

Mengatur Bandwidth menggunakan ALTQ (FreeBSD)

Abstraksi

ALTQ adalah sebuah paket untuk mengatur trafik, meliputi framework antrian dan beberapa disiplin antrian lanjut seperti CBQ, RED, WFQ dan RIO. ALTQ juga mendukung RSVP dan diffserv. ALTQ dapat juga dikonfigurasi dalam berbagai cara baik untuk penelitian dan juga operasi. Bagaimanapun menggunakan ALTQ memerlukan pengertian teknis untuk meng'setup secara benar.

1. Dasar Antrian

Pada pokoknya, setiap skema pengatur trafik meliputi pengatur antrian. Sejumlah besar disiplin antrian disarankan untuk mencapai persyaratan seperti kelayakan, perlindungan, batas performansi, kemudahan implementasi atau administrasi.

1.1 Komponen Antrian

Gambar 1, mengilustrasikan antrian yang berhubungan dengan blok-blok fungsional pada suatu router. Tiap blok fungsional akan dibutuhkan untuk membangun layanan tertentu namun tidak juga selalu diperlukan untuk layanan lainnya. Kenyataannya, pada kebanyakan router yang digunakan saat ini tidak menggunakan seluruh blok fungsional.

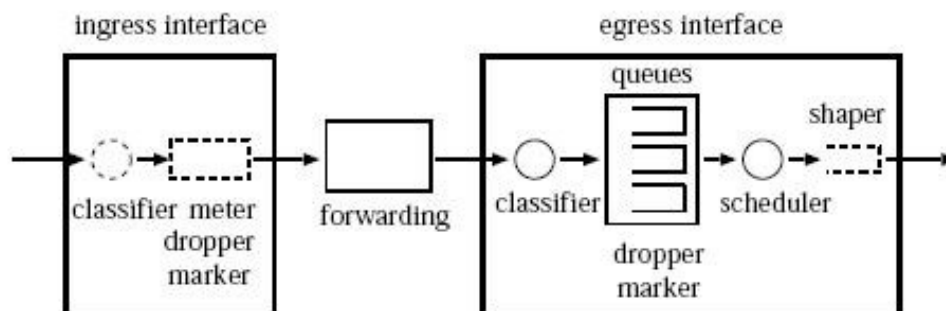


Figure 1: Queueing Architecture

Paket-paket berdatangan pada satu interface router (*ingress interface*), lalu diteruskan ke interface lainnya (*egress interface*). Sebuah router mempunyai blok fungsional pada ingress interface untuk menjaga paket-paket yang datang, namun blok fungsional yang utama berada di *egress interface*. Fungsi dari tiap blok di jelaskan berikut.

Classifier, *Paket classifier* menggolongkan paket-paket berdasarkan dari isi dari beberapa bagian pada suatu header paket (e.g , pengalamatan dan nomor-nomor port). Paket-paket disesuaikan dengan aturan tertentu untuk proses lebih lanjut.

Meters, *Traffic meters* mengukur sifat aliran trafik (e.g., bandwidth, menghitung paket). Hasil pengukuran karakteristik disimpan sebagai flow state dan digunakan oleh fungsi lain.

Markers, *packet markers* mengumpulkan nilai akurat beberapa bagian dari header paket. Nilai yang dicatat menjadi prioritas, informasi kongesti, tipe aplikasi, atau tipe informasi lainnya.

Droppers, *droppers* menggantikan beberapa atau seluruh paket dalam aliran trafik dengan tujuan untuk membatasi panjang antrian, atau sebagai informasi kongesti yang lengkap.

Queues, *queues* adalah buffer-buffer terhingga untuk menyimpan paket-paket backlogged. Suatu disiplin antrian mungkin dapat mempunyai antrian ganda untuk kelas trafik yang berbeda.

Schedulers, *schedulers* memilih paket untuk dikirimkan dari paket backlogged di antrian.

Shapers, *shapers* menunda beberapa atau seluruh paket di aliran trafik dengan maksud untuk membatasi laju puncak dari aliran. Suatu shaper biasanya memiliki ukuran buffer terhingga, dan paket mungkin saja ditukar jika tidak ada ruang yang cukup untuk menahan paket yang ditunda.

