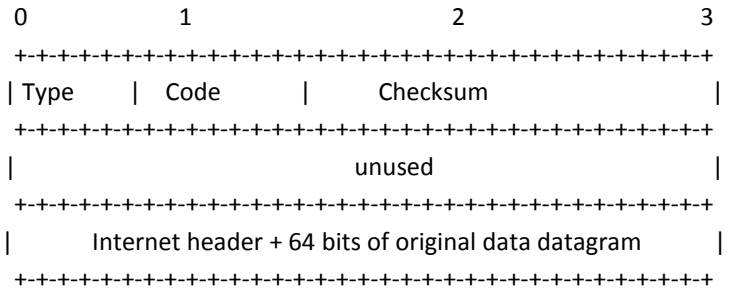
Konštanta *SOCK_RAW* špecifikuje, že chcem vytvoriť raw soket. Raw sokety nám dovolujú čítať a vytvárať pakety *ICMPv4*, *IGMPv4* a *ICMPv6*. Programy ping a traceroute využívajú raw sockety na kraftovanie vlastných paketov. Následne nastavím:

int on = 1;
setsockopt(socketIP,IPPROTO_IP,IP_HDRINCL,&on,sizeof(on));

S nastavením možnosti *IP_HDRINCL* máme možnosť vytvoriť si vlastnú IP hlavičku. Alokuje veľkosť dát pre ip a icmp hlavičku pomocou funkcie malloc. Do ip hlavičky pridám verziu = 4. Dôležité je nastaviť options ip hlavičky frag_off na *IP_DF*. Do protokolu priradíme *IPPROTO_ICMP*, keďže za ip hlavičkou nasleduje icmp hlavička. Zdrojovú adresu nastavím na 0. Týmto zabezpečím že zdrojovú ip adresu vyplní operačný systém.

V icmp hlavičke je podstatné nastaviť typ na *ICMP_ECHO*, code na 0. Do id paketu je priradené aktuálne id procesu. Do sequence nastavím aktuálnu sekvenciu-poradie odoslaného icmp paketu. Do checksum zavolám funkciu na výpočet kontrolného súčtu checksum, ktorú som prebral z RFC stránok.

Nastavíme časovač pomocou štruktúry timeval na 3 sekundy. Potom sa paket pošle pomocou funkcie sendto na zadanú ip adresu získanú skrz getaddrinfo. Funkciou select program počká na odpoveď. Pokiaľ odpoveď prišla získam ju pomocou funkcie recvfrom. Odpoveď príradíme do štruktúry s icmp a ip paketom. Zistíme typ icmp paketu. Pokiaľ dostanem echo reply zvýšime veľkosť paket. Pokiaľ sa príjme paket s typom Destination Unreachable a kódom Fragmentation needed naopak paket zmenším.



Štruktúra icmp paketu Destination Unreachable z RFC 772[3].

Veľkosť paketu sa hľada bínárným vyhľadávaním. Daný postup opakujem až dokým rozdiel hodnoty max a min z binárneho vyhľadávanie nie sú menšie ako 1. Inicializačná hodnota pre max hodnotu je 1500 ako pre ipv4 tak pre ipv6, keďže tak bolo definované zadanie úlohy. Je to taktiež hodnota ktorú dokáže ešte spracovať ethernet. Pokiaľ chceme hľadať väčšie veľkosti môžeme to špecifikovať prepínačom programu „-m veľkosť“. Veľosť min pri ipv4 som nastavil na 68 bytov. Pokiaľ sme na konci, vypíšem retazec “resume: %d”, kde %d je číslo MTU od zdroja k cieľu. U prijatého paketu kontrolujeme id paketu a sekvenciu paketu. Pokiaľ tieto dva identifikátory nespĺňajú podmienky očakávaného paketu, program pokračuje ďalej a znovu odošle ďalší paket s predošlou veľkosťou a na novo nastavenou sekvenciou paketu.

