BAB 2
LANDASAN TEORI

2. 1 TEKNOLOGI *WIRELESS*

Teknologi jaringan *wireless* terbentang luas mulai dari komunikasi suara sampai dengan jaringan data, yang mana membolehkan pengguna untuk membangun koneksi *wireless* pada jarak tertentu. Ini termasuk teknologi infrared, frekuensi radio, dan lain sebagainya. Perangkat yang umum digunakan untuk jaringan *wireless* termasuk di dalamnya adalah laptop, PDA, telepon selular, dan lain sebagainya. Teknologi *wireless* ini memiliki beberapa kegunaan, misalnya pengguna bergerak dapat menggunakan telepon selular untuk mengakses *e-mail*, pengguna dengan laptopnya biasa terhubung ke internet ketika berada di bandara, kafe, kereta api, dan tempat umum lainnya.

Karena kemampuannya dalam pengiriman data, manusia di seluruh dunia menggunakannya dalam berbagai aplikasi yang berkaitan dengan jaringan *wireless* dimana salah satunya adalah penggunaan dalam komunikasi data. Untuk mengatasi perkembangan penggunaan komunikasi data dengan jaringan *wireless*, diharapkan jaringan komunikasi di masa mendatang dapat menggunakan jaringan *wireless* dengan lebih baik dalam area lokal maupun area yang luas.

2. 1. 1 Keuntungan Dan Keterbatasan Pada Teknologi *Wireless*

Keuntungan utama dari jaringan *wireless* bagi pengguna adalah tidak diperlukannya ada tempat yang tetap dan memiliki kemampuan untuk berpindah secara bebas di dalam sebuah bangunan atau bahkan di luar bangunan. Dalam segi biaya jaringan *wireless* juga lebih menguntungkan dibandingkan dengan jaringan kawat, sebab pada jaringan kawat dibutuhkan kawat tembaga dalam jumlah yang cukup banyak, sehingga biaya yang dibutuhkan juga lebih besar.

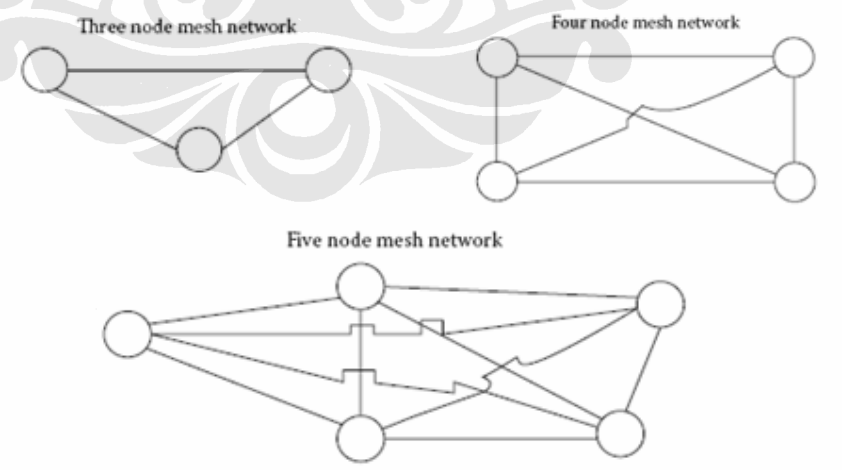
Selain keuntungan-keuntungan tersebut, jaringan *wireless* juga memiliki keterbatasan. Ada beberapa keterbatasan dari penggunaan jaringan *wireless* dibandingan jaringan kawat. Permasalahan utama adalah keterbatasan bandwidth, menurut standar IEEE 802.11a dan 802.11g yang memiliki kecepatan transfer data 54 Mbps. Dengan kata lain, dengan menggunakan jaringan *wireless* jauh lebih

lambat daripada jaringan kawat tembaga yang bekerja pada kecepatan 100 Mbps. Selain itu juga terdapat kekhawatiran pada jangkauan area yang mampu dicakupnya (*coverage*). Misalnya seperti berapa banyak *access point* yang diperlukan untuk satu bangunan. Satu *access point* dapat menjangkau 30 sampai 150 meter dan lebih tergantung pada struktur dari bangunan tersebut.

Namun semua keterbatasan-keterbatasan yang ada ini dapat ditangani dengan perencanaan dan desain yang baik pada jaringan *wireless* yang akan dibangun pada area tersebut.

2. 2 *WIRELESS MESH NETWORK*

Untuk memahami jaringan *mesh*, pertama kita perlu memiliki pemahaman tentang topologi *mesh*. Jika kita memiliki *node* dalam suatu jaringan, dimana istilah "*node*" menunjukkan perangkat komunikasi yang dapat membawa data dari satu *interface* ke *interface* yang lainnya, kemudian kemampuan dari setiap *node* untuk berkomunikasi dengan tiap-tiap *node* yang lain dalam jaringan yang menggambarkan topologi jaringan *mesh*. Kita dapat melihat struktur jaringan *mesh* dengan menyederhanakan jumlah *node* yang ada dalam jaringan dengan nilai n. Gambar 2.1 mengilustrasikan tiga, empat, dan lima buah *node* dalam struktur jaringan *mesh*, setiap node memiliki hubungan komunikasi dengan semua *node* ada dalam jaringan.



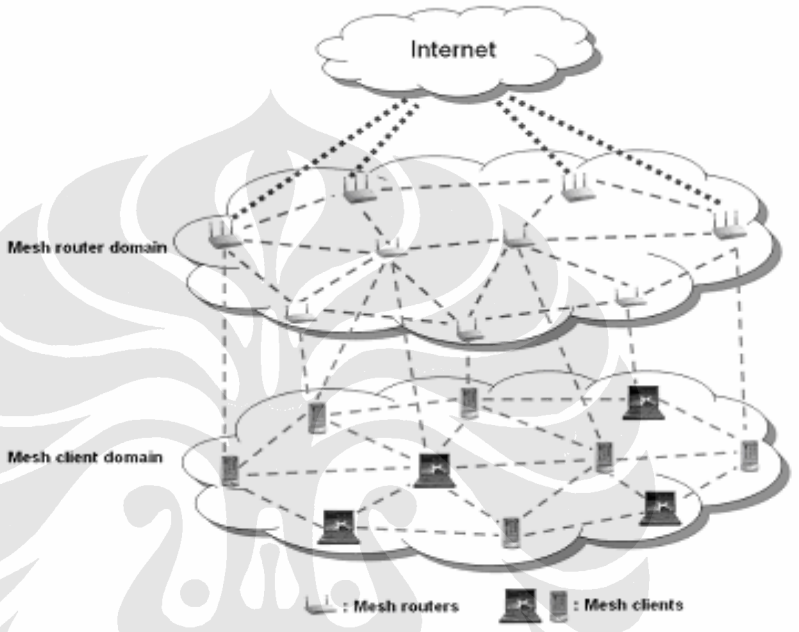
Gambar 2.1. Struktur jaringan *mesh* dengan tiga buah *node*, empat buah *node*, dan lima buah *node* dimana setiap *node* saling terhubung satu sama lain.

Hubungan antara setiap *node* menggambarkan sebuah *link*. Jika diperhatikan jumlah *link* terasosiasi dengan jaringannya, jumlah *link* meningkat seiring dengan meningkatnya jumlah *node*. Tiga buah *link* dibutuhkan untuk menghubungkan jaringan *mesh* yang memiliki tiga buah *node*, enam buah *link* dibutuhkan untuk menghubungkan jaringan *mesh* yang memiliki empat buah *node*, dan sepuluh buah *link* dibutuhkan untuk menghubungkan jaringan *mesh* yang memiliki lima buah *node*. Struktur jaringan *mesh* klasik yang setiap *node*-nya terhubung dengan *node* yang lain menjadi tidak praktis saat jumlah *node* dalam jaringan meningkat, karena *link* yang dibutuhkan untuk menghubungkan setiap *node* akan semakin meningkat pula. [1].

Dalam lingkungan *wireless*, frekuensi radio (pengirim atau penerima) dalam sebuah *node* memiliki kemampuan untuk berkomunikasi dengan *node* lainnya pada jumlah yang tak terbatas. Dengan demikian batasan fisik yang terasosiasi pada konektivitas kawat bukan menjadi persoalan dalam lingkungan *wireless*. Ini berarti proses untuk sebuah *node* berkomunikasi dengan *node* lainnya menjadi praktis dan relatif sederhana, karena sebuah *interface* tunggal dalam bentuk frekuensi radio (pengirim atau penerima) dapat menggantikan *interface-interface* yang dibutuhkan dalam lingkungan kawat. Dengan ketentuan *node* lainnya harus berada pada jarak transmisi agar komunikasi dapat terjadi.

