PENGGUNAAN GRAPH EMBEDDING (GEM) UNTUK PENJALURAN DAN PENYIMPANAN DATA PADA JARINGAN *SENSOR* TANPA INFORMASI GEOGRAFIK

Ray Suryadiptya – NIM : 13505012

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if15012@students.if.itb.ac.id

Abstrak

Dalam makalah ini diperkenalkan GEM (*Graph Embedding*) untuk jaringan *sensor*, sebuah infrastruktur untuk penjaluran noktah ke noktah dan penyimpanan data serta pemrosesan informasi dalam jaringan sensor. Infrastruktur ini tidak tergantung pada informasi geografik dan akan berjalan dengan baik walaupun ada hambatan secara fisik. Dalam GEM, dibuat sebuah graf berlabel yang bisa ditanamkan pada topologi jaringan awal dengan efisien dan merata. Pada graf tersebut, tiap noktah diberi label yang menunjukkan posisinya dalam topologi jaringan awal tersebut, hal ini akan memungkinkan pesan disalurkan dengan efisien pada jaringan, sedangkan tiap noktah hanya perlu mengetahui label noktah tetangganya.

Untuk menunjukkan bagaimana GEM bisa diaplikasikan, telah dibuat sebuah metode penanaman graf yang disebut VPCS (*Virtual Polar Coordinate Space*) serta algoritma penjaluran efisien yang menggunakan VPCS, disebut VPCR. Pada VPCS, sebuah pohon dengan gelang ditanamkan ke topologi jaringan, dan memberi label pada noktah-noktah sehingga terbentuk koordinat polar *virtual*. VPCR merupakan algoritma pertama untuk noktah ke noktah yang menjamin penyampaian, yang hanya membutuhkan tiap noktah mengetahui label noktah tetangganya dan tidak membutuhkan informasi geografik. Hasil simulasi memperlihatkan bahwa VPCR efektif dalam jaringan dinamis, berjalan lancar walaupun ada hambatan, serta cukup ringkas dalam hal besar dan kepadatan jaringan.

Keyword : Sensor networks, routing, data-centric storage, Graph Embedding, VPCS, VPCR

1. Pendahuluan

Penyebarluasan jaringan sensor sudah di mana-mana. Jaringan yang terdiri atas ribuan sensor sudah dibuat sebagai solusi ekonomis untuk masalah-masalah yang dihadapi : pengawasan arus lalu-lintas, pengawasan keamanan (struktur, api, dan lain-lain), pencatatan gempa, pengawasan polusi, pengawasan hewan liar, dan lain-lain. Tapi, jaringan ini menemukan masalah untuk diriset. Di satu pihak, penyebaran *sensor* besar-besaran dapat menyediakan pengukuran ppada area yang luas, tapi di lain pihak, karena peralatannya mempunyai kapasitas komputasi, memori, penyimpanan, jarak komunikasi, dan baterai yang terbatas, peralatan dan teknik tradisional tidak bisa diaplikasikan terhadap jaringan *sensor* ini. Kurangnya alat dan teknik untuk membuat komunikasi antar noktah *sensor* dan pengambilan data dari jaringan *sensor* membatasi guna dari jaringan *sensor*, dan membuat penyebaran jaringan secara besar-besaran menjadi tidak berguna.

Salah satu masalah utama dalam jaringan sensor adalah bagaimana cara mendapatkan data. Solusinya dapat dibedakan menjadi 3 kategori : penyimpanan lokal, penyimpanan eksternal, dan pendekatan data-sentris. Pada penyimpanan lokal, sebuah *query* harus dijalankan dalam network, membuat noktah yang berisi data yang sesuai dengan query mengirimkan kembali datanya ke pusat. Pada penyimpanan eksternal, data dikirim ke pusat tanpa menunggu sebuah *query* dikirim. Walaupun penyimpanan eksternal menghemat dalam pemakaian *query*, sistem ini akan menghamburkan tenaga jika data yang dikirimkannya tidak dibutuhkan. Pada cara ketiga, pendekatan data-sentris[5], setiap kejadian dinamai, dan sensor bekerja secara lokal untuk mendeteksi sebuah kejadian, seperti adanya api dalam gedung. Saat sebuah noktah mendeteksi adanya sebuah kejadian, mendeteksi noktah mana yang berhubungan dengan kejadian dengan nama

itu, dan menyimpan datanya di noktah tersebut. Noktah yang berfungsi untuk penyimpanan data tertentu biasanya ditentukan dengan membuat daftar nama kejadian, dan memetakannya pada noktah dalam jaringan. Saat pengguna ingin mengambil data, dia hanya mengirimkan *query* pada noktah yang bertanggungjawab menyimpan data yang sesuai dengan *query*. Dengan sistem ini, *query* tidak harus dikirimkan ke semua noktah pada jaringan, dan data yang tidak diinginkan tidak akan dikirim ke pusat. *Query* juga dapat diproses sebagian pada noktah yang menyimpan data, sehingga bisa membuat hanya data agregasi yang dikirim ke pusat, bukannya semua data yang ada pada noktah.

Dalam makalah ini diajukan penggunaan *Graph Embedding* pada jaringan *sensor*, yang menyediakan infrastruktur efisien untuk pemrosesan informasi data-sentris dan penyimpanannya dengan menggabungkan mengatasi masalah dalam penjaluran dan pemetaan nama data ke noktah. Mengenai dasar-dasar graf[4] tidak dipelajari lebih lanjut di sini. Dalam GEM, noktah sensor diberi label sehingga : (1) Noktah dapat menyalurkan data yang ditujukan ke label tertentu dengan efisien, hanya perlu mengetahui label noktah tetangganya, dan (2) Pemetaan data ke label dapat dilakukan dengan efisien untuk melakukan penyimpanan secara data-sentris. Dengan menggunakan metode penanaman graf dan pelabelan pada noktah, GEM memungkinkan komunikasi antar noktah serta penyimpanan dan agregasi data-sentris secara efisien.

Untuk mendemostrasikan bagaimana GEM bisa diaplikasikan, dibuat VPCS (*Virtual Polar Coordinate Space*), sebuah teknik penanaman graf yang menanamkan sebuah koordinat polar *virtual* ke topologi jaringan. VPCS berjalan dengan benar walaupun dibuat tanpa memperhatikan kondisi fisik. Meskipun begitu telah ditemukan 2 teknik untuk menaikkan kinerja dengan menyesuaikan ruang *virtual* dengan topologi jaringan. Cara pertama membutuhkan noktah untuk menentukan terlebih dahulu jarak dengan tetangganya, sedangkan cara kedua tidak perlu. Sudah dibuat juga VPCR, yaitu algoritma untuk membuat jalur dalam koordinat polar *virtual*. Aplikasi data-sentris, seperti penjaluran secara data-sentris, penyimpanan data-sentris, dan agregasi data-sentris dapat dibuat dengan menggunakan VPCS dan membuat komunikasi antar noktah dengan menggunakan VPCR.

2. Kerangka GEM

Penanaman graf adalah sebuah teknik dalam teori graf di mana graf tamu G dipetakan ke dalam graf H. Teknik ini sudah diaplikasikan ke dalam banyak masalah[1]. Teknik ini digunakan terutama untuk memetakan sebuah jaringan ke jaringan lain, seperti untuk mensimulasikan sebuah topologi menggunakan topologi lain. Dalam GEM, metode penanaman graf digunakan untuk mengambil topologi yang paling cocok dan memetakannya ke dalam topolgi jaringan asli.

