BAB II

LANDASAN TEORI

2.1 Pengertian Jaringan Sensor Nirkabel (JSN)

JSN adalah suatu infrastruktur jaringan nirkabel yang terdiri dari sejumlah besar *node* sensor yang tersebar di suatu area. Dewasa ini perkembangan JSN mengalami kemajuan yang pesat. Hal ini terjadi karena adanya suatu kebutuhan akan jaringan sensor yang memilik kriteria yang sangat baik dalam hal efisiensi operasional dan performansi. JSN menjadi suatu fenomena baik bagi dunia industri maupun kalangan akademis, karena aplikasi JSN yang mencakup berbagai bidang. Hal ini di dukung oleh fakta bahwa sekitar 98% prosesor bukan berada di dalam sebuah komputer PC/laptop seperti kebanyakan, namun terintegrasi dalam aplikasi militer, kesehatan, *remote control*, *chip robotic*, alat komunikasi dan mesin-mesin industri yang didalamnya telah di pasang sensor. (Stephanie, 2011).

Menurut Stephanie (2011) menjelaskan bahwa teknologi JSN dapat memonitor dan mengontrol temperatur, kelembaban, kondisi cahaya, level derau, pergerakan suatu objek dan lain sebagainya. Dapat disimpulkan bahwa JSN adalah sebuah penghubung antara lingkungan fisik (*physical world*) dan sensor (*digital world*).

Berikut adalah beberapa keuntungan yang bisa diperoleh dari teknologi JSN :

1. Praktis / ringkas karena tidak perlu ada instalasi kabel yang rumit dan dalam kondisi geografi tertentu sangat menguntungkan dibanding *wired sensor*. Sensor menjadi bersifat *mobile*, artinya pada suatu saat dimungkinkan untuk memindahkan sensor untuk mendapat pengukuran yang lebih tepat tanpa harus khawatir mengubah desain ruangan maupun susunan kabel ruangan.

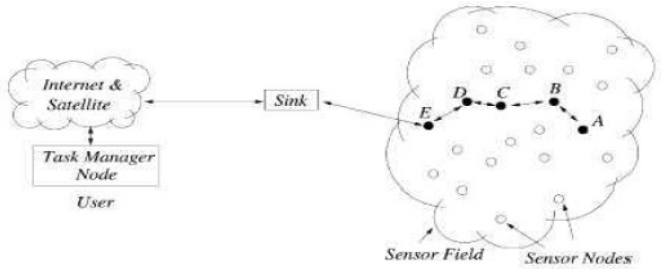
2. Meningkatkan efisiensi operasional.
3. Mengurangi total biaya sistem secara signifikan.
4. Dapat mengumpulkan data dalam jumlah besar.
5. Memungkinkan komunikasi digital 2 arah.

Menyediakan konektivitas internet yang secara global, kapanpun, dimanapun informasi tersebut dapat diakses melalui *server*, *laptop*, dsb.

2.2 Arsitektur Jaringan Sensor Nirkabel (JSN)

Pada JSN, *node* sensor disebar dengan tujuan untuk menangkap adanya gejala atau fenomena yang hendak diteliti. Jumlah *node* yang disebar dapat ditentukan sesuai kebutuhan dan tergantung beberapa faktor misalnya luas area, kemampuan sensing *node*, dan sebagainya. Tiap *node* memiliki kemampuan untuk mengumpulkan data dan meroutingkannya kembali ke *base station* serta berkomunikasi dengan *node* lainnya. *Node* sensor dapat mengumpulkan data dalam jumlah yang besar dari gejala yang timbul dari lingkungan sekitar.

Arsitektur JSN secara umum dapat direpresentasikan oleh Gambar 2.1.