Wireless mesh network merupakan jaringan komunikasi yang memiliki kemampuan dalam mengkonfigurasi dan mengorganisasi dirinya sendiri (*self-configured and self-organized*), dengan kata lain mampu membuat dan menjaga konektivitasnya apabila terjadi kerusakan pada salah satu *node*. Kemampuan ini selain membantu para pengguna untuk dapat selalu terhubung kapan saja dan dimana saja, juga akan membawa keuntungan lain seperti biaya pembuatan yang rendah, kemudahan dalam perawatan jaringan, tingkat *robustness* dan reliabilitas tinggi. *Wireless mesh network* terbentuk dari susunan node radio yang dapat saling terhubung satu sama lain. Node pada *wireless mesh network* terdiri dari *mesh router* dan *mesh client* (Gambar 2.2). Dalam *wireless mesh network* setiap *node* dapat berfungsi sebagai *router* dan *repeater*, yang akan meneruskan data melalui *node* lainnya sampai ke *node* tujuannya.

Wireless mesh network dikembangkan untuk mengantisipasi keterbatasan dan juga meningkatkan performansi dari *wireless adhoc network*, *wireless local area network* (WLAN), dan *wireless metropolitan area network* (WMAN). Dengan berbagai kelebihannya, *wireless mesh network* dapat digunakan untuk menyediakan layanan *wireless* untuk berbagai keperluan dan aplikasi baik untuk kepentingan pribadi, area lokal, kampus ataupun area metropolitan [2].



Gambar 2.2. *Node* pada *wireless mesh network* terdiri atas *mesh routers* dan *mesh clients*.

2. 2. 1 Arsitektur *Wireless Mesh Network*

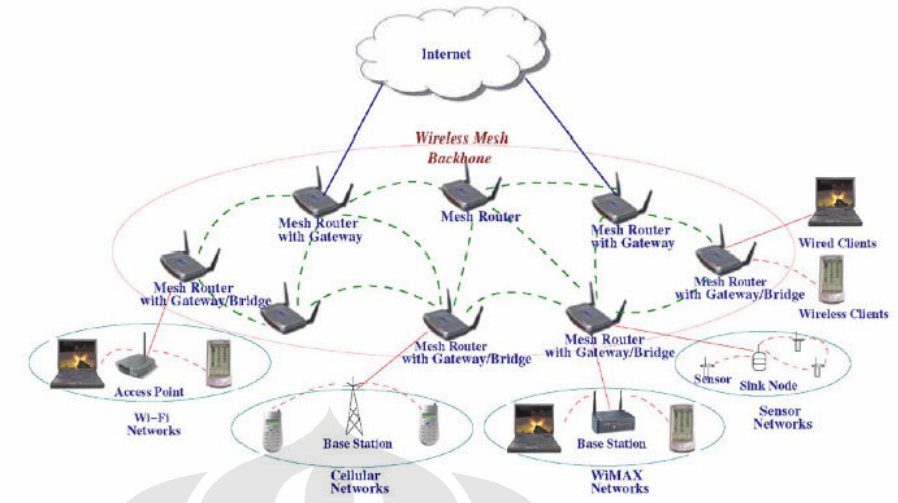
Pada umumnya *wireless mesh network* terdiri dari *mesh router* dan *mesh client* seperti yang terlihat pada Gambar 2.2. *Mesh router* biasanya dilengkapi dengan *multiple wireless interfaces* untuk meningkatkan fleksibilitas dari *mesh network*. Dibandingkan dengan *wireless router* konvensional, *wireless mesh router* dapat memiliki jangkauan area yang sama dengan daya transmisi yang jauh lebih rendah melalui komunikasi *multi-hop*. Walaupun dengan perbedaan tersebut, *wireless router* untuk *mesh* dan konvensional biasanya dibuat berdasarkan *platform hardware* yang sama. *Mesh router* dapat dibangun menggunakan *dedicated computer systems (embedded systems)*, ataupun juga *general purpose systems* (*desktop* PC dan *laptop*).

Untuk *mesh client* juga mempunyai fungsi tertentu pada suatu *mesh network*, dimana suatu *mesh client* juga mempunyai fungsi *router*. Namun *node* ini tidak mempunyai fungsi *gateway* ataupun bridge seperti yang ada pada *mesh router*. Oleh karena itu biasanya *mesh client* cukup dilengkapi dengan satu *wireless interface* sehingga *platform hardware* dan *software* yang dipakaipun menjadi lebih sederhana dibandingkan dengan *mesh router*. *Mesh client* dapat menjadi *node* bergerak yang umumnya dijalankan dengan baterai, sehingga pemakaian daya pada *mesh client* harus dibatasi. *Mesh client* dapat terdiri dari *desktop* PC, *laptop*, PDA, dan lain sebagainya. Berdasarkan fungsionalitas dari *node-node* tersebut arsitektur pada *wireless mesh network* dapat diklasifikasikan kedalam tiga jenis [2], yaitu :

1) *infrastructure wireless mesh network*,
2) *client wireless mesh network*,
3) *hybrid wireless mesh network*.

2. 2. 1. 1 *Infrastructure Wireless Mesh Network*

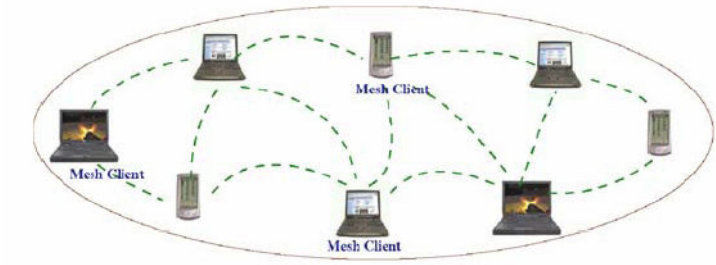
Pada tipe arsitektur ini terdapat beberapa *mesh router* yang membentuk sebuah infrastruktur bagi *client-client* yang terhubung dengannya. Seperti terlihat pada Gambar 2.3 dimana *wireless mesh network* dapat terkoneksi dengan *mesh client* ataupun dengan berbagai jenis teknologi *wireless* lainnya yang dalam hal ini bertindak sebagai *client* dari jaringan infrastruktur yang dibentuk oleh *mesh router*. Dengan kemampuan sebagai *gateway*, *mesh router* juga dapat dihubungkan ke internet. *Mesh router* harus mempunyai kemampuan untuk mengkonfigurasi serta memperbaiki hubungan antara *router* secara mandiri, sehingga hubungan tidak sampai terputus. Untuk *client* konvensional yang memiliki teknologi yang sama dengan *wireless mesh network*, mereka dapat terhubung langsung dengan *mesh router*. Sedangkan untuk *client* yang menggunakan teknologi *wireless* selain *wireless mesh network* maka mereka dapat terhubung dengan *mesh router* melalui sebuah *base station* yang terhubung dengan *mesh router* melalui kabel.



Gambar 2.3. Arsitektur *infrastructure wireless mesh network*.

