

YILDIZ TEKNİK ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

Tcp/Ip Pro., Dif. Ser. Mod. göre
Qua. of Ser. Sağ. bir Sist. Tas. ve Ger.

YÜKSEK LİSANS TEZİ

G. Eda Ağar

2005

**YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ**

**TCP/IP PROTOKOLÜNDE “DIFFERENTIATED
SERVICES” MODELİNE GÖRE “QUALITY OF SERVICE”
SAĞLAYACAK BİR SİSTEMİN TASARLANMASI ve
GERÇEKLENMESİ**

Bilgisayar Müh. Güneş Eda AĞAR

**FBE Bilgisayar Mühendisliği Anabilim Dalında
Hazırlanan**

YÜKSEK LİSANS TEZİ

Tez Danışmanı : Yrd. Doç. Dr. A. Gökhan YAVUZ (YTÜ)

İSTANBUL, 2005

YILDIZ TEKNİK ÜNİVERSİTESİ
KÜTÜPHANE VE DOKÜMANTASYON
DAİRE BAŞKANLIĞI

Yer No (DDC): *R.368 -106*

Kayıt No

2719

İSVEZİG!

Geldiği Yer

Fen Bilim Enst.

Tarih

19.10.05

Fiyat

İKT

3.80-

Fatura No

Ayniyat No

1/16

Ek

(Y) SÜV



YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

X1-76

TCP/IP PROTOKOLÜNDE “DIFFERENTIATED
SERVICES” MODELİNE GÖRE “QUALITY OF SERVICE”
SAĞLAYACAK BİR SİSTEMİN TASARLANMASI ve
GERÇEKLENMESİ

Bilgisayar Müh. Güneş Eda AĞAR

FBE Bilgisayar Mühendisliği Anabilim Dalında
Hazırlanan

YÜKSEK LİSANS TEZİ

Tez Danışmanı

Y. Dr. A. Gökhan YAVUZ (YTÜ)

RD. Doç. Dr. A. TEVFIK İNAN

İSTANBUL, 2005

Prof. Dr. Zülent Örenç

İÇİNDEKİLER

	Sayfa
KISALTMA LİSTESİ.....	v
ŞEKİL LİSTESİ.....	vi
ÇİZELGE LİSTESİ.....	vii
BAĞINTI LİSTESİ.....	viii
ÖNSÖZ.....	ix
ÖZET.....	x
ABSTRACT.....	xi
1. GİRİŞ.....	1
2. INTEGRATED SERVICES (INTSERV).....	2
2.1 Integrated Services Yapısı.....	2
2.2 RSVP.....	3
2.3 Garanti Edilmiş Servis (Guaranteed Service).....	6
2.4 Kontrollü Yük Servisi (Controlled Load Service).....	7
2.5 “Leaky Bucket” ve “Token Bucket” Trafik Şekillendirme Algoritmaları.....	7
2.6 Integrated Services ve Differentiated Services Karşılaştırması.....	9
3. DIFFERENTIATED SERVICES (DIFFSERV).....	11
3.1 Differentiated Services Alanı.....	11
3.2 Differentiated Services Yapısı.....	13
3.3 Per-Hop-Behaviour.....	15
4. LINUX DIFFSERV DESTEĞİ.....	18
4.1 Geleneksel Trafik Kontrol Mekanizmaları ile Linux Trafik Kontrol Bileşenleri Arasındaki İlişki.....	19
4.1.1 Biçimlendirme (Shaping).....	19
4.1.2 Zamanlama (Scheduling).....	20
4.1.3 Sınıflandırma (Classifying).....	20
4.1.4 Paket Limitleme (Policing).....	20
4.1.5 Paket Reddetme (Dropping).....	21
4.1.6 İşaretleme (Marking).....	21
4.2 Linux’taki Trafik Kontrol Bileşenleri.....	21
4.2.1 Kuyruk Disiplinleri.....	21
4.2.1.1 Sınıflı Kuyruk Disiplinleri.....	22
4.2.1.2 Sınıfsız Kuyruk Disiplinleri.....	27
4.2.2 Filtreler.....	30

5.	TASARIM.....	31
5.1	DiffServ Konfigürasyonu.....	31
5.1.1	Kenar Yönlendirici Konfigürasyonu.....	32
5.1.2	İç Yönlendirici Konfigürasyonu	32
5.2	İstemci-Sunucu Uygulamaları Arasındaki İletişim.....	34
6.	UYGULAMA	36
6.1	LDAP Dizin Sunucusu.....	36
6.1.1	Veri Şeması	37
6.1.2	LDAP Dizin Sunucusuna Erişim Fonksiyonları	40
6.1.2.1	my_ldap_init Fonksiyonu	40
6.1.2.2	ldap_final Fonksiyonu.....	40
6.1.2.3	ldap_get_attributeFonksiyonu.....	40
6.2	İç Yönlendirici Konfigürasyonu	40
6.2.1	İç Yönlendirici İlk Konfigürasyon Kodu	40
6.2.2	İç Yönlendirici Konfigürasyon Güncelleme Kodu	41
6.2.3	İç Yönlendirici Sunucu Kodu	42
6.2.3.1	set_dsmark Fonksiyonu.....	42
6.2.3.2	set_phb Fonksiyonu	42
6.2.3.3	modify_phb Fonksiyonu	43
6.2.3.4	set_af_class Fonksiyonu	43
6.2.3.5	delete_af_class Fonksiyonu	43
6.2.3.6	modify_af_class Fonksiyonu	43
6.3	Kenar Yönlendirici Konfigürasyonu.....	43
6.4	Trafik Kontrol Fonksiyonları	43
6.4.1	modify_queue Fonksiyonu.....	44
6.4.2	modify_filter Fonksiyonu	44
6.4.3	modify_cbq_class Fonksiyonu.....	44
6.4.4	modify_htb_class Fonksiyonu	44
7.	TEST	45
8.	SONUÇ	57
	KAYNAKLAR.....	59
	İNTERNET KAYNAKLARI.....	61
	EKLER	62
Ek 1	“TC” komutları ile iç yönlendirici konfigürasyon komutları.....	62

Ek 2	Dizin sunucusu fonksiyonları.....	62
Ek 3	Veri yapıları	62
Ek 4	Trafik kontrol fonksiyonları.....	62
ÖZGEÇMİŞ		76

CSZ	Class-Specific Zoning
Diffsrv	Differentiated Services
DS	Differentiated Services
DSOP	Diffsrv Code Point
DSMARK	Diffsrv field marker
EF	Expedited Forwarding
ESFO	Extended Stochastic Fair Queuing
EWMA	Exponential Weighted Moving Average
FIFO	First In First Out
GRED	Generalized Random Early Detection
HTB	Hierarchical Token Bucket
IETF	Internet Engineering Task Force
IntServ	Integrated Services
LDAP	Lightweight Directory Access Protocol
PHB	Per Hop Behavior
QoS	Quality of Service
RED	Random Early Detection
RFC	Request for Comments Protocol
SFO	Stochastic Fair Queuing
TBF	Token Bucket Filter
TC	Traffic Control
TEQL	Traffic Equilibrium
Tos	Type of Service
WRR	Weighted Round Robin