Disiplin antrian secara umum didefinisikan sebagai satu set blok fungsional pada *egress interface*, dan biasanya terdiri dari struktur antrian yang spesifik, mekanisme penjadwalan dan suatu mekanisme dropper. Blok fungsional yang dijelaskan disini adalah konseptual dan memungkinkan variasi kombinasi yang luas.

Disiplin Antrian

Pengalokasian bandwidth salah satu hal yang paling penting dalam disiplin antrian. Alokasi bandwidth yang rata dapat dicapai dengan menggunakan disiplin antrian yang tepat. Mekanisme yang sama juga aliran yang keliru, dengan demikian menjaga trafik lainnya.

Tujuan penting lainnya adalah untuk mengontrol waktu tunda dan *jitter* (variasi waktu acak) dimana merupakan hal yang penting untuk membangun aplikasi-aplikasi yang *real-time*. Hal yang mungkin untuk membatasi waktu tunda dan *jitter* dari suatu aliran dengan mengembalikan resource jaringan yang dibutuhkan. Izin kontrol diperlukan untuk memutuskan apakah resource yang diminta dapat di alokasikan. Juga diperlukan untuk mengatur laju aliran balik dalam arti pembentukan. Laju kedatangan haruslah lebih rendah dari laju kepulangan (balik) untuk menghindari waktu tunda yang disebabkan oleh aliran trafik sendiri. Suatu *leaky bucket* adalah mekanisme sederhana *shaper* dengan ukuran buffer yang terhingga. Mekanisme *shaper* lain yang populer adalah *token bucket* yang membolehkan ledakan (*burst*) kecil dengan ukuran ledakan yang dapat di konfigurasi. *Token bucket* dapat mengakomodasi aliran-aliran trafik dengan karakteristik ledakan sehingga lebih sesuai untuk trafik internet saat ini.

Tujuan lain disiplin antrian adalah menghindari kongesti. TCP mempertimbangkan *packet loss* sebagai tanda kongesti. Suatu router dapat memberitahukan TCP kongesti secara lengkap dengan meng'drop paket secara sengaja. Daftar berikut menjelaskan beberapa disiplin antrian.

FIFO, disiplin antrian yang paling sederhana adalah **FIFO** (*First-In-First-Out*) yang hanya mempunyai antrian tunggal dan sebuah drop-tail dropper sederhana.

PQ (*Priority Queueing*) memiliki antrian-antrian ganda yang diasosiasikan dengan prioritas berbeda. Sebuah antrian dengan prioritas tertinggi selalu di layani terlebih dahulu. Antrian prioritas merupakan bentuk tersederhana dari antrian preferensial. Bagaimanapun, trafik prioritas rendah mudah rusak hingga ada mekanisme untuk mengatur trafik prioritas tinggi.

WFQ (*Weighted Fair Queueing*) adalah suatu disiplin antrian yang menandai suatu antrian bebas untuk tiap aliran. WFQ dapat memberikan alokasi bandwidth yang rata dalam waktu kongesti, dan melindungi suatu aliran dari aliran yang lain. Bobot dapat ditandai ke tiap antrian untuk memberikan proporsi berbeda dalam suatu kapasitas jaringan.

SFQ (*Stochastic Fairness Queueing*) adalah aproksimasi dari WFQ. WFQ termasuk sukar untuk di implementasikan karena besarnya jumlah antrian yang diperlukan sebagai jumlah dari bertambahnya aliran-aliran. Pada SFQ, fungsi hash digunakan untuk memetakan suatu aliran ke satu antrian yang telah fix, dan memungkinkan untuk dua aliran berbeda di petakan ke dalam antrian yang sama.

CBQ (*Class Based Queueing*), dapat mencapai penyekatan dan pembagian link bandwidth dengan struktur golongan. Tiap golongan mempunyai antrian tersendiri dan ditandai, dimana juga membagi bandwidth. CBQ dapat mengatur penggunaan bandwidth dari suatu golongan. Golongan *'child'* dapat di konfigurasi untuk meminjam bandwidth dari golongan induknya selama kelebihan bandwidth tersedia.

RED (*Random Early Detection*) adalah mekanisme dropper yang menurut ke rata-rata panjang antrian. RED menghindari sikronisasi trafik dimana paket hilang TCP dalam satu waktu. RED juga membuat TCP menyimpan antrian pendek. RED dapat berlaku adil dalam arti paket tersebut di drop dari aliran-aliran dengan probabilitas yang proporsional ke buffer mereka. RED tidak memerlukan status per-aliran, bersifat skala dan cocok untuk *backbone routers*.