Penanaman graf didefinisikan sebagai berikut. Penanaman sebuah graf G (graf tamu) terdiri atas 2 pemetaan: (1) Sebuah fungsi α memetakan noktah pada G satu per satu ke noktah di H. (2) Sebuah fungsi ρ membuat pada tiap sisi $\{u,v\} \in E(G)$ sebuah jalur pada H yang menghubungkan noktah $\alpha(u)$ dan $\alpha(v)$.
Dalam GEM, ide pananaman graf ke jaringan *sensor* diaplikasikan dalam dua langkah. Pertama, kita harus memilih sebuah graf tamu G yang sudah berlabel yang bisa digunakan untuk penjaluran dan penyimpanan data-sentris yang efektif. Langkah kedua adalah menanamkan graf tamu G ke topologi jaringan *sensor* yang nyata, H.

Pemilihan Graf Tamu. Langkah pertama adalah memilih graf tamu yang dapat digunakan untuk penjaluran dan penyimpanan data-sentris. Graf tersebut harus mempunyai aspek berikut :

- *Routing* : Graf berlabel G harus mempunyai sifat yang membolehkan pengguna mengirim pesan secara efisien dari satu noktah ke noktah lain walaupun noktah tersebut hanya mengetahui label tetangganya. Karena itu, label pada noktah harus mempunyai informasi secara implisit mengenai topologi jaringannya.
- Pemetaan untuk tabel *hash* terdistribusi : Harus ada fungsi yang secara efisien memetakan sebah kunci K ke sebuah label dalam ruang label L.
- *Low embedding overhead* : penanaman graf G ke dalam topologi H harus terdistribusi, tanpa membuat overhead yang tak perlu.
- Toleransi kesalahan : Jika terjadi kegagalan dalam sebuah noktah pada jaringan sensor, graf tamu harus bisa mencari jalur lain, atau noktah tersebut harus terlihat sehingga penanaman graf dapat dibuat ulang untuk memperbaiki graf tamu.

Cara kedua adalah membuat sebuah penanaman yang dapat mensimulasikan graf tamu G dengan

menggunakan topologi jaringan aslinya, H. Ada beberapa cara untuk penanaman graf[1], tapi yang akan dipakai adalah dilasi. Secara ringkas, kita menginginkan fungsi penjaluran sisi ρ untuk memetakan sisi pada G ke jalur terpendek di H. Secara ideal, setiap sisi di G akan dipetakan ke sisi yang berhubungan di H. Tapi, saat koneksi yang benar tidak ada di H, sisi tersebut harus disimulasikan oleh sebuah jalur.

Dapat dilihat bahwa dengan penggunaan penanaman graf dengan cara ini bisa berguna sebagai infrastruktur dan memungkinkan banyak aplikasi dijalankan secara efisien :

Memungkinkan Penyimpanan Secara Data-sentris. Untuk memungkinkan penyimpanan data-sentris, sebuah data dengan nama k, dapat dipetakan ke label f(k) dan kemudian disalurkan dan disimpan pada noktah dengan label tersebut. Pengecekan untuk data dengan nama tersebut dapat dilakukan dengan menghitung f(k) dan mengirimkan query ke noktah yang sama. Karena graf berlabel G ditanamkan ke graf H, penjaluran pada ruang label dapat hampir sama efisiennya dengan penjaluran jarak terpendek pada topologi jaringan aslinya. Hal ini sangat berbeda dengan distribusi tabel hash untuk sistem terdistribusi melalui internet.

Memungkinkan Penjaluran Noktah ke Noktah. Untuk memungkinkan penjaluran noktah ke noktah, kita bisa mengimplementasi sebuah mekanisme pencari untuk menemukan label dari noktah saat ini menggunakan penyimpanan data-sentris. Asumsikan noktah mempunyai tanda pengenal n, dan labelnya adalah L(n). Node n dapat menyimpan labelnya pada tabel hash terdistribusi, pada noktah yang bersangkutan dengan f(n) dengan cara yang sama dengan menyimpan data. Saat noktah lain ingin berkomunikasi dengan n, noktah itu bisa mengirimkan pencarian ke noktah dengan label f(n), mengambil data label n, yaitu L(n). Saat noktah tersebut mempunyai label L(n) milik n, noktah tersebut dapat mengirimkan pesan ke L(n) dan akan disalurkan melalui graf ke n.

3. VPCS (Virtual Polar Coordinate Space)

Untuk mendemostrasikan cara mengaplikasikan penanaman graf pada jaringan *sensor*, dibuat *Virtual Polar Coodinate Space*. Dalam VPCS, graf pohon bergelang ditanamkan ke dalam sebuah topologi jaringan. Hal ini dilakukan dengan cara memberikan tiap noktah sebuah tingkat, yaitu banyaknya loncatan menuju akar dari pohon, dan sebuah jajaran sudut *virtual*, yang mengidentifikasikan sebuah noktah secara unik dalam sebuah tingkat. Sebuah mekanisme juga dibuat agar memungkinkan pemberian sudut *virtual* secara konsisten dalam topologi jaringan. Hasil dari proses ini adalah semua label dapat dilihat dan dimengerti dengan mendefinisikan ruang koordinat polar *virtual*. Pada bagian ke-4 nanti, sifat ini akan memungkinkan untuk memakai *greedy forwarding* untuk mencapai penjaluran yang efisien.

3.1. Dasar dari membuat VPCS

Langkah pertama untuk membuat virtual polar coordinate space, atau VPCS, adalah menanamkan sebuah pohon bergelang. Kita dapat mengikuti algoritma yang sudah terkenal untuk membuat sebuah pohon merentang.
Untuk membuat sebuah pohon merentang, pertama kita memilih sebuah noktah yang berfungsi sebagai akar pohon. Noktah manapun dapat dipilih, tapi jika ada noktah yang berfungsi sebagai pusat sebaiknya itu yang dipilih. Akar akan memberi pernyataan bahwa dia terdapat pada tingkat 0 dalam pohon, noktah-noktah yang berada dalam jarak radio dari akar langsung menjadi anak pohon. Noktah-noktah ini kemudian memberi pernyataan bahwa mereka terdapat pada tingkat 1 dari pohon. Noktah akar yang mendapat pernyataan dari noktah-noktah tersebut menandai pengirim pernyataan sebagai anaknya. Noktah lain yang belum tergabung dalam pohon akan menandai pengirim pernyataan sebagai *parent*. Jika sebuah noktah mendapat lebih dari 1 pernyataan, noktah tersebut dapat memilih yang manapun untuk ditandai sebagai *parent*, walaupun begitu, sebaiknya dipilih noktah yang pernyataannya diterima dengan kekuatan sinyal terbesar. Proses ini berlanjut secara rekursif sampai semua noktah yang bisa dicapai tergabung dalam pohon.

Setelah sebuah pohon dibuat, informasi besar tiap sub-pohon dikirimkan menuju akar. Noktah daun memulai hal ini dengan mengirimkan noktah *parent*-nya sebuah informasi sub-pohon dengan besar satu. Setelah sebuah noktah menerima pesan dari anak-anaknya, dia menambahkan satu ke jumlah tersebut dan mengirimkan jumlah sub-pohon itu ke noktah *parent*. Setelah akar mendapat ukuran besar sub-pohon dari tiap anak-anaknya, dia sudah mempunyai cukup informasi untuk mulai memberi sudut virtual ke tiap anak-anaknya. Untuk memulai, akar diberikan jajaran sudut yang dipakai. Dalam sebuah koordinat polar geometris nilainya adalah

0 sampai 2π. Tapi karena sebaiknya pembagian dapat dilakukan tanpa berhubungan dengan pecahan, jajaran seperti 0 sampai 2^{16} -1 atau 0 sampai 2^{32} -1 lebih sesuai. Akar kemudian akan memberikan tiap anak sebuah subset dari jajaran tersebut. Ukuran subset yang diberikan ke tiap anak sebanding dengan ukuran sub-pohon anak tersebut. Sebagai contoh, jika anak-anaknya mempunyai ukuran sub-pohon 10, 20 dan 15, maka jajaran yang diberikan adalah $\frac{10}{45}$, $\frac{20}{45}$, dan $\frac{15}{45}$ dari jajaran sudut. Penyeimbangan ini memberikan sub-pohon yang lebih besar lebih banyak jajaran sudut untuk dicocokan. Hal ini juga berguna untuk memastikan jajaran sudut tidak habis sebelum mencapai semua daun dari pohon. Tiap anak kemudian memberikan anak-anaknya subset dari jajaran sudut yang dipunyainya, dan terus berlaku secara rekursif sampai semua noktah mendapat jajaran sudut. Sebagai contoh untuk penomoran, lihat gambar 1(a).

