Optimisasi Pemrosesan Paket Data pada Gateway Subnetwork dengan Menggunakan Algoritma Greedy

Nitho Alif Ibadurrahman (13513072)[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]nithoalif@arc.itb.ac.id

***Abstrak*—Algoritma *greedy* merupakan algoritma yang umum digunakan untuk persoalan - persoalan optimasi. Algoritma *greedy* ini dapat diterapkan pada pemrosesan paket data yang dilakukan oleh sebuah *gateway*, yang berada pada sebuah *subnet*, untuk meningkatkan performansi pengiriman/*transfer data* antar komputer/mesin, baik yang berada dalam sebuah *subnet*, maupun antar *subnet* lainnya.**

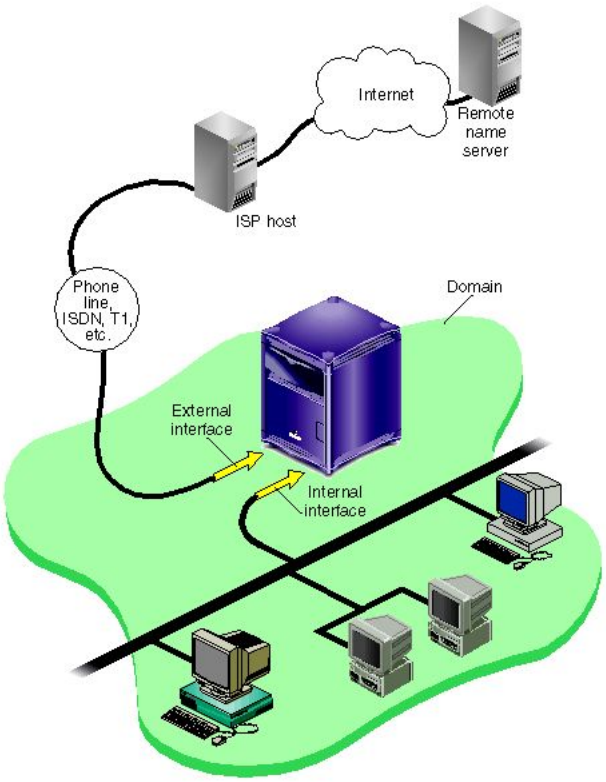
***Kata Kunci*—Optimisasi Paket *Data*, *Subnet*, *Gateway*, *Greedy*.**

I. PENDAHULUAN

Saat ini, *internet* merupakan bagian integral yang tidak dapat dipisahkan dalam kehidupan manusia sehari - hari. Seiring dengan berjalannya waktu, penggunaan *internet* akan semakin tinggi. Pada tahun 2011 saja, setidaknya terjadi perpindahan data sebesar 3 juta TB data dalam kurun waktu satu bulan [1].

Internet sendiri merupakan perpaduan antara komponen - komponen yang disebut dengan jaringan, yang tersebar di seluruh dunia. Komponen - komponen inilah yang menjadi sumber dan tujuan dari perpindahan *data* tersebut. Di dalam suatu jaringan, mungkin juga terdapat jaringan lain yang lebih kecil. Jaringan yang lebih kecil ini disebut subjaringan (*subnetwork*), atau yang lebih dikenal dengan *subnet* [2].

Pada sebuah *subnet*, setidaknya terdapat satu buah komputer/mesin yang bekerja sebagai penghubung antara jaringan (*subnet*) tersebut dengan jaringan (*subnet*) lain. Komputer/mesin inilah yang disebut dengan *gateway*. *Gateway* ini juga dapat bekerja sebagai *router* yang mampu meneruskan paket - paket data dari komputer/mesin yang ada dibawahnya (secara hierarkis) [3]. Gambar 1. Menunjukkan alur penerusan paket data (*routing*) mulai dari pengirim, melewati sebuah *gateway*, hingga ke komputer/mesin tujuan.



