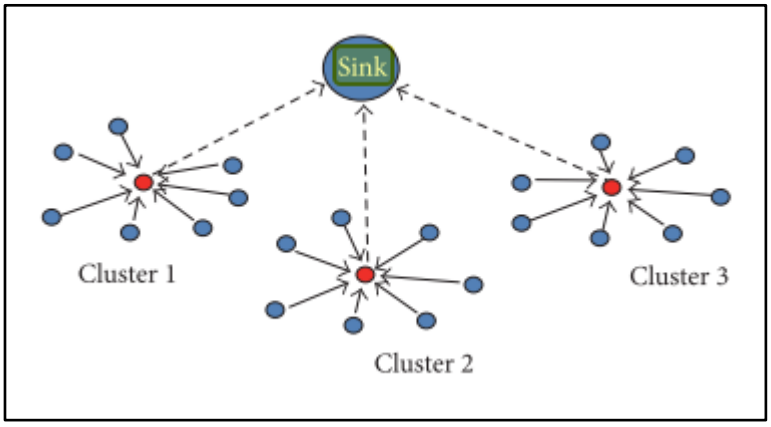
BAB 2
LANDASAN TEORI

2.1 Protokol Pada Jaringan Sensor Nirkabel

Seperti alat jaringan pada umumnya, JSN memiliki protokol tersendiri untuk saling berkomunikasi dan menentukan *routing* satu sama lain. Salah satu protokol yang menjadi inti pada penelitian ini dan merupakan yang paling banyak diadopsi oleh JSN adalah protokol *Low-Energy Aware Cluster Hierarchy* (LEACH). Seperti namanya, LEACH merupakan protokol *routing* yang dapat memperpanjang masa hidup jaringan tersebut, selain itu LEACH memiliki sifat *clustering*, *adaptive*, dan, *self-organizing* (Almomani dkk., 2016).

Pada protokol LEACH, JSN dibagi menjadi tiga buah hierarki simpul yang saling terkoneksi yaitu *sink node* atau *Base Station* (BS), *Cluster Head* (CH), dan *member node* seperti yang dapat dilihat pada Gambar 2.1. Data yang didapatkan oleh setiap sensor *member node* akan dikumpulkan oleh CH masing-masing klaster, kemudian akan diteruskan kepada BS untuk diproses lebih lanjut (Almomani dkk., 2016).



Gambar 2.1 Struktur Simpul Pada Protokol LEACH (Almomani dkk., 2016)

Dengan menggunakan hierarki seperti ini, protokol LEACH dapat meminimalisir penggunaan energi dan jumlah pesan yang dikirimkan karena masing-masing sensor pada klaster hanya akan mengirimkan informasinya kepada CH masing-masing dan bukan secara langsung kepada BS (Arora dkk., 2016).

Pada protokol LEACH terdapat 2 proses operasi yaitu *setup-phase* dan *state-phase*. Pada *setup-phase*, klaster-klaster dan CH dibentuk. Klaster-klaster dibentuk berdasarkan kekuatan sinyal tiap sensor sedangkan CH dibentuk secara efisien agar setiap sensor mendapat giliran untuk menjadi CH, sehingga konsumsi energi setiap sensor dapat secara rata disebar. CH yang dipilih hanya akan mempertahankan perannya untuk satu ronde. Untuk menentukan giliran sensor yang menjadi CH, sensor ke ‘n’ akan menghasilkan angka acak antara 0 dan 1 dan membandingkannya dengan *threshold* T(n). Sebuah sensor akan menjadi CH jika angka yang dihasilkan lebih kecil dari nilai T(n). *Threshold* ini memastikan hanya sebagian pecahan dari sensor, p, akan menjadi CH, dan sensor yang telah menjadi

CH pada ronde 1/p terakhir tidak akan dipilih kembali menjadi CH. Persamaan *Threshold* dapat dinotasikan pada Persamaan (2.1) (Arora dkk., 2016).

$$T(n) = \begin{cases} \frac{p}{1-p} * \left(r \bmod \frac{1}{p}\right) & if\ n \in G \\ 0 & otherwise \end{cases} \quad (2.1)$$

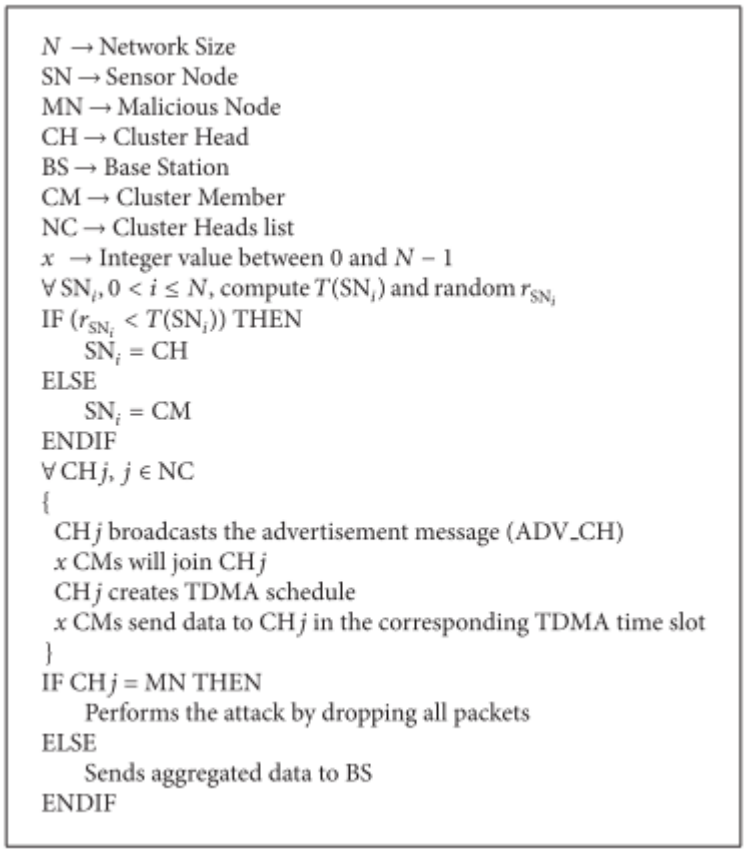
Nilai r adalah ronde yang sedang berjalan, G adalah sensor yang belum menjadi CH pada ronde 1/p terakhir.

