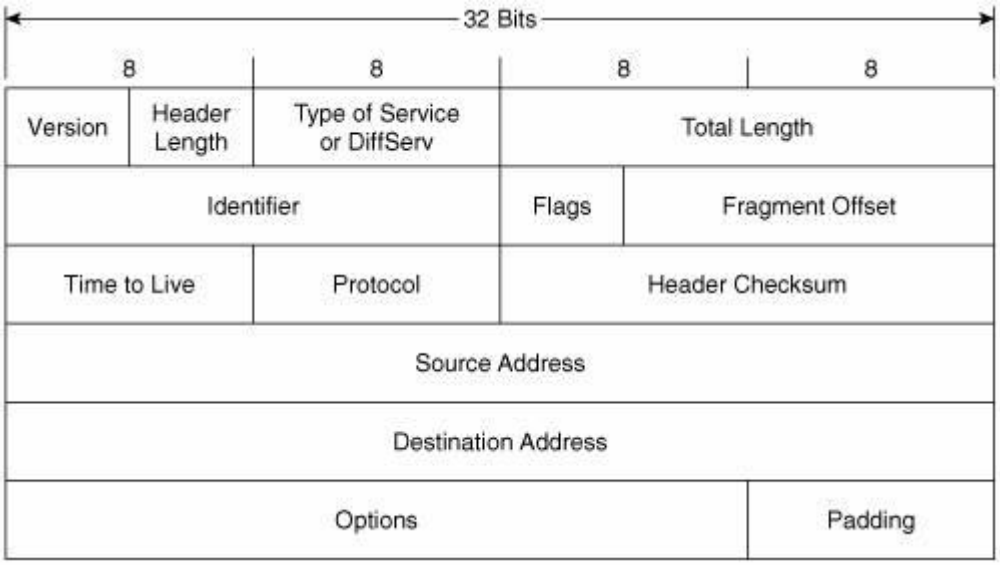
2. LANDASAN TEORI

2.1. Protokol IP (Internet *Protocol*)

"Sebuah datagram IP terdiri dari bagian *header* dan bagian teks. *Header* mempunyai bagian tetap sebesar 20 *byte* dan bagian *optional* yang panjangnya dapat berubah-ubah" (Tanenbaum, 1997, p.55). Format *header* ditunjukkan pada Gambar 2.1.



Gambar 2.1. *Header* IP

Sumber : Tanenbaum (1997, p. 55)

Format *header* ditransmisikan dari arah kiri ke kanan (*big endian*), dimana bit orde tinggi *field Version* berangkat terlebih dahulu. (SPARC adalah *big endian*, sedangkan Pentium adalah *little endian*). Pada mesin *little endian*, diperlukan konvensi *software* baik pada transmisi maupun pada penerimaan.

Field Version mengawasi versi protokol pada datagram. Dengan memasukkan versi pada setiap datagram akan dimungkinkan untuk mempunyai transmisi antara dua buah versi, dimana sebagian mesin mengoperasikan versi lama dan mesin lainnya menjalankan versi baru (Tanenbaum, 1997).

Karena panjang *header* tidak konstan, maka *field* pada *header*, *IHL*, disediakan untuk memberitahukan panjang *header*. Panjang IHL itu sendiri 32-bit words. Panjang minimun adalah 5 yang dipakai apabila tidak ada *option*, nilai

maksimum *field* 4 bit ini adalah 15, yang membatasi *header* pada 60 *byte*, sehingga *field option* mempunyai panjang 40 *byte*. Untuk beberapa *option*, seperti *option* yang mencatat *route* yang dipakai paket, 40 *byte* sangat tidak mencukupi. Hal ini akhirnya menyebabkan *option* menjadi tidak berguna.

Field Type of Service memungkinkan *host* memberitahu subnet tentang jenis layanan yang diinginkan. Bermacam-macam kombinasi reliabilitas dan kecepatan juga dimungkinkan. Bagi suara yang didigitasi, pengantaran yang cepat lebih disenangi dibanding pengantaran yang akurat. Sedangkan untuk transfer data, tranmisi bebas *error* lebih penting daripada transmisi yang cepat .

Field itu sendiri terdiri dari (dari kiri ke kanan) *field* tiga bit *Precedence*, tiga buah frag, D, T, dan R dan dua bit yang tidak dipakai. *Field Precedence*, merupakan prioritas, yang nilainya dimulai pada 0 (normal) hingga 7 (paket kontrol jaringan). Tiga buah bit mengizinkan *host* untuk menspesifikasikan tentang apa yang paling harus diperhatikan dari set (*Delay, Throughput, Reliability).* Dalam teori, *field-field* mengizikan router untuk membuat pilihan. Misalnya, suatu *link* satelit dengan *throughput* yang tinggi dan *delay* yang tinggi atau saluran sewa yang mempunyai *throughput* yang rendah dan *delay* yang rendah. Pada prakteknya, router-router yang tersedia saat ini mengabaikan *field Type of Service* (Tanenbaum, 1997).

Total Length meliputi segala hal dalam *delay-header* dan data. Panjang maksimumnya adalah 65.535 *byte*. Saat ini, limit atas bersifat *tolerable*, tapi jaringan gigabit masa depan akan membutuhkan datagram yang lebih besar.

Field Identification diperlukan untuk mengizinkan *host* tujuan menentukan datagram pemilik *fragment* yang baru datang. Semua *fragment* suatu datagram berisi nilai *Identification* yang sama.

Setelah *field* tersebut terdapat bit yang tidak dipakai dan kemudian diikuti oleh dua buah *field* 1-bit. *DF* singkatan dari *Don't Fragment.* Bit ini merupakan perintah ke router agar tidak melakukan fragmentasi terhadap datagram karena mesin tujuan tidak memiliki kemampuan untuk menggabungkan kembali potongan-potongan. Misalnya, bila suatu komputer di-*boot*, ROM-nya akan meminta *memory image* mengirimkan kepadanya sebagai datagram tunggal. Dengan menandai datagram dengan bit *DF*, pengirim akan mengetahui bahwa

datagram tersebut akan tiba dalam satu potong saja dan juga diartikan bahwa datagram harus menghindari jaringan paket kecil pada lintasan terbaik dan mengambil lintasan optimal yang lainnya. Semua mesin diharuskan dapat menerima *fragment* 576 *byte* atau yang lebih kecil.