Gambar 1. Ilustrasi *internet gateway*, pada suatu *subnet*.
Sumber : http://techpubs.sgi.com/library/dynaweb_docs/0620/SGI_Admin/books/Gateway_IG/sgi_html/ch01.html

Dikarenakan cara kerjanya yang demikian, sebuah *gateway* harus terus menerus melayani komputer/mesin yang ada dibawahnya. Oleh karena itu, performansi maksimal dari suatu jaringan/*subnet* sangatlah bergantung kepada *gateway*-nya. Jika komputer/mesin - mesin yang ada dibawahnya perlu mengirimkan paket data dalam jumlah besar, seperti yang terjadi pada transfer file, pada saat yang bersamaan, *gateway* harus mencoba memilah-milah mana yang harus diteruskan terlebih dahulu.

Layaknya persoalan knapsack (*knapsack problem*), *gateway* harus menentukan paket mana saja yang terlebih dahulu harus diteruskan dalam rentang waktu tertentu. Pada makalah ini penulis akan mencoba menganalisis strategi optimisasi pemrosesan paket *data* tersebut dengan menggunakan algoritma *greedy*.

II. LANDASAN TEORI

A. Algoritma Greedy

Algoritma *greedy* merupakan metode yang paling populer untuk memecahkan persoalan optimasi. Prinsip dari *greedy* sendiri ialah "*take what you can get now!*". Dengan kata lain, prinsip dari algoritma *greedy* ialah mencoba untuk mengambil semua yang dapat diambil pada suatu saat tertentu (*greedy*/rakus). Algoritma *greedy* mencoba membangun sebuah solusi melalui urutan langkah - langkah, yang masing - masing langkahnya merupakan pengembangan dari solusi parsial yang sudah didapatkan sampai saat itu. Hal ini dilakukan sampai dengan solusi keseluruhan dicapai (*complete*) [4].

Terdapat banyak pilihan yang perlu dieksplorasi pada setiap langkah solusi. Oleh karena itu, pada setiap langkah harus dibuat keputusan yang terbaik dalam menentukan pilihan. Keputusan yang telah diambil pada suatu langkah tidak dapat diubah lagi pada langkah selanjutnya. Pendekatan yang digunakan di dalam algoritma *greedy* ialah:

1. Mengambil pilihan yang terbaik yang dapat diperoleh pada saat itu tanpa memperhatikan konsekuensi ke depan (prinsip "*take what you can get now!*");
2. Berharap bahwa dengan memilih optimum lokal pada setiap langkah akan berakhir dengan optimum global.

Pada setiap langkah diperoleh optimum lokal. Bila algoritma berakhir, kita berharap optimum lokal menjadi optimum global. Secara garis besar, terdapat tiga aturan dalam setiap pengambilan langkah, yaitu:

1. *Feasible* — memenuhi batasan persoalan;
2. *Local optimum* — harus merupakan pilihan terbaik diantara semua pilihan yang ada;
3. *Irrevocable* — jika sudah dilakukan, pilihan yang telah diambil tersebut tidak dapat dikembalikan dan diubah lagi.

Berikut ini merupakan komponen - komponen penyusun yang ada pada algoritma *greedy* [5]:

a) Himpunan kandidat
   Berisi elemen-elemen pembentuk solusi.
b) Himpunan solusi
   Berisi kandidat-kandidat yang terpilih sebagai solusi persoalan.
c) Fungsi seleksi (*selection function*)
   Memilih kandidat yang paling memungkinkan mencapai solusi optimal. Kandidat yang sudah dipilih pada suatu langkah tidak pernah dipertimbangkan lagi pada langkah selanjutnya.
d) Fungsi kelayakan (*feasible*)
   Memeriksa apakah suatu kandidat yang telah dipilih dapat memberikan solusi yang layak, yakni kandidat tersebut bersama-sama dengan himpunan solusi yang sudah terbentuk tidak melanggar kendala (constraints) yang ada. Kandidat yang layak dimasukkan ke dalam himpunan solusi, sedangkan kandidat yang tidak layak dibuang dan tidak pernah dipertimbangkan lagi.
e) Fungsi objektif
   Fungsi yang memaksimumkan atau meminimumkan nilai solusi (misalnya panjang lintasan, keuntungan, dan lain-lain).