Biaya energi yang digunakan untuk konstruksi ini cukup rendah. Jika semua proses digabung, tiap noktah harus mengirimkan 3 pesan saat pembuatan ruang *virtual*, saat pembuatan pohon, saat mengukur besar dari sub-pohon, dan saat pemberian jajaran sudut *virtual*.

3.2. Kebutuhan Konsistensi

Agar VPCS bisa berfungsi dengan baik, kondisi berikut harus dipertahankan :

1. Tiap noktah harus mempunyai *parent* (kecuali noktah akar).
2. Tiap noktah harus diberi tingkatan yang setara dengan tingkatan *parent*-nya ditambah satu.
3. Tiap noktah harus diberi jajaran sudut *virtual* yang merupakan subset dari jajaran sudut *virtual parent*-nya.
4. Tak boleh ada 2 anak yang mempunyai jajaran sudut yang bertumpukkan.

Sudah jelas bahwa algoritma di bagian 3.1 sudah memenuhi kebutuhan di atas. Akan diperlihatkan nanti bahwa kondisi ini memungkinkan penjaluran dan penyimpanan data-sentris yang efektid.

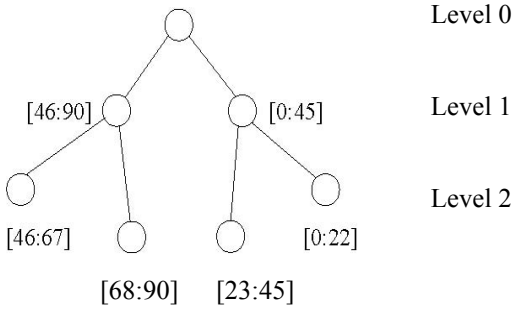
3.3. Mensejajarkan VPCS dengan Topologi Jaringan tanpa Informasi Geografik

Metode yang telah dideskripsikan sejauh ini untuk memberikan ruang virtual sudah cukup untuk menyediakan penjaluran noktah ke noktah yang dapat diandalkan, yang akan dideskripsikan pada 4.1 dan 4.2. Tetapi, untuk mengambil dari keunggulan hubungan silang pada struktur pohon bergelang, kita harus memberi label pada noktah sehingga noktah akan mengetahui hubungan silang yang sebaiknya digunakan, jika ada. Untuk mengatasi hal ini, ruang koordinat virtual akan disejajarkan dengan topologi jaringan. Secara spesifik, jika kita mengikuti "gelang" dalam sebuah tingkat pada pohon, sudutnya akan bertambah atau berkurang (sampai kembali ke tempat semula). Untuk ilustrasi hal ini, lihat gambar 1(b). Ditunjukkan pada bagian 4.3 bahwa hal ini akan memungkinkan kita menggunakan hubungan bersilangan untuk melakukan penjaluran yang lebih efisien. Untuk melakukan pensejajaran ini dalam sistem yang terdistribusi, tiap parent harus menentukan urutan dari anak-anaknya dalam gelang. Banyak cara untuk melakukan hal ini, akan dideskripsikan 2 cara diantaranya di bawah ini. *Naïve Scheme* membutuhkan noktah untuk bisa mencari jarak antara dirinya dengan tetangganya sehingga bisa mumbuat sistem koordinat lokal. *Improved Scheme* tidak tergantung pada asumsi lokalisasi dan mendapatkan hasil yang lebih baik.

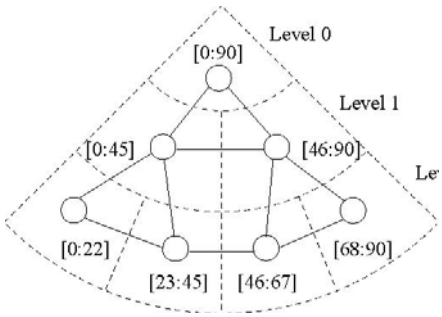
Naïve Scheme : Menggunakan informasi jarak.
Pertama kita melihat kasus di mana noktah-noktah dapat memperkirakan jarak antara mereka dan noktah tetangga. Cara yang dapat digunakan adalah mengukur perbedaan waktu anata sinyal radio dan sinyal ultrasonik. Tiap noktah kemudian dapat menggunakan informasi ini untuk menentukan koordinat tetangganya dalam sistem koordinat lokal, seperti yang dilihat pada [6]. Setelah itu noktah tersebut dapat mengurutkan anak-anaknya dengan sudut geometris. Sayangnya, kesalahan kecil dalam perkiraan jarak dapat menghambat noktah anak diurutkan secara benar. Apalagi, sub-pohon dapat bersilangan, sehingga lebih masuk akal jika kita mempertimbangkan sudut seluruh sub-pohon noktah anak tersebut, dibandingkan hanya sudut noktah anak saja.

Improved Scheme : Menggunakan sistem koordinat global. Sekarang akan dideskripsikan cara menggunakan koordinat sistem global untuk mengatasi masalah di atas, dan cara membuat pengurutan tanpa melokalisasi data.

Kita membuat sistem koordinat global menggunakan segitiga dari akar dan 2 buah sederhana noktah lain. Noktah akar hanya



(a) Pemberian jajaran sudut sederhana



(b) Pemberian urutan jajaran sudut, dengan adanya hubungan silang

Gambar 1: Contoh pemberian jajaran sudut

Menggunakan heuristik sederhana untuk memilih noktah lain sebagai referensi seperti 3 noktah yang tidak sejajar dan tidak terlalu dekat satu sama lain. Untuk tiap noktah menghitung posisinya, noktah perlu mengetahui jarak ke noktah yang direferensikan, dan jarak antar noktah referensi. Dengan begitu, kita mengukur jarak saat lompatan antar noktah dengan jarak terpendek, bukannya langsung mengukur jarak seperti pada sistem koordinat lokal.

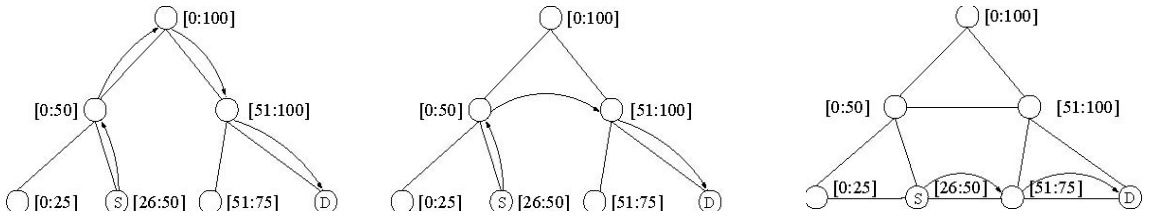
Kita pertama-tama membuat sebuah pohon merentang sementara dari 2 noktah referensi. Setiap noktah kemudian mengetahui jaraknya,dalam loncatan jaringan, dari tiap noktah referensi : sama dengan tingkatan noktah dalam pohon merentang noktah referensi. Salah satu noktah referensi kemudian bisa mengirimkan jarak antara dirinya dan noktah referensi yang satunya dan mengirimkannya ke noktah akar. Noktah akar kemudian membanjiri jaringan dengan jarak antara akar dengan 2 noktah referensi. Dengan informasi ini tiap noktah dapat menghitung posisinya di dalam jaringan.
Karena jaraknya diukur dalam bentuk loncatan jaringan, hasilnya tidak akurat. Ada kemungkinan noktah yang satu tingkat mempunyai koordinat yang sama, hal ini diatasi dengan mengurutkan anak-anak dengan sudut dari pusat massanya. Pusat massa bisa dilihat sebagai titik di tengah sebuah sub-pohon dari noktah, yang didefinisikan dengan menghitung rata-rata dari koordinat pada sebuah sub-pohon. Mengurutkan noktah anak sesuai sudut dari pusat massa ini menyelesaikan masalah ketidakakuratan, karena kesalahan pada tiap noktah akan dirata-ratakan di pusat massa.