State-phase adalah pada saat tiap sensor bekerja secara stabil dan mengirimkan informasi yang didapat kepada masing-masing CH pada rentang waktu yang telah ditentukan (Arora dkk., 2016).

2.2 Denial-of-Service

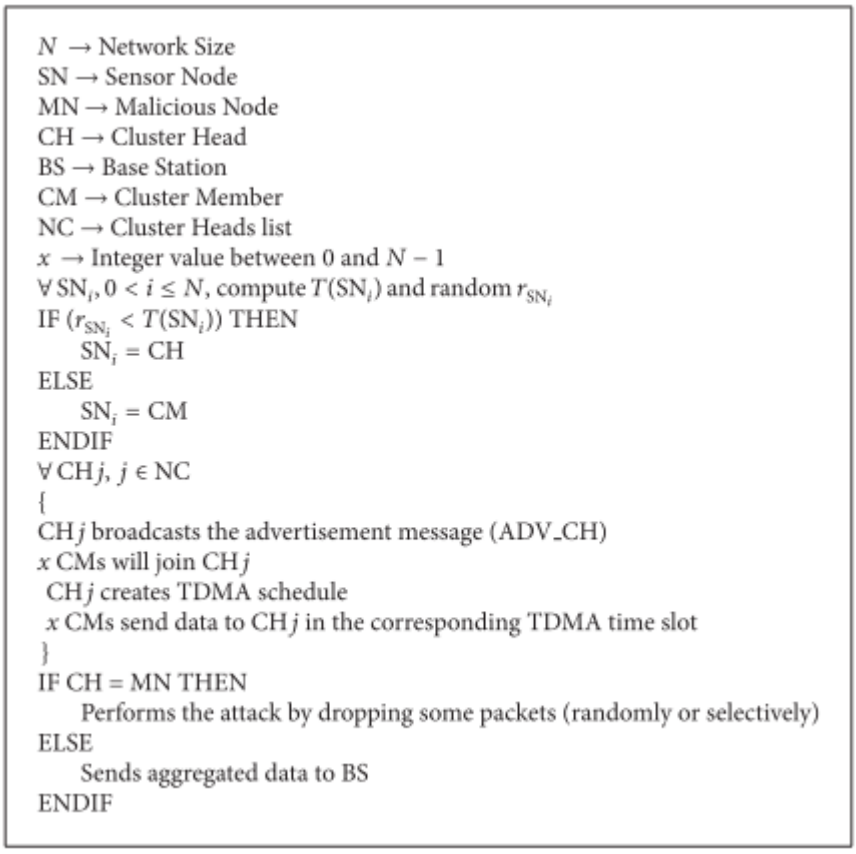
Protokol LEACH merupakan protokol *routing* yang berarti protokol ini berjalan pada lapisan jaringan JSN. Pada protokol ini, terdapat empat tipe serangan *DoS* yang dapat dilakukan, yaitu *Blackhole Attack, Grayhole Attack, Flooding Attack,* dan *Scheduling* atau TDMA *Attack* (Le dkk., 2018).

Blackhole Attack merupakan serangan yang terjadi ketika seorang penyerang mendeklarasikan dirinya sebagai CH pada awal ronde ketika pemilihan CH berlangsung. Sensor-sensor *member node* pada suatu klaster tersebut akan mengirimkan informasi yang didapatkan kepada CH penyerang, sebelum seharusnya diteruskan kepada BS, sehingga penyerang dapat menjatuhkan setiap paket informasi yang diterimanya dan tidak meneruskannya kepada BS (Le dkk., 2018). Model *pseudocode* dari *Blackhole Attack* dapat dilihat pada Gambar 2.2.



Gambar 2.2 Model dari *Blackhole Attack* (Almomani dkk., 2016)

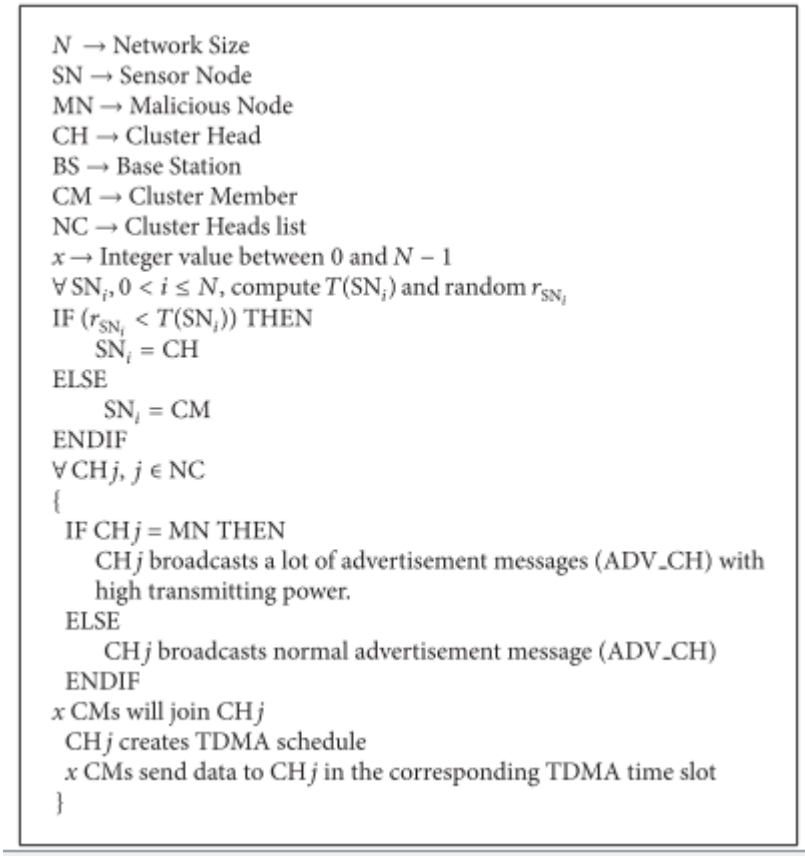
Grayhole Attack merupakan serangan yang serupa dengan *Blackhole Attack*, namun pada serangan ini penyerang mendeklarasikan dirinya sebagai CH kepada sensor-sensor *member node* yang lain, sehingga ketika CH penyerang tersebut menerima paket informasi, paket tersebut akan dijatuhkan dan tidak diteruskan kepada BS (Le dkk., 2018). Model *pseudocode* dari *Grayhole Attack* dapat dilihat pada Gambar 2.3.