Gambar 2.1 Arsitektur JSN secara umum

Node sensor yang berukuran sangat kecil disebar dalam jumlah besar di suatu area sensor. *Node* sensor tersebut memiliki kemampuan untuk saling berkomunikasi dan

meroutingkan data yang dikumpulkan ke *node* lain yang berdekatan. Data yang akan dikirim melalui transmisi radio akan diteruskan menuju *Base Station* (BS) yang merupakan penghubung antara *node* sensor dan user. Informasi tersebut dapat diakses melalui berbagai *platform* seperti koneksi internet atau satelit sehingga memungkinkan user untuk dapat mengakses secara *realtime* melalui *remote server*.

2.3 *Mode* pada *Node Sensor*

Urutan aktivasi *mode-mode* pada *node* sensor dapat direpresentasikan oleh Gambar 2.2.

Gambar 2.2 Urutan *mode* pada *node* sensor

Pada Gambar 2.2 menunjukkan bahwa *mode-mode* tersebut dijalankan dengan urutan tertentu. Tiap *mode* memiliki karakteristik yang berbeda tergantung dari aktivitas yang sedang dilakukan *node*, apakah sedang melakukan proses transmisi atau sedang *standby* dan seterusnya. Hal ini mengakibatkan energi yang digunakan tiap *mode* juga berbeda-beda. Semakin ke kiri, maka *mode* tersebut mengeluarkan energi yang semakin rendah, begitu juga sebaliknya. Suatu *mode* harus melalui *mode* disampingnya jika ingin berganti *mode*. Misalnya *mode off* harus melalui *mode sleep* dan *standby* terlebih dahulu jika akan melakukan transmisi. Apabila tidak ada aktivitas observasi atau transmisi, sebaiknya dijalankan *mode off* atau *sleep*. Hal ini perlu diperhatikan karena proses

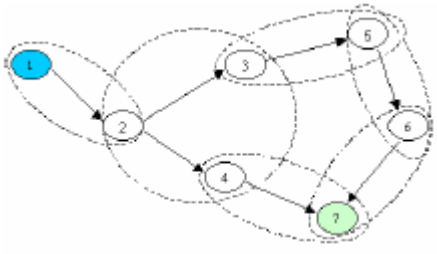
transmisi dan observasi cenderung menggunakan energi yang lebih besar. Pada *mode ready*, *node sensor* dapat melakukan kedua *mode* baik *transmit* dan *receive*.

2.4 Protokol *Routing On-demand Routing*

2.4.1 *Dynamic Source Routing* (DSR)

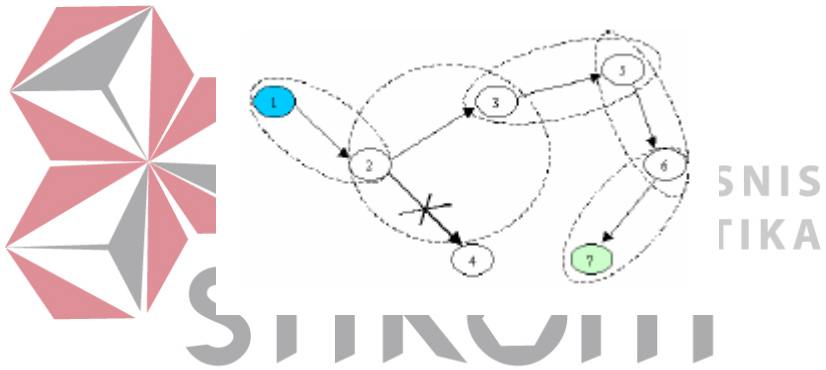
DSR merupakan protokol dimana *node* sumber menentukan rute paket yang dikirim setelah mengetahui serangkaian rute yang lengkap. Proses *routing* pada protokol ini terdiri atas 2 mekanisme, yaitu *Route Discovery* dan *Route Maintenance*. *Route discovery* yaitu kondisi dimana *node* ingin mengirimkan paket data ke tujuan yang belum diketahui rutenya, sehingga sumber mengirim *route request* (RREQ). RREQ akan melakukan proses *flooding*, yaitu proses pengiriman data atau *control message* ke setiap *node* pada jaringan untuk mencari rute tujuan. RREQ akan tersebar ke seluruh *node* dalam jaringan. Tiap *node* akan mengirim paket RREQ ke *node* lain kecuali *node* tujuan. Kemudian *node-node* yang menerima RREQ akan mengirim paket *route reply* (RREP) ke *node* yang mengirim RREQ tersebut. Setelah rute ditemukan, *node* sumber mulai mengirim paket data.