2. 2. 1. 2 *Client Wireless Mesh Network*

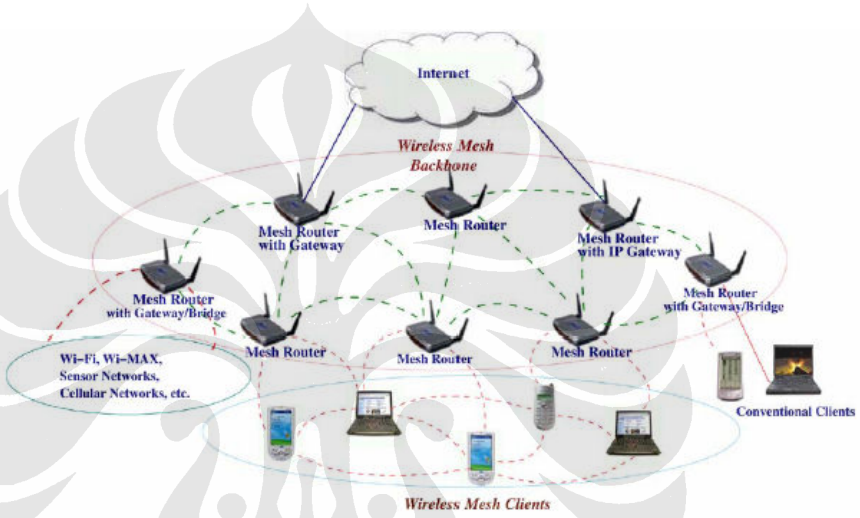
Client wireless mesh network menyediakan jaringan *peer-to-peer* diantara *node mesh client*, dimana *mesh client* dapat bertindak sebagai *host* dan *router*. Pada jenis arsitektur ini, jaringan terbentuk dari sekumpulan *node mesh client* yang dapat melakukan fungsi *routing* dan konfigurasi serta menyediakan aplikasi *end-user* pada pengguna jaringan, seperti yang terlihat pada Gambar 2.4. Tipe arsitektur ini tidak memerlukan *mesh router*, oleh karena itu tingkat mobilitasnya menjadi lebih tinggi bila dibandingkan dengan tipe arsitektur *infrastructure wireless mesh network*. Pada *client wireless mesh network*, paket yang dikirimkan ke suatu *node* tujuan akan melalui serangakaian lompatan (*hops*) melalui beberapa *node* untuk mencapai tujuan. Oleh karena itu, apabila dibandingkan dengan tipe *infrastructure wireless mesh network*, maka *node mesh client* pada tipe ini memerlukan kebutuhan akan aplikasi *end-user* yang lebih tinggi karena harus memiliki kemampuan *routing* serta konfigurasi sendiri.



Gambar 2.4. Arsitektur *client wireless mesh network*.

2. 2. 1. 3 *Hybrid Wireless Mesh Network*

Tipe ini merupakan bentuk gabungan dari dua tipe arsitektur lainnya yaitu *client* dan *Infrasructure wireless mesh network*. Seperti yang terlihat dalam Gambar 2.5, *mesh client* dapat terhubung pada jaringan melalui *mesh router* sekaligus tetap berhubungan langsung dengan *mesh client* lainnya. Selain itu jaringan *infrastructure* dari *mesh router* juga dapat terhubung dengan jaringan teknologi *wireless* lainnya.



Gambar 2.5. Arsitektur *hybrid wireless mesh network*.

2. 2. 2 Karakteristik *Wireless Mesh Network*

Wireless mesh network memiliki beberapa karakteristik umum yang sangat mempengaruhi kinerjanya [2], seperti berikut ini.

1. *Multi-hop wireless network*.

   Karakteristik ini berguna untuk meningkatkan area jangkauan dari jaringan tanpa harus mengorbankan kapasitas kanal. Selain itu berguna pula untuk menyediakan bentuk layanan *Non Line-of-Sight* (NLOS).

2. Kemampuan *self-forming, self-healing, self-organizing* serta mendukung *ad-hoc networking*.

   Karakteristik ini menambah performansi dari *wireless mesh network* karena membawa sifat fleksibel dalam jaringan, pembuatan dan konfigurasi yang mudah, serta *fault tolerance*.

3. Tingkat mobilitas tergantung dari jenis *node*.

   Mesh router biasanya memiliki tingkat mobilitas yang lebih rendah dibanding dengan sebuah *mesh client* yang dapat bersifat *fixed* maupun *mobile*.

4. Dapat mengakses ke berbagai jenis teknologi jaringan lainnya.

   Dalam *wireless mesh network* sebuah jaringan infrastruktur dari *mesh router* dapat terhubung baik ke *mesh client*, internet maupun dengan berbagai jenis teknologi jaringan lainnya.

5. Dependensi terhadap pemakaian daya tergantung dari jenis *node*.

   Mesh router tidak memiliki suatu batasan tertentu dalam pemakaian daya karena biasanya bersifat *fixed* dan terhubung langsung dengan sumber daya. Sedangkan bagi *mesh client* kemungkinan memerlukan suatu protokol tertentu untuk mengatur pemakaian daya karena seringkali *mesh client* bersifat *mobile*.

2. 2. 3 Protokol Dalam *Wireless Mesh Network*

Protokol-protokol yang digunakan dalam suatu sistem *wireless mesh network* dapat dibagi menjadi beberapa bagian menurut lapisan (*layer*), mulai dari *physical layer* hingga *application layer*.

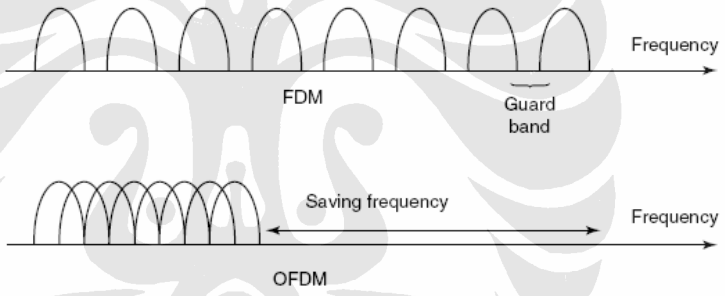
2. 2. 3. 1 *Physical Layer Protocol*

Permasalahan utama pada *physical layer* untuk *wireless mesh network* terletak pada pemilihan teknologi *radio* yang tepat. Pemilihan teknologi *radio* dapat didasarkan pada pertimbangan teknologi dan pertimbangan ekonomi.

Pertimbangan teknologi mengacu pada efisiensi spektrum, *physical layer data rate*, dan kemampuan beroperasi pada saat terjadi interferensi. Jenis *protocol* yang paling banyak digunakan untuk physical *layer* dalam sebuah sistem *wireless mesh network* saat ini adalah *Ortoghonal Frequency Division Multiplexing* (OFDM) yang memungkinkan pengiriman data berkecepatan tinggi dalam lingkungan yang *mobile*. OFDM memiliki *physical layer data rate* maksimum sebesar 54 Mbps, pada jaringan yang padat dengan tingkat interferensi yang tinggi hal ini mungkin tidak dapat tercapai. OFDM merupakan perkembangan dari

sistem *Frequency Division Multiplexing* (FDM) dimana yang dapat mengirimkan berbagai sinyal secara simultan dengan cara membagi-bagi mereka ke dalam beberapa *band* frekuensi yang berbeda (*subcarrier*).

Dalam FDM terdapat *guard band* yang berfungsi untuk mengurangi interferensi antar frekuensi yang berbeda, hal ini menyebabkan adanya bandwidth yang terbuang percuma, sedangkan OFDM menerapkan sistem yang lebih efisien dimana OFDM tidak lagi menggunakan *guard band* untuk mengurangi interferensi melainkan dengan memodulasi sinyal secara *orthogonal* (lihat Gambar 2.6). Dengan cara ini OFDM dapat menyediakan lebih banyak ruang pada bandwidth untuk dipakai. OFDM menggunakan *Fast Fourier Transform* (FFT) yang dan *Inverse* FFT untuk mengubah serial data ke dalam bentuk *multiple channel*. OFDM mempunyai 256 *subchannel (carrie*r), dan sinyal asli akan dibagi-bagi menjadi 256 *subcarrier* dan ditransmisikan secara paralel.



Gambar 2.6. Perbandingan antara FDM dan OFDM.

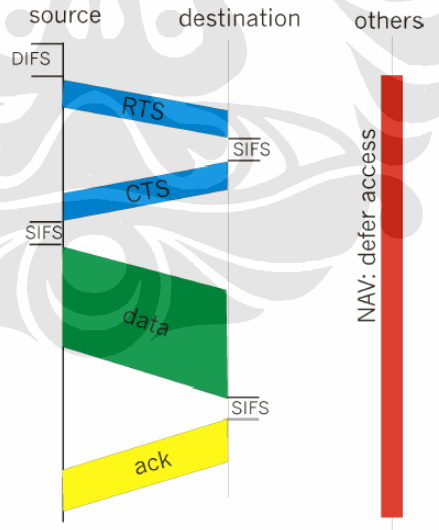
Pertimbangan ekonomi atau sosial mengarah pada harga peralatan yang murah dan dapat memberikan kemudahan bagi pengguna. Sebagai contoh adalah bukti suksesnya IEEE 802.11b berbasis *wireless mesh network* dimana murahnya kartu *interface* jaringan memberikan kontribusi atas suksesnya perkembangan *wireless mesh network.* Oleh sebab itu pada saat pemilihan teknologi *physical layer*, desain jaringan harus memperhatikan aplikasi dan kebutuhan penggunanya.