N → Network Size
SN → Sensor Node
MN → Malicious Node
CH → Cluster Head
BS → Base Station
CM → Cluster Member
NC → Cluster Heads list
x → Integer value between 0 and $N - 1$
$\forall \mathrm{SN}_i, 0 < i \leq N$, compute $T(\mathrm{SN}_i)$ and random r_{SN_i}
IF ($r_{\mathrm{SN}_i} < T(\mathrm{SN}_i)$) THEN
    SN_i = CH
ELSE
    SN_i = CM
ENDIF
$\forall$ CHj, $j \in$ NC
{
CHj broadcasts the advertisement message (ADV_CH)
x CMs will join CHj
CHj creates TDMA schedule
x CMs send data to CHj in the corresponding TDMA time slot
}
IF CH = MN THEN
    Performs the attack by dropping some packets (randomly or selectively)
ELSE
    Sends aggregated data to BS
ENDIF

Gambar 2.3 Model dari *Grayhole Attack* (Almomani dkk., 2016)

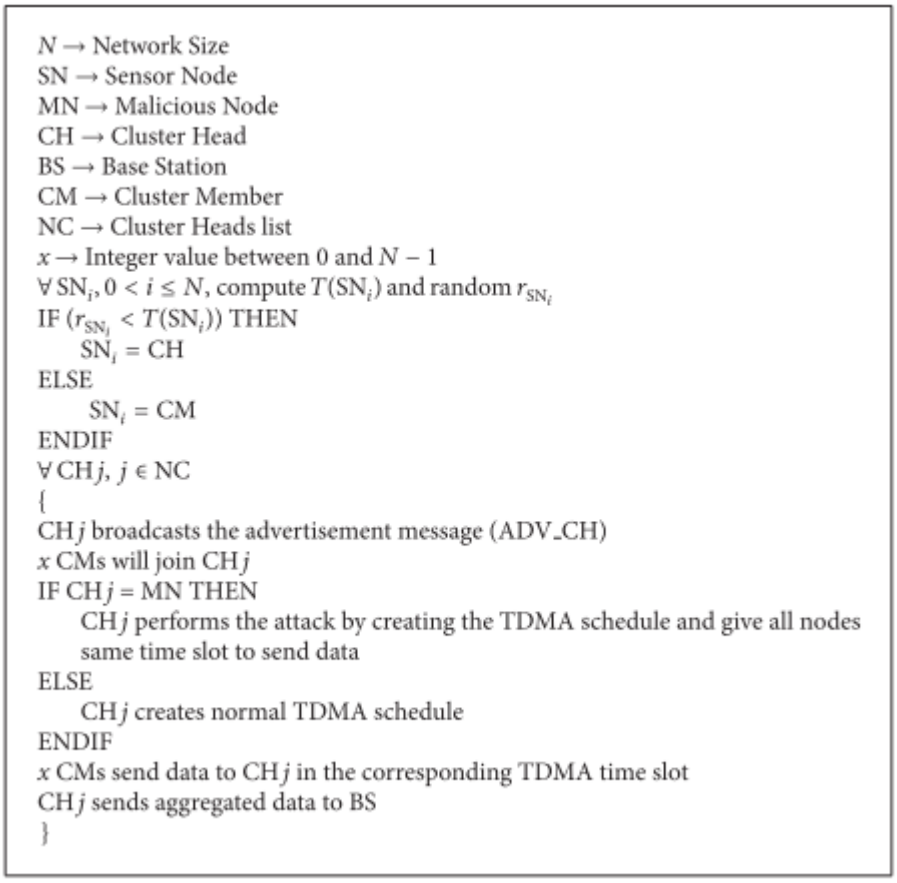
Flooding Attack merupakan serangan yang terjadi ketika penyerang mengirimkan paket *advertising CH* (ADV CH) dalam jumlah yang sangat besar kepada banyak sensor-sensor *member node*, sehingga ketika sensor tersebut menerima paket ini, energi dan waktu yang habis digunakan untuk menerima paket akan lebih besar daripada yang digunakan untuk menentukan CH yang akan dipilih. Selain itu, penyerang juga akan membuat sensor-sensor *member node* untuk memilih simpul yang salah untuk menjadi CH (simpul yang letaknya berjauhan), sehingga juga akan menguras energi sensor-sensor tersebut (Le dkk., 2018). Model *pseudocode* dari *Flooding Attack* dapat dilihat pada Gambar 2.4.



$N \rightarrow$ Network Size
SN $\rightarrow$ Sensor Node
MN $\rightarrow$ Malicious Node
CH $\rightarrow$ Cluster Head
BS $\rightarrow$ Base Station
CM $\rightarrow$ Cluster Member
NC $\rightarrow$ Cluster Heads list
$x \rightarrow$ Integer value between 0 and $N - 1$
$\forall \mathrm{SN}_i, 0 < i \leq N$, compute $T(\mathrm{SN}_i)$ and random r_{SN_i}
IF ($r_{\mathrm{SN}_i} < T(\mathrm{SN}_i)$) THEN
    SN_i = CH
ELSE
    SN_i = CM
ENDIF
$\forall$ CHj, $j \in$ NC
{
 IF CHj = MN THEN
    CHj broadcasts a lot of advertisement messages (ADV_CH) with high transmitting power.
 ELSE
    CHj broadcasts normal advertisement message (ADV_CH)
 ENDIF
x CMs will join CHj
 CHj creates TDMA schedule
 x CMs send data to CHj in the corresponding TDMA time slot
}

Gambar 2.4 Model dari *Flooding Attack* (Almomani dkk., 2016)

Scheduling atau TDMA *Attack* terjadi ketika CH penyerang mengirimkan perintah *Time-Division Multiple Access* (TDMA) yang salah sehingga waktu untuk mengirim paket setiap sensor-sensor *member node* akan dilakukan pada waktu yang sama. Hal ini akan mengakibatkan tabrakan paket sehingga paket tersebut gagal untuk dikirimkan (Le dkk., 2018). Model *pseudocode* dari *Scheduling Attack* dapat dilihat pada Gambar 2.5.