4. VPCR : Sebuah Algoritma untuk VPCS

Pada bagian ini dideskripsikan VPCRm yaitu sebuah algoritma penjaluran yang dibuat dalam VPCS. VPCR menjalurkan dari noktah manapun ke titik manapun dalam VPCS, di mana sebuah titik didefinisikan dengan tingkatan dan sudut. Akan didiskusikan 3 algoritma, yaitu penjaluran *naïve-tree* yang tidak memakai hubungan silang dari pohon bergelang, penjaluran *smart-tree*, hasil perbaikan dari *naïve-tree*, dan yang terakhir adalah VPCR, yang menggunakan g*reedy forwarding* untuk memakai hubungan silang pada pohon bergelang.

4.1 Penjaluran *Naïve Tree*

Dalam algoritma ini, sebuah pohon merentang dibuat dengan pusat pada akar. Agar pesan bisa disalurkan menuju pusat, tiap noktah mengirimkan pesan ke parent-nya sampai pesan tersebut tiba di pusat. Pesan dapat dikirim dari noktah manapun pada jaringan dengan menggunakan jumlah minimum loncatan. Dengan VPCS, algoritma ini dapat diperbesar untuk menyelesaikan penjaluran noktah ke noktah. Tiap pesan disalurkan menuju ke puncak pohon sampai ke ancestor dari tujuan. Pada tiap loncatan, noktah dapat menentukan bahwa tujuan berada dalam sub-pohonnya atau tidak dengan cara mengecek sudut virtual tujuan ada dalam jajaran sudutnya, dan apakah tingkatan tujuan lebih tinggi dari tingkatan saat ini. Jika tidak, maka tujuan tidak terdapat dalam sub-pohon noktah, sehingga pesan diteruskan ke parent. Jika tujuan berada dalam sub-pohon noktah, maka digunakan metode yang sama untuk menentukan di sub-pohon mana terdapat tujuan, dan meneruskan pesan ke sub-pohon tersebut. Diilustrasikan pada gambar 2(a).
Bagian yang tidak efektif dari algoritma ini adalah pesan harus menuju ancestor dari tujuan terlebih dahulu, jadi jalurnya akan lebih pendek jika menghindari penjaluran ke atas sebisa mungkin. Selain membuat jalannya lebih pendek,



(a) Naive Tree, 4 loncatan (b) Smart Tree, 3 loncatan (c) VPCR, 2 loncatan

Gambar 2: Sebuah paket dikirimkan dari S ke D menggunakan 3 algoritma penjaluran. Smart Tree dan VPCR menggunakan loncatan ke tetangga.

ini juga berfungsi dalam menghindari lalu-lintas noktah di bagian atas menjadi penuh.

4.2. Penjaluran Smart Tree

Pada kasus di mana algoritma *naïve tree* langsung mengirim ke atas, algoritma *smart tree* mengecek tetangganya apakah tetangganya merupakan *ancestor* dari tujuan. Jika ternyata tetangganya adalah ancestor tujuan, maka pesan langsung dikirimkan ke tetangganya tersebut, tanpa dikirimkan ke atas terlebih dahulu. Dapat dilihat pada gambar 2(b) hal ini akan lebih sedikit menggunakan loncatan. Metode ini bisa juga ditambah lebih lanjut jika tiap noktah menyimpan data dari tetangga dari tetangganya, sehingga data langsung dikirimkan ke noktah tersebut dalam 2 loncatan. Dapat juga sebuah noktah menyimpan topologi dari seluruh jaringan sehingga pesan selalu melalui jarak terpendek, tapi data yang harus disimpan sangat banyak.

4.3 Virtual Polar Coordinate Routing (VPCR)

Smart tree hanya menyediakan jalan pintas apabila mencapai noktah yang dekat dengan ancestor dari tujuan. Untuk noktah yang jauh, pesan akan tetap dikirimkan ke atas hingga mencapai noktah yang bisa mengefektifkan *smart-tree*. VPCR mencoba menggunakan hubungan silang dalam pohon bergelang untuk menyalurkan secara lateral dalam pohon, walaupun noktah saat ini tidak mengetahui secara eksplisit tentang ancestor dari tujuan. Hal ini dilakukan dengan asumsi sudut virtual akan bertambah atau berkurang saat melalui gelang pada pohon bergelang dengan pengurutan pada 3.3. Pada kasus ini, tiap label dapat dilihat sebagai pendefinisian ruang dalam VPCS, seperti pada gambar 1(b).

VPCR menggunakan ruang koordinat polar ini untuk penjaluran yang lebih efisien. Saat smart-tree diharuskan mengirim pesan ke atas, VPCR mengecek apakah ada tetangganya yang lebih dekat dengan jajaran sudut tujuan dibandingkan jajaran sudut saat ini. Jika ya, maka pesan langsung dikirimkan (*greedily forwarded*) lebih dekat ke jajaran sudut dari tujuan. Gambar 2(c) memperlihatkan bahwa jarak yang ditempuh akan lebih pendek secara signifikan dari algoritma yang lain.

Terkadang pesan akan mencapai minimum lokal, di mana tak ada noktah lain yang mempunyai sudut lebih dekat dengan tujuan. Hal ini terjadi karena ada hubungan silang yang hilang dalam pohon bergelang. Saat ini terjadi, kita bisa menggunakan struktur pohon untuk mencari jalur sebagai pengganti *link* yang hilang. Secara praktis, hal ini dapat dilakukan dengan cara mengirim pesan ke noktah *parent*. Pada akhirnya pesan akan mencapai noktah di mana hubungan silangnya tidak rusak, sehingga *greedy forwarding* dapat terus berlangsung, atau mencapai ancestor dari tujuan di mana pesan langsung disalurkan menurun untuk mencapai tujuan.

Jika VPCS sudah disejajarkan dengan topologi jaringan, sebagai hasilnya paket-paket data akan mengikuti lengkungan dari sumber ke tujuan. Untuk jarak pendek, lengkungan tersebut tidak lebih panjang dari garis lurus, tapi untuk sudut melebihi 115°, jarak dari sumber ke pusat kemudian menuju tujuan akan lebih pendek daripada lengkungannya. Karena itu pada implementasi saat ini kita melanjutkan pesan ke atas pohon daripada mengikuti lengkungan jika ada kasus seperti ini.

Sifat penting dari algoritma ini adalah selama VPCS konsisten, maka paket akan selalu sampai pada tujuannya. Jika tak ada noktah di dekatnya yang mempunyai jarak lebih dekat ke tujuan, maka paket akan dikirimkan ke *parent*.

Sifat penting lainnya dari VPCR adalah loop penjaluran tidak mungkin bisa dilakukan. Paket tidak pernah dikirimkan ke noktah yang mempunyai jarak lebih jauh dengan tujuan, paket selalu disalurkan ke noktah yang jaraknya lebih dekat dengan tujuan, jika gagal akan dikirim ke *parent*. Karena jajaran sudut dari *parent* merupakan superset dari sudut saat ini, jaraknya tidak lebih jauh dari noktah saat ini, dan paket tidak mungkin dikirimkan kembali ke anak karena jaraknya tidak lebih pendek dari noktah *parent*.