Field Time to live adalah *counter* yang digunakan untuk membatasi umur paket. *Field* ini diharapkan dapat menghitung waktu dalam detik, yang memungkinkan umur maksimum 255 detik. *Counter* harus diturunkan pada setiap *hop* dan diturunkan beberapa kali bila berada bila berada pada antrian router dalam waktu yang lama. Pada kenyataanya, *field* ini hanya menghitung *hop.* Ketika *counter* mencapai nol, paket dibuang dan paket peringatan dikirimkan kembali ke *host* sumber. *Feature* ini mencegah datagram agar tidak kehilangan arah, sesuatu yang mungkin terjadi bila *routing table* rusak.

2.2. Alamat IP

Setiap *host* dan router di *Internet* memiliki alamat IP, yang meng-*encode* nomor jaringan dan nomor *host*. Kombinasinya bersifat unik, tidak ada dua mesin yang memiliki IP sama. Semua alamat IP mempunyai panjang 32 bit dan digunakan dalam *field-field Source address* dan *Destination address* paket IP. Format alamat IP ditunjukkan pada Gambar 2.2. Mesin-mesin yang terhubung ke jaringan yang banyak mempunyai alamat-alamat yang berbeda pada masing-masing jaringan.

Kelas Alamat	Nilai oktet pertama	Bagian untuk *Network Identifier*	Bagian untuk *Host Identifier*	Jumlah jaringan maksimum	Jumlah *host* dalam satu jaringan maksimum
Kelas A	1–126	W	X.Y.Z	126	16,777,214
Kelas B	128–191	W.X	Y.Z	16,384	65,534
Kelas C	192–223	W.X.Y	Z	2,097,152	254
Kelas D	224-239	Multicast IP Address	Multicast IP Address	Multicast IP Address	Multicast IP Address
Kelas E	240-255	Dicadangkan; eksperimen	Dicadangkan; eksperimen	Dicadangkan; eksperimen	Dicadangkan; eksperimen

Gambar 2.2. Format Alamat IP

Sumber : Wikipedia (2009)

Format-format kelas A, B, C, dan D mengizinkan hingga 126 jaringan dengan masing-masing 16 juta *host*, 16.382 jaringan hingga 64 K *host*, 2 juta jaringan, (misalnya LAN-LAN), dengan masing-masing 254 *host*, dan *multicast*, dimana datagram ditujukan ke sejumlah *host*. Alamat-alamat yang berawal dengan 11110 dicadangkan untuk dipakai di masa datang. Saat ini puluhan ribu jaringan telah terhubung ke Internet, dan jumlahnya berlipat ganda dua setiap tahunnya. Nomor jaringan diberikan oleh NIC (*Network Information Center*) untuk menghindari terjadinya konflik.

Alamat jaringan, yang berupa nomor dengan panjang 32 bit, biasanya ditulis dalam bentuk notasi desimal bertitik. Dalam format ini, setiap 4 *byte* ditulis dalam bentuk bilangan desimal, mulai dari 0 sampai 255. Misalnya alamat heksadesimal C0290614 ditulis sebagai 192.41.6.20. Alamat IP terendah adalah 0.0.0.0 dan yang tertinggi adalah 255.255.255.255.

Alamat khusus 0.0.0.0 digunakan oleh *host* ketika *host-host* sedang di-*boot* tapi setelah itu tidak digunakan. Alamat IP yang menggunakan angka 0 sebagai nomor jaringannya berkaitan dengan jaringan saat itu. Alamat-alamat ini memungkinkan mesin untuk merujuk ke jaringannya sendiri tanpa harus mengetahui nomornya (tapi mesin-mesin itu harus mengetahui kelasnya agar mengetahui jumlah angka 0 yang akan dilibatkan). Alamat yang seluruhnya terdiri dari angka 1 dapat melakukan *broadcasting* pada jaringan lokal, biasanya suatu LAN. Alamat dengan nomor jaringan tertentu dan *field host*-nya berangka 1 seluruhnya mengizinkan mesin untuk mengirimkan paket-paket *broadcast* ke LAN luar dimanapun di Internet. Terakhir, semua alamat dengan bentuk 127.xx.yy.zz dicadangkan untuk pengujian *loop* balik. Paket yang dikirimkan ke alamat itu tidak diteruskan ke kabel namun paket-paket tersebut diproses secara lokal dan diperlakukan sebagai paket yang datang. Hal ini memungkinkan paket dikirimkan ke jaringan lokal namun tanpa pengirim mengetahui nomornya. *Feature* ini juga dipakai untuk *debugging software* jaringan.

2.3. Internet *Multicast*

Komunikasi IP normal berada di antara sebuah pengirim dan sebuah penerima. Akan tetapi, beberapa aplikasi akan bermanfaat apabila suatu proses

mampu mengirim ke sejumlah besar penerima secara bersamaan. Misalnya adalah *updating database* terdistribusi, pengiriman harga saham ke sejumlah pialang, dan penanganan panggilan telepon konferensi (melibatkan banyak pihak) digital.

IP mendukung *multicasting,* dengan menggunakan alamat-alamat kelas D. Setiap alamat kelas D mengidentifikasikan sekelompok *host.* Dua puluh depalan bit bisa dipakai untuk pengidentifikasian kelompok, sehingga lebih dari 250 juta kelompok dapat berada dalam saat yang bersamaan. Ketika sebuah proses mengirimkan suatu paket ke alamat kelas D, usaha yang terbaik dilakukan dengan mengirimkan paket itu ke semua anggota kelompok yang mempunyai alamat, tapi tidak ada jaminan yang diberikan. Sebagian anggota mungkin tidak akan mendapatkan paket (Tanenbaum, 1997).

Dua jenis kelompok alamat didukung oleh alamat permanen dan alamat sementara. Kelompok permanen selalu ada dan tidak harus disetel terlebih dahulu. Setiap kelompok permanen memiliki alamat kelompok permanen. Beberapa contoh alamat kelompok permanen adalah:

- 224.0.0.1 Semua sistem pada suatu LAN
- 224.0.0.2 Semua router pada suatu LAN
- 224.0.0.5 Semua router OSPF pada suatu LAN
- 224.0.0.6 Semua router OSPF wakil pada suatu LAN

Kelompok sementara harus dibuat sebelum kelompok-kelompok itu bisa digunakan. Suatu proses dapat meminta *host*-nya untuk bergabung dengan kelompok, kelompok itu tidak berada lagi di *host.* Setiap *host* mengawasi kelompok mana dimiliki proses.