Pseudocode dari algoritma *greedy* ialah sebagai berikut:

```
function greedy(input C:
himpunan_kandidat)→ himpunan_solusi
{ Menentukan solusi optimum dari
persoalan optimasi dengan algoritma
greedy
 Masukan: himpunan kandidat C
 Keluaran: himpunan solusi S
}

Deklarasi
   x : kandidat
   S : himpunan_kandidat

Algoritma:
{inisialisasi S dengan kosong}
S ← {}

while ((not SOLUSI(S)) and (C ≠ {})) do
   {pilih sebuah kandidat dari C}
   x ← SELEKSI(C)
   {elemen himpunan kandidat berkurang 1}
   C ← C - {x}
   if (LAYAK(S ∪ {x})) then
     S ← S ∪ {x}
   endif
endwhile
{SOLUSI(S) sudah diperoleh or C = {}}

if (SOLUSI(S)) then
   return S
else
   write "tidak ada solusi"
endif
```

B. Knapsack Problem

Knapsack problem merupakan sebuah persoalan yang didefiniskan sebagai berikut:

a) Terdapat n buah barang yang masing - masing memiliki berat sebesar w_1, w_2, w_3, ..., w_n dan nilai sebesar v_1, v_2, v_3, ..., v_n.
b) Terdapat sebuah tas ransel dengan kapasitas W.

Persoalan ini mendefinisikan bagaimana cara terbaik untuk memilih barang yang dapat dimasukkan ke dalam tas ransel, agar mendapatkan keuntungan/nilai sebesar - besarnya tanpa melebihi kapasitas dari ransel tersebut. Secara matematis, persoalan in dapat dirumuskan sebagai berikut:

Maksimasi:

$$F = \sum p_i \, x_i \;;$$

Dengan kendala (*constraint*):

$$\sum w_i \, x_i \leq W \;;$$

Yang dalam hal ini:

$x_i = 0$ atau 1, i = 1, 2, ..., n

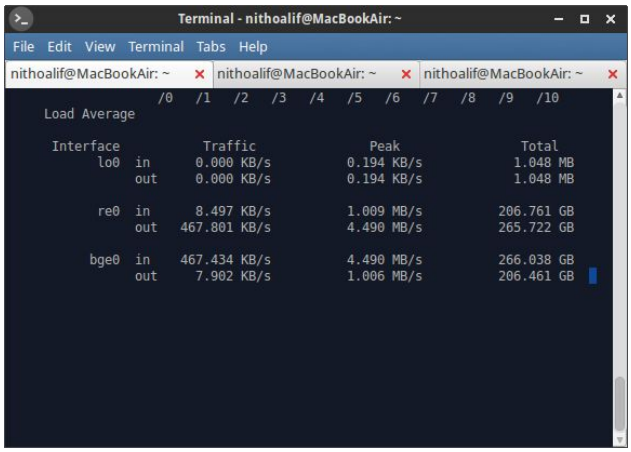
III. Analisis Dan Pembahasan

3.1. Percobaan

Pada analisis kali ini, penulis akan menginstansiasi permasalahan dengan mencoba mengisolasi paket *data* yang masuk dan keluar dari suatu *gateway*. Disini penulis akan mesimulasikan paket *data* yang keluar-masuk suatu gateway dengan dengan cara men-*transfer* suatu *file* dari *sebuah server* ke sebuah *client*. Penulis juga akan mempergunakan bantuan kakas yang disebut dengan `tcpdump` dan `systat` guna melihat paket data lang lewat, serta beban dari sebuah *gateway*. Sistem yang akan dipergunakan penulis ialah tiga buah komputer yang berjalan dengan sistem operasi *FreeBSD*. Ketiga komputer tersebut memiliki peran sebagai berikut:

1. *Gateway*
   Komputer *gateway* yang akan di-*monitor* untuk melihat detail paket *data* yang harus diproses. Komputer ini menggunakan *IP address* `167.205.3.1`.
2. *Server*
   Komputer/mesin yang akan digunakan sebagai penyedia *file*. Komputer ini menggunakan *IP address* `167.205.3.120`.
3. *Client*
   Komputer/mesin yang akan digunakan sebagai penerima *file*. Komputer ini menggunakan *IP address* `167.205.3.64`.