Namun dalam perkembangannya, lebih banyak lagi protokol yang dapat digunakan pada *physical layer* dari sebuah sistem *wireless mesh network* seperti *Code Division Multiple Access* (CDMA), *Ultra Wideband Access* (UWB) ataupun *Multiple Input Multiple Output* (MIMO) yang dapat digunakan sesuai dengan kebutuhan.

2. 2. 3. 2 *Medium Access Control (MAC) Layer Protocol*

Protokol yang dikembangkan untuk digunakan pada MAC *layer* untuk sebuah sistem *wireless mesh networking* adalah *protocol* MAC berbasis IEEE 802.11 yaitu *Carrier Sense Multiple Access* with *Collision Avoidance* (CSMA/CA) dengan paket kontrol RTS/CTS (*Ready to Send/Clear to Send*). *Frame* yang akan dikirimkan membawa informasi waktu perkiraan untuk *frame* tersebut agar dapat sampai pada tujuan. Waktu perkiraan inilah yang akan dipakai oleh *node* yang lain dalam menentukan waktu minimum pemakaian jalur atau *network location vector* (NAV).

Dengan menggunakan paket kontrol RTS/CTS, saat pengirim akan mengirimkan *frame* maka ia akan mengirimkan paket RTS yang berisi informasi waktu yang dibutuhkan untuk mengirimkan data dan sinyal *Acknowledgement* (ACK). Penerima yang menerima paket RTS tersebut akan mengirimkan paket CTS sebagai balasan yang menandakan bahwa pengirim dapat mulai mengirimkan data. *Node* lain yang mengetahui adanya sinyal RTS/CTS ini dapat menunggu hingga waktu pengiriman selesai untuk menghindari adanya *collision*. Hal tersebut diperlihatkan pada Gambar 2.7 dibawah ini.



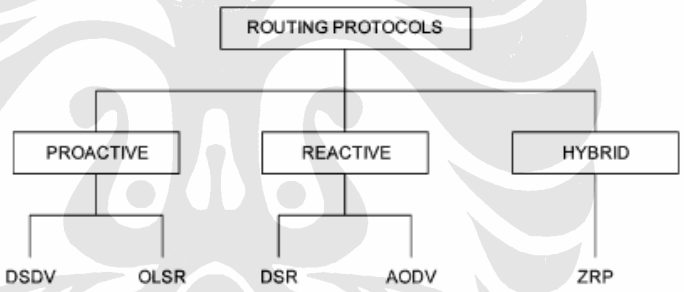
Gambar 2.7. CSMA/CA dengan paket kontrol RTS/CTS [3].

2. 2. 3. 3 *Network Layer Protocol*

Protokol yang dipakai dalam *network layer* pada suatu *wireless mesh network* banyak mengadaptasi *routing protocol* yang digunakan dalam *wireless*

ad-hoc network yang dibagi menjadi tipe *proactive, reactive* dan *hybrid* (lihat Gambar 2.8). Protokol *routing* tipe *proactive* mengirimkan informasi seperti keterangan *node* tetangga, rute dan lain-lain secara *broadcast* dalam periode tertentu. Hal ini memungkinkan waktu *set-up* yang cepat, namun meningkatkan penggunaan *overhead*. Contoh dari tipe *protocol* ini adalah *Destination Sequenced Distance Vector* (DSDV) dan *Optimized Link State Routing* (OLSR).

Pada protokol *routing* tipe *reactive* rute hanya dibangun berdasarkan adanya permintaan, sehingga mengurangi pemakaian *overhead* pemilihan rute namun menimbulkan *delay* yang cukup besar pada saat pengiriman *frame* pertama. Contoh dari protokol *routing* ini adalah *Dynamic Source Routing* (DSR) dan *Ad-hoc On-demand Distance Vector* (AODV). Sedangkan untuk protokol *routing* tipe *hybrid* merupakan gabungan dari kedua tipe tersebut diatas. Contoh dari protokol *routing* ini adalah *Zone Routing Protocol* (ZRP).



Gambar 2.8. Klasifikasi protokol *routing* dalam *wireless mesh network*.

2. 2. 3. 4 *Transport Layer Protocol*

Protokol TCP yang selama ini diadaptasi untuk *wireless mesh network* walaupun tetap digunakan namun tidak memberikan hasil yang maksimal. Kekurangan dari TCP dalam *wireless mesh network* karena pada TCP yang *standard* tidak membedakan *loss* yang diakibatkan oleh *congestion* maupun *non-congestion*, sehingga bila terjadi *non-congestion loss* maka troughput jaringanpun menurun drastis. Selain itu apabila terjadi perbaikan jalur setelah mengalami kerusakan, protokol TCP konvensional tidak dapat diperbaiki dengan cepat. Untuk itu menutupi kekurangan-kekurangan ini, sedang dikembangkan *protocol-protocol* berbasis TCP yang dapat dipakai untuk *wireless mesh network*.

2. 2. 3. 5 *Application Layer Protocol*

Hampir semua dari *protocol-protocol* yang bekerja pada *layer* 7 pada OSI *layer* dapat digunakan dengan baik pada *wireless mesh network*. Terdapat tiga fungsi utama aplikasi yang dapat digunakan dalam suatu *wireless mesh network* yaitu akses internet, penyimpanan informasi terdistribusi, pertukaran informasi antara jaringan teknologi *wireless* lainnya.

2. 2. 4 *Routing* Dalam *Wireless Mesh Network*

Wireless mesh network merupakan *trend* baru dalam komunikasi *wireless* yang menjanjikan flesibilitas tinggi, keandalan, dan perfoma diatas konvensional *wireless local network* (WLAN). *Wireless mesh network* dan jaringan *ad-hoc* bergerak menggunakan konsep komunikasi yang sama antara *node*-nya, namun menitikberatkan pada aspek yang berbeda. Jaringan *ad-hoc* bergerak (*mobile ad-hoc network* atau MANET) memiliki latar belakang akademis dan memusatkan pada perangkat pengguna, mobilitas, dan kemampuan *ad-hoc*. WMN memiliki latar belakang bisnis dan memusatkan pada perangkat statis (biasanya infrastruktur), keandalan, dan kapasitas jaringan. Fungsi utama dari jaringan *multihop ad-hoc* sama seperti *wireless mesh network* yaitu kemampuan *routing*-nya. Protokol *routing* menyediakan jalur yang dibutuhkan melalui *wireless mesh network*, sehingga *node* dapat terhubung dengan baik atau memiliki jalur terbaik pada *multiple wireless hop*. Protokol *routing* harus memperhitungkan kesulitan pada lingkungan *radio* dengan keadaan yang sering kali berubah dan harus mendukung komunikasi yang handal serta yang efisien pada jaringan *mesh*.

Karena *wireless mesh network* memiliki fitur serupa dengan jaringan *ad-hoc wireless*, protokol *routing* yang dikembangkan untuk MANET dapat diterapkan pada *wireless mesh network*. Sebagai contoh, *microsoft mesh network* yang dibuat berdasarkan pada *Dynamic Source Routing* (DSR), dan masih banyak perusahaan lainnya seperti penggunaan *Ad-hoc On-demand Distance Vector* (AODV).

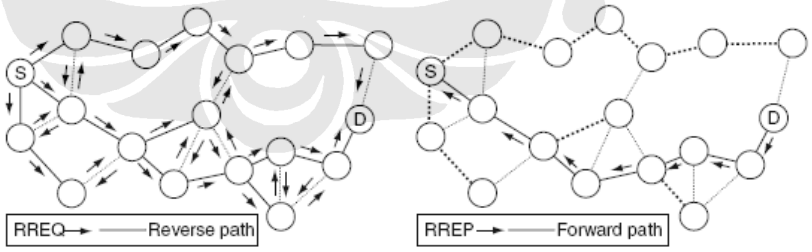
2. 2. 5 Protokol *Routing Ad hoc On-demand Distance Vector (AODV)*

AODV adalah protokol *routing* yang popular dari MANET, yang merupakan protokol *routing reactive*. Pada protocol *routing* ini jalur diatur

berdasarkan permintaan, dan hanya jalur yang aktif yang akan dipertahankan. AODV sudah distandarisasi oleh IETF dalam percobaan RFC 3561 [4].

AODV menggunakan mekanisme permintaan-balasan (*request-reply*) sederhana untuk menemukan jalurnya. Saat jalur ke tujuan tidak diketahui, AODV membuat paket permintaan jalur dan menyiarkannya ke tetangganya. Pesan permintaan jalur berisi identitas sumber (*source ID*), identitas tujuan (*destination ID*), urutan nomor sumber (*source sequence number*), urutan nomor tujuan (*destination sequence number*), penghitung *hop* (*hop count*), dan identitas penyiaran (*broadcast ID*) [5].