KISALTIMA LİSTESİ

AF	Assured Forwarding	3
ATM	Asynchronous Transfer Model	5
BE	Best Effort	5
CBQ	Class Based Queuing	5
CSZ	Clak-Scott-Zhang	9
DiffServ	Differentiated Services	11
DS	Differentiated Services	13
DSCP	DiffServ Code Point	13
DSMARK	DiffServ field marker	14
EF	Expedited Forwarding	14
ESFQ	Extended Stochastic Fair Queuing	14
EWMA	Exponential Weighted Moving Average	17
FIFO	First In First Out	17
GRED	Generalized Random Early Detection	18
HTB	Hierarchical Token Bucket	19
IETF	Internet Engineering Task Force	27
IntServ	Integrated Services	24
LDAP	Lightweight Directory Access Protocol	24
PHB	Per Hop Behaviour	25
QoS	Quality of Service	25
RED	Random Early Detection	26
RSVP	Resource Reservation Protocol	28
SFQ	Stochastic Fair Queueing	28
TBF	Token Bucket Filter	28
TC	Traffic Control	34
TEQL	Traffic Equaliser	39
ToS	Type of Service	39
WRR	Weighted Round Robin	39

ŞEKİL LİSTESİ

Şekil 2.1 İstemci sistemi ve yönlendirici sisteminde RSVP (RFC 2205) [Braden vd., 1997]... 3	3
Şekil 2.2 Kaynak rezervasyonu için RSVP sinyalleşmesi [Bai vd.] 5	5
Şekil 2.3 Akış Tanımlayıcı ve bileşenleri 5	5
Şekil 2.4 “Leaky Bucket” ve “Token Bucket” algoritmaları [Tanenbaum, 2002]..... 9	9
Şekil 3.1 RFC 1349’da tanımlanan ToS oktet yapısı 11	11
Şekil 3.2 Differentiated Services alanı 13	13
Şekil 3.3 DiffServ – Temel Yapısı [Majalainen, 2002] 14	14
Şekil 3.4 Paket sınıflandırıcısı ve trafik koşullandırma elemanları 14	14
Şekil 3.5 DS alanındaki bitlerin anlamları 17	17
Şekil 4.1 Linux’ta paket işleme akışı [2]..... 18	18
Şekil 4.2 Kuyruk disiplinleri, sınıflar ve filtreler 19	19
Şekil 4.3 DSMARK ile paket işaretleme [Balliache, 2003]..... 27	27
Şekil 5.1 İç yönlendirici DiffServ konfigürasyonu 34	34
Şekil 5.2 İstemci sunucu arasındaki zamana bağlı iletişim..... 35	35
Şekil 6.1 Örnek DS alanı ortamı 36	36
Şekil 6.2 LDAP şeması 38	38
Şekil 6.3 “interface” nesne sınıfının özellikleri 38	38
Şekil 6.4 “coreConfig” ve “coreTrans” nesne sınıflarının özellikleri 39	39
Şekil 6.5 “edgeConfig” ve “edgeTrans” nesne sınıflarının özellikleri..... 39	39
Şekil 7.1 Test Yerel Ağı (I)..... 45	45
Şekil 7.2 Test Yerel Ağı (II)..... 53	53
Şekil Ek3.1 packet_info veri yapısı..... 70	70
Şekil Ek3.2 pkt_type veri yapısı 70	70
Şekil Ek3.3 device_info veri yapısı..... 70	70
Şekil Ek3.4 phb_info veri yapısı 71	71
Şekil Ek3.5 phb_type veri yapısı 71	71
Şekil Ek3.6 request_type veri yapısı 71	71
Şekil Ek3.7 be_info veri yapısı 71	71
Şekil Ek3.8 af_class_info veri yapısı 72	72
Şekil Ek3.9 router_info veri yapısı 72	72

ÇİZELGE LİSTESİ

Çizelge 3.1 Öncelik (Precedence) alanı değerleri	12
Çizelge 3.2 Servis tipi (ToS) alanı değerleri	12
Çizelge 3.3 DSCP değerleri [Manner, 2003]	13
Çizelge 3.4 ToS ve DSCP değerleri	17
Çizelge 6.1 Ek AF Sınıfları	42
Çizelge 7.1 Test 1.1 sonuçları	46
Çizelge 7.2 Test 1.2 sonuçları	46
Çizelge 7.3 Test 1.3 sonuçları	47
Çizelge 7.4 Test 1.4 sonuçları	47
Çizelge 7.5 Test 1.5 sonuçları	47
Çizelge 7.6 Test 1.6 sonuçları	48
Çizelge 7.7 Test 1.7 sonuçları	49
Çizelge 7.8 Test 2.1 sonuçları	50
Çizelge 7.9 Test 2.2 sonuçları	50
Çizelge 7.10 Test 2.3 sonuçları	50
Çizelge 7.11 Test 2.4 sonuçları	51
Çizelge 7.12 Test 2.5 sonuçları	51
Çizelge 7.13 Test 2.6 sonuçları	51
Çizelge 7.14 Test 2.7 sonuçları	52
Çizelge 7.15 Test 3.1 sonuçları	54
Çizelge 7.16 Test 3.2 sonuçları	55
Çizelge 7.17 Test 3.3 sonuçları	55
Çizelge 7.18 Test 3.4 sonuçları	56
Çizelge 7.19 Test 3.5 sonuçları	56

BAĞINTI LİSTESİ

$C + \rho S = MS$ (2.1)	8
$ToS = (skb \rightarrow tc_index \& mask) deęer$ (4.1)	26
Yeni DS deęeri = (Eski DS deęeri & maske) deęer (4.2)	26
$P = probability * \frac{avg - min}{max - min}$ (4.3)	28
$limit \geq MTU * burst + max$ (4.4)	28

ÖNSÖZ

Her konuda bana destek olan aileme, yüksek lisans eğitimi ve tez geliştirme süresince bilgisini ve zamanını esirgemeyen tez danışmanım Yrd. Doç. Dr. A. Gökhan Yavuz'a, Prof. M. Yahya Karsligil'e ve bölüm başkanımız Prof. Oya Kalıpsız'a teşekkürü borç bilirim.

ÖZET

Günümüzde Internet “en iyiyi sağlama” (Best Effort) mantığına göre servis sağlamakta, ağ ortamındaki tüm paketlere eşit servis verilmektedir. Internet kullanımının ve birbirinden farklı özellikler gösteren ve farklı gereksinimlere ihtiyaç duyan uygulama ve veri tiplerinin artması, servis kalitesinin (Quality of Service-QoS) sağlanması ihtiyacını doğurmuştur. Tez çalışmasında farklı trafik tiplerine farklı servis kalitesi sağlamak amacıyla Internet Task Engineering Force (IETF) tarafından geliştirilmiş olan Differentiated Services modeli temel alınarak servis kalitesi sağlayacak bir sistem gerçekleştirilmiştir.