2. Instalasi ALTQ

FreeBSD yang saya pakai adalah FreeBSD 5.2.1-RELEASE sehingga memerlukan patch pada kernel untuk instalasinya. Dan untuk patch FreeBSD 5.2.1-RELEASE digunakan *altq-freebsd-5.2-release-beta2.tar.gz*, untuk patch harus disesuaikan dengan versi FreeBSD yang anda gunakan, sedangkan patch bisa download di <http://www.rofug.ro/projects/freebsd-altq/> . Bila anda menggunakan FreeBSD 5.3, ALTQ sudah terintegrasi pada kernel sehingga tidak diperlukan patch, anda bisa langsung loncat ke tahap no. 4. Tahapan Instalasi sebagai berikut:

1. **Pastikan kernel sudah ter'install**, periksa direktori */usr/src/sys*. Jika kernel belum terinstall, anda bisa gunakan program *sysinstall* untuk instalasi kernel (tutorial lengkap bagaimana instalasi kernel ada di <http://purwakarta.org/flash/freebsdgw.pdf>).

2. **Download dan ekstrak patch kernel** (sesuaikan dengan versi FreeBSD).

```
#fetch http://purwakarta.org/flash/freebsd/altq-freebsd-5.2-release-beta2.tar.gz
#tar zxvf altq-freebsd-5.2-release-beta2.tar.gz -C /usr/src
```

3. **Patching kernel**, ubah user menjadi root

Kita simpan kernel baru pada direktori */usr/src/sys-altq* :

```
#su
password:
#cd /usr/src
#mkdir sys-altq
#tar cvf - . | (cd ../sys-altq; tar xf -)
```

Patch kernel :

```
#cd /usr/src/sys/sys-altq
# patch -p0 < /usr/src/altq-freebsd-5.2-release-beta2/sys-altq/sys-altq-freebsd-5.2-release.diff
#cp -rf /usr/src/altq-freebsd-5.2-release-beta2/sys-altq/altq/ .
```

Jika versi kernel anda tidak didukung (e.g, -stable), coba dengan patch terdekat ke FreeBSD versi anda. Perintah “*patch*” mungkin saja gagal mem'patch beberapa file. Anda dapat mencari file yang gagal dengan perintah berikut:

```
#find . -name “*.rej” -print
```

Anda juga perlu untuk mengedit file config kernel ALTQ.

4. Edit Konfigurasi Kernel ALTQ

```
#cd i386/conf
#cp ALTQ ALTQ.asli
Backup file konfigurasi kernel.
```

```
#vi ALTQ
```

Pilihlah metoda antrian yang akan anda gunakan ,

```
# ALTQ options
options    ALTQ                # Alternate queueing
options    ALTQ_CBQ         # Class Based Queueing
options    ALTQ_WFQ        # Weighted Fair Queueing
options    ALTQ_FIFOQ      # FIFO queueing
options    ALTQ_RED         # Random Early Detection
options    ALTQ_FLOWVALVE # Flowvalve for RED (needs RED)
options    ALTQ_RIO        # Triple RED for DiffServ (needs RED)
options    ALTQ_LOCALQ     # Local use
options    ALTQ_HFSC       # Hierarchical Fair Service Curve
options    ALTQ_IPSEC      # Check IPSEC in IPv4
options    ALTQ_CDNR       # DiffServ traffic conditioner
options    ALTQ_BLUE       # Blue by Wu-Chang Feng
options    ALTQ_PRIQ       # Priority Queue
options    ALTQ_NOPCC      # Don't use Processor Cycle Counter
                                # Enable this for SMP
#options    ALTQ_DEBUG     # For debugging
```

Save file, kemudian konfigurasi, kompilasi dan install kernel baru. Setelah proses instalasi kernel baru selesai, FreeBSD harus direstart agar dapat menggunakan kernel yang baru.

```
#config ALTQ  
#cd ../../compile/ALTQ  
#make depend  
#make  
#make install  
#shutdown -r now
```