Pada saat ada permintaan jalur dari sebuah *node*, maka *node* tersebut akan memancarkan (*broadcast*) sebuah paket untuk meminta rute. Setiap *node* lain yang berada pada jaringan akan meneruskan paket tersebut serta mencatat dari mana paket tersebut berasal dan berapa *hop* yang telah dilalui. Apabila paket permintaan tersebut telah sampai pada *node* yang dituju atau sampai pada *node* yang mengetahui jalur menuju *node* yang dituju maka *node* tersebut akan mengirimkan paket balasan berisi informasi rute yang diperlukan. *Node* yang ingin mengirimkan paket akan menyeleksi paket balasan yang masuk dengan melihat jumlah *hop* terkecil, atau bila memiliki jumlah yang sama maka yang dipilih adalah paket yang paling dahulu datang. Hal ini dapat dilihat pada Gambar 2.9. Apabila ada kerusakan rute pada saat pengiriman maka akan ada pesan kerusakan yang dikirimkan ke *node* pengirim.



Gambar 2.9. Pencarian rute AODV: disebelah kiri adalah rute permintaan (*route request*) dan disebelah kanan adalah rute balasan (*route reply*) [6].

Setiap *node* mengamati status setiap *link*-nya, dan saat terjadi perubahan koneksi pada *link* atau apabila ada kerusakan rute pada saat pengiriman *node* akan membuat pesan kesalahan rute dan memberitahukan anggotanya tentang rute yang

tidak dapat dicapai. AODV mengandalkan skema *layer medium access control* (MAC) atau menggunakan paket *beacon* dengan *periodic interval* untuk mengetahui status koneksi dengan tetangganya.

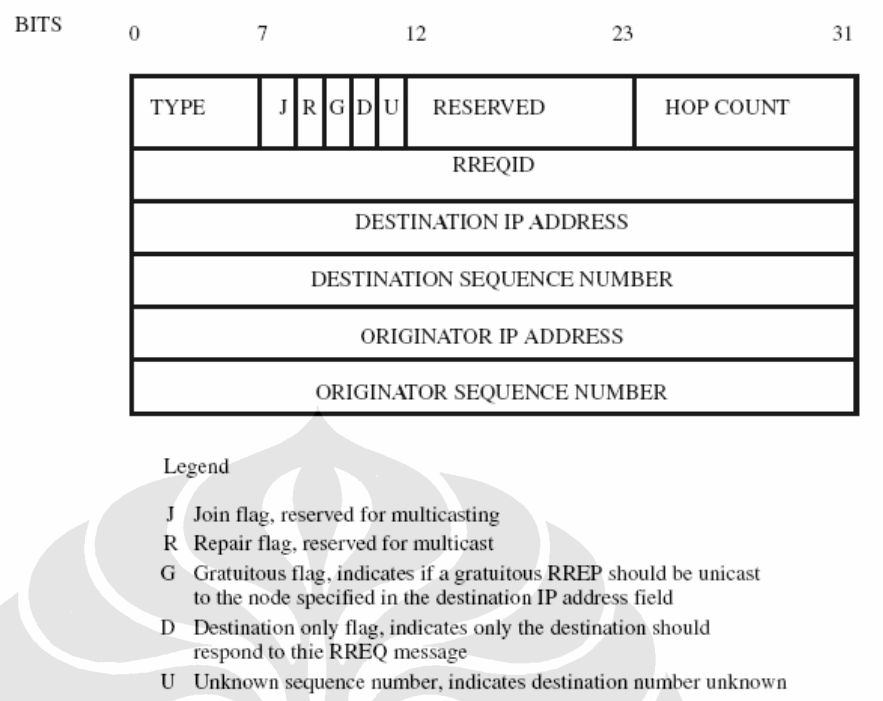
Keunggulan dari AODV adalah membuat rute hanya dalam daerah sebarannya, yang mengurangi pesan kontrol periodik berlebih yang terasosiasi dengan protokol *routing proactive*. Kekurangan dari AODV adalah terletak pada *node* penghubung (*hop*) dapat mengarahkan ke rute yang kurang efisien jika urutan nomor sumber berumur tua (rendah) dan *node* penghubung memiliki urutan nomor tujuan yang lebih tinggi. Juga rute paket balasan (RREP) yang banyak dalam menanggapi satu rute paket permintaan (RREQ) dapat meyebabkan kontrol berlebih yang tinggi [5].

Dalam proses pencarian jalurnya AODV menggunakan mekanisme format pesan (*massage formats*) dalam bentuk paket-paket. Format pesan tersebut adalah *Route Request* (RREQ), *Route Reply* (RREP), *Route Error* (RERR), dan HELLO [1] [4].

2. 2. 5. 1 *Route Request (RREQ)*

Pada saat sebuah *node* ingin berkomunikasi dengan *node* lain, *node* tersebut akan memancarkan (*broadcast*) sebuah paket *route request* (RREQ) ke *node* tetangganya yang akan diteruskan ke *node* lainnya sampai ke *node* tujuannya. Paket RREQ yang dikirimkan suatu *node* ke *node* yang lain berisi RREQ ID, *destination IP address*, *destination sequence number*, *originator IP address*, dan *originator sequence number* (Gambar 2.10).

Pada saat proses pengiriman, setiap *node* yang menerima paket RREQ akan memeriksa *Destination IP Addres*-nya apakah *node* tersebut adalah *node* tujuan. Jika *node* tersebut adalah *node* tujuan maka penjaluran berhenti sampai di *node* tersebut, dan sebaliknya apabila *node* tersebut bukanlah *node* tujuan, ia akan meneruskan paket RREQ tersebut ke *node* tetangganya. Proses ini akan berulang terus hingga paket RREQ menemui *node* tujuan atau apabila jumlah *hop count* telah memenuhi batas. *Sequence number* digunakan untuk menunjukkan tingkat *up-to-date* informasi pada paket tersebut, makin besar nilai *sequence number* maka makin baru informasi yang ada pada paket tersebut.

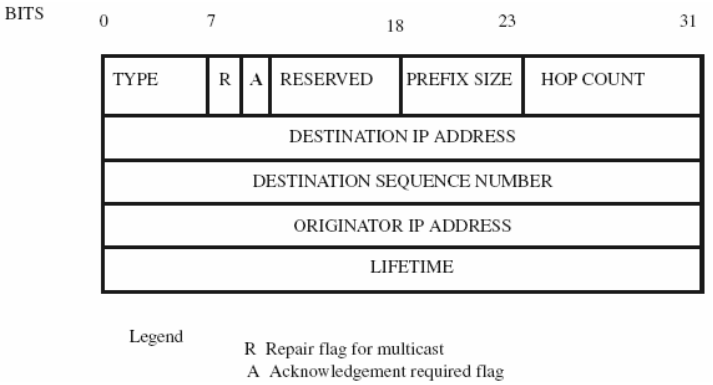


Gambar 2.10. Format paket pesan *route request* (RREQ) [1].

2. 2. 5. 2 *Route Reply (RREP)*

Setelah paket RREQ sampai pada *node* tujuannya akan membentuk jalur yang menuju ke *node* yang mengirimnya, jalur ini akan digunakan kembali untuk mengirim paket RREP. Paket RREP merupakan paket yang dikirimkan oleh *node* tujuan atau *node* yang mempunyai informasi jalur menuju *node* tujuan. *Node* yang mengirimkan paket akan menyeleksi paket balasan yang masuk dengan melihat jumlah *hop* terkecil, atau bila memiliki jumlah yang sama maka yang dipilih adalah paket yang paling dahulu datang.

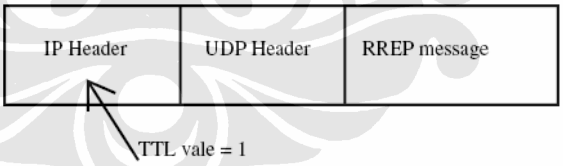
Pada Gambar 2.11 paket format RREP, terdapat bit A yang dapat diset apabila *node* tujuan memerlukan sinyal *Acknowledgement* dari RREP yang telah dikirimkan. Ada dua jenis *field* yang berbeda dari format paket RREP yaitu *Prefix Size* dan *Lifetime*. *Prefix Size* apabila diset dengan nilai diluar nol maka akan menyatakan bahwa *node* selanjutnya akan dapat dipakai oleh setiap *node* yang memiliki rute yang sama dengan *node* tujuan. Sedangkan *Lifetime* yang terdiri dari 32 bit menunjukkan waktu *valid* dari suatu rute (dalam *millisecond*) bagi setiap *node* yang menerima paket RREP.