Gambar 2.5 Model dari *Scheduling Attack* (Almomani dkk., 2016)

2.3 SMOTE

Metode SMOTE adalah sebuah metode berbasis matematika yang dapat dilakukan untuk menyeimbangkan data yang tidak seimbang. Metode SMOTE tidak hanya akan melakukan duplikasi sederhana data yang berada pada kelas minoritas, tetapi menggunakan teknik interpolasi untuk menghasilkan sebuah data sintetis baru yang lebih optimal. Interpolasi yang dilakukan adalah antara beberapa data yang berasal dari kelas minoritas dan merupakan cara yang memanfaatkan hubungan antar fitur-fitur pada dataset bukan hanya hubungan antar data, maka dari itu prosedur ini disebut sebagai prosedur *feature space* (Fernandez dkk., 2018).

Cara implementasi metode SMOTE adalah pertama-tama tentukan sebuah data sampel, X, yang diambil dari kelas minoritas. Setelah itu tentukanlah sejumlah N data tetangga terdekatnya (biasanya berjumlah lima). Data tetangga terdekat bisa didapatkan dengan menghitung jarak *euclidean* dari data sampel X tersebut. Data sampel X dan N merupakan data yang berkorelasi sehingga interpolasi acak dapat dilakukan untuk memperoleh sampel data hasil interpolasi (p_i). Notasi rumus untuk melakukan interpolasi dapat dilihat pada Persamaan (2.2) (Tan dkk., 2019).

$$p_i = X + rand(0,1) * (y_i - X), i = 1, 2, \dots, N \quad (2.2)$$

Nilai X merupakan data sampel yang diambil dari kelas minoritas, *rand*(0,1) merupakan angka acak antara interval 0 dan 1, dan y_i merupakan data sampel tetangga yang ke-i, dari data tetangga terdekat N (ditentukan secara acak atau bebas).

Sebagai penjelasan lebih lanjut, simulasi metode SMOTE dapat dilihat pada contoh berikut.

Diasumsikan terdapat *dataset* yang berdimensi dua, yang merupakan sebuah koordinat pada diagram Kartesius.

Data X dicontohkan sebagai data pada koordinat (8,4).

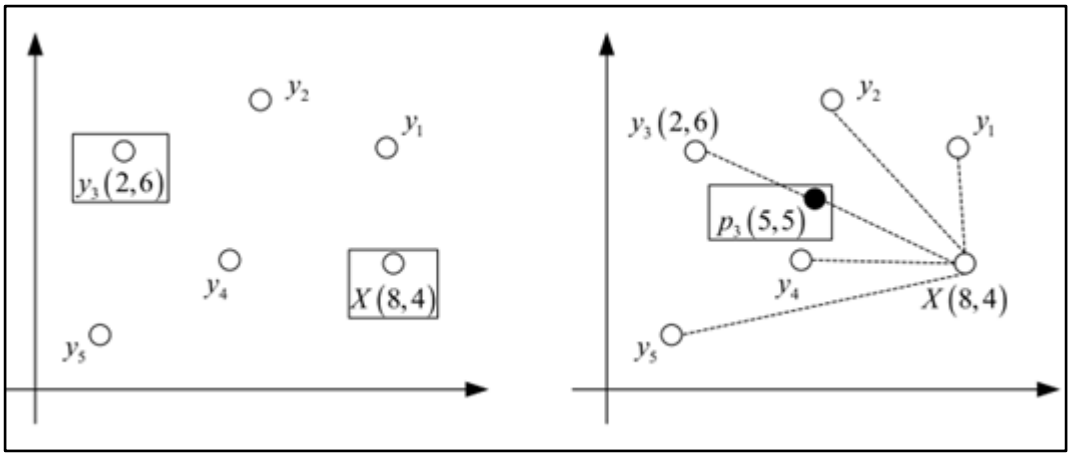
Nilai dari angka acak dicontohkan bernilai 0.5.

Data tetangga X yang dipilih (y_i) dicontohkan berkoordinat (2,6).

Berdasarkan nilai-nilai tersebut dan rumus pada Persamaan (2.2), didapatkan hasil seperti pada Persamaan (2.3).

$$p_3 = X + rand(0,1) * (y_3 - X) = (8,4) + 0.5 * \big((2,6) - (8,4)\big) = (5,5) \quad (2.3)$$

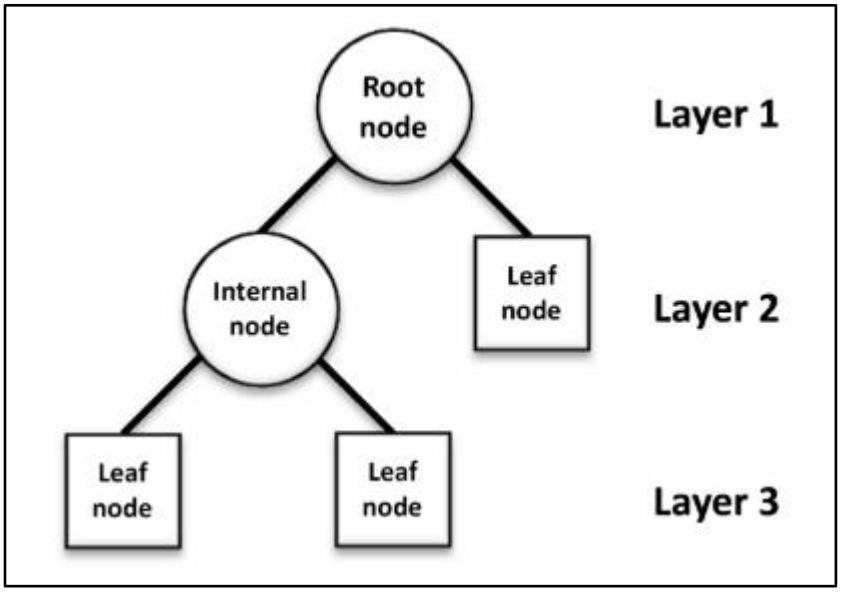
Berdasarkan perhitungan di atas didapatkan bahwa data sintetis baru yang dihasilkan adalah (5,5). Gambar 2.6 memperlihatkan visualisasi kasus di atas dalam bentuk koordinat.