Trafik kontrol konfigürasyonlarının Linux tabanlı ağ yönlendiricilerine merkezi bir noktadan dağıtımı sağlanarak yönlendirici konfigürasyonlarının uzaktan otomatik olarak uygulanması gerçekleştirilmiştir. Linux’taki mevcut “tc” trafik kontrol arayüz programı, trafik kontrol elemanlarının konfigürasyon işlemlerinin komut satırından gerçekleştirilmesini sağlamaktadır. Trafik kontrol elemanlarının konfigürasyonunun doğrudan uygulama içinden yapılabilmesi için bir programlama arabirimi geliştirilmiştir. Trafik kontrol konfigürasyonlarının saklanması için merkezi bir LDAP dizin sunucusu kullanılmıştır.

Linux kernel’da ilk olarak gerçekleştirilen servis kalitesi yöntemlerinden biri olmasından dolayı trafik kontrolünde CBQ (Class Based Queuing) algoritmasının kullanımı yaygındır. 2001 yılı sonlarında Martin Devera tarafından CBQ algoritmasına alternatif olarak HTB (Hierarchical Token Bucket) algoritması geliştirilmiştir. DiffServ konfigürasyonu tasarlanırken CBQ ve HTB algoritmaları göz önüne alınmıştır. CBQ algoritması kullanılarak gerçekleştirilen konfigürasyonda, trafik tiplerine atanan bant genişliklerinde sapmalar görülmüştür. Bunun sonucunda CBQ yerine CBQ algoritmasından daha doğru değerler verdiği görülen HTB algoritması kullanımı tercih edilmiştir.

Anahtar Kelimeler : Linux, Servis Kalitesi, Differentiated Services, CBQ, HTB

ABSTRACT

Today's Internet only provides "Best Effort" service. Traffic is processed as quickly as possible, but there is no guarantee as to timeliness or actual delivery. The growth in Internet usage and the increase in the number and variety of applications having different service requirements, rapidly developed demands for service quality. Developed by the Internet Engineering Task Force (IETF), Differentiated Services (DiffServ) is a model which can be used to provide QoS (Quality of Service) in IP networks. In this thesis, with the aim to provide different service quality to different traffic types, a system that provides QoS based on "Differentiated Services" is implemented.

In the implemented system, Linux based routers can be configured remotely by automatic distribution and deployment of traffic control configurations. The "tc" traffic control program in the Linux kernel, provides an interface to configure traffic control elements in command line. In order to configure traffic control elements from within an application program, a programming interface has been developed. An LDAP directory server is used for centrally storing traffic control configurations.

Since CBQ (Class Based Queuing) algorithm is one of the firstly developed service quality methods in Linux kernel, CBQ algorithm is commonly used for traffic control. Near the end of year 2001, HTB (Hierarchical Token Bucket) algorithm was developed by Martin Devera as an alternative to CBQ. In this thesis, both CBQ and HTB algorithms have been considered. With the traffic control configuration designed using CBQ algorithm, it has been seen that there is an amount of deviation in the requested bandwidth allocations. As a result, HTB algorithm which gave better results in bandwidth usages, has been preferred to be used instead of CBQ.

Keywords : Linux, Quality of Service, Differentiated Services, CBQ, HTB

1. GİRİŞ

İnternet'in kullanımı artarken, uzaktan öğrenim, video konferans, e-ticaret, VoIP (Voice over IP) gibi yeni uygulamaların da geliştirilmesi ile daha yüksek bant genişliklerine gereksinim duyulmaktadır. Her uygulamanın bant genişliği, gecikme ve gecikme sürelerindeki değişim bakımlarından kendine özgü özellikleri mevcuttur. Farklı uygulamaların farklı gereksinimlere sahip olmasına rağmen, günümüzde İnternet "en iyiyi sağlama" (Best Effort) mantığına göre servis sağlamakta, ağ ortamındaki tüm paketlere eşit servis verilmektedir. Trafik mümkün olan en kısa sürede fakat zamanında iletim ve hedefe ulaşma garantisi olmadan işlem görmektedir. Bu durum çoğu uygulama için kabul edilebilir değildir.

İnternet kullanımının ve birbirinden farklı özellikler gösteren ve farklı gereksinimlere ihtiyaç duyan uygulama ve veri tiplerinin artması, servis kalitesi (Quality of Service-QoS) sağlanması ihtiyacını doğurmuştur. İnternet Task Engineering Force (IETF) servis kalitesi gereksinimi için farklı model ve mekanizmalar geliştirmiştir. Bunların belli başlıları arasında Integrated Services (IntServ) modeli ve Differentiated Services (DiffServ) modeli bulunmaktadır. Paket akışı başına sınıflandırmanın yapıldığı IntServ yönteminin tersine, DiffServ'de paketler "birleştirme (aggregation)" yöntemi ile sınıflandırılırlar. DiffServ modelinde servis kalitesi ile ilgili bilgi, IP başlığındaki DSCP değerinde saklanır. Aynı DiffServ Code Point (DSCP) değerine sahip tüm trafiğe aynı şekilde davranılır. Differentiated Services modeli, Integrated Services modelinin tersine yönlendiricilerde durum (state) bilgisi tutulmasını gerektirmediğinden ölçeklenebilirlik bakımından daha iyi özelliklere sahiptir ve ağ sağlayıcıları arasında daha yaygın olmaya başlamıştır.

Bu tez çalışmasında Differentiated Services modeli temel alınmıştır. Differentiated Services modeline göre yerel ağda servis kalitesi sağlayacak bir sistem gerçekleştirilmiştir. DiffServ konfigürasyonları merkezi bir LDAP dizin sunucusunda tutularak merkezden yönlendiricilere konfigürasyon dağıtımı sağlanmıştır. Linux'taki mevcut "tc" trafik kontrol arayüz programı, trafik kontrol elamanlarının konfigürasyon işlemlerinin komut satırından gerçekleştirilmesini sağlamaktadır. Bu bakımdan konfigürasyon dağıtımının doğrudan uygulama içinden gerçekleştirilebilmesi için trafik kontrol elamanlarının konfigürasyon işlemlerini gerçekleştirecek programlama arabirimi gerçekleştirilmiştir.

2. ve 3. bölümlerde Integrated Services ve Differentiated Services modeli, 4. bölümde Linux'taki DiffServ desteği hakkında ayrıntılı bilgi verilmiştir. 5. ve 6. bölümlerde DiffServ tasarımı ve uygulama detayları anlatılmıştır. 7. bölümde uygulamanın test sonuçlarına yer verilmiştir.

2. INTEGRATED SERVICES (INTSERV)

RFC 1633'te [Braden vd., 1994] tanımlanmış olan Integrated Services modeli kaynak rezervasyonu mantığına dayalıdır. RSVP (Resource Reservation Protocol) (RFC2205) [Braden vd., 1997] kullanılarak RSVP-destekli yönlendiricilerdeki kaynakları dinamik olarak rezerve etme yöntemiyle paket akışı tabanlı bir servis kalitesi sunar. RSVP sinyalleşme protokolünde, bilgiyi gönderecek olan taraf, göndereceği uca kadar olan tüm ara noktalardan belirli bir iletim hızını, bant genişliğini veya bunlara ait sınırları rezerve etmelerini ister. RSVP ile ilgili detaylı bilgi bölüm 2.2'de verilmiştir.