Multicasting diimplementasikan oleh router *multicast* khusus, yang dapat atau tidak dapat ditempatkan bersama-sama router *standard.* Sekitar satu menit, setiap router *multicast* mengirimkan suatu *multicast hardware* (yaitu, *data link layer*) ke *host-host* yang terdapat di LAN-nya (alamat 224.0.0.1) yang meminta *host* untuk melaporkan kelompok yang sedang dimiliki proses-proses. *Host* mengirimkan jawaban semua alamat kelas D yang diminatinya.

Paket *query* dan respon ini menggunakan protokol yang disebut *Internet Group Management Protocol* (IGMP), yang tidak jelas kesamaanya dengan ICMP. IGMP hanya mempunyai dua jenis paket : *query* dan *response*, yang masing-

masing menggunakan format tetap yang sederhana yang berisi beberapa kontrol informasi pada word pertama *file payload* dan alamat kelas D pada word keduanya. Protokol ini dijelaskan pada RFC 1112 (Tanenbaum, 1997).

Routing multicast dilaksanakan dengan *spanning tree*. Setiap router *multicast* saling bertukar informasi dengan tetangga-tetangga dengan menggunakan protokol vektor jarak yang dimodifikasi supaya masing-masing router membentuk *spanning tree* per kelompok yang mencakup seluruh anggota kelompoknya. Terdapat banyak metode optimasi yang digunakan untuk memangkas diagram pohon agar dapat mengurangi router dan jaringan yang tidak dikehendaki oleh kelompok-kelompok tertentu. Protokol dapat sering menggunakan *tunneling* supaya tidak mengganggu simpul-simpul di dalam *spanning tree.*

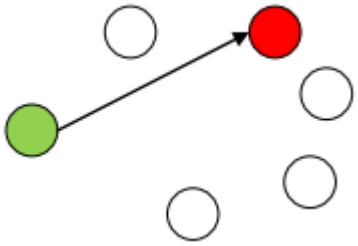
2.4. Pengalamatan *Interface*

Pengalamatan *interface* dibagi ke dalam 4 jenis, yaitu *unicast, multicast, broadcast* dan *anycast*.

2.4.1. *Unicast Address*

Alamat *unicast* pada IPv4 berada dalam kelas A-C. Sebuah alamat *unicast* secara unik mengidentifikasikan sebuah *nterface* yang memiliki alamat IP. Misalkan terdapat sebuah komputer dengan *ethernet card* yang memilik IP *address* 10.77.0.1. Dalam satu *network* yang sama, tidak bole ada dua *interface* yang memilik IP *address* yang sama. Jika terdapat IP *address* yang sama, biasanya sistem operasi akan mengeluarkan peringatan bahwa terdapat IP *address* yang kembar.

Sebuah paket yang dikirimkan ke sebuah *unicast address* akan dikirimkan ke *interface* yang memiliki alamat tersebut, sehingga relasi pengiriman paket ini adalah *one-to-one.* Pengiriman pada alamat *unicast* terlihat pada Gambar 2.3.

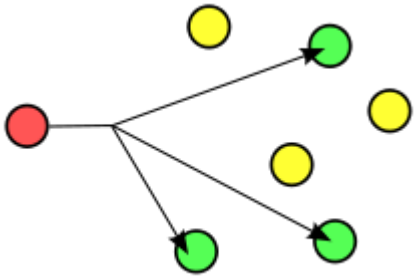


Gambar 2.3. Pengiriman pada alamat *Unicast*

Sumber : Wikipedia (2009)

2.4.2. *Multicast Address*

Alamat *multicast* juga tersedia pada IPv4. Alamat *multicast* pada IPv4 berada dalam kelas D. Sebuah alamat *multicast* mengidentifikasikan sebuah grup yang terdiri dari beberapa *interface* yang memiliki alamat IP. Sebuah paket yang dikirimkan ke sebuah alamat *multicast* akan diproses oleh semua anggota dari grup *multicast* tersebut, seperti terlihat pada Gambar 2.4 :



Gambar 2.4. Pengiriman pada alamat *Multicast*

Sumber : Wikipedia (2009)

Keuntungan penggunaan alamat *multicast* ini adalah pengirim hanya perlu mengirimkan data sebanyak satu kali kepada semua *user* yang tergabung dalam grup *multicast* tersebut. Sehingga pengiriman pada alamat *multicast* ini dapat menghemat *bandwith.* Misalkan seorang *user* ingin melakukan *streaming* data video yang berukuran 1 *megabytes/s.* Jika pengirimannya bersifat *unicast* dan terdapat 5 *user*, berarti dibutuhkan 5x1 *megabytes* tiap detiknya untuk mengirimkan data kepada *user* yang lain. Selain itu, pada satu waktu, hanya 1 *user* yang dapat menerima data. Sehingga, waktu yang dibutuhkan untuk mengirimkan data ke semua *user* yang lain adalah *n* kali dari waktu yang

dibutuhkan untuk pengiriman pada satu *user*. Hal ini sangat tidak efisien, karena memakan waktu dan *bandwith* yang besar.

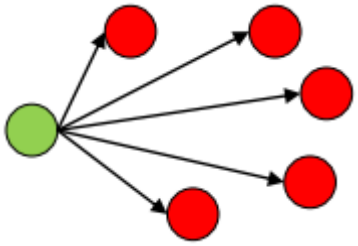
Dengan pengiriman data pada alamat *mutlicast*, *user* yang mengirimkan data hanya perlu mengirimkan data sekali saja pada sebuah alamat *multicast*. *User* lainnya yang hendak menerima data tersebut harus *listen* pada alamat *multicast* dan nomor *port* yang dikehendaki. Dengan demikian, jumlah *user* yang menerima data pada sebuah paket *multicast* tidak berpengaruh terhadap jaringan yang ada.

Namun kelemahan dari *multicast address* ini adalah setiap paket yang dikirimkan menggunakan *User Datagram Protocol I* (UDP) dimana paket yang dikirimkan tidak dijamin akan sampai kepada penerima. Belum tentu semua *user* menerima paket yang sama.

Mengenai alokasi alamat *multicast* yang dapat digunakan untuk aplikasi dapat dilihat secara lengkap pada RFC 3171 untuk IPv4.