Gambar 2.3 merupakan ilustrasi mekanisme kerja *Route Discovery* menurut Dicky Rachmad P. (2007).



Gambar 2.3 Mekanisme *Route Discovery*

Route Maintenance adalah mekanisme dimana sumber mendeteksi adanya perubahan topologi jaringan sehingga pengiriman paket mengalami kongesti. Hal ini disebabkan ketika salah satu *node* yang terdaftar dalam rute sebelumnya bergerak menjauh dari *range node* yang lain. Saat masalah tersebut terdeteksi, paket *route error* (RERR) akan dikirim ke *node* pengirim. Saat RERR diterima, hop ke *node* yang menjauh akan dihilangkan dari *route cache*. Kemudian rute lain yang masih tersimpan di *cache* akan digunakan. Saat tidak ada rute yang tersisa, protokol DSR akan melakukan proses *route discovery* untuk menemukan rute baru. Gambar mekanisme *Route Maintenance* dapat dilihat pada Gambar 2.4. (Pambudi , Dicky R, 2007)



Gambar 2.4 Mekanisme *Route Maintenance*

Keuntungan dari penggunaan algoritma DSR adalah sebagai berikut :

1. *Intermediate node* tidak perlu memelihara secara *up to date* informasi *routing* pada saat melewatkan paket, karena setiap paket selalu berisi informasi *routing* di dalam *header*nya.
2. DSR menghilangkan proses *periodic route advertisement* dan *neighbour detection.*
3. DSR memiliki kinerja yang paling baik dalam hal *throughput, routing overhead* (pada paket) dan rata-rata panjang *path*.

Sedangkan, kerugian dari protokol *routing* ini adalah sebagai berikut :

1. Mekanisme *route maintenance* tidak dapat memperbaiki *link* yang rusak atau *down*.
2. DSR memiliki delay yang buruk untuk proses pencarian rute baru.

Menurut Dicky Rachmad P. (2007), penggunaan protokol DSR akan sangat optimal pada jumlah *node* yang kecil atau kurang dari 200 *node*.

2.4.2 *Ad Hoc On-demand Distance Vector* (AODV)

Menurut Dwi Nofianti, dkk (2011), AODV merupakan sebuah protokol *routing On-Demand* yang hanya akan membangun rute antar *node* jika diinginkan oleh *source node*. Rute akan disimpan selama masih dibutuhkan oleh *source node* (proses pemeliharaan rute). AODV menggunakan *sequencial number* untuk memastikan bahwa rute yang dihasilkan adalah rute *loop-free* dan memiliki informasi *routing* yang paling *update*.

AODV menciptakan suatu rute dengan menggunakan *Route Request* (RREQ) dan *Route Reply* (RREP). Ketika *source node* menginginkan suatu rute menuju *destination node*, tetapi belum mempunyai rute yang benar, maka *source node* akan menginisialisasi *route discovery process* untuk menemukan rute ke *destination node*. *Source node* akan mem-*broadcast* paket RREQ menuju *neighbour node*. RREQ paket berisi *source address*, *destination address*, *hop counter*, *source* and *destination sequence number*, dan *broadcast* ID. Nilai *broadcast* ID akan bertambah satu setiap suatu *source node* mengirimkan RREQ yang baru dan digunakan untuk mengidentifikasi sebuah paket RREQ. Jika *node* yang menerima RREQ memiliki informasi rute menuju *destination node*, maka *node* tersebut akan mengirim paket RREP kembali menuju