2.1 Integrated Services Yapısı

Integrated Services yapısı, "Integrated service modeli" ve "Referans implementasyon modeli" olmak üzere aşağıda açıklanmış olan iki ana elemandan oluşur.

Integrated service modeli : Integrated service modeli, "Kontrollü Yük Servisi" (Controlled Load Service) ve "Garanti Edilmiş Servis" (Guaranteed Service) olmak üzere iki tip servis tanımlar. Kullanılacak servis tipi uygulamaların servis kalitesi gereksinimlerine bağlıdır. Servis tipleri ile ilgili bilgi bölüm 2.3 ve 2.4'te bulunmaktadır.

Referans implementasyon modeli : Referans implementasyon modeli, Integrated Services modelinin gerçekleştirilmesi için aşağıdaki dört mekanizmayı tanımlar :

1. Katman 3 paket zamanlaması (Layer 3 scheduling)
2. Sınıflandırma
3. Kabul kontrolü (admission control)
4. Kaynak rezervasyonu

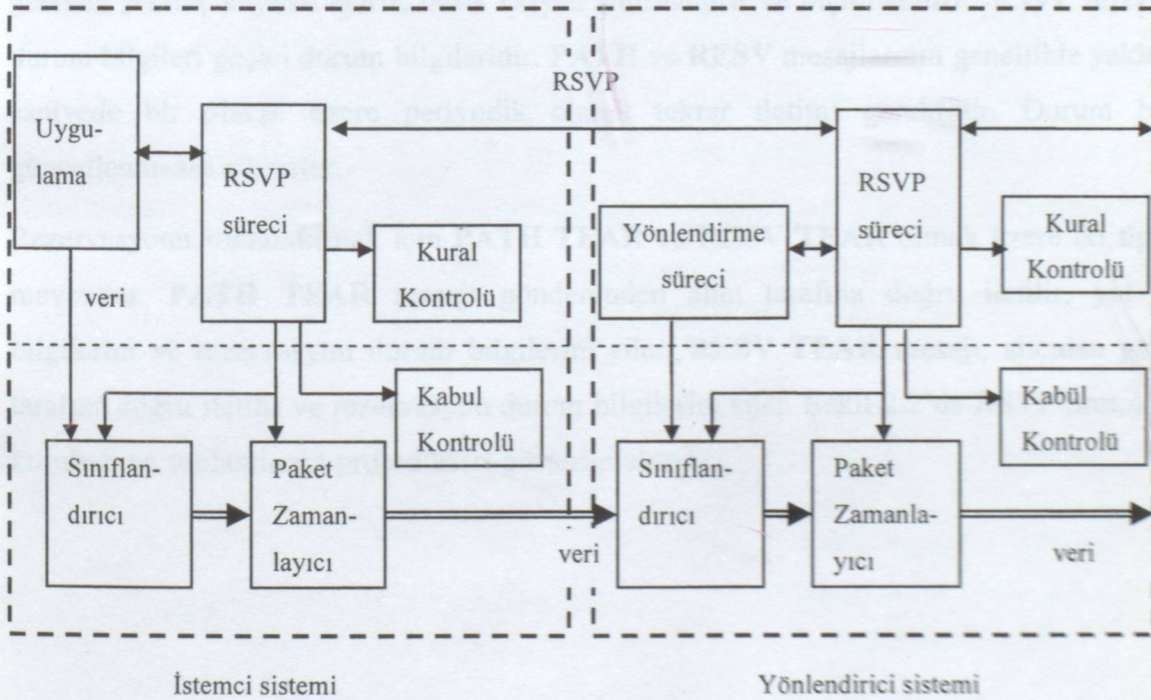
Sınıflandırma, paket zamanlaması (scheduling) ve kabul kontrolü, trafik kontrol mekanizmalarıdır. Sınıflandırıcı, paketlerin servis kalitesi gereksinimlerine göre ait oldukları sınıfı belirler. Zamanlayıcı (scheduler), paketleri servis kalitesi gereksinimlerine göre işleme sokar. Ağdaki her ağ elemanı, yeterli kaynağın bulunup bulunmadığını ve trafik akışının ilgili servisi istemeye hakkı olup olmadığını belirlemek için gelen trafik akışlarına kabul kontrolü (admission control) ve kural kontrolü (policy control) uygular.

RSVP sinyalleşme protokolü, IntServ'deki kaynak rezervasyonu için tasarlanmıştır. Bununla birlikte, IntServ başka mekanizmalar da kullanabilir. IntServ yapısı ve RSVP birbirlerinden bağımsız olarak da kullanılabilir.

2.2 RSVP

Resource Reservation Protocol (RSVP) (RFC2205) [Braden vd., 1997], uygulamaların servis kalitesi gereksinimlerini ağ elemanlarına iletmek için kullanabilecekleri bir sinyalleşme protokolüdür. RSVP, servis kalitesi parametrelerinin yalnızca iletim işlemi için kullanılır, servis kalitesi ile ilişkili herhangi bir fonksiyon sağlamaz. Bir başka deyişle, taşıdığı servis kalitesi isteği bilgilerini yorumlamaz. RSVP, iletişim oturumunun başlangıcında uygulama tarafından başlatılır. İletişim oturumu, hedef IP adresi, iletim (transport) katmanı protokol tipi ve hedef port numarası ile tanımlanır. Herbir RSVP paketi ait olduğu oturumun detaylarını içerir. Kaynak atama (provisioning), RSVP'den bağımsız bir işlemdir. IntServ modeli bakımından düşünüldüğünde, kaynak atama (provisioning) işlemi yani belirli bir trafik akışı için gereksinim duyulan kaynakların kabulü/reddi işlemi IntServ'e ait bir fonksiyondur.

Rezervasyon kurulumu esnasında, RSVP isteği, "kabul kontrolü" ve "kural kontrolü" yerel karar modüllerine aktarılır. Kontrollerin ikisi de başarılı olursa ilgili servis kalitesini sağlamak için paket sınıflandırıcıda ve bağlantı katmanı arayüzünde (ör. paket zamanlayıcıda) ilgili parametreler atanır. Kontrollerden herhangi biri başarısız olursa RSVP protokolü, isteği gönderen uygulamaya hata mesajı gönderir. Şekil 2.1'de RSVP'nin kontrol modülleri ile etkileşimi gösterilmektedir.