2.4.3. *Broadcast address*

Alamat *broadcast* hanya terdapat pada IPv4 dan merupakan alamat terakhir pada sebuah *network/subnetwork*. Data yang dikirimkan pada alamat *broadcast* akan diterima oleh semua *interface* yang memiliki IP yang berada pada satu *network/subnetwork* yang sama. Misalkan terdapat sebuah komputer dengan alamat IP 10.77.0.1 dan *subnet mask* 255.0.0.0, alamat *broadcast*-nya adalah 10.255.255.255, karena alamat tersebut merupakan alamat terakhir dari *network* tersebut. Jika komputer tersebut mengirimkan data ke alamat 10.255.255.255, data akan diterima oleh semua komputer, mulai dari 10.0.0.1 sampai 10.255.255.254 yang menggunakan *subnet mask* 255.0.0.0, seperti terlihat pada Gambar 2.5.

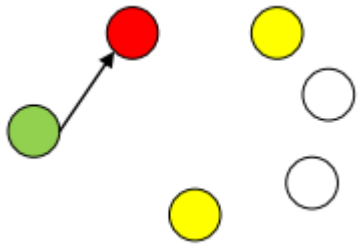


Gambar 2.5. Pengiriman pada alamat *Broadcast*

Sumber : Wikipedia (2009)

2.4.4. *Anycast address*

Anycast address hanya terdapat pada IPv6 dan biasanya diberikan pada beberapa *interface*. Sebuah paket yang dikirimkan ke sebuah alamat *anycast* akan dikirimkan hanya pada satu *device* saja, biasanya dikirimkan pada *interface* yang paling dekat, seperti terlihat pada Gambar 2.6 :



Gambar 2.6. Pengiriman pada alamat *Anycast*

Sumber : Wikipedia (2009)

2.5. Protokol-Protokol *Transport Internet*

"Internet memiliki dua buah protokol utama dalam *transport layer*, sebuah protokol *connection oriented* dan sebuah protokol *connectionless.* Protokol *connection oriented* tersebut adalah TCP. Sedangkan protokol *connectionless* adalah UDP. Karena UDP pada dasarnya merupakan IP dengan tambahan *header* yang pendek" (Tanenbaum, 1997, p.135).

Transmission Control Protocol (TCP) secara spesifik dirancang untuk menyediakan aliran *byte end-to-end* yang *reliable* melalui *Internetwork* yang tidak *reliable*. Suatu *Internetwork* dibedakan dengan jaringan tunggal karena bagian-bagian yang tidak sama dapat memiliki topologi, *bandwith*, *delay*, ukuran paket, dan parameter-parameter lainnya yang sangat berbeda. TCP telah dirancang agar mampu beradaptasi secara dinamis dengan *Internetwork* dan mempunyai daya tahan dalam menghadapi berbagai macam kegagalan.

Secara formal TCP didefinisikan dalam RFC 793. Dengan berjalannya waktu, berbagai *error* dan ketidak konsistenan telah dapat terdeteksi, persyaratan-persyaratanpun dalam beberapa bidang telah banyak berubah. Uraian dan perbaikan *bug* dibahas secara mendalam dalam RFC 1122. Sedangkan pengembangannya diberikan dalam RFC 1323.

Setiap mesin yang mendukung TCP memiliki *transport entity* TCP, yaitu proses pengguna atau bagian *kernel* yang mengatur aliran TCP dan *interface* bagi bagi IP *layer*. *Entity* TCP menerima aliran data pengguna dari proses-proses lokal, kemudian memecahnya menjadi beberapa potong yang tidak melebihi 64 *Kilobyte* (pada prakteknya, biasanya sekitar 15000 *byte*), dan mengirimkan setiap potongan itu sebagai datagram IP yang terpisah. Ketika datagram IP yang berisi data TCP tiba di mesin, datagram tersebut diserahkan kepada *entity TCP, yang* menyusun kembali menjadi aliran *byte* orisinilnya. Untuk mudahnya, kadang-kadang kita hanya menggunakan "TCP" untuk mengartikan *transport entity* TCP (potongan *software*) atau protokol TCP (kumpulan peraturan). Dari koneksi jelas TCP yang mana yang dimaksud. Misalnya di dalam konteks "pengguna memberikan data ke TCP". Maka yang dimaksud TCP disini adalah *transport entity* TCP.

IP *layer* tidak menjamin datagram akan dikirimkan dengan benar, karena itu hal ini tergantung pada TCP dalam memberikan *timeout* dan menstransmisi ulang datagram tersebut bila diperlukan. Datagram yang tiba dapat juga mempunyai urutan yang salah, ini juga merupakan tugas TCP untuk menyusunnya kembali menjadi pesan-pesan dengan urutan yang benar. Singkatnya, TCP harus menyediakan reliabilitas yang diperlukan oleh sebagian besar pengguna dan yang tidak dapat diperoleh dari IP.

2.5.1. Protokol TCP

Setiap *byte* pada koneksi TCP memiliki nomor urut 32 bit-nya sendiri. Bagi *host* yang berkecepatan penuh pada sebuah LAN 10 Mbps, secara teoritis nomor urut membutuhkan waktu sekitar satu jam, namun pada prakteknya membutuhkan waktu yang lebih lama lagi. Nomor urut digunakan baik untuk *acknowlegment* maupun untuk mekanisme jendela, yang menggunakan *field-field header* terpisah (Tanenbaum, 1997).

Entity TCP pengirim dan penerima saling bertukar data dalam bentuk *segment*, sebuah *segment header* berukuran tetap 20 *bytes* (ditambah bagian optional) yang diikuti oleh nol atau lebih *byte-byte* data. *Software* TCP memutuskan besar *segment* yang seharusnya. *Software* ini dapat mengakumulasi data dari beberapa tulisan menjadi sebuah *segment* atau memotong-motong data

dari sebuah penulisan menjadi beberapa *segment*. Terdapat dua hal yang membatasi ukuran *segment*. Pertama, setiap *segment,* termasuk *header* TCP, harus dengan *payload* IP 65.535 *byte.* Kedua, setiap jaringan memiliki *maximum transfer unit* atau MTU, dan setiap *segment* harus pas dengan besarnya MTU. Pada prakteknya, umumnya MTU adalah beberapa ribu *byte* karena itu menentukan batas maksimum ukuran *segment*. Bila sebuah *segment,* melewati sejumlah jaringan tanpa diframnetasikan terlebih dahulu dan kemudian menemui jaringan yang MTU-nya lebih kecil dari *segment*, maka router pada batas memfragmentasikan *segment* menjadi dua *segment* atau lebih yang berukuran lebih kecil