Gambar 2.11. Format paket pesan route reply (RREP) [1].

2. 2. 5. 3 *Pesan HELLO*

Sebuah *node* dapat memeriksa informasi konektifitas dengan *node* tetangganya dengan cara memancarkan (*broadcasting*) paket pesan *HELLO* secara periodik selama jalur tersebut merupakan jalur yang aktif. Paket *HELLO* merupakan sebuah paket RREP dengan nilai *Time To Live* (TTL) adalah 1 yang ditentukan pada *IP header* dari pesan. Dalam paket RREP *destination IP address* diisi dengan *IP address* dari *node* yang mengirimkan pesan *HELLO*. Kemudian dengan nilai *field destination sequence number* diisi dengan nilai *sequence* terakhir dari *node* tersebut serta nilai *field hop count* diset dengan nilai 0. Gambar 2.12 menunjukkan format dari paket pesan *HELLO*.

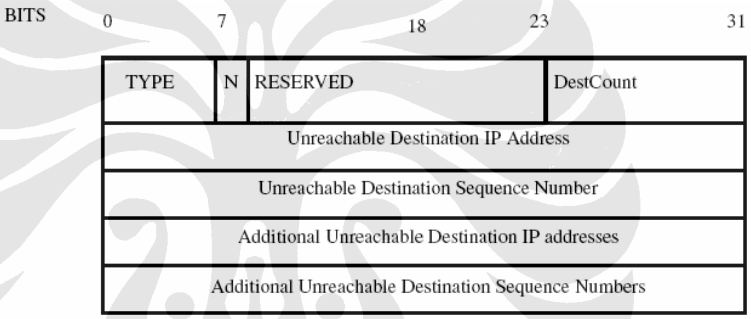


Gambar 2.12. Format paket pesan *HELLO* [1].

Jika suatu *node* tidak dapat menerima paket *HELLO* dari *node* tetangganya maka tabel rute (*routing table*) yang ada tersebut akan dihapus datanya dari *node* tetangga tersebut. Dan apabila hal tersebut terjadi pada saat pembuatan hubungan maka *node* tersebut akan mengirimkan paket RRER.

2. 2. 5. 4 *Route Error (RERR)*

Paket RRER memungkinkan AODV untuk menyesuaikan rute saat *node* tersebut berpindah tempat ataupun terjadi kerusakan pada suatu rute. Saat suatu *node* tidak menerima lagi paket *HELLO* dari *node* tetangganya, maka *node* tersebut akan menghapus rute dari *node* yang tidak mengirim paket *HELLO* itu dari tabel jalurnya. Kemudian *node* tersebut akan mengirimkan paket pesan RRER ke *node* tetangganya yang menggunakan *node* tersebut sebagai jalur ke *node* yang tidak mengirim paket *HELLO* tadi dan diteruskan ke *node-node* yang lainnya. Setelah RRER diterima maka jalur yang ada pada setiap *node* untuk menuju pada *node* yang tidak mengirim paket *HELLO* tersebut akan dihapus dari tabel jalurnya. Gambar 2.13 memperlihatkan format untuk paket pesan RRER.



Gambar 2.13. Format paket pesan RRER [1].

Dest count field menandakan jumlah *node* tujuan yang tidak dapat dicapai yang termasuk di dalam paket tersebut. *Dest count field* ini harus memiliki nilai minimal adalah 1. Apabila suatu *node* menerima paket RRER maka *node* tersebut akan menghapus semua informasi rute pada tabel jalur yang berkaitan dengan *node* tesebut.

2. 2. 5. 5 *Informasi Yang Terdapat Pada Tabel Routing*

Setiap tabel *routing* yang terdapat pada AODV terdiri dari informasi sebagai berikut ini [4].

1. *Destination IP address*

   Bagian ini berisi informasi alamat IP (internet protokol) dari *node* yang merupakan tujuan *routing*-nya.

2. *Destination sequence number*

   Bagian ini berisi informasi yang menunjukan nomor urut *routing* dari *node* tujuan pada tabel *routing*.

3. *Valid destination sequence number flag*

   Bagian ini berisi informasi yang menunjukan status *flag* pada *destination sequence number*.

4. *Interface*

   Bagian ini berisi informasi tentang *interface* jaringan yang digunakan untuk mencapai *node* tujuan.

5. *Hop count*

   Bagian ini berisi informasi yang menunjukan keterangan jumlah *hop* yang akan dilalui untuk mencapai *node* tujuannya.

6. *Next hop*

   Bagian ini berisi informasi yang menunjukan keterangan *hop* selanjutnya yang akan dilalui saat menuju ke *node* tujuan.

7. *List of precursors*

   Bagian ini berisi daftar tentang *node-node* tetangganya yang dapat memberikan jalur untuk meneruskan paket.

8. *Lifetime*

   Bagian ini berisi informasi yang menunjukan waktu berakhirnya atau waktu penghapusan jalur tersebut.

9. *State*

   Bagian ini berisi informasi yang menunjukan status jalur tersebut, seperti dapat digunakan (*valid*), tidak dapat digunakan (*invalid*), perlu perbaikan (*repairable*), dalam perbaikan (*being repaired*).

2. 3 *AD HOC ON-DEMAND DISTANCE VECTOR–UPPSALA UNVERSITY* (AODV-UU)

AODV-UU adalah implementasi protokol *routing* AODV pada Linux, yang dikembangkan oleh Universitas Uppsala, Swedia. AODV-UU ini berjalan sebagai *user-space daemon* dan memelihara *kernel* tabel *routing*. AODV-UU ditulis

dengan bahasa pemrograman C dan telah di terbitkan dibawah GNU *General Public License* (GPL).

AODV-UU mengimplementasikan hampir semua hal pada AODV. Salah satu tujuan dari AODV-UU adalah untuk memenuhi implementasi dari AODV yang sesuai dengan *draft* terakhir dan tujuan ini menopang pengembangan perangkat lunak yang berkelanjutan.

Kebutuhan sistem dari AODV-UU lebih sederhana, dengan menggunakan *distro* linux versi kernel 2.4.x bersama dengan menggunakan kartu jaringan *wireless* (ini juga memungkinkan digunakan pada jaringan kawat). Sebagai tambahan, AODV-UU dapat di *cross-compile* untuk digunakan pada *platform* ARM, sehingga AODV-UU ini dapat digunakan pada PDA yang popular seperti COMPAQ iPAQ dan Sharp Zaurus [7].

2. 3. 1 Konfigurasi

AODV-UU menyediakan banyak pilihan perintah untuk mengatur operasinya. Pilihan ini disediakan sebagai parameter pada *command line* untuk aodvd *routing daemon* [7]. Perintah pilihan yang tersedia adalah sebagai berikut:

1) *daemon mode* (-d, --*daemon*): menjalankannya di *backgound*,
2) *force gratuitous* (-g, --*force-gratuitous*): memaksa *gratuitous flag* untuk diterapkan pada semua RREQ,
3) *help* (-h, --*help*): menampilkan informasi bantuan,
4) *interfac*e (-i, --*interface*): menspesifikasikan *interface* jaringan mana yang akan digunakan pada AODV-UU. *Default*-nya adalah *interface* awal jaringan *wireless* tersebut,
5) *hELLO jittering* (-j, --*hello-jitter*): menonaktifkan *jitter* dari pesan *HELLO*.
6) *logging* (-l, --*log*): mengaktifkan *logging* ke AODV-UU *logfile*,
7) *routing table logging* (-r N, --*log*-rt-*table* N): mencatat isi dari tabel *routing* ke tabel *routing logfile* setiap N detik,
8) *N HELLOs* (-n N, --n-*hellos* N): memerlukan pesan *N HELLO* untuk diterima dari sebuah *node* sebelum ditetapkan sebagai *node* tetangganya,
9) *uni-directional hack* (-u, --*unidir-hack*): memperbolehkan pendeteksian dan penghindaran dari *link uni-directional*. Ini masih fitur percobaan,

10) *gateway mode* (-w, --*gateway-mode*): memperbolehkan dukungan internet *gateway*. Ini juga masih fitur percobaan,
11) *disabling of expanding ring search* (-x, --*no-expanding-ring*): menonaktifkan perluasan daerah pencarian untuk RREQ, yang normalnya digunakan untuk membatasi penyebaran RREQ didalam jaringan,
12) *no wait-on-reboot* (-D, --*no-worb*); menonaktifkan penundaan selama 15 detik *wait-on-reboot* pada saat *startup*,
13) *version information* (-V, --*version*); menampilkan informasi versi dan *copyright*.