5. Kasus Dinamis dalam VPCS

Topologi jaringan mungkin berubah seiring dengan waktu. Noktah dapat tidak berfungsi, noktah baru mungkin ditambahkan. Saat hal-hal seperti ini terjadi, VPCS harus diubah agar tetap konsisten. Saat membuat perubahan dalam VPCS, konsistensi pada bagian 3.2 harus tetap, selain itu ada 2 tujuan yang harus diingat, yaitu stabilitas dan kesejajaran dengan topologi jaringan.

Tergantung cara komunikasi noktah ke noktah yang digunakan, perubahan koordinat mungkin mahal. Pada kasus penyimpanan data, perubahan sudut mungkin menyebabkan pemindahan data dari satu noktah ke noktah lain. Selain itu, perubahan koordinat membutuhkan komunikasi antar noktah yang akan berubah koordinatnya, yang membutuhkan energi. Karena itu perubahan pada VPCS harus sesedikit mungkin sambil tetap konsisten.
Saat membuat jalur menggunakan VPCR, makin dekat VPCS menggambarkan topologi, semakin efektif penyampaian datanya. Karena itu korelasi anatara VPCS dengan topologi harus dipertahankan. Sangat sulit mencapai kedua tujuan ini, salah satu cara menjaga adalah me'reboot' sebagian atau seluruh jaringan, yaitu menghapus jalur dan *address* kemudian melakukan ulang algoritma 3.1. Cara ini cara terbaik menjaga korelasi, tetapi sangat mahal karena banyak noktah yang berganti posisi. Karena itu cara ini tak akan digunakan kecuali pada kasus di mana perubahan jarang terjadi. Oleh sebab itu kinerja penjaluran yang lebih baik akan mengurangi biaya harus membuat ulang sebagian VPCS saat topologi berubah.

Algoritma yang digunakan lebih mengutamakan pengurangan jumlah noktah yang harus diganti *address*nya dibandingkan menjaga korelasi VPCS dan jaringan. Sebagai hasilnya hanya sedikit noktah yang harus diganti *address*nya saat terjadi perubahan akibat rusaknya atau penambahan noktah.

5.1. Penanganan Kerusakan Noktah

Alasan yang paling banyak dikemukakan bertanggunjawab atas rusaknya suatu noktah biasanya karena kehabisan energi, beberapa mungkin karena lingkungan yang tidak memadai. Agar VPCS menjadi solusi yang dibsa dipraktekkan, VPCS harus bisa menangani masalah kerusakan dengan *overhead* minimum.

Saat sebuah noktah P rusak, anak-anaknya kehilangan parent, melanggar konsistensi pada bagian 3.2. Tiap anak C dapat mencari noktah lain, P' yang terhubung dengan pohon sebagai parent-nya, tapi ini akan melanggar konsistensi ke-3, karena jajaran sudut milik P' tidak mempunyai sudut milik C. Untuk memperbaiki hal ini, P' harus menambah sudut milik C ke jajaran sudut miliknya. Parent dari P' juga harus melakukan hal yang sama, dan akan berlaku rekursif hingga mencapai *ancestor* terendah dari P dan P'. Saat ini *ancestor* tersebut melanggar konsistensi ke-4, yaitu mempunyai jajaran sudut yang bertumpukkan, sebagai *ancestor* dari P dan P'. Hal ini dapat diperbaiki dengan cara menghilangkan jajaran sudut C dari anak *ancestor* tersebut, dan berlaku rekursif sampai *parent* dari P menghilangkan jajaran sudut C dari jajaran sudutnya. Terakhir, agar konsistensi ke-2 dipenuhi, maka C harus mengubah tingkatannya menjadi tingkatan P' ditambah satu. Anak dari C juga harus melakukan hal yang sama, hingga mencapai daun dari pohon.

Terkadang noktah yang terputus tidak bisa mencapai noktah untuk dibuat *parent*, tapi salah satu keturunannya bisa, pada saat itu, sang anak menjadi *parent* dari noktah *parent*nya, dan pergantian itu terus berlangsung hingga mencapai noktah yang terputus itu. Saat mengubah hubungan parent-anak, jajaran sudut harus tetap konsisten. Untuk itu, keturunan yang terhubung dengan pohon harus mengambil jajaran sudut dari noktah yang terputus, dan memberikan (bekas) *parent*nya jajaran sudut baru, dikurangi jajaran sudut anak lainnya. Noktah itupun kemudian mengirimkan hal yang sama ke (bekas) *parent*nya hingga mencapai noktah yang terputus. Saat balikan dari pohon sudah terbentuk, akar baru dari sub-pohon membuat noktah yang terhubung dengan pohon menjadi parentnya, dan melakukan algoritma yang sudah dideskripsikan.

Kasus terakhir yang mungkin adalah tidak ada noktah yang bisa dihubungkan, pada kasus ini, sub-pohon menjadi terpisah dan tidak ada cara untuk menghubungkannya kembali.

Ada beberapa hal yang mungkin menjadi masalah dalam algoritma ini. Masalah pertama adalah beberapa noktah mungkin mempunyai jajaran sudut yang tidak bisa diteruskan. Sebagai contoh, jika sebuah noktah mempunyai jajaran 50-100, dan jajaran 60-70 dihilangkan, yang tersisa adalah 50-59 dan 71-100. Walaupun masih bisa diterima dalam VPCS dan VPCR, hal ini membuat penyimpanan dari sudut tetangga menjadi lebih kompleks. Jajaran sudut dari sebuah noktah mungkin terdiri dari angka yang tidak diteruskan. Masalah lainnya adalah saat algoritma perbaikan pohon, ada noktah lain yang mungkin rusak, hal ini mungkin membuat VPCS tidak konsisten. Jika jajaran sudut menjadi terlalu terpisah, atau VPCS menjadi tidak konsisten, sub-pohon yang terpengaruh dapat me-*reboot* dirinya sehingga bagian sub-pohon tersebut kembali konsisten.

5.2. Penambahan Noktah

Untuk banyak aplikasi, penambahan noktah ke jaringan diperlukan, baik itu untuk memperbaiki, meningkatkan daerah pengawasan *sensor*, ataupun menggunakan *sensor* tipe baru. Agar noktah baru bisa menjadi bagian dari VPCS, noktah itu harus tergabung dalam pohon dan diberikan tingkatan serta jajaran sudut. Seperti pada perbaikan, proses penambahan noktah lebih diutamakan pada penghematan dibandingkan konsistensi korelasi VPCS dan jaringan.

Saat noktah baru ditambahkan ke dalam jaringan VPCS, noktah baru ini harus diberikan jalur dan jajaran sudut, untuk melakukan hal ini, noktah baru memilih parent dari tetangganya. Parentnya tersebut kemudian memberikan tingkatan dirinya ditambah satu ke noktah baru. Kemudian parent ini harus menghilangkan sebuah jajaran sudut dari anaknya untuk diberikan ke noktah baru. Anaknya kemudian menghilangkan jajaran sudut tersebut dari keturunannya, terus berlangsung sampai daun dari pohon.

Jika noktah baru dapat mencari noktah terhubung untuk digunakan sebagai parent, dan dapat mencapai noktah yang terpisah dari jaringan, jaringan yang sebelumnya terputus dapat terhubung kembali melalui noktah baru ini. Noktah baru memberi jajaran sudut ke anak-anaknya sama seperti sewaktu pembuatan pohon pertama kali.