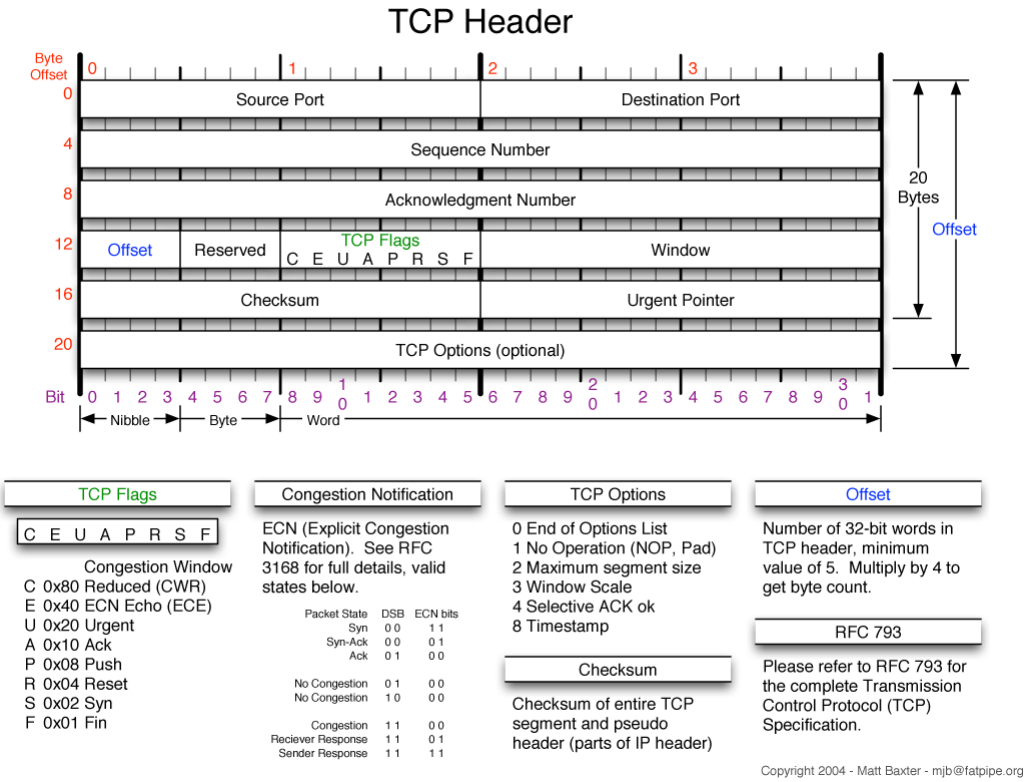
Sebuah *segment* yang terlalu besar bagi jaringan yang disinggahinya daoat dipecah menjadi beberapa *segment* oleh router. Masing-masing *segment* baru mendapatkan TCP dan *header* IP-nya sendirim sehingga fragmentasi yang dilakukan router menambah *overhead* total (karena setiap *segment* tambahan menambah 40 *byte* formasi *header* tambahan).

Protokol dasar yang digunakan oleh *entity* TCP merupakan protokol jendela geser. Ketika pengiriman mentransmisikan sebuah *segment,* pengirim juga menghidupkan timer, pada saat *segment* tiba di tempat tujuan, *entity* TCP penerima mengirimkan kembali sebuah *segment* (dengan data bila ada, bila tidak ada maka tanpa data) yang berkaitan dengan nomor *acknowlegment* yang sama dengan nomor urut berikutnya yang akan diterimanya. Bila *timer* pengirim berhenti sebelum sebelum *acknowlegment* diterima, maka pengirim mentransmisikan *segment* lagi.

Walaupun protokol ini kelihatannya sederhana, terdapat banyak masalah yang rumit. Misalnya, karena *segment* dapat difragmentasikan, maka mungkin saja bagian *segment* tiba dan diberi *acknowlegment* oleh *entity* TCP penerima, namun *segment-segment* lainnya hilang. *Segment* dapat juga dalam keadaan tidak terurut, maka *byte-byte* 3072-4095 dapat tiba tapi tidak diberi *acknowlegment* karena *byte-byte* 2048-3071 belum muncul. *Segment* juga dapat di-*delay*-kan dalam interval waktu yang lama hingga pengirimnya *timeout* dan mentransmisi ulang *segment-segment* tersebut. Bila *segment* yang ditransmisi ulang itu mengambil *route* yang berbeda dengan *route* asalnya dan difragmentasikan secara

berbeda, bit-bit dan potongan-potongan *segment* orisinil dan *segment* dulplikat dapat tiba secara sporadis. Hal ini memerlukan administrasi yang hati-hati agar memperoleh aliran *byte* yang *realiabe.* Terakhir, dengan semakin banyaknya jaringan yang tergabung dengan *Internet*, maka mungkin sebuah *segment* mempunyai jaringan yang sedang macet (atau putus) dalam lintasannya.

TCP harus disiapkan untuk menghadapi masalah-masalah ini dan mengatasinya dengan cara efisien. Banyak usaha yang lebih dicurahkan untuk mengoptimasi untuk kerja aliran TCP, termasuk dalam menghadapi masalah-masalah jaringan.



Gambar 2.7. Header TCP

Sumber : Tanenbaum (1997, p. 138)

Gambar 2.7 menjelaskan *layout* segmen TCP. Setiap segmen dimulai dengan sebuah format tetap *header* 20 *byte.* Header ini diikuti oleh *option header*. Setelah *option*, bila ada, diikuti oleh *byte* data hingga 65.535 – 20 – 20 = 65.515, dimana angka 20 pertama adalah *header* IP dan yang kedua adalah *header* TCP.

Segmen yang tanpa data adalah legal dan umumnya digunakan untuk *acknowlegment* dan pesan-pesan kontrol.

Field Source port dan *field Destination port* mengidentifikasi *end point* lokal daripada koneksi. Setiap *host* dapat mengambil keputusan untuk dirinya sendiri tentang cara mengalokasi *port*-nya yang dimulai pada 256. Suatu *port* ditambah alamat IP *host* membentuk TSAP 48 *bit* yang unik. Nomor *socket* sumber dan tujuan bersama-sama mengidentifikasi koneksi.

Field Sequence number dan *field Acknowlegment number* membentuk fungsi seperti biasanya. Dengan catatan bahwa *Acknowlegment number* menspesifikasi *byte* berikutnya yang diharapkan. Keduanya mempunyai panjang 32 bit karena setiap *byte* data diberi nomor di dalam aliran TCP.