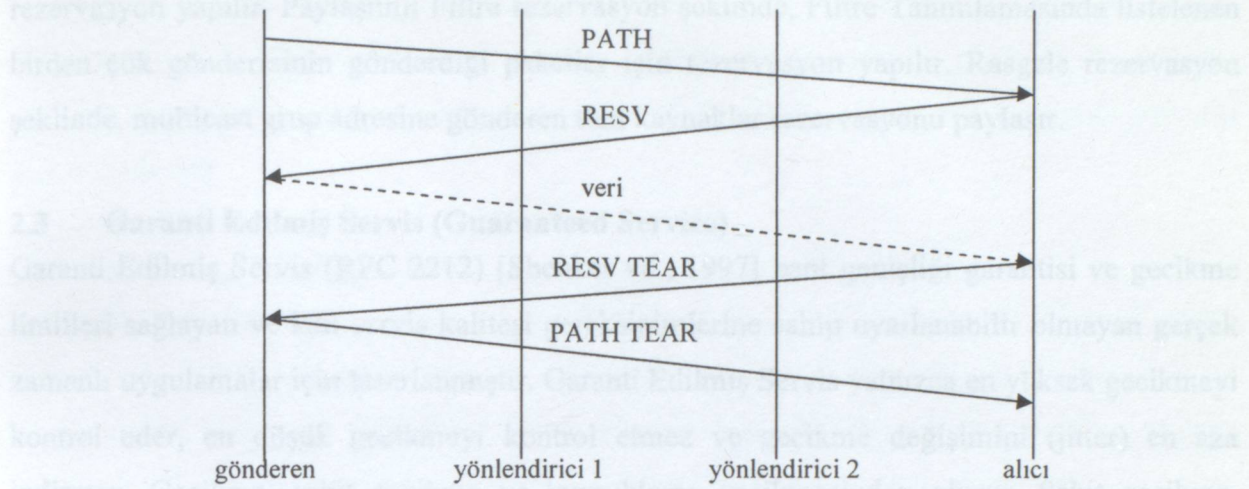


Şekil 2.1 İstemci sistemi ve yönlendirici sisteminde RSVP (RFC 2205) [Braden vd., 1997]

RSVP iletişim oturumu, trafik kaynağının kaynaktan hedefe doğru **PATH** mesajı göndermesi ile başlar. **PATH** mesajı, kaynağın ürettiği akışa ait trafik karakteristiklerinin bulunduğu Sender TSpec (sender traffic specification - gönderen trafik tanımlaması) bilgisini içerir. Yol üzerindeki her bir ara yönlendirici **PATH** mesajını, yönlendirme protokolü tarafından belirlenen bir sonraki düğüme gönderir. **PATH** mesajının gönderildiği tüm ağ elemanları **PATH** mesajının doğruluğunu kontrol eder. Bir hata belirlenirse yönlendirici **PATH** mesajını gözardı edip **PATH** mesajını üreten trafik kaynağına **PATH-ERR** mesajı iletir. Eğer **PATH** mesajı geçerli ise yönlendirici yol (path) durum bilgisini saklar. Yol durum bilgisi, Sender TSpec bilgisini ve **RESV** mesajlarının ters yönde iletimi için kullanılacak olan bir önceki yönlendiricinin IP adresi (Phop – previous hop address) bilgisini içerir.

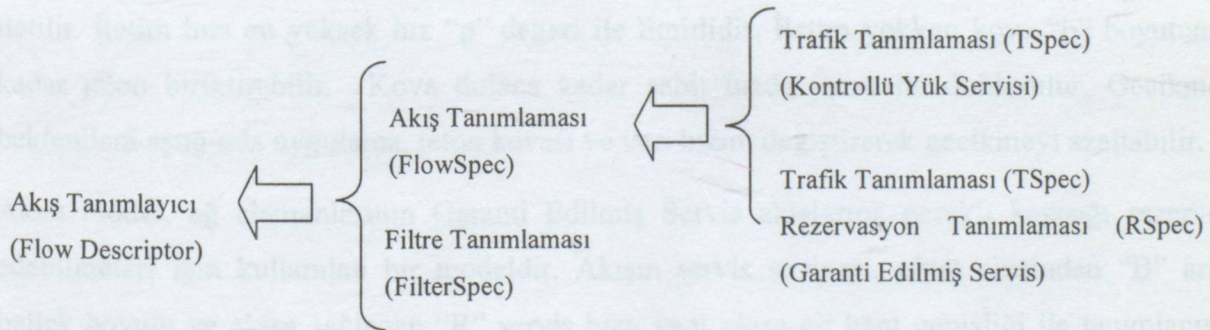
İletişim oturumunun alıcı tarafı rezervasyonu onaylarsa **PATH** mesajı ile aynı yolu kullanılmak suretiyle gönderen tarafına **RESV** mesajı iletilir. Yol üzerindeki her bir ara yönlendirici isteği kabul veya red edilebilir. Reddedilirse, iletişim oturumunun alıcı tarafına **RESV-ERR** mesajı iletilir, kabul edilirse, servis kalitesi durum bilgisi yönlendiriciye yüklenir. Durum bilgileri alıcının kendine has servis kalitesi gereksinimlerine aittir. Gönderen tarafının **RESV** mesajını alması kaynakların rezerve edildiğini ve verinin transfer edilebileceğini gösterir. Durum bilgisine göre yol üzerindeki ağ elemanları trafik akışları için gereken şekilde kaynak ayırır, trafik akışını sınırlandırır ve biçimlendirir. RSVP rezervasyon durum bilgileri geçici durum bilgileridir. **PATH** ve **RESV** mesajlarının genellikle yaklaşık 30 saniyede bir olmak üzere periyodik olarak tekrar iletimi gereklidir. Durum bilgileri güncellenmezse silinirler.

Rezervasyonu sonlandırmak için **PATH TEAR** ve **RESV TEAR** olmak üzere iki tip mesaj mevcuttur. **PATH TEAR** mesajı gönderenden alıcı tarafına doğru iletilir, yol durum bilgilerini ve rezervasyon durum bilgilerini siler. **RESV TEAR** mesajı, alıcıdan gönderen tarafına doğru iletilir ve rezervasyon durum bilgilerini siler. Şekil 2.2’de RSVP protokolünün kurulum ve sonlandırma prosedürleri gösterilmektedir.



Şekil 2.2 Kaynak rezervasyonu için RSVP sinyalleşmesi [Bai vd.]

RESV mesajı, Akış Tanımlaması (FlowSpec) ve Filtre Tanımlamasından (FilterSpec) oluşan Akış Tanımlayıcı (Flow Descriptor) bilgilerini içerir. Akış Tanımlaması (FlowSpec), Kontrollü Yük Servisi veya Garanti Edilmiş Servis olmak üzere gereksinim duyulan servis tipini, istenen servis kalitesine ait servis parametrelerini içeren RSpec (reserve specification - rezervasyon tanımlaması) bilgisini ve servis isteminde bulunacak trafik akışına ait parametrelerin tanımlandığı TSpec (traffic specification - trafik tanımlaması) bilgisini içerir. Akış Tanımlayıcı'da RSpec bilgisinin mevcut olması, istenen servis tipinin Garanti Edilmiş Servis olmasına bağlıdır. Filtre Tanımlaması (FilterSpec) ise sınıflandırma işleminde kullanılacak parametreleri tanımlar, bir başka deyişle, rezervasyonu kullanacak paketleri belirler. Akış Tanımlayıcı'nın bileşenleri Şekil 2.3'te gösterilmektedir.