Gambar 2.6 Representasi Contoh Kasus Metode SMOTE (Tan dkk., 2019)

2.4 Decision Tree

Decision tree adalah metode pembelajaran yang memanfaatkan konsep diagram berbentuk pohon terbalik. Struktur *decision tree* mulai dari paling atas yang disebut akar atau *root* dan akan tumbuh ke arah bawah hingga berakhir pada simpul daun atau *leaf*. Secara umum, visualisasi struktur *decision tree* dapat dilihat pada Gambar 2.7. Algoritma *decision tree* ini akan bergerak *top-down* dengan menentukan klasifikasi yang paling baik dari sebuah data yang masuk berdasarkan variabel-variabel penentu yang ada pada data (Guleria dkk., 2014).



Gambar 2.7 Visualisasi Struktur *Decision Tree* (Chiu dkk., 2016)

Untuk membentuk *decision tree* digunakan dua konsep utama, yaitu *Entropy* dan *Information Gain*. Konsep *Entropy* adalah mengukur tingkat kemurnian data. Tingkat kemurnian data ini dapat ditentukan berdasarkan tingkat homogenitas pada data yang ada. Semakin kecil tingkat homogenitas, nilai *Entropy* akan semakin besar, dan sebaliknya. Semakin besar nilai *Entropy*, maka artinya konten informasi yang ada pada suatu data atau atribut tersebut semakin besar (Guleria dkk., 2014). Notasi *Entropy* dapat dilihat pada Persamaan (2.4).

$$H(S) = \sum_{i=1}^{n} -P_i \log_2 P_i \tag{2.4}$$

Nilai S adalah sampel dari sejumlah n kali jumlah pelatihan dan nilai P_i adalah peluang terjadinya sebuah kejadian berdasarkan atribut atau fitur data yang ada.

Information Gain didapatkan berdasarkan nilai *Entropy*. *Information Gain* berarti seberapa besar sebuah atribut atau fitur pada data menentukan suatu

klasifikasi yang dibentuk. Atribut yang memiliki nilai *Information Gain* terbesar pada suatu simpul akan dipilih untuk menjadi atribut penguji pada simpul tersebut. Atribut yang memiliki *Information Gain* terbesar dari seluruh atribut yang ada, akan dijadikan *root* pada *decision tree* tersebut (Guleria dkk., 2014). Notasi *Information Gain* dapat dilihat pada Persamaan (2.5).

$$Gain(S, A_i) = H(S) - \sum_{v \in Values(A_i)} P(A_i = v) H\ (S_v) \qquad (2.5)$$

Nilai *S* adalah sampel data, nilai A_i merupakan atribut atau fitur dari data yang ada, nilai *H(S)* adalah *entropy* sampel data, nilai *P(Ai = v)* adalah peluang kejadian berdasarkan atribut atau fitur data yang ada.

2.5 Random Forest

Metode *Random Forest* dapat digunakan untuk berbagai hal seperti klasifikasi dan regresi. *Random Forest* bekerja dengan cara membentuk klaster-klaster *decision tree* sejumlah K, lalu menghasilkan sebuah kelas yang merupakan rata-rata prediksi yang dihasilkan oleh masing-masing *decision tree* (Noshad dkk., 2019). Notasi deskripsi K-*decision tree* dapat dilihat pada Persamaan (2.6).

$$\{h(X, \theta_k), k = 1,2, \dots, K\} \qquad (2.6)$$

Nilai *h* adalah fungsi *decision tree*, nilai *X* merupakan sampel data, nilai K adalah jumlah *decision tree* yang ada dan θ_k merupakan vektor acak yang secara identik terdistribusi (Tan dkk., 2019).

Algoritma *Random Forest* secara umum bekerja dengan langkah-langkah berikut (Tan dkk., 2019).

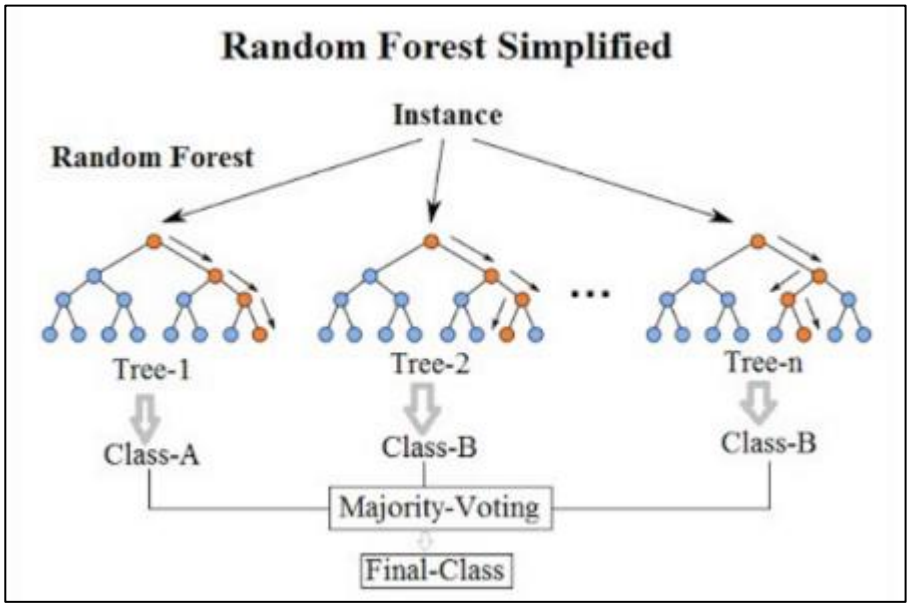
1. Berdasarkan sampel data dari data latih (*training set*) diimplementasikan metode *random repeated sampling*, kemudian sejumlah K *tree* untuk regresi klasifikasi dibentuk.

2. Sejumlah *m* fitur dari total *n* fitur dari *training set*, dimana $m \leq n$, secara acak dipilih. Dengan menghitung informasi yang terkandung pada setiap *m* fitur, fitur yang paling menentukan klasifikasi dipilih untuk dilakukan *node splitting*.