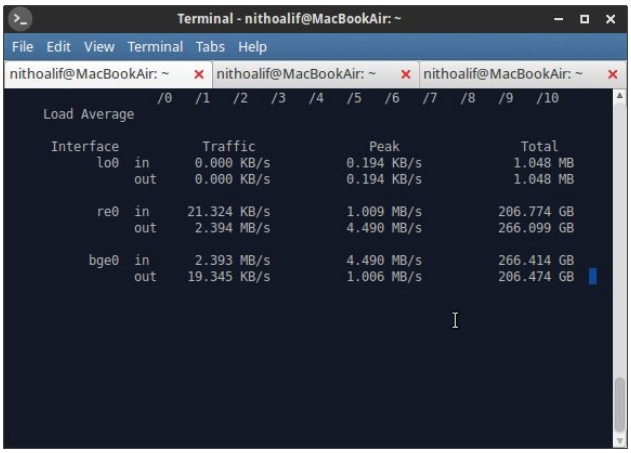
Untuk mengetahui keadaan awal dari sebuah *gateway*, penulis akan melakukan perintah `systat -ifstat 1` guna mengetahui beban (*load*) sebuah *Network Interface Card* (NIC) pada *gateway* saat ini. Gambar 2 menunjukkan beban dari dari NIC *gateway*.



Gambar 2. Load awal NIC pada *gateway*.

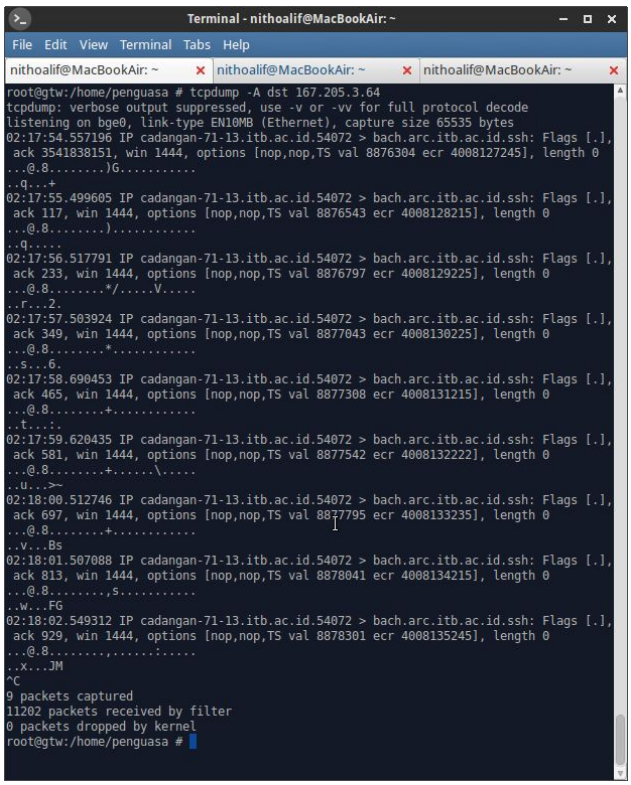
Pada *gateway* tersebut terdapat 2 NIC dan 1 loopback. NIC `re0` digunakan sebagai *uplink*, atau sebagai penghubung ke jaringan/*subnet* lain. NIC `bge0` digunakan sebagai penghubung antar komputer/mesin dibawah *gateway* tersebut (*downlink*). Fokus penulis saat ini ialah pada beban *gateway* yang dihasilkan akibat komunikasi antar komputer/meisn dalam satu *subnet*. Dengan demikian, NIC yang akan menjadi fokus analisis ialah `bge0`.

Komputer yang bertindak sebagai *server* akan diatur sebagai *FTP Server*, sehingga komputer yang bertindak sebagai *client* dapat mengambil file pada komputer Server tersebut dengan protokol FTP. Setelah *client* mengirimkan pesan untuk mengambil file di *server*, yang sudah pasti melalui *gateway* untuk diteruskan ke *server*, penulis dapat mengetahui beban yang harus ditanggung *gateway* tersebut. Gambar 3 menunjukkan beban pada saat proses transfer *file* berlangsung.



Gambar 3. Load di NIC pada saat transfer.

Dari kedua gambar diatas, dapat diketahui bahwa terjadi peningkatan beban sekitar 1.9 MB/s pada NIC `bge0` pada saat proses transfer file. Gambar 4 meunjukkan hasil dari paket yang keluar masuk *gateway* pada saat proses transfer file. Gambar berikut diambil dengan menggunakan perintah `tcpdump -A dst 167.205.3.64`.