source node. Tetapi jika tidak mengetahuinya, maka *node* tersebut akan mem-*broadcast* ulang RREQ ke *node tetangganya* setelah menambahkan nilai *hop counter*. *Node* yang menerima RREQ dengan nilai *source address* dan *broadcast* ID yang sama dengan RREQ yang diterima sebelumnya akan membuang RREQ tersebut. *Source sequence number* digunakan oleh suatu *node* untuk memelihara informasi yang *valid* mengenai *reverse path* (jalur balik) menuju *source node*. Pada saat RREQ mengalir menuju *node* tujuan yang diinginkan, dia akan menciptakan *reverse path* menuju ke *node*, setiap *node* akan membaca RREQ dan mengidentifikasi alamat dari *neighbour node* yang mengirim RREQ tersebut. Ketika *destination node* atau *node* yang memiliki informasi rute menuju *destination* menerima RREQ maka *node* tersebut akan membandingkan nilai *destination sequence number* yang dia miliki dengan nilai *destination sequence number* yang ada di RREQ. Jika nilai *destination sequence number* yang ada di RREQ lebih besar dari nilai yang dimiliki oleh *node*, maka paket RREQ tersebut akan di*broadcast* kembali ke *neighbour node*, sebaliknya jika nilai *destination sequence number* yang ada di *node* lebih besar atau sama dengan nilai yang ada di RREQ maka *node* tersebut akan mengirim route reply (RREP) menuju *source node* dengan menggunakan *reverse path* yang telah dibentuk oleh RREQ. *Intermediate node* yang menerima RREP akan meng*update* informasi *timeout* (masa aktif rute) jalur yang telah diciptakan. Informasi rute *source* ke *destination* akan dihapus apabila waktu *timeout*nya habis.

Dalam AODV, setiap *node* bertanggung jawab untuk memelihara informasi rute yang telah disimpan di dalam *routing table*-nya. Pada saat pengiriman data, apabila terjadi perubahan topologi yang mengakibatkan suatu *node* tidak dapat dituju dengan menggunakan informasi rute yang ada di *routing table*, maka suatu *node* akan

mengirimkan *route error packet* (RRER) ke *neighbour node* dan *neighbour node* akan mengirim kembali RRER demikian seterusnya hingga menuju *source node*. Setiap *node* yang memperoleh RRER ini akan menghapus informasi yang mengalami *error* di dalam *routing table*-nya. Kemudian *source node* akan melakukan *route discovery process* kembali apabila rute tersebut masih diperlukan. DSR merupakan protokol dimana *node* sumber menentukan rute paket yang dikirim setelah mengetahui serangkaian rute yang lengkap. Proses *routing* pada protokol ini terdiri atas 2 mekanisme, yaitu *Route Discovery* dan *Route Maintenance*. *Route discovery* yaitu kondisi dimana *node* ingin mengirimkan paket data ke tujuan yang belum diketahui rutenya, sehingga sumber mengirim *route request* (RREQ). RREQ akan melakukan proses *flooding*, yaitu proses pengiriman data atau *control message* ke setiap *node* pada jaringan untuk mencari rute tujuan. RREQ akan tersebar ke seluruh *node* dalam jaringan. Tiap *node* akan mengirim paket RREQ ke *node* lain kecuali *node* tujuan. Kemudian *node-node* yang menerima RREQ akan mengirim paket *route reply* (RREP) ke *node* yang mengirim RREQ tersebut. Setelah rute ditemukan, *node* sumber mulai mengirim paket data.

2.5 Parameter *QoS*

Parameter-parameter yang digunakan untuk menghitung *Quality of Service* (*QoS*) adalah sebagai berikut :

1. *Delay*

   Delay adalah waktu yang dibutuhkan oleh sebuah paket data, terhitung dari saat pengiriman oleh *transmitter* sampai saat diterima oleh *receiver*. Berikut adalah formula untuk menghitung besarnya *delay* : (Rifiani, Vina dan Hadi, M. Zen S., dkk. 2009)

Waktu tunda (t) = (T_r – T_s) detik

$0 \leq t \leq T$

dimana :

T_r = Waktu penerimaan paket (detik)

T_s = Waktu pengiriman paket (detik)