TCP *header length* menyatakan berapa buah *word* 32 *bit* yang terdapat di dalam *header* TCP. Informasi ini diperlukan karena *field Option* mempunyai panjang yang berubah-ubah, karena itu *header*-pun akan berubah-ubah pula. Secara teknik, *field* ini benar-benar mengindikasikan awal data di dalam segmen, yang diukur dalam *word* 32 *bit,* namun nomor ini merupakan panjang *header* dalam *word,* karena itu efeknya sama saja.

Berikutnya terdapat *field* 6 *bit* yang tidak digunakan. Kenyataa bahwa *field* ini tetap utuh dalam beberapa dekade merupakan ujian tentang kualitas TCP. Sedikit protokol akan memerlukan *field* ini untuk menghilangkan *bug* di dalam rancangan yang orisinil.

Setelah itu terdapat *flag* 1 *bit.* URG disetel ke 1 apabila *Urgent pointer* sedang digunakan. *Urgent pointer* digunakan untuk mengindikasi *offset byte* dari nomor urut saat tertentu dimana *urgent* data ditemukan. Fasilitas ini adalah untuk menggantikan pesan-pesan *interrupt.* Seperti telah disebutkan di atas, fasilitas ini merupakan cara penting untuk mengizinkan pengirim memberi sinyal kepada penerima tanpa menyebabkan keterlibatan TCP itu sendiri.

Bit ACK disetel ke 1 untuk mengindikasikan bahwa *acknowlegment number* adalah valid. Bila ACK sama dengan 0, segmen tidak mengandung suatu *acknowlegment* sehingga *acknowlegment number* akan diabaikan.

Bit PSH mengindikasikan data yang di-PUSH. Penerima dalam hal ini diminta untuk mengantarkan data ke aplikasi ketika data itu sampai dan tidak mem-*buffer*-kannya sampai *buffer* penuh telah diterima.

Bit RST dipakai untuk menyetel ulang koneksi yang telah menjadi kacau sehubungan dengan terjadinya *crash* dan sebab-sebab lainnya. Hal ini juga dipakai untuk membuang segmen yang tidak valid atau menolak suatu usaha untuk membuka suatu koneksi. Pada umumnya, bila anda mendapat segmen yang mempunya *bit RST*-nya dalam keadaan *on,* maka sedang terjadi masalah.

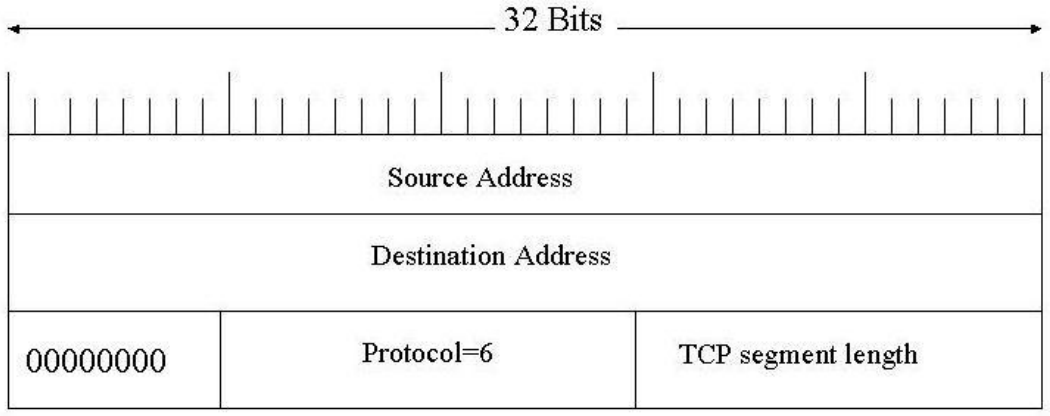
Bit SYN digunakan untuk membentuk koneksi. Permintaan koneksi memiliki SYN = 1 dan ACK = 0 untuk menandakan bahwa *field acknowlegment piggyback* sedang tidak dipakai. Jawaban koneksi tidak berkaitan dengan *acknowlegment*, maka jawaban ini mempunyai SYN = 1 dan ACK = 1. Pada dasarnya, *bit* SYN digunakan untuk menandakan *CONNECTION REQUEST* dan *CONNECTION ACCEPTED*, sedangkan *bit* ACK digunakan untuk membedakan antara kemungkinan yang bisa terjadi (Tanenbaum, 1997).

Bit FIN dipakai untuk melepaskan koneksi. *Bit* ini menspesifikasi bahwa pengirim juga tidak mempunya data lainnya yang akan ditransmisikan. Akan tetapi, setelah koneksi ditutup, proses dapat melanjutkan menerima data. Segmen SYN dan FIN memiliki nomor urut dan karena itu dijamin akan diproses dengan urutan yang benar.

Kontrol aliran dalam TCP ditangani dengan menggunakan jendela geser yang ukuranya dapat berubah-ubah. *Field Window* menyatakan berapa banyak *byte* yang dapat dikirimkan ang berawal pada saat *byte* di-*acknowlegde*. *Field Window* 0 adalah legal dan menyatakan bahwa *byte* sampai dengan *Acknowlegment number*-1 telah diterima, namun penerima saat itu sedang tidak memerlukan data lainnya dan tidak menginginkan menerima data lainnya pada saat itu. Izin untuk mengirimkan dapat diberikan kemudian dengan mengirimkan segmen yang mempunyai *acknowlegment number* yang sama dan *field Window* yang tidak nol.

Checksum juga disiapkan untuk reliabilitas ekstrim. *Field* ini memberi *checksum header*, data, dan *pseudo-header* konseptual yang ditunjukkan pada Gambar 2.8. Ketika membentuk komputasi ini, *field Checksum* TCP disetel ke nol,

dan *field* data ditambahkan dengan *byte* nol tambahan bila panjangnya merupakan bilangan ganjil. Algoritma cukup menambahkan seluruh *word 16 bit* dalam komplemen 1 dan kemudian mengambil komplemen 1 jumlahnya. Sebagai akibatnya, ketika penerima membentuk kalkulasi seluruh segmen, termasuk *field Checksum*, maka hasilnya sama dengan nol.