Gambar 4. Paket yang melalui gateway dan diteruskan ke client (IP 167.205.3.64).

Dari ketiga gambar di atas dapat disimpulkan bahwa *gateway* akan mengangung beban yang cukup signifikan setiap kali ada permintaan. Permasalahan pemrosesan permintaan (*request*) di *gateway* suatu *subnet* sendiri dapat dimodelkan dengan *knapsack problem*. Hal ini dikarenakan keduanya merupakan merupakan bentuk dari *decision problem*. Salah satu penyelesaian dari *knapsack problem* ialah dengan menggunakan algoritma *greedy*. Oleh karena itu, disini penulis akan mencoba menguraikan solusi dari permasalahan tersebut dengan menggunakan algoritma *greedy*.

3.1. Uraian

Pada dasarnya, sebuah paket *data* memiliki dua atribut, yaitu *control information* dan *user data* [7]. Pada penguraian solusi kali ini, penulis akan menggunakan tiga pendekatan pada algoritma *greedy*, yang di antaranya:

a) Greedy berdasarkan ukuran/panjang paket
   Pendekatan dengan memperhatikan ukuran/panjang (*length*) paket yang ada pada *user data* pada sebuah paket (*Greedy by length/size*).
b) Greedy berdasarkan prioritas
   Pendekatan dengan memperhatikan waktu kadaluarsa paket (*Time-To-Live*/TTL) yang ada pada *control information* pada sebuah paket (*Greedy by priority*).
c) Greedy berdasarkan pemecahan menjadi paket - paket yang lebih kecil
   Pendekatan dengan mencoba memecah suatu pake menjadi paket - paket lain dengan ukuran yang lebih kecil dan membandingkannya dengan TTL (*Greedy by fractional size/gain*).

3.1.A. Definisi Kasus

Untuk mempermudah analisis kasus diatas, penulis mencoba menyederhadakan persoalan dengan contoh berikut. Dimisalkan pada sebuah *subnet* terdapat sebuah *gateway* dengan NIC yang hanya mampu meneruskan *request* sebesar 4096 *bytes* dalam kurun waktu 600 ms. Pada saat yang bersamaan, terdapat lima *request* yang masuk ke dalam *gateway* tersebut. Masing - masing request tersebut memiliki atribut sebagai berikut:

Paket ke-	Length (*bytes*)	TTL (ms)
1	48	60
2	1024	180
3	512	150
4	4096	360
5	768	120

A. Solusi Greedy by Length/Size

Jika sebuah *gateway* menggunakan pendekatan *greedy by length*, yaitu dengan mengambil semua paket *data* dengan ukuran terbesar pada saat pemrosesan *request*, selama paket tersebut belum kadaluarsa, penulis akan mendapatkan *data* sebagai berikut:

Pemrosesan ke-	Paket ke-	Besar paket yang telah diproses (bytes)	Sisa Waktu (ms)
0	0	0	600
1	4	0 + 4096 = 4096	600 - 360 = 240
2	2	4096 + 1024 = 5120	240 - 180 = 60
3	1	5120 + 48 = 5168	60 - 60 = 0

Total paket yang dapat diproses dengan pendekatan ini ialah 5168 *bytes*, dengan sisa waktu 0 ms. Di bawah ini merupakan *pseudocode* yang digunakan untuk mendapatkan *data* di atas; dengan *sumTTL* merupakan sebuah fungsi yang menghasilkan penjumlahan TTL dari suatu himpunan paket; sedangkan *maxBytes* adalah fungsi yang menghasilkan suatu paket dengan ukuran *bytes* terbesar dari suatu himpunan paket.

```
function greedyBytes(input C:
himpunan_paket, maxTTL: TTL_maksimal)→
himpunan_paket
{ Mengembalikan himpunan paket dengan
ukuran - ukuran terbesar
}

Deklarasi
   x : paket
   S : himpunan_paket

Algoritma:
{inisialisasi S dengan kosong}
S ← {}

while (sumTTL(S) < max_TTL) and (C ≠ {})
do
   {pilih sebuah kandidat dari C}
   x ← maxBytes(C)
   {elemen himpunan kandidat berkurang 1}
   C ← C - {x}
   S ← S ∪ {x}
endwhile

return S
```