Verzia IPV6

Podobný postup som použil pre IPv6 verziu, teda funkciu *findMTU6*. Namiesto *AF_INET* a *IPPROTO_ICMP* som použil *AF_INET6* a *IPPROTO_ICMPV6*. Pomocou funkcie

int hops = 30;
setsockopt(socketIP,IPPROTO_IPV6,IPV6_UNICAST_HOPS,&on,sizeof(on));

Nastavíme ip hlavičke počet skokov na 30. Hned za tým je opäť funkcia

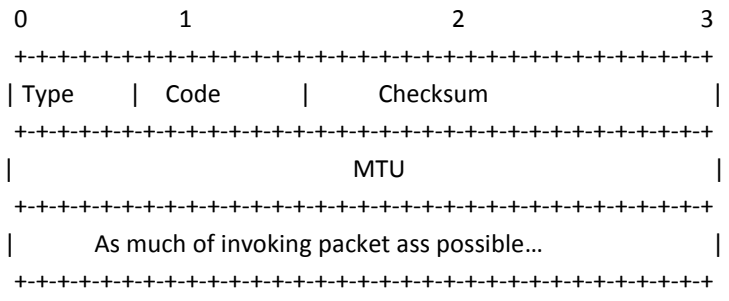
setsockopt(socketIP,IPPROTO_IPV6, IPV6_PMTUDISC_PROBE,&on,sizeof(on));

Ktorá zaručí aby zdrojový počítač pakety nefragmentoval. V našom sme použili radšej túto špecifikáciu keďže *IPV6_DONTFRAG*, ktorú definuje RFC3543[8] není na referenčnom linuxe

implementovaný. Rozdiel medzi ipv4 a ipv6 verziou je, že pri ipv6 verzii už nemusím vytvárať vlastnú ip hlavičku. Túto doplní sám operačný systém. Ja už iba nastavujem pomocou setsockopt ďaľšie vlastnosti ako je hop times. Vystačíme si teda už iba s icmpv6 hlavičkou, ktorú musíme opäť naplniť. Do štruktúry icmp6_hdr položku icmp6_hdr->icmp6_type nastavím na *ICMP6_ECHO_REQUEST*, položku code na 0, do icmp6_id priradím opäť číslo nášho procesu z *getpid()*. Nastavím taktiež sekvenciu icmp paketu, ktorú každým odoslaním zvýšim. V icmpv6 verzií už nevyplňám kontrolný súčet, ten vypočíta a doplní operačný systém. Nastavím štruktúru timeval takto:

timeval timeout;
timeout.tv_sec = TIMEOUT; // TIMEOUT == 3 sekundy
timeout.tv_usec = 0;

Tým povieme že, na odpoveď sa čaká tri sekundy. Následne sa zavolá funkcia sendto s ktorou pošleme na danú ipv6 adresu datagram. Pokiaľ nám vráti funkcia chybu, teda že paket nebol odoslaný, zmenšíme paket a pokúšame sa paket poslať znovu. Na zmenšenie a zväčšenie paketu používa program funkcie *reducePacketSize* a *risePacketSize*. Po sendto nasleduje select ktorý čaká 3 sekundy. Pokiaľ select neprešiel, vytvoríme nový datagram so zvýšenou sekvenciou. Pokiaľ select prešiel prejdem k príjmaniu dát pomocou recvfrom funkcie. Táto funkcia vráti dĺžku dát ktoré sme prijali a dáta vloži do bufferu cez ukazateľ. Dáta priradíme do icmp hlavičky. Pokiaľ má icmp hlavička typ *PACKET_TOO_BIG*, zmenšíme paket, pokiaľ dostaneme od cieľa odpoveď *ECHO_REPLY*, zvýšime paket. Pri *PACKET_TOO_BIG* využívam položku *MTU*, ktorá sa nachádza v pakete. Touto hodnotou nastavíme novú veľkosť paketu. Pokiaľ sa stane že paket obsahuje typ *DEST_UNREACH* teda že cieľ je nedostupný ukončíme program vypísaním chybovej hlášky.



Štruktúra icmp paketu *PACKET_TOO_BIG* z *RFC 4443*[2].