Gambar 2.8. *Pseudoheader* yang termasuk dalam *checksum* TCP
Sumber : Tanenbaum (1997, p. 139)

Pseudoheader terdiri dari alamat IP 32 *bit* mesin sumber dan tujuan, nomor protokol untuk TCP, dan hitungan *byte* untuk segmen TCP (termasuk *header*). Dengan melibatkan *pseudoheader* kedalam perhitungan *checksum* TCP akan menolong mendeteksi paket yang diantarkan secara salah. Tapi dengan melakukan hal ini melanggar hierarki protokol karena alamat IP yang berada didalamnya sebenarnya milik IP *layer*, bukan milik TCP *layer*.

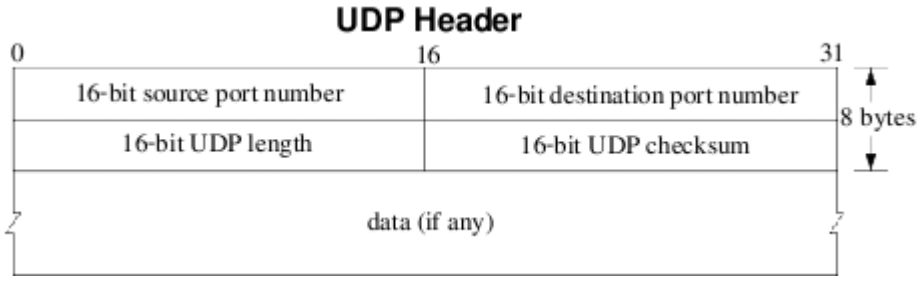
Field Option dirancang untuk menyediakan fasilitas tambahan yang tidak dicakup oleh *header* biasa. *Option* yang paling penting adalah *option* yang mengizinkan setiap *host* untuk menspesifikasi *payload* TCP maksimum yang ingin diterimanya. Dengan menggunakan segmen yang lebih besar akan lebih efisien dibanding dengan menggunakan segmen yang lebih kecil karena *header* 20 *byte* dapat dibayar melalui data yang berukuran lebih besar, namum *host* yang kecil mungkin tidak akan mampu menangani segmen yang berukuran sangat besar. Selama persiapan koneksi, masing-masing sisi dapat mengumumkan nilai segmen maksimumnya dan melihat nilai maksimum yang dimiliki sisi lainnya. Maka nilai

terkecil di antara kedua nilai maksimum itu yang akan digunakan. Bila *host* tidak menggunakan *option* ini, maka nilainya disetel ke nilai *default payload* 536 *byte*. Semua *host* Internet diharuskan dapat menerima segmen TCP 536 + 20 = 556 *byte*.

Bagi saluran yang mempunyai *bandwith* yang tinggi, *delay* yang besar, atau keduanya, jendela 64 KB seringkali menjadi masalah tersendiri. Pada saluran T3 (44,736 Mbps), diperlukan waktu 12 milidetik meng-*output*-kan jendela 64 KB penuh. Bila *delay* propagasi pulang pergi sama dengan 50 milidetik (nilai yang umum bagi serat optik antar benua), maka pengirim akan berada dalam keadaan *idle* selama ¾ untuk menunggu *acknowlegment.* Pada koneksi satelit, situasi malah semakin memburuk. Ukuran jendela yang lebih besar akan mengizinkan pengirim memompa data keluar secara terus-menerus, namun dengan menggunakan *field Window size*, tidak ada cara untuk mengekspresikan ukuran seperti itu. Dalam RFC 1323, *option Window scale* telah diajukan pemakaiannya. *Option* ini mengijinkan pengirim dan penerima melakukan negosiasi faktor skala jendela. Bilangin ini memungkinkan kedua sisi menggeser *field Window size* sampai 16 *bit* ke kiri, sehingga memungkinkan jendela menjadi berukuran sampe 2^{32} *byte*. Sekarang sebagian besar implementasi TCP mendukung *option* ini.

2.5.2. Protokol UDP

Suite protokol *Internet* juga mendukung protokol *transport connectionless*, yaitu UDP (*User Datagram Protocol)*. UDP menyediakan cara bagi aplikasi untuk mengirimkan datagram IP yang dikemas dan mengirimkan datagram ini tanpa melakukan pembentukan koneksi. Banyak aplikasi *client-server* yang memiliki sebuah *request* dan sebuah *response* menggunakan UDP dibandingkan dengan membuat koneksi terlebih dahulu dan kemudian melepaskan koneksi. UDP dijelaskan dalam RFC 768 (Tanenbaum, 1997).



Gambar 2.9. Header UDP

Sumber : Tanenbaum (1997, p. 150)

Segment UDP terdiri dari *header* 8 *byte* yang diikuti oleh data. *Header* ini ditunjukkan pada Gambar 2.9. Dua buah *port* mempunyai fungsi yang sama seperti yang dikerjakannya pada TCP, yaitu mengidentifikasi *endpoint* pada mesin sumber dan mesin tujuan. *Field length* UDP meliputi *header* 8 *byte* dan data. *Field checksum* UDP meliputi *pseudoheader* format yang sama dengan yang ditunjukkan pada Gambar 2.9, *header* UDP, dan data UDP yang ditambahkan ke jumlah genap *byte* bila diperlukan. Hal itu merupakan optional dan disimpan sebagai 0 bila tidak dihitung (*true* yang dihitung sebagai 0 disimpan seluruhnya sebagai *byte-byte* 1, yang sama dengan komplemen 1). Mematikan perhitungan ini cukup merugikan kecuali kualitas data tidak dijadikan pertimbangan yang utama (misalnya pidato yang didigitasi) (Tanenbaum, 1997).

2.6. *Windows Registry*

Registry, dalam *platform* sistem operasi Mircosoft Windows 32-bit, merupakan sebuah basis data yang disusun secara hierarkis yang mengandung informasi mengenai konfigurasi sebuah sistem, mulai dari konfigurasi perangkat keras, perangkat lunak, asosiasi ekstensi berkas dengan aplikasinya hingga preferensi pengguna. *Registry* merupakan pengganti berkas-berkas konfigurasi *.INI yang digunakan dalam sistem Windows 16-bit (Windows 3.x dan Windows *for Workgroup). Registry*, pertama kali diperkenalkan di dalam sistem Windows 16-bit sebagai penampung informasi mengenai pemetaan/asosiasi ekstensi berkas dengan aplikasinya, dan kemudian dikembangkan menjadi basis data dengan cakupan yang luas pada sistem-sistem operasi keluarga Windows NT. *Registry* juga kemudian digunakan pada sistem operasi kelas rumahan, yaitu Windows 95, Windows 98 dan Windows ME, tapi memang implementasi yang cukup bagus dari *registry* terdapat di dalam keluarga sistem operasi Windows NT (Wikipedia, 2009).