6. Evaluasi VPCS dan VPCR

Pada bagian ini, kinerja algoritma VPCS dan VPCR akan dievaluasi. Parameter yang digunakan untuk percobaan terdapat pada tabel berikut :

Jarak	40 m
Besar Jaringan	400 m x 400 m
Kerapatan	Rata-rata 15 per tingkat
Noktah	509
Ketetanggaan	2 loncatan

Untuk menunjukkan keefektifan VPCR dalam penjaluran jaringan ang besar, dipilih sebuah jaringan yang cukup besar dalam hubungannya dengan jarak radio dari noktah. Sebuah paket dikirim melewati satu sisi jaringan yang harus melakukan loncatan setidaknya 10 kali. Sebuah paket dikirim dari satu ujung jaringan ke ujung lain setidaknya membutuhkan 15 loncatan. Jaringan yang dipilih memiliki rata-rata 15 tetangga untuk memastikan topologi yang dibuat secara acak dapat terhubung dengan baik.
Besar ketetanggaan dipilih 2 loncatan untuk penggunaan *smart tree* dan algoritma VPCR. Besarnya ketetanggaan menambah keefektifan penjaluran dengan *drawback* penambahan pernyataan pada noktah. Akan ditunjukkan pada bagian 6.9 bahwa ketetanggaan 2 loncatan memberikan keseimbangan yang baik antara jumlah pernyataan yang harus disimpan dan kinerja penjaluran.

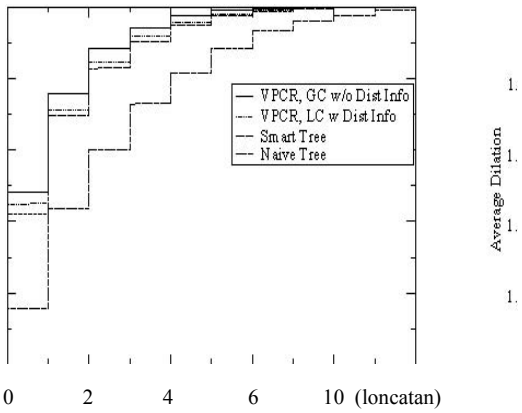
Hasil evaluasi didapat dari [3] dengan simulator khusus buatan mereka sendiri. Simulator ini menggunakan asumsi model radio ideal dan layer MAC ideal, jika da noktah dalam jarak radio, mereka bisa berhubungan tanpa adanya kehilangan paket. Semua hasil didapat dari menjalankan tes pada 5 topologi acak dan merata-ratakan hasilnya. Pada tiap topologi, tiap noktah diletakkan secara acak kecuali akar dari pohon, yang diletakkan di tengah jaringan. Saat topologi jaringan acak memiliki kurang dari 95% noktah yang tidak dapat dijangkau oleh akar, topologi tersebut tidak digunakan dan diganti dengan yang lain.

Pada simulasi VPCR digunakan baik metode sistem koordinat lokal maupun sistem koordinat global, yang dideskripsikan di bagian 3.3. Pada

hasil yang berlabel "VPCR, LC w Dist Info" digunakan naïve scheme, yang mengasumsikan tiap noktah bisa menemukan jarak antar noktah yang bertetangga dan menggunakan informasi tersebut untuk membentuk sistem koordinat lokal pada tiap noktah. Pada hasil yang berlabel "VPCR, GC w/o DistInfo" digunakan improved scheme, yang tidak mengasumsikan tiap noktah dapat mencari jarak antar tetangga, tetapi menggunakan noktah terpilih sebagai referensi untuk membuat sebuah sistem kordinat global. Untuk percobaan ini, kedua noktah referensi diletakkan pada ujung jaringan, terpisah 90 derajat.

6.1 Kinerja Mendekati Jarak Terpendek

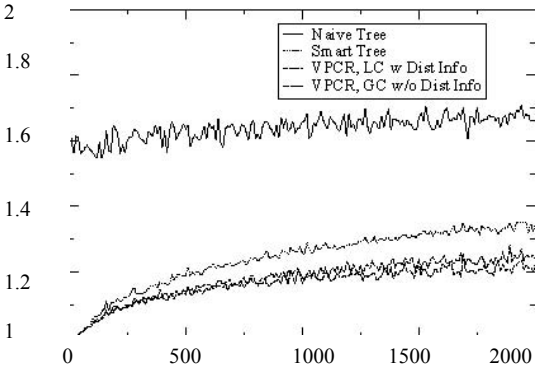
Gambar 3 menunjukkan fungsi distribusi kumulatif untuk menghitung banyaknya loncatan tambahan tiap paket dibanding jarak terpendek. Dengan menggunakan pengurutan koordinat global, jalur VPCR menyalurkan 75% paketnya dengan 0 atau 1 loncatan tambahan. Walaupun tanpa hubungan silang, algoritma smart tree menyalurkan 70% paketnya dengan 0 atau 1 loncatan tambahan.



Gambar 3: Penambahan loncatan dibandingkan jarak terpendek

6.2. Perbandingan terhadap Jaringan Besar

Berikutnya akan diperiksa perbandingan VPCR dengan besarnya jaringan. Untuk percobaan ini, jumlah noktah akan ditambah dengan mempertahankan kerapatannya. Karena jalur rata-rata bertambah panjang saat jaringan bertambah besar, yang diukur adalah rata-rata perpindahan, bukan jumlah loncatan terhadap jarak terpendek. Perpindahan sebuah paket dihitung dengan jumlah loncatan yang sebenarnya terjadi dibagi dengan jumlah loncatan pada jarak terpendek.



Gambar 4: Perpindahan pada jaringan berbanding dengan besarnya

Gambar 4 menunjukkan rata-rata perpindahan pada jaringan dari 10 sampai 200 noktah. Karena semua variasi dari VPCR berbasis pada pohon, maka overheadnya berkembang secara logaritma dengan besar jaringan.

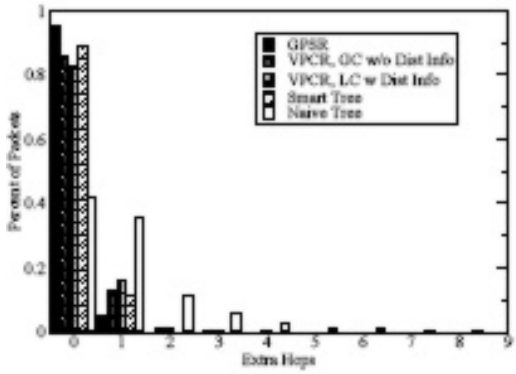
6.3 Perbandingan dengan GPSR

Gambar 5 membandingkan algoritma penjaluran pada makalah ini dengan GPSR. Gambar ini memperlihatkan berapa tambahan loncatan terhadap jalur terpendek. Parameter dan hasil tes untuk GPSR diambil dari [2]. Luas jaringannya 2250m dan 450m, dengan 112 noktah yang mempunyai jarak radio 250m. Perlu diketahui bahwa pada percobaan GPSR digunakan model jalur acak, sementara pada simulasi lain, noktahnya tetap tidak bergerak. Karena itu digunakan hasil dengan mobilitas terendah, yaitu saat waktu selanya 120s.

Semua kecuali algoritma naïve bekerja dengan baik di sini. Hal yang perlu diperhatikan adalah algoritma smart tree sedikit lebih baik untuk percobaan ini dibandingkan dengan VPCR untuk jaringan ini. Hal ini terjadi karena jaringan tidak lebih lebar dari jarak radio noktah. Jadi pesan yang dilempar akan mencapai sebuah noktah dalam ketetanggaan dari tujuan tanpa harus naik ke puncak pohon. Dalam hal ini, greedy forwarding dari VPCR sulit untuk lebih baik, dan mungkin akan melakukan kesalahan. Walaupun begitu, sangat sedikit paket yang melebihi satu langkah dari jalur terpendek untuk kedua algoritma ini.