B. Solusi Greedy by Priority

Sebuah *gateway* bisa saja menggunakan pendekatan *greedy by priority*, yaitu dengan mengasumsikan bahwa semakin banyak paket yang diproses, semakin besar pula ukuran paket yang bisa diproses. Hal ini dilakukan dengan cara mengambil semua paket *data* dengan TTL terkecil (prioritas paling besar) pada saat pemrosesan *request*. *Gateway* yang demikian akan menghasilkan *data* sebagai berikut:

Pemrosesan ke-	Paket ke-	Besar paket yang telah diproses (*bytes*)	Sisa Waktu (ms)
0	0	0	600
1	1	0 + 48 = 48	600 - 60 = 540
2	5	48 + 768 = 816	540 - 120 = 420
3	3	816 + 512 = 1328	420 - 150 = 270
4	2	1328 + 1024 = 2352	270 - 180 = 90

Total paket yang dapat diproses dengan pendekatan ini ialah 2352 *bytes*, dengan sisa waktu 90 ms. Di bawah ini merupakan *pseudocode* yang digunakan untuk mendapatkan data di atas; dengan *sumTTL* merupakan sebuah fungsi yang menghasilkan penjumlahan TTL dari suatu himpunan paket; sedangkan *minTTL* adalah fungsi yang menghasilkan suatu paket dengan ukuran TTL terkecil dari suatu himpunan paket.

```
function greedyTTL(input C:
himpunan_paket, maxTTL: TTL_maksimal)→
himpunan_paket
{ Mengembalikan himpunan paket dengan
TTL - TTL terkecil
}

Deklarasi
   x : paket
   S : himpunan_paket

Algoritma:
{inisialisasi S dengan kosong}
S ← {}

while (sumTTL(S) < max_TTL) and (C ≠ {})
do
   {pilih sebuah kandidat dari C}
   x ← minTTL(C)
   {elemen himpunan kandidat berkurang 1}
   C ← C - {x}
   S ← S ∪ {x}
endwhile

return S
```

B. Solusi Greedy by Fractional Size/Gain

Dikarenakan bentuk dari sebuah paket bergantung kepada siapa yang mengirim dan standar yang digunakan, dapat diasumsikan bahwa sebuah paket yang utuh merupakan gabungan dari beberapa paket yang lebih kecil, dengan persebaran yang merata (*uniform*). Sebuah *gateway* dapat memproses *request* melalui pendekatan *greedy by gain*, yaitu dengan membandingkan antara besarnya paket suatu *data*, dengan TTL-nya. Setelah itu *gateway* akan memproses paket - paket tersebut dimulai dengan perbandingan terbesar, baru kemudian dilanjutkan dengan yang lebih kecil. Dengan demikian, *data* paket - paket *request* akan menjadi seperti ini:

Paket ke-	Length (*bytes*)	TTL (ms)	Gain (*bytes*/ms)
1	48	60	0.8
2	1024	180	5.688
3	512	150	3.413
4	4096	360	11.37
5	768	120	6.4

Berikut ini merupakan data hasil pemrosesan dari *gateway* yang menggunkanan pendekatan *greedy by fractional size/gain*:

Pemrosesan ke-	Paket ke-	Besar paket yang telah diproses (bytes)	Sisa Waktu (ms)
0	0	0	600
1	4	0 + 4096 = 4096	600 - 360 = 240
2	5	4096 + 768 = 4864	240 - 120 = 120

3	1	4864 + 48 = 4912	120 - 60 = 60

Total paket yang dapat diproses dengan pendekatan ini ialah 4912 *bytes*, dengan sisa waktu 60 ms. Di bawah ini merupakan *pseudocode* yang digunakan untuk mendapatkan *data* di atas; dengan *sumTTL* merupakan sebuah fungsi yang menghasilkan penjumlahan TTL dari suatu himpunan paket; sedangkan *maxGain* adalah fungsi yang menghasilkan suatu paket dengan perbandingan antara besar paket dan TTL yang terbesar dari suatu himpunan paket.

```
function greedyGain(input C:
himpunan_paket, maxTTL: TTL_maksimal)→
himpunan_paket
{ Mengembalikan himpunan paket dengan
perbandingan antara ukuran paket dengan
TTL terbesar
}

Deklarasi
   x : paket
   S : himpunan_paket

Algoritma:
{inisialisasi S dengan kosong}
S ← {}

while (sumTTL(S) < max_TTL) and (C ≠ {})
do
   {pilih sebuah kandidat dari C}
   x ← maxGain(C)
   {elemen himpunan kandidat berkurang 1}
   C ← C - {x}
   S ← S ∪ {x}
endwhile

return S
```