5. Membuat ALTQ devices

```
# cd /usr/src/altq-freebsd-5.2-release-beta2/  
#sh MAKEDEV.altq all
```

6. Memuat module ALTQ

Module kernel altq dibuat secara otomatis dan terinstall menjadi bagian dari kernel. Untuk membuat module altq KLD adalah :

```
#cd /usr/src/sys-altq/modules/altq  
#make  
#make install
```

Sehingga kini anda dapat dengan mudah memuat module dengan perintah:

```
#kldload altq_cbq
```

Untuk melihat module yang sudah termuat di kernel :

```
#kldstat -v
```

Untuk melepas (unload) module :

```
#kldunload altq_cbq
```

7. Instalasi tools ALTQ

```
# cd /usr/src/altq-freebsd-5.2-release-beta2/  
#make  
#make install
```

8. Konfigurasi file altq.conf

Agar lebih mudah memahami bagaimana membuat konfigurasi altq.conf

kita lihat pada contoh direktori `altq.conf.samples`. Contoh file `cbq.bandwidthtest` :

```
# this is the setting used for the bandwidth guarantee test (Fig. 7)
# in the ALTQ paper
#
interface en0 bandwidth 134M cbq
class cbq en0 root_class NULL priority 0 admission none pbandwidth 100
class cbq en0 def_class root_class borrow priority 2 pbandwidth 95 default
#
class cbq en0 tcp_class0 def_class priority 3 pbandwidth 8
filter en0 tcp_class0 0 0 0 6790 6
class cbq en0 tcp_class1 def_class priority 3 pbandwidth 16
filter en0 tcp_class1 0 0 0 6791 6
class cbq en0 tcp_class2 def_class priority 3 pbandwidth 24
filter en0 tcp_class2 0 0 0 6792 6
class cbq en0 tcp_class3 def_class priority 3 pbandwidth 32
filter en0 tcp_class3 0 0 0 6793 6
```

Pembahasan :

Perintah Interface :

```
interface if_name [bandwidth bps] [tbrsize bytes] [sched_type] [discipline-specific-options]
```

```
interface en0 bandwidth 134M cbq
```

- perintah *interface* menentukan interface network mana yang akan diatur oleh ALTQ.
- opsi *bandwidth* menentukan bandwidth (bits per second) pada interface tadi. Laju maksimum yang dibolehkan pada interface.
- *cbq*, tipe dari disiplin antrian. Dapat di ganti dengan *blue*, *cbq*, *fifoq*, *hsfc*, *priq*, *red*, *rio*, atau *wfq*.

Perintah Class:

```
class sched_type if_name class_name parent_name [red/rio] [ecn] [cleardscp] [discipline-specific-options]
```

```
class cbq en0 root_class NULL priority 0 admission none pbandwidth 100
```

Perintah Class yang lebih lengkap :

```
class sched_type if_name class_name parent_name [admission cntlload|none] [priority pri]
[bandwidth percent] [exactbandwidth bps] [borrow] [default] [control] [maxburst count]
[minburst count] [maxdelay msec] [packetize bytes] [maxpacketize bytes] [red/rio] [ecn]
[flowvalve] [cleardscp]
```