Pokiaľ má icmp hlavička akýkoľvek iný typ, program pokračuje ďalej a odošle znova paket s inkrementovaným sekvenčným číslom, pričom sa veľkosť paketu nezmení. V ipv6 sa kontrolojem id paketu iba pokiaľ je paket typu *ECHO_REPLY*, keďže *PACKET_TOO_BIG* neobsahuje položku pre id. Tento cyklus vytvorenie paketu, odoslanie a prijatie opakujeme dovtedy, dokým rozdiel hodnoty min a max je väčší ako 1. Ku výsledku pričítame ešte veľkosť ip hlavičky čo je 40 bytov, pretože tú zdroj dopĺňa automaticky sám pri ipv6. Uvedené binárne vyhľadávanie je presnejšie ako použitie tabuľky s dopredu definovanými **hodnotami**.

Použitie programu

Program preložte s priloženým súborom Makefile. Spustenie programu:

./mypmtud ipv6.google.com # spustenie s doménovým menom

./mypmtud -m 500 seznam.cz # program môžeme spustiť aj s prepínačom m

./mypmtud 2001::212:213:21::21 # môžeme spustiť program aj s ip adresou

Prepínač “m” nastavuje maximálnu možnú veľkosť paketu.

./mypmtud -h # vypíše nápovedu

Pre prehľadnosť som do programu pridal niekoľko dodatočných výpisov pre prehľadnosť “In ipv4 version” alebo “In ipv6 version”, podľa toho aký typ adresy nám ako prvé vráti funkcia getaddrinfo. Program vypíše pri nájdení PMTU reťazec “resume: %d\n” kde %d je nalezené číslo. Okrem toho, pokiaľ programu prejde časovač pri čakáni na paket vo funkcii select, vypíše sa “Time expires!...3sec\n”. Program taktiež dodatočne vypisuje reťazec “Try to send: %d\n”, kde %d je veľkosť aktuálne odosielaného paketu. Program sa ukončí v prípade že:

1. nájdeme hľadané MTU

2. programu príde icmp paket destination unreachable

3. nastane chyba pri vytváraní soketu

4. pri zle zadaných parametroch, prípadne zlej adrese

Z toho vyplíva, že program sa neustále pokúša odoslať dáta pri hľadaní aj keď vyprší časovač, prípadne icmp paket iného typu ako ECHO REPLY/PACKET_TOO_BIG/FRAG_NEEDED, podobne ako sa neustále snaží odoslať pakety program Ping.

Príklad behu programu, pri tomto behu je skutočná PMTU 1300:

./mypmtud -m 1500 google.com
In ipv4 version
Try to send: 784
Time expires!...3sec
Try to send: 784
Try to send: 1142
Try to send: 1321
Try to send: 1231

...
Try to send: 1300
Try to send: 1300
resume: 1300

Záver

Program *mypmtud* slúži na zistenie maximálnej možnej veľkosti datagramu odoslaného zo zdrojového pc na cieľ. Program je preložiteľný a odtestovaný na referenčnom systéme **linux Ubuntu 12**. Program bol odtestovaný jak v "domácom prostredí" kde ipv6 bola implementovaná skrz *Teredo*, tak v laboratórii *Cisco*, kde bola postavená vlastná sieť s počítačmi. Taktiež bol program testovaný na virtuálnej sieti *GNS3[7]*. Program bol vytvorený do predmetu ISA na VUT FIT v akademickom roku 2012/2013.

Použitá literatúra

[1] STEVENS, W. *UNIX network programming*. 2nd ed. Upper Saddle River: Prentice Hall PTR, c1998, xx, 1009 s. ISBN 01-349-0012-X.

[2] [online]. [cit. 2012-10-27]. Dostupné z: http://tools.ietf.org/html/rfc4443

[3] [online]. [cit. 2012-10-27]. Dostupné z: http://tools.ietf.org/html/rfc792

[4] [online]. [cit. 2012-10-27]. Dostupné z: http://tools.ietf.org/html/rfc1191

[5] [online]. [cit. 2012-10-27]. Dostupné z: http://tools.ietf.org/html/rfc2923

[6] [online]. [cit. 2012-10-27]. Dostupné z: http://tools.ietf.org/html/rfc4821

[7] [online]. [cit. 2012-11-17]. Dostupné z: http://www.gns3.net/download/

[8] [online]. [cit. 2012-10-27]. Dostupné z: http://www.ietf.org/rfc/rfc3542.txt