3.1.B. Hasil Analisis Kasus

Dari ketiga pendekatan yang berbeda di atas, akan didapatkan hasil perbandingan sebagai berikut:

Pendekatan	Besar paket yang dapat diproses (bytes)	Sisa Waktu (ms)
Greedy by size	5168	0
Greedy by priority	2352	90
Greedy by fractional size/gain	4912	60

Dari tabel di atas, dapat disimpulkan bahwa untuk contoh kasus ini, algoritma *greedy* dengan pendekatan panjang/ukuran paket akan memberikan hasil paling baik. Hal ini dikarenakan ketidakjelasan hubungan antara ukuran suatu paket dengan TTL-nya. Besar TTL dari suatu paket seluruhnya merupakan kebijakan dari sang pengirim. Oleh sebab itu, pendekatan dengan prioritas (TTL terkecil) dan pendekatan dengan pemecahan paket menjadi paket - paket yang lebih kecil (pembagian dengan TTL) akan menuaikan hasil yang tidak tentu.

IV. KESIMPULAN

Dikarenakan ketidakteraturan dari *Time-To-Live* (TTL) suatu paket *data*, pemrosesan paket *data* dengan algoritma *greedy* oleh suatu *gateway* dalam *subnet*, paling baik dilakukan dengan pendekatan ukuran paket terbesar (*greedy by length/size*).

V. UCAPAN TERIMA KASIH

Pertama - tama penulis ingin mengucapkan syukur kepada Tuhan Yang Maha Esa, karena hanya oleh karena rahmat-Nya penulis dapat menyelesaikan tulisan ini. Penulis juga ingin megucapkan berterima kasih kepada dosen mata kuliah IF2211 Strategi Algoritma, yaitu Ibu Dr. Nur Ulfa Maulidevi, S.T., M.Sc. dan Bapak Dr. Ir. Rinaldi Munir atas bimbingan dan jasa beliau yang selama ini telah mengajar dan memberikan ilmu bagi penulis, sehingga penulis mampu membuat tulisan ini. Tak lupa juga penulis berterima kasih atas rekan - rekan mahasiswa Teknk Informatika 2013, serta Kru Amateur Radio Club yang senantiasa memberikan dorongan serta semangat kepada penulis.

REFERENSI

[1] University of Minnesota. (2009). *Minnesota Internet Traffic Studies (MINTS)*. Diakses pada tanggal 4 Mei 2015, dari: http://www.dtc.umn.edu/mints/

[2] J. Mogul, Internet Engineering Task Force (IETF). (1985). *RFC 950 - Internet Standard Subnetting Procedure*. Diakses pada tanggal 4 Mei 2015, dari: http://tools.ietf.org/html/rfc950

[3] A. Anderson, The Linux Documentation Project. (1996). *Gateways*. Diakses pada tanggal 4 Mei 2015, dari: http://tldp.org/LDP/nag/node30.html#SECTION004430000

[4] Munir, Rinaldi. (2004). *Diktat Strategi Algoritmik*. Bandung : Departemen Teknik Informatika, Institut Teknologi Bandung.

[5] A. Levitin. (2012). *The Design & Analysis of Algorithms*. New Jersey : Pearson Education, Inc.

[6] D. Pisinger, dkk. (2004). *Knapsack Problems*. Springer-Verlag Berlin Heidelberg.

[7] The TCP/IP Guide. Understanding The OSI Reference Model: An Analogy. Diakses pada tanggal 4 Mei 2015, dari: http://www.tcpipguide.com/free/t_UnderstandingTheOSIReferenceModelAnAnalogy.htm

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 5 Mei 2015

Nitho Alif Ibadurrahman (13513072)