Şekil 2.3 Akış Tanımlayıcı ve bileşenleri

RESV mesajı, Akış Tanımlayıcı bilgilerinin yanında rezervasyon şekli bilgisini de içerir. Filtre (Fixed Filter - FF), Belirtilmiş Paylaşımlı Filtre (Shared Explicit - SE) ve Rasgele Filtre

(Wildcard Filter - WF) olmak üzere üç rezervasyon şekli mevcuttur. Sabit filtre rezervasyon şeklinde, Filtre Tanımlamasında belirtilmiş olan tek bir göndericinin gönderdiği paketler için rezervasyon yapılır. Paylaşımlı Filtre rezervasyon şeklinde, Filtre Tanımlamasında listelenen birden çok göndericinin gönderdiği paketler için rezervasyon yapılır. Rasgele rezervasyon şeklinde, multicast grup adresine gönderen tüm kaynaklar rezervasyonu paylaşır.

2.3 Garanti Edilmiş Servis (Guaranteed Service)

Garanti Edilmiş Servis (RFC 2212) [Shenker vd., 1997] bant genişliği garantisi ve gecikme limitleri sağlayan ve katı servis kalitesi gereksinimlerine sahip uyarlanabilir olmayan gerçek zamanlı uygulamalar için tasarlanmıştır. Garanti Edilmiş Servis yalnızca en yüksek gecikmeyi kontrol eder, en düşük gecikmeyi kontrol etmez ve gecikme değişimini (jitter) en aza indirmez. Gecikme, sabit gecikme ve kuyruklama gecikmesinden oluşur. Sabit gecikme, seçilen yol ile ilgili iletim gecikmeleri vb. özelliklerden oluşur ve yol kurulum mekanizması (ör: RSVP) tarafından tanımlanır. Kuyruklama gecikmesi ise jeton kovası (token bucket) parametreleri ve uygulamanın istediği servis hızı gibi değerlere göre Garanti Edilmiş Servis tarafından belirlenir.

Trafik kaynağı ve ağ elemanları farklı davrandıklarından jeton kovası (token bucket) modeli ve akıcı (fluid) model olmak üzere iki ayrı model ile tanımlanırlar.

Jeton kovası modeli sayesinde, özellikle jeton kovası boyutu “b” ve jeton hızı “r” parametreleri sayesinde, Garanti Edilmiş Servis’in sağlayacağı kuyruklama gecikmesi hakkında uygulamaların önceden bilgisi olur ve bu parametreleri kontrol edebilirler. Uygulamanın kaynağı kova (bucket) içinde jeton (token) bulunduğu sürece veri gönderebilir. Kaynak, jeton kovası hızı “r” değerinden daha büyük hızda paket gönderiyor da olsa paketler jeton kovası hızı “r” ile gönderilir. Bu durumda paketler ara bellekte saklanır ve “r” hızında iletilir. İletim hızı en yüksek hız “p” değeri ile limitlidir. İletim yokken kova “b” boyutuna kadar jeton biriktirebilir. Kova dolana kadar sabit hızda jeton ile doldurulur. Gecikme beklenileni aştığında uygulama, jeton kovası ve veri hızını değiştirerek gecikmeyi azaltabilir.

Akıcı model, ağ elemanlarının Garanti Edilmiş Servis akışlarına gerekli kaynağı rezerve edebilmeleri için kullanılan bir modeldir. Akışın servis seviyesi, oktet cinsinden “B” ara bellek boyutu ve akışa sağlanan “R” servis hızı, yani akışa ait bant genişliği ile tanımlanır. Ara bellek, iletim hızı “p”nin akış servis hızı “R”yi aştığı durumlar için gereklidir. [Rexhepi ve Heijenk, 2000]

Garanti Edilmiş Servis, TSpec ve RSpec ile tanımlanır. TSpec, jeton kovası hızı “r”, kova boyutu “b”, en yüksek hız “p”, en düşük temin edilen birim “m” ve en büyük datagram

boyutu “M” bilgilerini içerir. En yüksek hız “p”, kaynağın trafik gönderebileceği en yüksek hızdır. En düşük temin edilen birim “m”’in anlamı “m”’den daha küçük datagramların “m” boyutunda kabul edilmesidir. RSpec, istenen servis hızı “R” ve “S” slack değerinden oluşur. “S” slack değeri, “R” rezervasyon hızı kullanıldığında gerçekleşen gecikme ile istenilen gecikme arasındaki farkı gösterir. Bu değer kullanılarak belirli bir gecikme garantisi sağlanabilir. [1]

2.4 Kontrollü Yük Servisi (Controlled Load Service)

Kontrollü Yük Servisi (Controlled Load Service - CL) (RFC 2211) [Wroclawski, 1997] ağdaki tıkanıklık koşullarına yüksek duyarlılıklı uyarlanabilir gerçek zamanlı uygulamalar için tasarlanmıştır. Kontrollü yük servisinin bu uygulamalara sağladığı tek fonksiyon, ağda fazla yük/tıkanıklık mevcut değilken gerçekleştirecek trafik iletimini, ağda fazla yük bulunduğu durumda da aynı sınırlar içinde sağlamaktır. Kontrollü yük servisi, paket kaybı ve gecikme gibi servis kalitesi kontrol parametrelerini kabul etmez ve kullanmaz. Kontrollü yük servisi isteminde bulunan uygulamalar, ağ ortamının düzgün çalıştığı kabul edilmek koşuluyla, kendi yarattıkları trafiğin başarıyla iletilmesini ve iletim gecikmesinin başarılı iletilen trafiğin minimum iletim gecikmesine yakın olmasını bekler.

Kontrollü yük isteğinin ulaştığı ağ elemanları, rezervasyon isteyen tarafın TSpec bilgisindeki istenen trafik özelliklerini sağlamak için gerekli bant genişliğini ve paket işleme kaynaklarını sağlamalıdır. İsteğin sağlanıp sağlanamayacağı ile ilgili olarak ağ elemanının kullandığı metod yerel bir konudur, kontrol parametreleri ve mesaj formatları uyumlu olması koşuluyla uygulama bağımlı olabilir. Bunun için ölçüm tabanlı yaklaşımlar veya zamanlama (scheduling) mekanizmaları kullanılabilir. [Rexhepi ve Heijenk, 2000]

2.5 “Leaky Bucket” ve “Token Bucket” Trafik Şekillendirme Algoritmaları

Trafik şekillendirme algoritmalarından “Leaky Bucket” ve “Token Bucket” algoritmaları birçok trafik kontrol mekanizmasının temelini oluşturmaktadır. Aşağıda bu algoritmaların çalışma prensiplerine yer verilmiştir.

Orjinal “Leaky bucket” algoritmasının uygulanması oldukça basittir. Algoritma sonlu bir kuyruk kullanır. Paket geldiğinde kuyruksa yer varsa kuyruğun sonuna eklenir, yoksa gözardı edilir. Kuyruk boş değilse her saat darbesinde bir paket iletilir.

“Byte-counting leaky bucket” benzer şekilde uygulanabilir. Bir sayaç her saat darbesinde n değerine getirilir. Kuyruktaki ilk paket, sayacın o anki değerinden daha az oktet içeriyorsa paket gönderilir ve sayaç gönderilen oktet kadar azaltılır. Sayaç yeteri kadar büyükse ek

paketler iletebilir. Sayacın değeri kuyruktaki bir sonraki paketin uzunluğunun altına düştüğünde iletim bir sonraki saat darbesine kadar durur. Sayaç her saat darbesinde n değerine getirildiğinden sayaçtaki artan oktetler kaybedilmiş olur.