T = Waktu simulasi (detik)

t = Waktu pengambilan sampel (detik)

2. *Packet Loss Ratio* (PLR)

Packet loss ratio adalah prosentase banyaknya paket yang hilang selama proses transmisi ke tujuan. Paket hilang terjadi ketika satu atau lebih paket data yang melewati suatu jaringan gagal mencapai tujuannya. Berikut adalah formula untuk menghitung besarnya PLR : (Rifiani, Vina dan Hadi, M. Zen S., dkk. 2009)

PLR = $\left(\frac{Pd}{Ps}\right)$ x 100%

$0 \leq t \leq T$

dimana :

Pd = Paket yang mengalami *drop* (paket)

Ps = Paket yang dikirim (paket)

T = Waktu simulasi (detik)

t = Waktu pengambilan sampel (detik)

3. *Utilisasi Bandwidth*

Utilisasi Bandwidth merupakan parameter yang menunjukkan prosentase suatu sumber daya yang digunakan. Dalam hal ini sumber daya yang dimaksud

adalah *bandwidth* suatu *link* yang menghubungkan antara kedua sisi yaitu sisi pelanggan dan *provider*. *Utilisasi bandwidth* suatu *link* menunjukan rasio antara ukuran *bandwidth* total terpakai oleh pelanggan dengan *bandwidth* yang tersedia, untuk mendapatkan nilai *utilisasi bandwidth* dilakukan dengan menggunaka 2 formula. Berikut adalah formula-formula untuk menghitung besarnya *Utilisasi Bandwidth* : (Jusak, 2011)

$$\textit{Bandwidth} \text{ terpakai} = \frac{Length}{(T - t_0)}$$

Dimana : t_0 = 0 jika waktu awal adalah 0 (nol)

Dengan :

Length = jumlah total paket yang dikirim dan diterima pada saat proses pengiriman data berlangsung (byte)

T = Waktu akhir penerimaan data (detik)

t_0 = Waktu awal pengiriman data (detik)

Hasil dari rumus tersebut (*Bandwidth* terpakai) digunakan untuk menghitung *utilisasi bandwidth* dengan rumus berikut : (Jusak, 2011)

$$\textit{Utilisasi Bandwidth} = \frac{\sum Bandwidth\ terpakai * 8\ bit}{Bandwidth\ sistem}$$

Dengan :

$\sum$ *Bandwidth* terpakai = total *bandwidth* yang terpakai

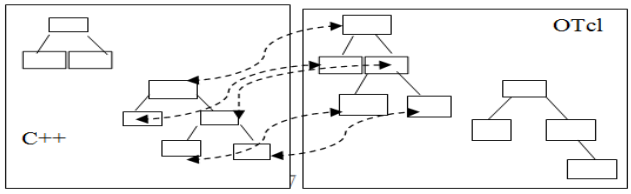
Bandwidth sistem = besarnya *bandwidth* yang disediakan oleh sistem yang digunakan.

2.6 NS-2 (*Network Simulator 2*)

2.6.1 Konsep NS-2

Network Simulator dibangun dengan menggunakan 2 bahasa pemrograman, yaitu C++ dan Tcl/OTcl. C++ digunakan untuk *library* yang berisi *event scheduler*, protokol dan *network component* yang diimplementasikan pada Network Simulator 2.34 yang digunakan oleh penulis. Tcl/OTcl digunakan pada *script* yang ditulis oleh *user* dan pada *library* sebagai simulator objek. OTcl juga nantinya akan berperan sebagai interpreter. Hubungan antar bahasa pemrograman dapat dideskripsikan seperti Gambar 2.5. (Wirawan, Andi B. dan Indarto, Eka : 2004)

Bahasa C++ digunakan pada library karena C++ mampu mendukung runtime simulasi yang cepat, meskipun simulasi melibatkan simulasi jumlah paket dan sumber data dalam jumlah besar.