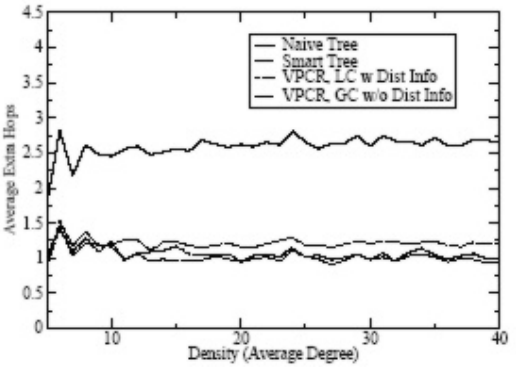
6.4. Perbandingan terhadap Kerapatan Jaringan

Gambar 6 menunjukkan bahwa algoritma yang berbeda bekerja pada kerapatan yang berbeda.



Gambar 5 : Histogram loncatan tambahan yang diperlukan dari lintasan terendek

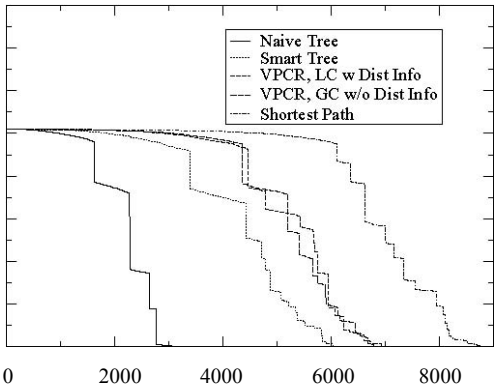
Dalam percobaan ini kerapatan diukur dalam rata-rata derajat, yang merupakan rata-rata jumlah tetangga ang dimiliki tiap noktah. Untuk percobaan ini luas jaringan dipertahankan pada 400x400m, jumlah noktah diubah untuk mendapatkan kerapatan yang berbeda. Percobaan ini menunjukkan bahwa kerapatan jaringan hanya memberi akibat yang kecil pada kinerja VPCR dan variasinya.



Gambar 6: Rata-rata loncatan tambahan untuk kerapatan yang berubah

6.5. Keseimbangan Beban

Salah satu masalah pada algoritma *naïve tree* adalah kecenderungan mengirimkan paket ke bagian atas pohon. Karena tiap noktah mempunyai energi yang terbatas, maka noktah-noktah bagian atas tersebut akan cepat kehabisan energi, mengakibatkan akar pohon terputus dari jaringannya. Algoritma *smart tree* dan VPCR mencoba mencegah penjaluran ke puncak pohon sebesar algoritma *naïve tree*, jadi kedua algoritma ini tidak bermasalah begitu besar dalam hal ini.
Untuk mengevaluasi keseimbangan beban pada



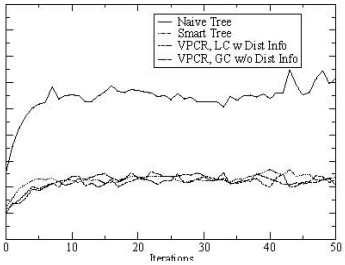
Gambar 7: Lifetime Jaringan

VPCR, paket disalurkan dari dan ke noktah acak dalam jaringan, tapi membatasi noktah hanya dapat meneruskan 200 paket sebelum rusak. Gambar 7 menunjukkan angka noktah hidup yang dapat dicapai oleh akar setelah paket dikirim dan ada noktah yang rusak. Seperti yang sudah diperhitungkan, noktah yang dapat dicapai oleh algoritma naïve tree berkurang secara drastis karena noktah pada bagian atas rusak. Algoritma smart tree dan VPCR bertahan cukup lama, tapi tak ada yang bertahan sampai semua paket disalurkan.

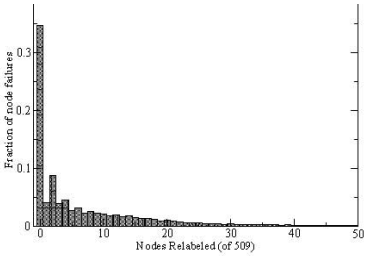
6.6. Kemampuan Adaptasi terhadap Jaringan Dinamis

Penambahan dan pengurangan noktah cenderung mempengaruhi pohon. Sewaktu ruang sudut diatur agar pohon tetap konsisten, VPCS mungkin akan kurang konsisten dalam hubungannya dengan topologi jaringan. Apalagi saat sebuah noktah memilih parent baru, sebisa mungkin noktah yang harus diatur ulang koordinat polar virtualnya diminimalisir, yang mungkin menyebabkan cabang pohon berkembang ke arah yang salah dan saling bertumpukkan.

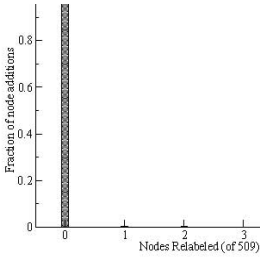
Agar bisa menentukan efek dari penambahan dan penghilangan noktah dari pohon yang terus-menerus, dilakukan percobaan berikut. Dimulai dengan pohon yang baru dibuat, 20% dari noktahnya dibuat rusak. Setelah melakukan algoritma perbaikan pohon, rata-rata dari loncatan tambahan yang dilakukan paket pada jaringan tersebut diukur. Kemudian secara acak ditambahkan noktah baru dengan jumlah yang sama dengan yang rusak di dalam jaringan dan menggabungkan mereka dengan pohon. Rata-rata loncatan tambahan diukur kembali setelah penggabungan ini. Proses ini dilakukan sebanyak 50 kali, dan hasilnya terlihat di gambar 8.



Gambar 8: Rata-rata loncatan tambahan saat penghilangan dan penggantian noktah



Gambar 9: Histogram noktah yang dilabel ulang saat noktah rusak



Gambar 10: Histogram noktah yang dilabel ulang saat noktah diganti

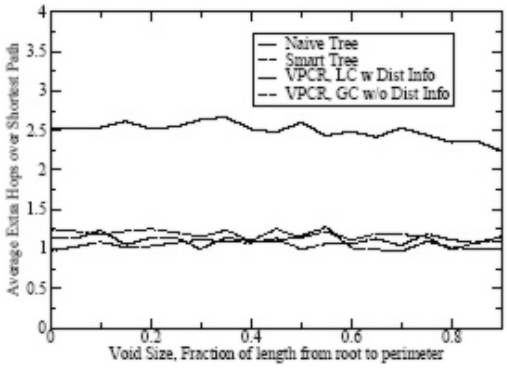
Kinerja penjaluran terus menurun sampai iterasi ke-10, setelah itu perbedaannya tidak bergitu jauh. Hal ini terjadi karena algoritma perbaikan dan penambahan pohon saat ini lebih dititikberatkan pada penghematan dibandingkan memperta-hankan kesejajaran VPCS dengan topologi jaringan. Percobaan ini berarti korelasi antara koordinat fisik dan koordinat polar virtual jauh berkurang setelah perbaikan yang berulang-ulang, dan banyak kerusakan pada waktu yang sama dapat menyebabkan adanya kekosongan pada jaringan karena pohonnya harus memutari kosongan ini. Bahkan, walaupun adanya tambahan noktah baru, keanehan pada struktur pohon tetap ada.

Untuk memeriksa bahwa algoritma perbaikan dan penambahan cukup ringan untuk dipakai, jumlah noktah yang harus dilabel ulang diukur. Gambar 9 menunjukkan histogram dari jumlah noktah yang harus dilabel ulang setelah tiap kerusakan. 35% dari kerusakan noktah tidak membutuhkan pelabelan ulang untuk memperbaiki pohon. Nilai tengah dari jumlah noktah yang harus dilabel ulang adalah 3, walaupun ada pada beberapa kasus seluruh jaringan harus dilabel ulang.
Gambar 10 menunjukkan histogram dari noktah yang harus dilabel ulang setelah penambahan noktahh. 99% penambahan noktah tidak membutuhkan pelabelan ulang. Noktah terbanyak yang harus dilabel ulang sewaktu penambahan adalah 13.