“Leaky bucket” algoritması, trafik ne kadar patlamalı (bursty) şekilde gelse de katı bir şekilde belirli bir ortalama hızda iletim sağlar. Birçok uygulama için trafik patlaması gerçekleştiğinde iletim hızının artabiliyor olması daha uygundur. Bu bakımdan “token bucket” algoritması gibi daha esnek ve veri kaybetmeyen bir algoritma gereksinimi mevcuttur. “Token bucket” algoritmasında jetonları (token) sayan, her ΔT sürede bir değeri bir arttırılan ve paket gönderildiğinde bir azaltılan bir sayaç kullanılır. Sayaç sıfır değerini aldığı anda paket gönderilemez. “Byte-count” (oktet-saymalı) versiyonunda da sayaç her ΔT sürede bir k oktet arttırılır ve gönderilen paket uzunluğu kadar azaltılır.

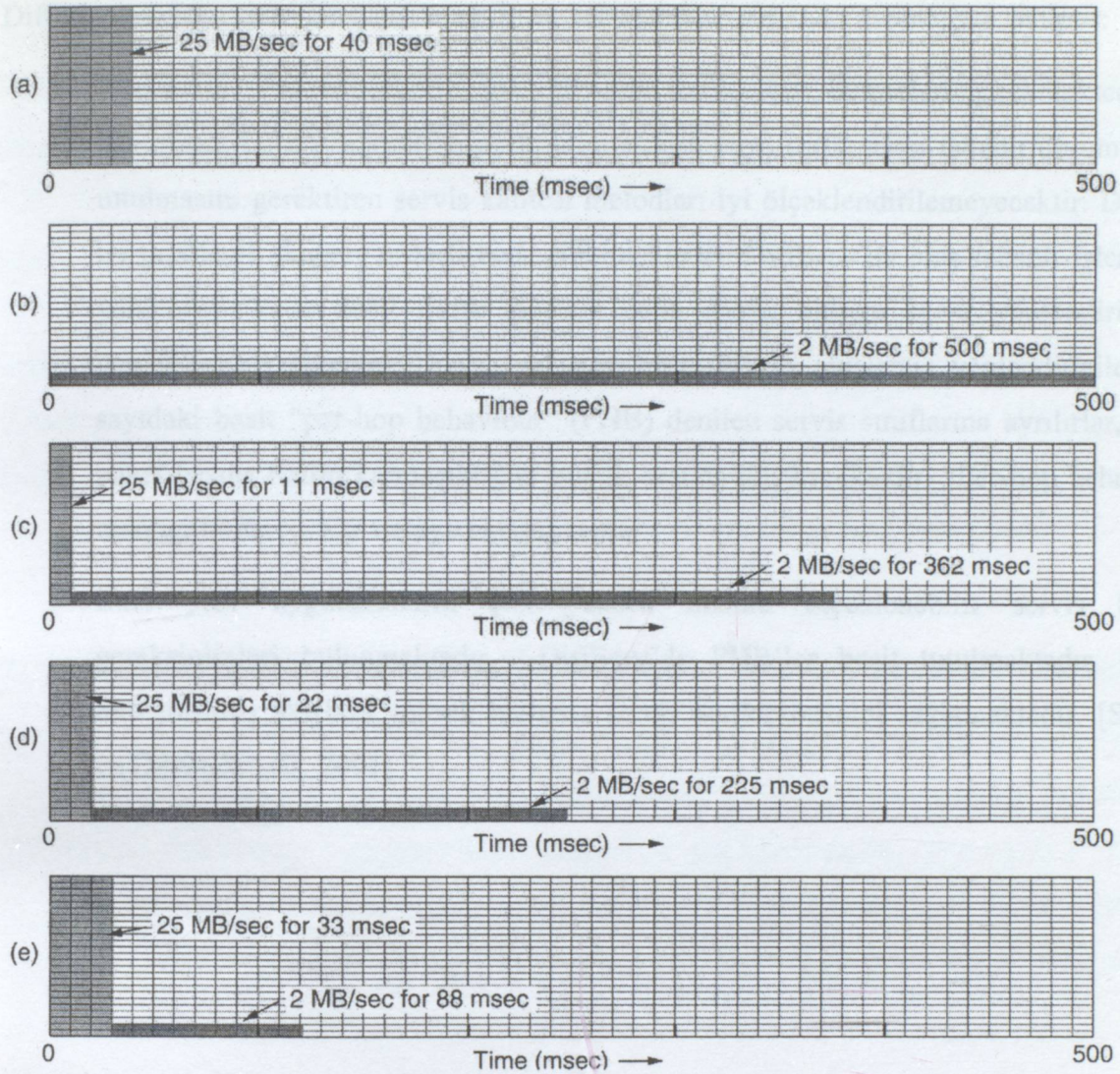
“Token bucket” algoritması “leaky bucket” algoritmasından daha farklı bir trafik şekillendirme sağlar. “Leaky bucket” algoritması boş kalan sunucuların sonraki olası büyük trafik patlamalarının iletimi için birikim gerçekleştirmelerine izin vermez. “Token bucket” algoritması ise n maksimum “bucket” boyutuna kadar izin verir. Bu sayede n adet pakete kadar trafik patlamalarının bir defada iletimi gerçekleştirilebilir ve ani trafik patlamalarına daha hızlı cevap verilmesi sağlanır.

“Token bucket” algoritması trafik patlamalarına ayarlanabilir bir maksimum boyuta kadar izin verir. Patlama süresi S aşağıdaki (2.1) bağıntısı ile elde edilebilir.

$$C + \rho S = MS \quad (2.1)$$

(2.1) bağıntısındaki C “token bucket” kapasitesini, ρ oktet/s cinsinden jeton (token) gelme hızını ve M oktet/s cinsinden maksimum çıkış hızını göstermektedir. Bu bağıntıdan da görüldüğü gibi patlama süresi hesaplanılırken hem “bucket” kapasitesi hem de o süre içinde gelecek jeton miktarı göz önüne alınmalıdır.