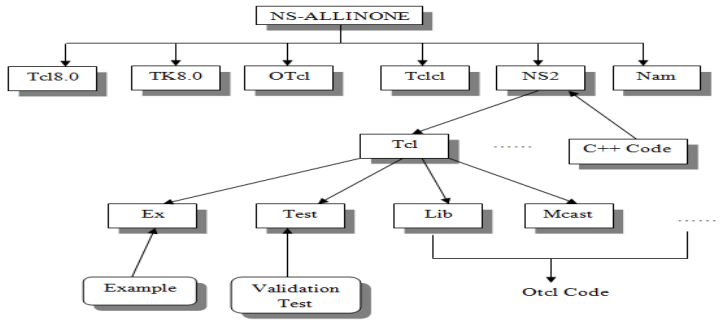


Gambar 2.5 Hubungan Tcl/OTcl dengan C++

Bahasa Tcl memberikan respon *runtime* yang lebih lambat daripada C++, tetapi jika terdapat kesalahan, respon Tcl terhadap kesalahan *syntax* dan perubahan *script* berlangsung dengan cepat dan interaktif. *User* dapat mengetahui letak kesalahannya yang dijelaskan pada *console*, sehingga *user* dapat memperbaiki dengan cepat. Oleh karena itu, bahasa ini dipilih untuk digunakan pada skripsi simulasi.

2.6.2 Komponen Pembangunan NS-2

Instaler NS versi NS-allinone berisi keseluruhan komponen wajib dan optional yang dibutuhkan oleh simulasi. Masing-masing komponen tersebut terdapat pada folder NS-allinone yang diinstal. Gambar 2.6 adalah gambar komponen-komponen pembangun NS-2.



Gambar 2.6 Komponen Pembangun NS-2

Keterangan:

1. TCL: Tool Command Language
2. Otcl: Object TCL
3. TK: Tool Kit
4. Tclcl: Tcl/C++ Interface
5. NS2: NS versi 2
6. Nam: Network Animator

2.6.3 Cara Membuat dan Menjalankan Skrip NS-2

Membuat skrip simulasi NS sangat mudah. Skrip simulasi bisa dibuat dengan menggunakan program teks *editor* yang ada pada linux, dan disimpan dalam sebuah folder dengan ekstensi .tcl.

Contoh:

Simulasi1.tcl

Berikut ini akan dijelaskan dasar-dasar bahasa Tcl yang berguna dalam membangun simulasi :

***Syntax* Dasar**

Syntax dasar perintah tcl yaitu :

```
Command Arg1 arg2 arg3 …
```

Command tersebut bisa berupa nama dari *built in command*, atau sebuah prosedur Tcl. Contoh :

```
Expr 2*3
Puts *ini adalah contoh command*
```

Variable* dan *Array

Untuk membuat *variable*, digunakan perintah `set`. Adapun contoh penggunaannya adalah sebagai berikut :

```
Set x *ini adalah contoh variable*
Set y 20
```

Pemanggilan *variabel* dilakukan dengan menggunakan tanda `$` seperti contoh di bawah ini :

```
Puts “ $x, semuanya berjumlah $y ”
```

Sedangkan pembuatan *array* ditandai dengan menggunakan tanda kurung setelah nama *array* tersebut yang dapat dituliskan sebagai berikut:

```
Set opts (bottleneckinkrate) 1.2 Mb
Set opts (ENC) *on*
Set n(0) [$ns node]
Set n(1) [$ns node]
```

Untuk menjalankan simulasi yang telah dibuat adalah dengan cara sebagai berikut :

[root @ accessnet your_folder]# ns Simulasi1.tcl

2.6.4 Tahap-tahap Membangun Simulasi NS-2

Pembangunan simulasi NS-2 dilakukan secara bertahap. Berikut ini merupakan contoh tahap-tahap dasar pembuatan simulasi NS-2 :