- perintah *class* menentukan kelas penjadwalan paket untuk *CBQ*, *HSFC*, atau *PRIQ*.
- *root_class* merupakan nama untuk kelas'nya (*class_name*), penamaan harus unik.
- *parent_class* harus didefinisikan terlebih dahulu, *PRIQ* tidak mempunyai susunan kelas dan *parent_name* haruslah di isi NULL untuk kelas *PRIQ*.
- *priority*, prioritas tertinggi adalah nilai terbesar. Nilai maksimum adalah 7 dan minimum 0, defaultnya 1.
- *admission*, tipe pengontrolan untuk perizinan dan tipe QoS. *cntlload* (controlled load service) untuk *RSVP*, sebaliknya *admission* haruslah di isi *NONE*. Defaultnya adalah *none*.
- *percentase* bandwidth interface yang di alokasikan untuk kelas ini. Biasanya dapat naik ke 100 persen pada tiap level susunan kelas, walaupun jumlah lainnya dapat di spesifikasikan guna eksperimen.
- *borrow*, kelas tersebut dapat meminjam bandwidth dari kelas induknya ketika kelas tersebut melebihi batas. Jika opsi “borrow” tidak dinyatakan maka ketika terjadi overlimit maka paket akan ditunda atau di drop.
- *exactbandwidth*, mengspesifikasikan bandwidth dalam bits-per-second sebagai alternatif *pbandwidth*.
- *default*, kelas default. Ketika opsi ini dinyatakan maka seluruh paket yang tidak sesuai dengan beberapa kriteria akan ditandai ke kelas ini. Sebaiknya satu kelas pada tiap interface di definisikan sebagai kelas default.
- *control*, kelas control. Ketika opsi ini dinyatakan (*RSVP*, *IGMP*, dan *ICMP*) ditandai ke kelas ini. Ketika kelas control tidak dispesifikasikan maka kelas default akan di buat dengan parameter default. Jika kelas control dispesifikasikan, ia harus di daftar sebelum kelas default. Sebaiknya hanya satu kelas pada tiap interface sebagai kelas control.
- *maxburst*, maksimal ledakan (burst) paket back-to-back yang dibolehkan pada kelas ini. Defaultnya 16 namun nilai default otomatis berkurang ke 4 ketika bandwidth kecil (sekitar 1 Mbps).
- *minburst*, minimum burst digunakan untuk memperoleh keadaan konstan. Parameter ini berguna untuk membantu menghitung “offtime” untuk kelas'nya. Offtime adalah banyaknya waktu suatu kelas untuk menunggu diantara paket-paket. Default nya adalah 2.
- *maxdelay*, menspesifikasikan dalam millisecond dan digunakan untuk mendapatkan ukuran maksimal antrian dari kelas tersebut. Jika tidak ditentukan nilai yang digunakan adalah 30 packets.
- *packetsize*, rata-rata ukuran paket dalam bytes untuk digunakan pada *CBQ* penghitungan-penghitungan over-/under-limit. Nilai defaultnya adalah MTU dari interface.
- *maxpacketize*, ukuran paket maksimal dalam bytes untuk kelas'nya. Nilai

default'nya adalah MTU dari interface.

- **red**, meng'enable kan RED pada kelas antriannya.
- **rio**, , meng'enable kan RIO pada kelas antriannya.
- **ecn**, meng'enable kan RED/ECN pada kelas antriannya.
- **flowvalve**, meng'enable kan RED/flow-valve (a.ka red-penalty-box) pada kelas antriannya.
- **cleardscp**, membersihkan “*diffserv codepoint*” pada header IP.

Perintah filter :

```
filter if_name class_name [name fltr_name] [ruleno num] filter_values
```

```
filter_values: dst_addr [netmask mask] dport src_addr [netmask mask] sport proto [tos value  
[tosmask value]] [gpi value]
```

```
filter en0 tcp_class 0 0 0 0 6790 6
```

- Perintah filter untuk menggolongkan paket-paket ke dalam penjadwalan kelas.
- Nilai filter 0 digunakan sebagai wildcard.
- dsport dan sport, nomor port dari tujuan dan nomor port sumber.
- Protocol number = 6, dimana 6 berarti TCP.

Untuk konfigurasi lebih lanjut berikut contoh file altq.conf :

CBQ Example

```
#  
# cbq configuration for vx0 (10Mbps ether)  
# give at least 40% to TCP  
# limit HTTP from network 133.138.1.0 up to 10%, use RED.  
# other traffic goes into default class  
#  
interface vx0 bandwidth 10M cbq  
#  
class cbq vx0 root_class NULL priority 0 pbandwidth 100  
class cbq vx0 def_class root_class borrow pbandwidth 95 default  
class cbq vx0 tcp_class def_class borrow pbandwidth 40  
    filter vx0 tcp_class 0 0 0 0 6  
class cbq vx0 csl_class tcp_class pbandwidth 10 red  
    filter vx0 csl_class 0 0 133.138.1.0 netmask 0xffffffff 80 6  
    filter vx0 csl_class 133.138.1.0 netmask 0xffffffff 0 0 80 6  
#  
# sample filter6 command  
#  
    filter6 vx0 csl_class ::0 0 d000:a:0:123::/64 80 6
```