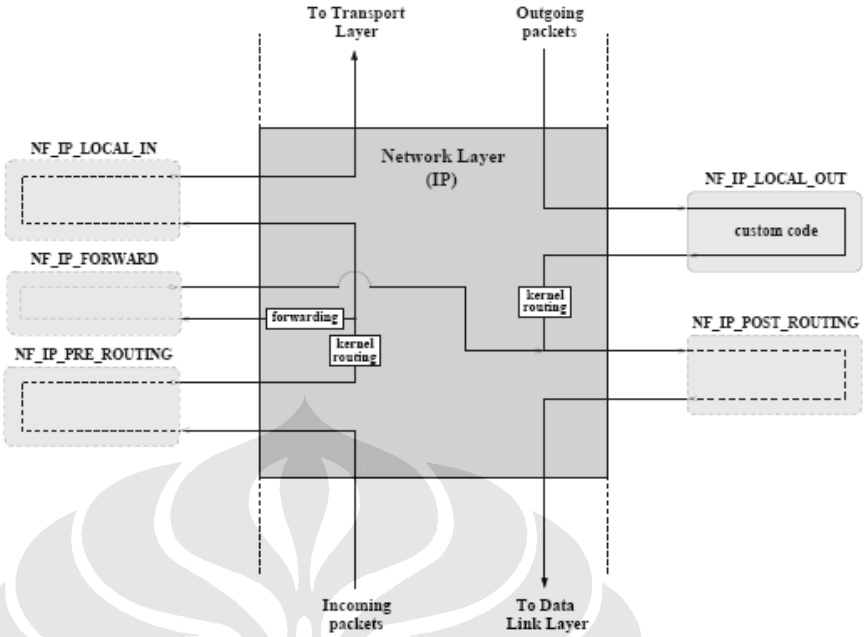
2. 3. 2 Interaksi Dengan Internet Protokol

Mengingat AODV merupakan protokol yang reaktif, maka penentuan rute dilakukan berdasar pada permintaan. Hal ini membutuhkan implementasi protokol *routing* yang mampu meneruskan permintaan (*request*) ke tujuan saat rute tujuan tersebut tidak ada. Penghapusan rute pada tabel *routing* yang telah lewat batas waktunya membutuhkan dukungan dari setiap *host* untuk pemantauan paketnya.

Implementasi AODV sebelumnya seperti *Multicast ad-hoc* (Mad-hoc) AODV tidak dapat melakukan hal tersebut diatas. Hal ini menghambat pengoperasian pada protokol yang berorientasi koneksi seperti TCP, dimana awal paket sangat vital untuk *setup* koneksi. Sejak itu, penanganan paket pada linux telah mengalami perkembang yang sangat cepat. Khususnya *software* yang disebut Netfilter telah dikembangkan, yang dapat memberikan penanganan paket yang sangat fleksibel. AODV-UU menggunakan Netfilter untuk pemprosesan semua paket dan keperluan modifikasi [7].

2. 3. 3 Netfilter *Framework*

Netfilter adalah sebuah *framework kernel* linux yang digunakan untuk mengelola paket. Untuk setiap protokol jaringan, beberapa *hook* diberikan. *Hook* ini berhubungan dengan tempat-tempat yang telah ditentukan dengan baik didalam *stack* protokol dan memperbolehkan memasukkan kode yang telah disesuaikan untuk pengelolaan paket dalam bentuk modul *kernel* (lihat Gambar 2.14).



Gambar 2.14. Netfilter *hooks* untuk IP. Pengiriman paket pada *hook* ini dapat diterima dan dimodifikasi dengan kode *segment* yang telah disesuaikan (modul *kernel*).

Setiap paket yang datang pada *hook* akan dihantarkan ke dalam sebuah *code segment* yang telah dicatat pada *hook* tersebut. Hal ini memungkinkan paket untuk dirubah, dibuang atau dirutekan ulang oleh kode *segment* modifikasi tersebut. Ketika sebuah paket diproses (dan mungkin dirubah), sebuah *verdict* (keputusan) harus dikembalikan ke *Netfilter*. *Verdict* ini menginstruksikan *Netfilter* untuk melakukan beberapa aksi pada paket yang berhubungan, seperti untuk membuang paket atau membiarkan untuk melanjutkan perjalanannya melalui *stack* protokol.

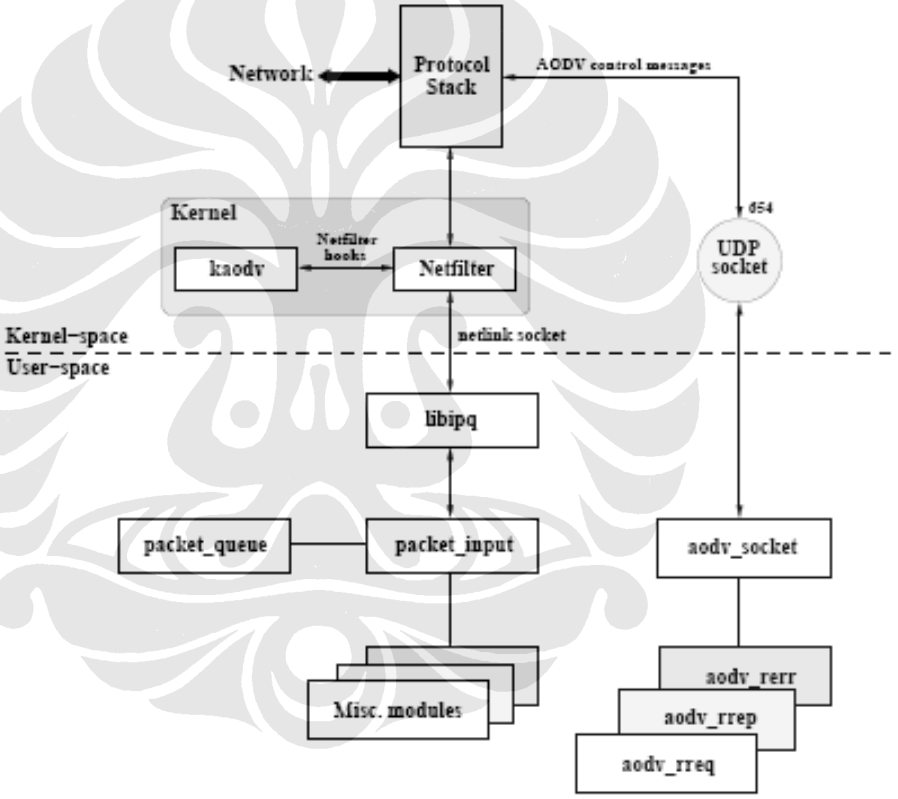
Netfilter juga menawarkan kemampuan untuk memproses paket dalam *user-space*. Dengan mengembalikan antrian keputusan khusus (*special queue verdict)*, paket diantrikan pada *kernel* dan informasi tentang paket dikirim ke *user-space* melalui *netlink socket*. Paket yang antri akan tetap diantrikan sampai aplikasi *user-space* (disisi lain dari soket) mengembalikan *verdict* menandakan aksi yang diinginkan untuk paket-paket ini.

Didalam AODV-UU, sebuah komponen modul *kernel* mendengarkan secara konstan kedua paket *inbound* dan *outbound*, dengan mendaftarkan dirinya pada *netfilter*. Paket yang diantrikan sesuai kebutuhan memperbolehkan pemrosesan

user-space dilakukan oleh AODV-UU *routing daemon*. Hal ini memperbolehkan penetuan rute sesuai permintaan dari operasi AODV menjadi nyata. Pada akhirnya, *netfilter* memperbolehkan implementasi secara mandiri dari perubahan *kernel*. Ini adalah keuntungan besar didalam pemeliharaan *software* [7].

2. 3. 4 Penanganan Paket

Penanganan paket (*packet handling*) mampu membedakan antara paket data dan pesan kontrol AODV, dan menanganinya secara terpisah menggunakan modul perangkat lunak yang berbeda (lihat Gambar 2.15) [7].



Gambar 2.15. Penanganan paket pada AODV-UU. Paket data dan pesan kontrol AODV ditangani secara terpisah.