Langkah 1: Mendefinisikan *Variable* Global

Menurut Wirawan & Indarto (2004), dalam membangun simulasi JSN apabila membutuhkan *variable-variable* global yang akan digunakan oleh keseluruhan program, maka *variable-variable* harus didefinisikan terlebih dahulu dengan menggunakan perintah `set<spasi>nama`*`variabel`*`(identitas`*`variabel`*`) <spasi>`*`value`*. Contoh pendefinisian tersebut adalah sebagai berikut :

```
set val(chan) Channel/WirelessChannel;
set val(prop) Propagation/TwoRayGround;
set val(netif) Phy/WirelessPhy;
set val(mac) Mac/802_11;
set val(ifq) Queue/DropTail/PriQueue;
set val(ll) LL;
set val(ant) Antenna/OmniAntenna;
set val(ifqlen) 50;
set val(nn) 5;
set val(rp) AODV;
set val(seed) 0;
set val(x) 300;
set val(y) 300;
set val(stop) 200;
set val(mobility) Static;
```

Variable-variable yang didefinisikan di atas memiliki fungsi yang berbeda-beda sesuai dengan kebutuhan pembuatan simulasi. Keterangan dari *variable-variable* di atas adalah sebagai berikut :

1. Chan

   Merupakan tipe *channel* yang digunakan dalam simulasi, seperti *channel wireless*.

2. Prop

   Merupakan model propagasi. Model propagasi bisa bernilai *OneWayGround* atau *TwoWayGround*.

3. Netif

   Merupakan tipe jaringan *wireless* yang digunakan.

4. Mac

   Merupakan tipe MAC yang digunakan sesuai dengan *channel* yang digunakan.

5. Ifq

   Merupakan tipe antarmuka antriannya, yang menunjukkan perlakuan *node* terhadap paket apabila memori yang digunakan telah penuh.

6. Ll

   Merupakan T\tipe *link layer*.

7. Ant

   Merupakan model antena *node* yang digunakan.

8. Ifqlen

   Merupakan ukuran maksimum antrian paket.

9. Nn

   Merupakan jumlah *node* yang digunakan.

10. Rp

    Merupakan protokol *routing* yang digunakan.

11. Seed

    Merupakan nilai s*eed* yang digunakan sebagai nilai awal dari penggunaan nilai *random*.

12. X

    Merupakan nilai topografi x.

13. Y

    Merupakan Nilai topografi y.

14. Stop

    Merupakan nilai waktu dimana simulasi akan dihentikan.

15. Mobility

    Merupakan mobilitas dari *node* apakah pergerakan *node* bersifat statis atau dinamis.

Langkah 2: Inisialisasi Simulasi

Menurut Wirawan & Indarto (2004), untuk memulai pembuatan simulasi sederhana, dapat menggunakan salah satu teks editor yang ada pada linux yang digunakan. Kemudian *file* tersebut disimpan dalam sebuah folder.

Simulasi NS dimulai dengan menuliskan skrip Tcl seperti di bawah ini :

```
#memanggil simulator object
Set ns [new Simulator]
#open file handle untuk simulator nam  trace data
Set nf [open out.nam w]
$ns namtrace-all $nf
#prosedur finish berguna untuk menyelesaikan simulasi
Proc finish {} {
#menutup file dan memulai nam (network animator)
Global ns nf
$ns flush-trace
Close $ns
Exec nam out.nam &
Exit 0
}
#mengeksekusi prosedur finish pada saat detik ke 5.0
$ns at 5.0 “finish”
#menjalankan simulasi
$ns run
```

Dimana baris yang diawali dengan tanda # dianggap sebagai komentar yang digunakan untuk menjelaskan masing-masing perintah.

Langkah 3 : Pembuatan Topologi

Topologi dibangun oleh *node* dan *link* yang dijelaskan sebagai berikut :

Node

Merupakan sebuah objek *node* pada NS-2 didefiniskan dengan command `$ns node`. Sebagai contoh pembuatan node pada NS-2 adalah sebagai berikut :

```
Set node [$ns node]
```

Link

Ada dua jenis link yang bisa digunakan pada NS-2, yaitu *simplex link* dan *duplex link.* Berikut ini adalah perintah pembuatan link beserta parameternya :

1. Untuk simplex link :

   ```
   $ns simplex-link <node1><node2><bw><delay><qtype>
   ```

   Link satu arah dari <node1> ke <node2>.