3. Setiap *tree* akan tumbuh hingga batas maksimal atau sampai batas yang telah ditentukan.

4. Dengan menggunakan semua *tree* yang telah dibentuk, sampel data baru dapat diklasifikasikan berdasarkan masing-masing *tree*. Hasil akhir dari klasifikasi ditentukan oleh jumlah dukungan (*vote*) yang diberikan oleh masing-masing *tree* tersebut.

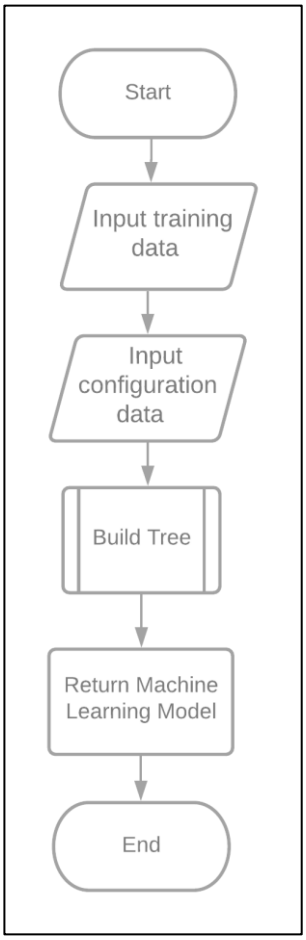
Gambar 2.8 memberikan visualisasi *tree* yang terbentuk dari langkah-langkah di atas.



Gambar 2.8 Visualisasi Algoritma Random Forest (Dang, 2019)

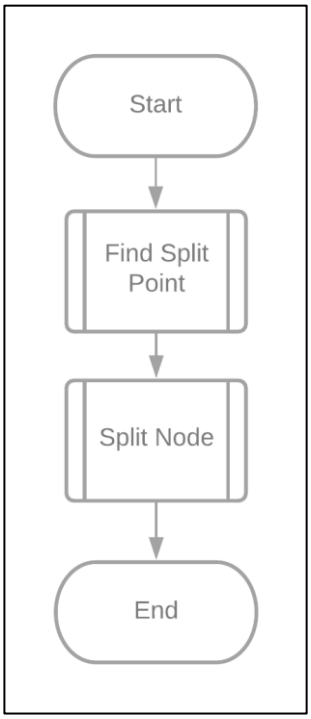
Cara kerja algoritma *Random Forest* untuk melakukan pelatihan data secara lebih lengkap dapat dijelaskan menggunakan diagram alur (*flowchart*) yang dapat dilihat pada Gambar 2.9 sampai dengan Gambar 2.12 (Breiman, 2001).

Gambar 2.9 merupakan *flowchart* utama dari algoritma Random Forest yang dapat digunakan untuk melatih model pembelajaran mesin menggunakan data latih yang telah ditentukan. Proses utama pada algoritma Random Forest adalah pembangunan *Decision Tree* yang menjadi penentu utama aturan-aturan yang digunakan untuk melakukan klasifikasi data. Sebelum melakukan pembuatan *Decision Tree*, dapat dilakukan konfigurasi parameter yang digunakan pada algoritma Random Forest. Setelah itu, proses pembuatan *Decision Tree* dapat dilakukan. Proses pembuatan *Decision Tree* pada akhirnya mengeluarkan model pembelajaran mesin yang dapat digunakan untuk klasifikasi.