Untuk aplikasi di mana banyak paket yang disalurkan, dan noktah jarang rusak, lebih baik digunakan algoritma yang cukup berat untuk memastikan penjaluran tetap seefisien mungkin.

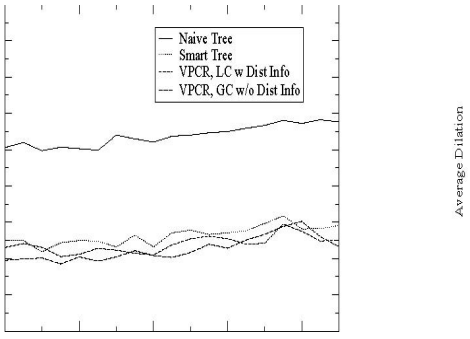
6.7. Susunan Jaringan yang Tidak Biasa

Dalam banyak aplikasi asli, noktah sensor tidak akan diletakkan secara merata pada sebuah daerah. Akan ada tembok yang menghalangi transmisi radio, atau adanya kekosongan yang

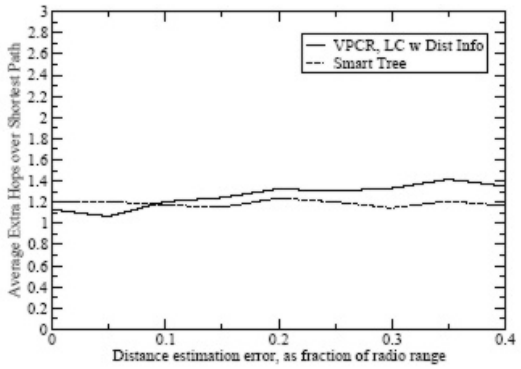


Gambar 11: Kinerja pada kekosongan yang melebar secara radial melintasi jaringan

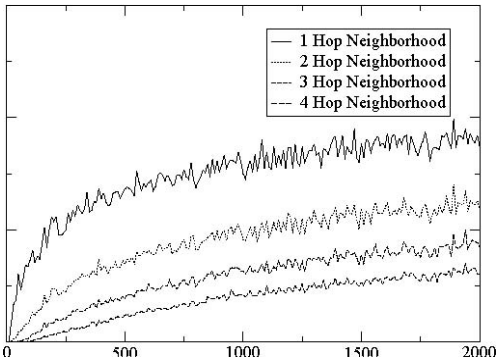
besar antar noktah. Untuk mengetes kinerja VPCR pada hal ini, disimulasikan 2 tipe kekosongan. Tipe pertama adalah kekosongan ada pada sebuah garis dari tengah jaringan (akar) ke arah pinggir. Gambar 11 menunjukkan kinerja VPCR saat adanya kekosongan hingga mencapai pinggir jaringan. Kekosongan tipe ini hanya berpengaruh sedikit pada kinerja VPCR.
Kemudian disimulasikan kekosongan yang paralel dengan sumbu y dari jaringan. Gambar 12 menunjukkan kinerja VPCR saat adanya kekosongan hingga mencapai kedua ujung jaringan. Kinerja sedikit berkurang karena kekosongan tipe ini berpengaruh pada cara pembuatan pohon merentang.



Gambar 12: Kinerja saat adanya kekosongan dari satu ujung jaringan ke ujung lain



Gambar 13: Perbandingan kinerja VPCR dengan infromasi jarak lokal dengan keakuratan informasi tersebut



Gambar 14: Perpindahan saat berbagai ukuran ketetanggaan digunakan

6.8. Efek dari Ketidaktepatan Informasi Jarak saat Penjaluran pada Skema I

Saat menggunakan informasi jarak untuk membuat koordinat sistem lokal (skema I pada 3.3), ketidaktepatan informasi jarak akan mengarah pada ketidaktepatan sistem koordinat, yang akan membuat VPCS kurang sejajar dengan topologi jaringan. Gambar 13 menunjukkan kinerja VPCR saat metode koordinat sistem lokal digunakan dengan tingkat keakuratan jarak yang berbeda-beda. Pecahan kesalahan di sini adalah pecahan dari jarak radio. Karena digunakan jarak radio 40m, kesalahan 10% berarti estimasi jarak akan beragam sampai 4m dari aslinya.

Hasil simulasi ini menunjukkan *greedy forwading* VPCR menguntungkan sampai kesalahan mencapai 10%, lebih dari itu *greedy forwarding* banyak melakukan kesalahan sehingga algoritma *smart tree* menghasilkan jarak yang lebih pendek. Kasus ini tidak berhubungan dengan skema II, di mana tak ada asumsi noktah mengukur jarak antara dia dan lainnya.

6.9. Memilih Ukuran Ketetanggaan

Gambar 14 menunjukkan pengaruh dari ukuran ketetanggaan yang berbeda dalam VPCR. Tiap loncatan tambahan yang ditambah ke ukuran jaringan memberi hasil yang semakin kecil. Walaupun ketetanggaan 2 loncatan jauh lebih efektif daripada ketetanggaan 1 loncatan, ketetanggaan 3 loncatan dan 4 loncatan hanya menghasilkan peningkatan yang relatif kecil.

Karena itu ketetanggaan 2 loncatan merupakan yang paling optimal, karena memberikan peningkatan yang cukup jauh dengan penambahan informasi tersedikit pada tiap noktah.

7. Kesimpulan

Pada makalah ini diajukan pemakaian GEM untuk jaringan *sensor*. GEM dapat digunakan untuk menyelesaikan sejumlah masalah pada jaringan *sensor*, termasuk penyimpanan secara data-sentris dan penjaluran noktah ke noktah secara umum. Telah didemostrasikan kegunaan konsep dari penanaman graf untuk membuat sebuah VPCS dalam sebuah jaringan *sensor*, dan VPCR dapat digunakan untuk penjaluran secara efisien dalam VPCS. VPCR adalah solusi pertama untuk penjaluran noktah ke noktah dan penyimpanan data yang hanya membutuhkan tiap noktah mengetahui label dari tetangganya dan tidak membutuhkan informasi geografik. VPCR efektif dalam jaringan dinamis, berjalan lancar walaupun ada hambatan, serta cukup ringkas dalam hal besar dan kepadatan jaringan. Diharapkan pendekatan penanaman graf ini memperlihatkan sebuah perspektif baru dalam membuat aplikasi jaringan *sensor*.

8. Daftar Pustaka

[1] Arnold L. Rosenberg, Lenwood S. Heath. 2000. Graph separators, with applications. Kluwer Academic/Plenum Publishers.

[2] Brad Karp, H. T. Kung. 2000. GPSR: Greedy Perimeter Stateless Routing for wireless networks. International Conference on Mobile Computing and Networking, pages 243–254.

[3] EECS. 2006. Electrical Engineering and Computer Sciences.

http://www.cs.berkeley.edu/ . Tanggal akses: 30 Desember 2006 pukul 14:00.

[4] Munir, Rinaldi. 2006. Matematika Diskrit. Program Studi Informatika, Institut Teknologi Bandung.

[5] Scott Shenker, Sylvia Ratnasamy, Brad Karp, Ramesh Govindan, Deborah Estrin. 2002. Data-centric storage in sensor nets. HOTNETS.

[6] Srdan Capkun, Maher Hamdi, Jean-Pierre Hubaux. 2001. GPS-free positioning in mobile ad-hoc networks. Hawaii International Conference on Systems Sciences.

[7] Wikipedia. 2006. http://en.wikipedia.org/wiki/Graph_embedding/ . Tanggal akses: 30 Desember 2006 pukul 14:00.

[8] Wolfram Math World. 2006. http://mathworld.wolfram.com/GraphEmbedding.html . Tanggal akeses: 30 Desember 2006 pukul 14:00.