2. Untuk duplex link :

   ```
   $ns simplex-link <node1><node2><bw><delay><qtype>
   ```

   Link dua arah dari <node1> ke <node2> dan sebaliknya.

Langkah 4 : Membuat aliran data

Proses pengiriman data pada NS-2 dilakukan dengan membuat *transport agent* dan aplikasi pembawanya. *Transport agent* dibuat berpasangan, satu berfungsi sebagai sumber data dan pasangannya sebagai tujuannya.

Pada simulasi ini, kita menggunakan paket TCP dengan menggunakan aplikasi FTP. Pengiriman data tersebut diawali dengan membuat *agent* pengirim data.

```
set tcp [new Agent/TCP/Newreno]
$tcp set class_ 2
set sink [new Agent/TCPSink]
$ns attach-agent $node_(1) $tcp
$ns attach-agent $node_(4) $sink
$ns connect $tcp $sink
```

```
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 10.0 "$ftp start"
```

#membuat objek simulator yang berupa TCP Agent

```
set tcp [new Agent/TCP]
```

#attach-agent berfungsi untuk mengambil object agent yang sudah didefinisikan. *Attach-agent* tersebut dibagi menjadi 2 bagian, yaitu bagian pengirim dan bagian penerima.

#node_(1) sebagai *node* sumber

```
$ns attach-agent $node_(1) $tcp
```

#node_(4) sebagai *node* tujuan

```
$ns attach-agent $node_(4) $sink
```

Untuk dapat mengalirkan data di antara 2 *node* tersebut, maka kedua *node* tersebut harus dihubungkan dengan perintah sebagai berikut :

```
$ns connect $tcp $sink
```

Kemudian dibuat aplikasi yang berjalan di atas *transport agent* tersebut. Pada contoh ini, kita gunakan generator trafik dengan fungsi FTP dengan perintah sebagai berikut :

```
set ftp [new Application/FTP]
$ftp attach-agent $tcp
```

Setelah itu dilakukan pengaturan waktu dimana data TCP akan dialirkan, dengan perintah sebagai berikut :

```
$ns at 10.0 "$ftp start"
```

Artinya bahwa data akan dikirim setelah detik ke 10.0.

Menurut Wirawan & Indarto (2004), ada 2 jenis TCP *Agent* yang didukung oleh NS-2, yaitu *One Way* TCP *Agent* dan *Two Way* TCP *Agent*. Dimana *One Way* TCP

Agent tidak menggunakan komunikasi 2 arah antara *node* sumber dan *node* tujuannya, sedangkan *Two Way* TCP *Agent* mengadakan kesepakatan dua arah dalam proses pengiriman datanya.

2.7 *Practical Extraction and Report language* (Perl)

Menurut Jusak (2011), Perl merupakan bahasa pemrograman yang banyak digunakan untuk pemrosesan data *file* ASCII pada system UNIX. Bahasa pemrograman ini dibuat oleh Larry Wall dengan tujuan untuk memudahkan tugas-tugas administrasi sistem UNIX. Saat ini, Perl telah berevolusi menjadi bahasa pemrograman dan merupakan salah satu sarana yang dapat digunakan untuk pemrograman web. Di bawah ini merupakan contoh sederhana prosedur queue.pl untuk mem*filter* data :

```
$infile=$ARGV[0];
open (DATA,"<$infile") || die "Can't open $infile $!";
while (<DATA>)
{
    @x = split(' ');
    if ($x[6] eq 'tcp')
{
print STDOUT "$x[0]           $x[1]                $x[2]
}
}
close DATA;
exit(0);
```

Prosedur queue.pl di atas dijalankan dengan perintah sebagai berikut :

```
Perl queue.pl simple.tr > queue
```

Yang berarti bahwa prosedur queue.pl digunakan untuk memfilter data dari file simple.tr yang kemudian hasilnya akan disimpan pada *file* baru bernama queue.