2. 3. 5 Kedatangan Paket

Ketika sebuah paket melintasi *stack* protokol, paket tersebut ditangkap oleh *netfilter hook* yang telah diatur oleh modul *kernel* AODV-UU, kaodv. Paket *non-*

IP selalu diterima, karena paket ini tidak memiliki hubungan dengan AODV-UU. Umumnya paket yang dibangkitkan selalu diantrikan, sebab sebuah rute harus ditentukan. Kontrol pesan AODV yang masuk juga selalu diterima, karena paket ini yang akan diproses pada soket UDP yang terpisah. Sedang paket yang masuk dalam antarmuka jaringan (*network interface*) lainnya, tetapi tidak diproses kemudian [7].

2. 3. 6 Pemrosesan Paket

Jika paket yang masuk adalah kontrol pesan AODV, verdict yang diterima dikembalikan ke modul libpq sehingga paket tersebut akan berakhir di soket UDP kontrol pesan AODV, diterima atau dikirim keluar, tergantung dari kondisi paket tersebut apakah paket *incoming* atau *outgoing* [7].

2. 3. 7 Pemrosesan Paket Data

Jika tujuan dari paket (ditentukan oleh alamat IP tujuan) adalah *host* yang sedang dikunjungi, maka paketnya merupakan paket *broadcast*, atau *mode internet gateway* telah di-*enable*-kan dan paket bukan merupakan paket *broadcast*, maka paket akan diterima. Ini berarti bahwa paket dalam kondisi seperti ini akan ditangani layaknya paket biasa oleh sistem operasi.

Selanjutnya, paket akan diteruskan, dimasukkan dalam antrian atau dibuang. Tabel *routing* internal dari AODV-UU digunakan untuk mengecek apakah rute yang aktif sekarang masih ada atau tidak. Jika rute masih ada, maka paket di-set dan diteruskan ke *hop* berikutnya. Dalam hal ini, paket di-*generate* secara lokal. ID paket yang unik daberikan dan akan digunakan untuk antrian paket yang tidak langsung sampai AODV-UU memutuskan sebuah aksi, dan *route discovery* dibuat. Jika paket tidak dibangkitkan (*generate*) secara lokal, dan tidak ada rute yang ditemukan, maka paket akan dibuang dan pesan RRER dikirimkan ke sumber [7].

2. 3. 8 Pemrosesan AODV *Control Message*

AODV *control message* diterima pada soket UDP (*port* 654) dan selanjutnya diproses. Tipe dari *field* AODV diperiksa, pesan dikonversi menjadi

tipe *message corresponding*, dan selanjutnya fungsi *correct handler* dijalankan [7].

2. 3. 9 Pengiriman AODV *Control Message*

Setiap AODV *control message* yang dibangkitkan oleh AODV-UU dikirimkan pada soket UDP. Ketika pesan ditangkap oleh *netfilter* untuk membangkitkan paket secara local kembali, *NF_IP_LOCAL_OUT*, diantrikan oleh *kernel* AODV dan diterima oleh modul *packet_input* dari AODV-UU melalui libpq. Modul *packet_input* akan mengembalikan sebuah *message* kembali libpq, dan paket selanjutnya akan ditangkap oleh *post-routing* dari *netfilter*, *NF_IP_POST_ROUTING*. Paket tersebut akan kembali diarahkan untuk memastikan penggunaan informasi *routing*, dan dikirim keluar oleh sistem [7].

2. 4 OPENWRT

OpenWrt merupakan program *firmware* berbasis Linux untuk *embedded device* seperti *wireless router*. OpenWrt hanya menyediakan konfigurasi minimal namun dengan kemampuan untuk mendukung paket-paket fitur tambahan, ini berarti *user* memiliki kemampuan untuk memodifikasi fiturnya, menghilangkan paket yang tidak diinginkan untuk menyediakan ruang bagi paket lain, dan untuk pengembang ini berarti dapat terfokus pada paket tanpa harus menguji dan membuang keseluruhan *firmware* [8]. Pada umumnya OpenWrt menggunakan *command-line* sebagai *interface*-nya, namun juga tersedia fitur pilihan *interface* GUI berbasis *web* dan untuk mengatur paket-paket fiturnya *OpenWrt* menggunakan sistem ipkg seperti yang terdapat pada *distro* Linux Debian [9].

OpenWrt diterbitkan pada januari tahun 2004 oleh tim proyek OpenWrt yang dibentuk dalam rangka mengembangkan sebuah *third-party firmware* yang jumlahnya masih sangat terbatas pada saat itu. Sebelum OpenWrt, telah banyak orang memodifikasi *firmware* Linksys berbasis Linux untuk tujuan tertentu, sehingga menyebabkan banyaknya pembuatan *firmware* yang telah dimodifikasi [10].

Pada awalnya OpenWrt merupakan *firmware* pengganti untuk Linksys WRT54G dan sejenisnya (Broadcom BCM947xx) [8], tetapi saat ini telah dapat

digunakan pada *platform* yang berbeda seperti Netgear, ASUS, D-Link, DELL dan lain-lain. Perkembangan OpenWrt sangat dipengaruhi akan kemudahannya dalam memodifikasi fitur-fitur tambahan diluar fitur-fitur yang telah disediakan oleh pihak manufaktur agar dapat digunakan sesuai dengan keperluan tertentu dari para pengguna. Hal ini dapat terjadi karena OpenWrt bersifat *opensource* karena dibuat berdasarkan GNU *General Public License*/Linux, sehingga setiap perubahan yang dibuat oleh pihak manufaktur harus didaftarkan dan dirilis melalui lisensi GPL [9].

Berdasarkan sifat *opensource* ini pula maka para pengguna dapat dengan bebas memodifikasi ataupun menambahkan fitur-fitur lain pada *router* sesuai dengan kebutuhan. Pada perangkat lunak OpenWrt terdapat dua versi utama yang telah dikembangkan yaitu :

1) whiterussian, dan
2) kamikaze.

2. 4. 1 WhiteRussian

Ini merupakan versi awal dari OpenWrt, versi yang paling banyak digunakan karena memiliki tingkat kestabilan tinggi karena telah dikembangkan lebih lama. Selain itu versi ini juga telah banyak memiliki dokumentasi maupun tutorial yang tersedia untuk mendukung pemakaiannya. Pengembangan terakhir dari whiterussian adalah WhiteRussian 0.9 yang dirilis pada tanggal 5 Februari 2007 [11].

2. 4. 2 Kamikaze

Versi ini merupakan versi terbaru dari OpenWrt, walaupun sudah stabil namun masih dalam pengembangan. Dibuat berdasarkan desain berbeda dengan versi yang terdahulu sehingga dapat bekerja pada pilihan jenis *wireless router* yang lebih luas, selain itu mempunyai *kernel* yang lebih baru. Seri terakhir dari kamikaze adalah Kamikaze 7.09 yang dirilis pada September 2007 [11].

2. 5 UoBWinAODV

UoBWinAODV Merupakan *protocol handler* AODV untuk Windows XP yang dikembangkan oleh Universitas Bremen. UoBWinAODV dibuat sesuai dengan RFC 3561 tentang protokol *routing* AODV. UoBWinAODV dibuat dalam bahasa C/C++ dengan menggunakan metode pembanding netfilter untuk mendapatkan informasi penting AODV. *Protocol handler* ini menggunakan PassThru *filter driver* yang dapat mengatur *driver* kartu jaringan secara langsung.

Melalui filter *driver* dapat dilakukan pengaturan distribusi paket dan pengumpulan informasi yang relevan. *Routing* daemon akan mengeksekusi program *user space* dan melewati IOCTL-*Systemcall* pada saat NDIS filter *driver* sedang berkomunikasi. Hal tersebut merupakan proses administrasi *routing* dari keseluruhan sistem setelah persyaratan AODV memungkinkan. NDIS filter *driver* yang digunakan berdasarkan pada yang dibuat oleh Thomas F. Divine [12].

Berdasarkan daftar bagian dari proyek yang terdapat pada RFC 3561, hal yang belum dapat di implementasikan adalah:

1) mencari tabel *routing* internal melalui pencocokkan *prefix* yang lama,
2) perbaikan lokal (*local repair*),
3) penjaluran subnet (subnet *routing*),
4) mengukur setelah *restarting*,
5) perbaikan dari hubungan *unidectional*,
6) pemberitahuan ICMP tujuan tidak dapat dicapai (*destination unreachable*) setelah pencarian rute gagal.