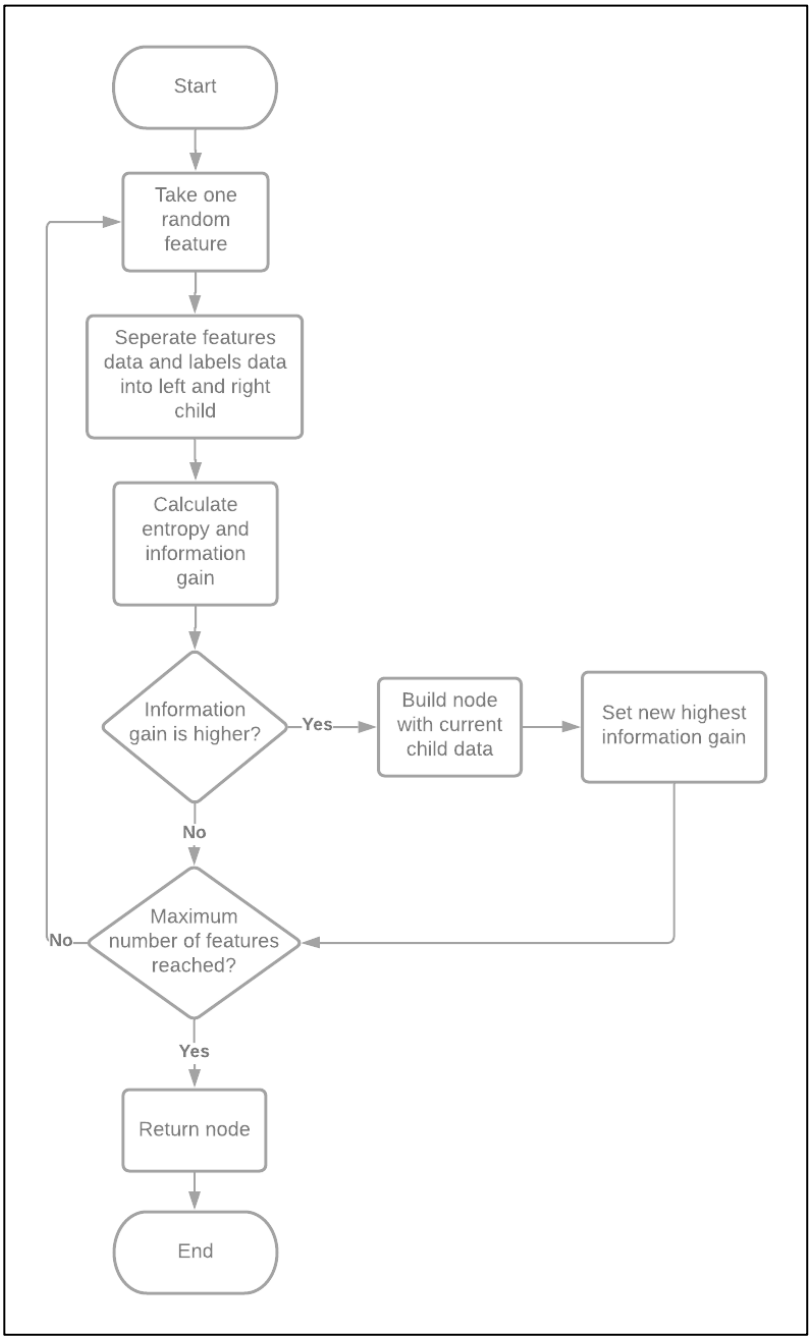
Gambar 2.9 *Flowchart* Random Forest

Gambar 2.10 merupakan *flowchart* pembentukan *Decision Tree* dari algoritma Random Forest. Proses pembentukan *Decision Tree* terdiri dari dua proses lainnya yaitu proses penentuan titik *best split* dan proses *split*. Penentuan titik *best split* berfungsi untuk menentukan posisi titik terbaik untuk melakukan *split* dengan mengukur nilai *information gain*, sedangkan proses *split* berfungsi untuk melakukan *split* simpul-simpul *Decision Tree* yang terbentuk dan akhirnya menentukan *terminal node* atau simpul yang menentukan klasifikasi data.



Gambar 2.10 *Flowchart* Build Tree

Gambar 2.11 merupakan *flowchart* penentuan titik *best split* yang dapat digunakan pada proses *split node* pada *Decision Tree* yang dibuat. Untuk memulai proses penentuan *best split*, sebuah sampel fitur atau salah satu atribut data acak dipilih dari data latih, kemudian dilakukan pemisahan data fitur dan data label menjadi *left child* dan *right child*. Setelah itu, dilakukan perhitungan *entropy* dan *information gain* untuk setiap nilai fitur yang dipilih pada data latih. Proses perhitungan *entropy* dan *information gain* dilakukan berulang kali hingga mencapai batas yang telah ditentukan. Penentuan titik *best split* dapat dilakukan dengan memilih simpul *Decision Tree* yang memiliki nilai *information gain* terbesar. Data simpul tersebut kemudian dikembalikan untuk dilanjutkan ke proses selanjutnya.

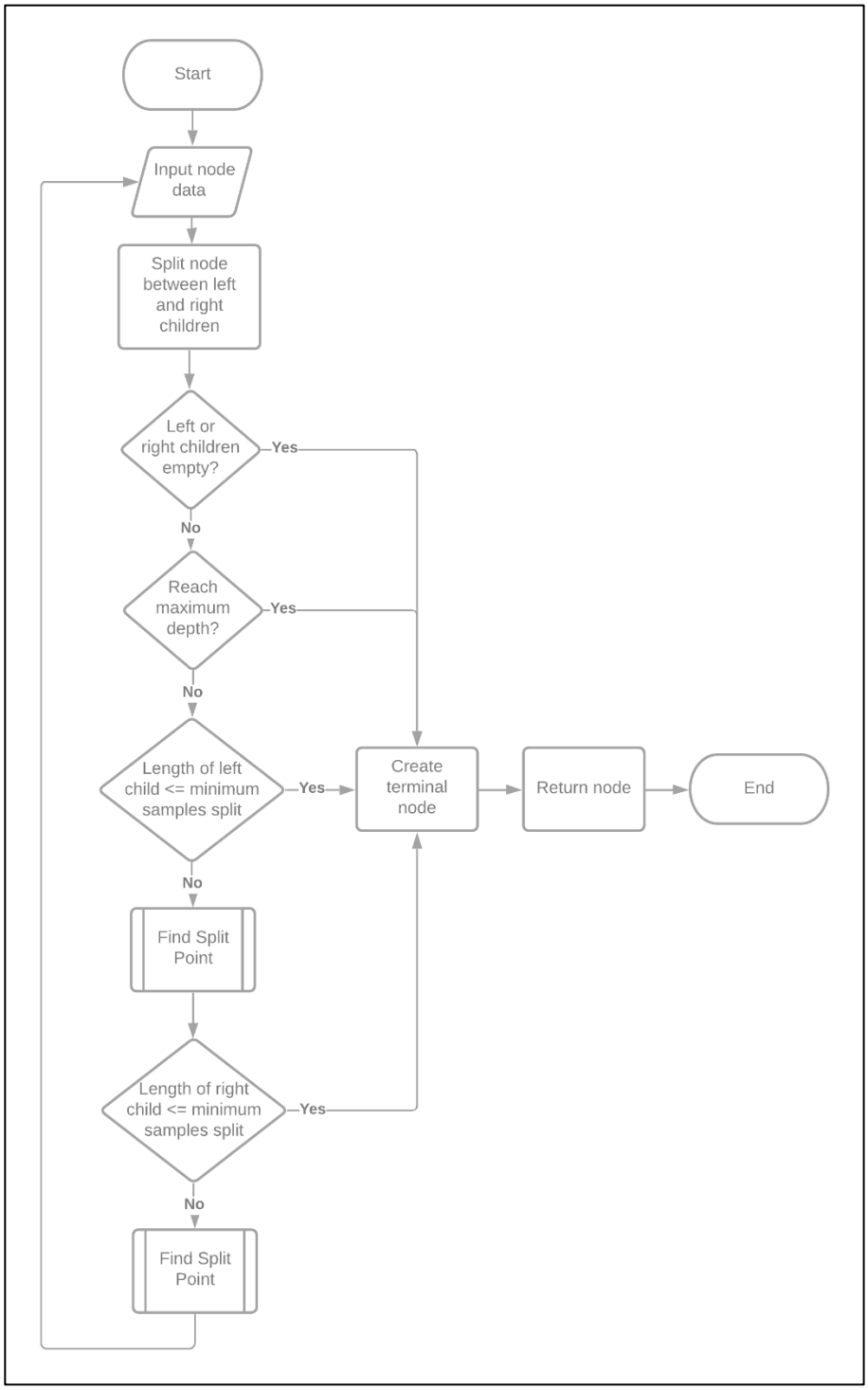


Gambar 2.11 *Flowchart* Find Split Point

Gambar 2.12 merupakan *flowchart split node* yang bertujuan untuk menentukan *terminal node* agar dapat melakukan klasifikasi. Proses *split node* dimulai dengan memisahkan *left child* dan *right child* yang telah ditentukan. Lalu dilakukan pengecekan nilai kepada kedua *child* tersebut. Jika salah satu data bernilai kosong, artinya *best split* pada simpul tersebut tidak dapat membedakan antara kedua kelas tersebut, maka langkah terbaik adalah membentuk *terminal node* berdasarkan kelas dengan jumlah terbanyak pada simpul.