Struktur *registry* agak mirip dengan struktur direktori dalam sistem berkas. Selain itu, *registry* juga dapat diakses dengan menggunakan sintakis yang sama dengan cara mengakses berkas, dengan menggunakan karakter garis miring

terbalik (*backslash*) untuk menandakan tingkatan hierarkis. Susunannya adalah seperti <*subtree*>\<*key*>\<*subkey...*>. sebagai contoh, My Computer \HKEY_LOCAL_MACHINE\Software\Microsoft\Windows, yang menunjuk kepada sebuah *subkey* yang memiliki nama "Windows" yang terdapat di dalam *subkey* Microsoft, yang terdapat didalam *key* dengan nama *software*, yang terdapat di dalam *subtree* HKEY_LOCAL_MACHINE**.**

Setiap *key* dan *subkey* tersebut dapat memiliki nilai yang dapat ditentukan atau nilai *default*, yang disebut sebagai *Value*. Akan tetapi, cara mengakses *Value* tidaklah sama dengan cara mengakses berkas, mengingat nama *value* dapat mengandung karakter *backslash* yang dapat menjadi ambigu ketika menggunakan cara baca seperti halnya mengakses sistem berkas. Adalah fungsi-fungsi dalam Windows 32-bit *Application Programming Interface/* API (Win32 API) yang dapat melakukan *query* dan manipulasi terhadap *value-value registry*, yang dilakukan dengan cara mengambil nama *value* secara terpisah dari *path key* yang merupakan *parent key*.

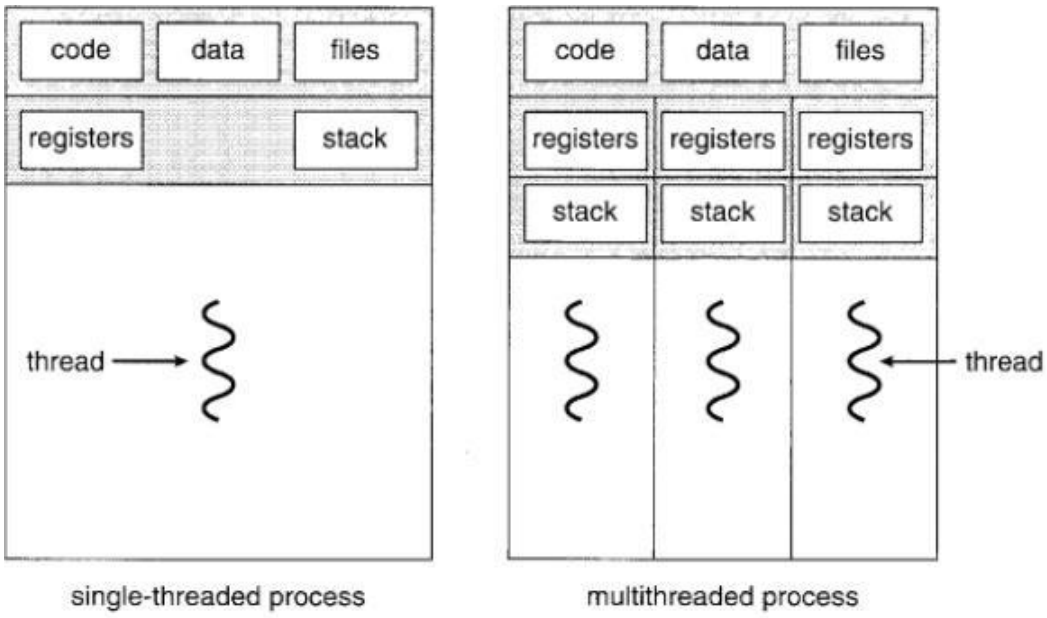
2.7. *Multithreading programming*

Thread adalah unit dasar penggunaan *Central Processing Unit* (CPU). *Thread* memiliki ID *thread*, *program counter*, sebuah set *register* dan sebuah *stack.* Satu proses dapat memiliki satu atau lebih *thread*. Jika sebuah proses memiliki lebih dari satu *thread*, proses tersebut dapat melakukan lebih dari satu pekerjaan dalam satu waktu.

Misalkan terdapat sebuah *file server* yang melayani permintaan *client* untuk melakukan transfer data. Sebuah *file server* yang ramai dapat saja memiliki banyak user yang mengakses *file server* tersebut secara bersamaan. Jika *file server* tersebut hanya menggunakan satu *thread*, *file server* tersebut hanya dapat melayani satu *user* saja pada satu waktu. Salah satu solusi adalah dengan membuat *server* tersebut menjalankan sebuah proses yang menerima permintaan. Ketika terdapat sebuah permintaan, *server* tersebut akan membuat proses yang terpisah untuk melayani permintaan tersebut. Namun pembuatan proses ini memakan waktu dan *resource* sehingga tidak efisien. Solusi yang lebih baik

adalah dengan membuat proses tersebut memiliki banyak *thread* untuk menerima permintaan.

Perbedaan proses *single-thread* dan *multithread* dapat dilihat pada Gambar 2.10 berikut.



Gambar 2.10. Proses single-threaded dan multithreaded

Sumber : Wikipedia (2009)

Beberapa keuntungan menggunakan *multithread :*

- Responsif

  Sebuah aplikasi yang menggunakan *multithread* memungkinkan aplikasi tetap berjalan dengan baik walaupun ada bagian dari aplikasi yang melakukan proses yang memakan waktu lama. Misalkan sebuah *web browser* dapat melakukan interaksi dengan *user* pada satu *thread* sedangan *thread* yang lain yang menampilkan gambar.

- *Resource sharing*

  Thread-thread pada aplikasi berbagi memori dan *resource* proses yang dimilikinya.

- *Economy*

  Lebih baik untuk membuat *thread-thread* yang berjalan pada satu proses dibandingkan dengan banyak proses yang berjalan bersamaan karena

pembuatan proses memakan waktu dan *resource* yang lebih banyak dibandingkan dengan pembuata *thread.*

- *Utilization of multiprocessor architectures*

  Keuntungan *multithreading* akan lebih terlihat pada *multiprocessor,* dimana *thread* dapat berjalan pada *processor* yang berbeda. Sebuah proses yang bersifat *single-thread* hanya dapat dijalankan pada satu *Cental Processing Unit* (CPU), walaupun CPU yang dimiliki lebih dari satu.