HFSC Example

```
#  
# hfsc configuration for hierachical sharing  
#  
interface pvc0 bandwidth 45M hfsc  
#  
# (10% of the bandwidth share goes to the default class)  
class hfsc pvc0 def_class root pshare 10 default  
#  
#    bandwidth share    guaranteed rate
```

```

#   CMU:      45%          15Mbps
#   PITT:     45%          15Mbps
#
class hfsc pvc0 cmu   root pshare 45 grate 15M
class hfsc pvc0 pitt root pshare 45 grate 15M
#
# CMU      bandwidth share   guaranteed rate
#   CS:    20%              10Mbps
#   other: 20%              5Mbps
#
class hfsc pvc0 cmu_other cmu   pshare 20 grate 10M
      filter pvc0 cmu_other 0 0 128.2.0.0 netmask 0xffff0000 0 0
class hfsc pvc0 cmu_cs   cmu   pshare 20 grate 5M
      filter pvc0 cmu_cs   0 0 128.2.242.0 netmask 0xfffff00 0 0
#
# PITT     bandwidth share   guaranteed rate
#   CS:    20%              10Mbps
#   other: 20%              5Mbps
#
class hfsc pvc0 pitt_other pitt  pshare 20 grate 10M
      filter pvc0 pitt_other 0 0 136.142.0.0 netmask 0xffff0000 0 0
class hfsc pvc0 pitt_cs   pitt  pshare 20 grate 5M
      filter pvc0 pitt_cs   0 0 136.142.79.0 netmask 0xfffff00 0 0

```

PRIQ Example

```

#
# priq configuration for fxp0 (100Mbps ether)
#   icmp: high priority
#   tcp:  medium priority
#   others: low priority
#
interface fxp0 bandwidth 100M priq
#
class priq fxp0 high_class NULL priority 2
      filter fxp0 high_class 0 0 0 0 1
class priq fxp0 med_class NULL priority 1
      filter fxp0 high_class 0 0 0 0 6
class priq fxp0 low_class NULL priority 0 default

```

WFQ Example

```
interface pvc0 bandwidth 134000000 wfq
```

FIFOQ Example

```
interface rl0 bandwidth 10M fifoq
```

Conditioner Example

```

#
interface fxp0
#
# a simple dropper
# discard all packets from 192.168.0.83
#
conditioner fxp0 dropper <drop>
      filter fxp0 dropper 0 0 192.168.0.83 0 0

#
# EF conditioner
# mark EF to all packets from 192.168.0.117
#
conditioner pvc1 ef_cdnr <tbmeter 6M 64K <mark 0xb8><drop>>
      filter fxp0 ef_cdnr 0 0 192.168.0.117 0 0
#
class priq fxp0 high_class NULL priority 2
      filter fxp0 high_class 0 0 0 0 1
class priq fxp0 med_class NULL priority 1
      filter fxp0 high_class 0 0 0 0 6
class priq fxp0 low_class NULL priority 0 default

```

WFQ Example

```
interface pvc0 bandwidth 134000000 wfq
```

FIFOQ Example

```
interface rl0 bandwidth 10M fifoq
```

Conditioner Example

```

#
interface fxp0
#
# a simple dropper
# discard all packets from 192.168.0.83
#
conditioner fxp0 dropper <drop>
    filter fxp0 dropper 0 0 192.168.0.83 0 0

#
# EF conditioner
# mark EF to all packets from 192.168.0.117
#
conditioner pvc1 ef_cdnr <tbmeter 6M 64K <mark 0xb8><drop>>
    filter fxp0 ef_cdnr 0 0 192.168.0.117 0 0

#
# AF1x conditioner
# mark AF1x to packets from 192.168.0.178
#
# AF11 (low drop precedence): less than 3Mbps
# AF12 (medium drop precedence): more than 3Mbps and less than 10Mbps
# AF13 (high drop precedence): more than 10Mbps
#
conditioner fxp0 af1x_cdnr <trtcm 3M 32K 10M 64K <mark 0x28><mark 0x30><mark 0x38>>
    filter fxp0 af1x_cdnr 0 0 192.168.0.178 0 0

```

9. Menjalankan altqd

```
#altqd -d -f/etc/altq.conf
```

Untuk melihat statistik altqd gunakan perintah altqstat.

```
#altqstat -w 1
```

Referensi :

- Arsip di Internet .
- altq.conf manual.
- Managing Traffic with ALTQ, Kenjiro Cho, Sony Computer Science Laboratories, Inc. Tokyo, Japan 1410022.

Mohon maaf bila masih banyak kekurangan dan kesalahan.

Semoga Bermanfaat.

Faiz.

Saran & Kritik :

faiz@purwakarta.org