Jika data tidak kosong, pengecekan kedalaman *tree* dilakukan. Jika kedalaman *tree* telah mencapai kedalaman maksimal yang telah ditentukan sebelumnya, maka *terminal node* dapat dibentuk. Jika kedalaman *tree* belum mencapai maksimal, maka pengecekan jumlah observasi pada *left child* dilakukan. Jika jumlah observasi pada *left child* kurang dari jumlah sampel minimal untuk *split*, maka *terminal node* dapat dibentuk. Jika jumlah observasi pada *left child* lebih banyak dari jumlah sampel minimal, maka *best split* perlu ditentukan kembali pada simpul *left child* tersebut, dan proses *split* diulang kembali. Hal yang sama juga dilakukan pada *right child*.

Proses ini kemudian diulang terus menerus hingga seluruh *branch* pada *decision tree* memiliki *terminal node*. Setelah seluruh proses selesai, *tree* telah selesai dibentuk dan dapat dikembalikan sebagai model pembelajaran mesin.



Gambar 2.12 *Flowchart* Split Node

Serupa dengan algoritma pembelajaran mesin lainnya, algoritma *Random Forest* merupakan algoritma yang sensitif terhadap nilai-nilai *hyperparameter* yang dimilikinya. Sehingga untuk mendapatkan hasil pelatihan pembelajaran mesin yang optimal, dapat dilakukan penyesuaian dan penyetelan terhadap sejumlah nilai-nilai *hyperparameter* tersebut. Jenis-jenis *hyperparameter* yang dimiliki oleh *Random Forest* meliputi jumlah *decision tree* yang ada, jumlah sampel minimal yang dibutuhkan untuk melakukan *split leaf node*, kedalaman *decision tree*, jumlah maksimal fitur yang dipertimbangkan dalam menentukan *split* (*mtry*), dan kriteria *split* yang digunakan. Setiap jenis *hyperparameter* tersebut dapat memiliki nilai yang berbeda-beda dan perlu disesuaikan untuk setiap kasus data yang digunakan (Daviran dkk., 2021; Probst dan Wright, 2019; Scornet, 2018). Untuk menentukan nilai *mtry* terdapat rumus tipikal yang biasa digunakan untuk membantu menentukan nilai optimal. Rumus tersebut dapat dilihat pada Persamaan (2.7) dan Persamaan (2.8). Persamaan (2.7) digunakan untuk menentukan nilai *mtry* jika ingin melakukan klasifikasi, sedangkan Persamaan (2.8) digunakan jika ingin melakukan regresi.

$$mtry = \lfloor \sqrt{(n(f))} \rfloor \tag{2.7}$$

$$mtry = \frac{n(f)}{3} \tag{2.8}$$

Pada Persamaan (2.7) dan Persamaan (2.8), *mtry* merupakan nilai jumlah maksimal fitur yang dipertimbangkan untuk *split*, sedangkan n(f) merupakan jumlah total fitur yang ada pada *dataset*.

2.6 Parameter Evaluasi Performa

Untuk mengevaluasi performa algoritma yang digunakan pada penelitian ini, digunakan enam buah parameter evaluasi performa, yaitu *True Positive Rate* (TPR), *True Negative Rate* (TNR), *False Positive Rate* (FPR), *False Negative Rate* (FNR), *accuracy*, dan *precision* (Almomani dkk., 2016).

TPR merepresentasikan seberapa besar data serangan yang benar diklasifikasikan sebagai serangan. Notasi TPR dapat dilihat pada Persamaan (2.9).

$$TPR = \frac{TP}{TP+FN} \tag{2.9}$$

TNR merepresentasikan seberapa besar data bukan serangan yang benar diklasifikasikan sebagai bukan serangan. Notasi TNR dapat dilihat pada Persamaan (2.10).

$$TNR = \frac{TN}{TN+ FP} \tag{2.10}$$

FPR merepresentasikan seberapa besar data bukan serangan yang salah diklasifikasikan sebagai serangan. Notasi FPR dapat dilihat pada Persamaan (2.11).

$$FPR = \frac{FP}{FP+ TN} \tag{2.11}$$

FNR merepresentasikan seberapa besar data serangan yang salah diklasifikasikan sebagai bukan serangan. Notasi FNR dapat dilihat pada Persamaan (2.12).

$$FNR = \frac{FN}{FN+TP} \tag{2.12}$$

Accuracy merepresentasikan seberapa besar klasifikasi yang tepat dilakukan. Notasi *Accuracy* pada permasalahan klasifikasi biner dapat dilihat pada Persamaan (2.13).

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{2.13}$$

Pada permasalahan klasifikasi banyak kelas, nilai *accuracy* keseluruhan dapat dicari dengan menggunakan notasi pada Persamaan (2.14) (Starovoitov dan Golub, 2020).

$$Accuracy = \frac{\sum_{i=1}^{N} TPi}{n} \tag{2.14}$$

Nilai *TPi* merupakan jumlah klasifikasi benar untuk kelas ke-i, sedangkan nilai *n* adalah jumlah seluruh data.

Precision merepresentasikan seberapa besar kasus positif yang diklasifikasikan secara benar. Notasi *Precision* dapat dilihat pada Persamaan (2.15).

$$Precision = \frac{TP}{TP+FP} \quad (2.15)$$

Pada notasi-notasi di atas, *TP* (*True Positive*) adalah jumlah sampel data serangan yang benar diklasifikasikan sebagai serangan. *TN* (*True Negative*) adalah jumlah sampel data yang bukan serangan yang benar diklasifikasikan sebagai bukan serangan. *FP* (*False Positive*) adalah jumlah sampel data yang bukan serangan, tetapi salah diklasifikasikan sebagai serangan. *FN* (*False Negative*) adalah jumlah sampel data serangan, tetapi salah diklasifikasikan sebagai bukan serangan (Almomani dkk., 2016).