Şekil 2.4 (a)’daki 40 ms boyunca 25MB/s hızla “leaky bucket”’a gelen trafiğin Şekil 2.4 (b)’de 500 ms boyunca 2 MB/s ortalama hızla iletileceği gösterilmektedir. Şekil 2.4 (c)’de ise “token bucket”’a $C = 250 \text{ KB}$, $\rho = 2 \text{ MB/s}$ ve $M = 25 \text{ MB/s}$ olmak üzere 1 MB’lık bir patlamanın geldiği durum gösterilmektedir. Patlama geldiğinde “bucket”ın dolu olduğu varsayımıyla (2.1) bağıntısına göre trafiğin 11ms 25MB/s hızla, kalan trafiğin ise 2 MB/s hızla iletileceği görülebilmektedir. Şekil 2.4 (d) ve Şekil 2.4 (e)’de sırasıyla $C = 500 \text{ KB}$ ve $C = 750 \text{ KB}$ olduğu durumdaki değerler görülebilmektedir. [Tanenbaum, 2002]



Şekil 2.4 “Leaky Bucket” ve “Token Bucket” algoritmaları [Tanenbaum, 2002]

2.6 Integrated Services ve Differentiated Services Karşılaştırması

Integrated Services yapısındaki problemler aşağıdaki gibidir:

- Yönlendiricilerde saklanan durum bilgisi miktarı trafik akışı adedi ile orantısal olarak artmaktadır. Bu durum yönlendiricilerde büyük bir saklama ve işlem yüküne neden olmaktadır. Bu nedenle Integrated Services yapısında Internet ortamı için ölçeklenebilirlik problemi mevcuttur.
- Yönlendiricilerdeki gereksinimler yüksektir. Tüm yönlendiricilerin RSVP, kabul kontrolü, sınıflandırma ve paket zamanlaması (packet scheduling) gerçekleştirmesi gereklidir.

Differentiated Services ise ölçeklenebilirlik bakımından aşağıdaki avantajlara sahiptir:

- Ağdaki iç yönlendiricilerin çok miktarda trafik akışı iletmeleri gerekmektedir. Bu bakımdan, bu noktalarda hesapsal karmaşıklık veya trafik akışı tabanlı durum bilgisi tutulmasını gerektiren servis kalitesi metodları iyi ölçeklendirilemeyecektir. DiffServ bu problemi akışları birleştirerek gidermektedir. DiffServ’de akış tabanlı işlemler iç yönlendiricilerde değil trafik akışının daha düşük olduğu kenar yönlendiricilerde gerçekleştirilmektedir. Bunun yanı sıra, birleştirilmiş akışlar iç yönlendiricilerde az sayıdaki basit “per-hop behaviour” (PHB) denilen servis sınıflarına ayrılırlar, bu da paket başına iletim karmaşıklığını büyük oranda düşürmektedir. “Per-hop behaviour” mantığı bölüm 3.3’te detaylı açıklanmıştır.
- Bazı yeni uygulamaların çok yüksek hızlara ölçeklenebilir servis kalitesi gereksinimleri bulunmaktadır. DiffServ’de PHB’ler basit tutulmaktadır. Bu da birleştirilmiş akışlara yüksek hızlarda servis verilebilmesini sağlamaktadır. [Stanisic ve Devetsikiotis, 2001]

3.1 Differentiated Services Alanı

İnternet paketli iletişim ortamı için servis kalitesini belirlemek üzere IP versiyon 4 bağlamında RFC 2475, RFC 2911, Type of Service (ToS) (servis tipi) oktetı mevcuttur. ToS oktetinin RFC 1349’da [14] tanımlanması yapılır. Şekil 3.1’de gösterildiği gibi, öncelik alanı servis tipi alanı ve kodlanmış 4 bitlik bir alandan oluşur. ToS oktetindeki her bir alanın değeri aşağıdaki gibidir:

0	1	2	3	4	5	6	7
Öncelik Alanı (Priority Field)				Servis Tipi Alanı (Type of Service Field)			
—				Geçikme (Delay)	Derinlik (Throughput)	Güvenilirlik (Reliability)	Maliyet (Cost)

Şekil 3.1 RFC 1349’da tanımlanan ToS oktet yapısı

3. DIFFERENTIATED SERVICES (DIFFSERV)

Differentiated Services (DiffServ) modeli Internet Engineering Task Force (IETF) DiffServ çalışma grubu tarafından tasarlanmıştır. DiffServ modeli, RFC 2474 [Nichols vd., 1998] ve RFC 2475'te [Blake vd., 1998] tanımlanmıştır. DiffServ modeli, paket akışlarını sınırlı sayıda sınıflara başka deyişle servis seviyelerine ayırır. Bu yaklaşım bazen servis sınıfı (Class of Service - CoS) olarak da adlandırılmaktadır. Differentiated Services kaynak ayırmak için herhangi bir sinyalleşme mekanizması tanımlamaz bunun yerine sağlanacak servis kalitesini (Quality of Service - QoS) tanımlamak için, paket bir Diffserv kod puanı (DiffServ Code Point - DSCP) ile işaretlenir. Bu kod puanı kullanılarak yönlendiricilerin IP paketini önceliklendirebilmesi sağlanır. DiffServ yapısının ana özellikleri aşağıdaki gibidir:

1. Birden çok paket akışı birleştirilmiş (aggregate) servis seviyelerine eşleştirilir.
2. Farklı servis seviyeleri kullanılarak uygulamalara servis kalitesi garantileri verilebilir.
3. Paket akışlarına ait herhangi bir durum (state) bilgisinin tutulmasına gerek duyulmaz. Paket akışı başına sınıflandırmanın yapıldığı IntServ yönteminin tersine, DiffServ'de paketler "birleştirme" (aggregation) yöntemi ile sınıflandırılırlar. Aynı DSCP değerine sahip tüm trafiğe aynı şekilde davranılır. Bu tür bir yaklaşım nedeniyle büyük ağlarda ve yüksek trafik yüklerinde daha iyi bir ölçeklenebilirlikten bahsedilebilmektedir. [Manner, 2003]

3.1 Differentiated Services Alanı

Internet paketlerine sağlanması gereken servis kalitesini belirtmek üzere IP versiyon 4 başlığında (IPv4, RFC 791) Type of Service (TOS) (servis tipi) oktetı mevcuttur. ToS oktetinin RFC 1349'da [14] tanımlanan yapısı, Şekil 3.1'de gösterildiği gibi, öncelik alanı, servis tipi alanı ve kullanılmayan 1 bitlik bir alandan oluşur. ToS oktetindeki son bit her zaman 0 değerine sahiptir.

0	1	2	3	4	5	6	7
Öncelik alanı (Precedence -3bit)			Servis tipi alanı (ToS -4bit)				0
			Gecikme (Delay)	Başarım (Throughput)	Güvenilirlik (Reliability)	Maliyet (Cost)	0

Şekil 3.1 RFC 1349'da tanımlanan ToS oktet yapısı

İlk 3 bit paketin önceliğini, sonraki 4 bit ise yönlendirme/iletim servisleriyle ilgili istenen servis tipini (Type of Service - ToS) belirtir. 3 bitlik öncelik alanı $8 (2^3)$ farklı değer atanabilmesine izin verir. ToS oktet yapısındaki öncelik alanında tanımlanabilecek değerler Çizelge 3.1’de gösterilmiştir.

Çizelge 3.1 Öncelik (Precedence) alanı değerleri

000	Rutin (Routine)
001	Öncelik (Priority)
010	Acil (Immediate)
011	Flaş (Flash)
100	Öncelikli flaş (Flash override)
101	Kritik (Critical)
110	İnternet kontrolü (Internetwork control)
111	Ağ kontrolü (Network control)

ToS oktet yapısındaki servis tipi alanında tanımlanabilecek değerler Çizelge 3.2’de gösterilmiştir.

Çizelge 3.2 Servis tipi (ToS) alanı değerleri

1000	Gecikmeyi en aza indir.
0100	İşlem hacmini en fazlaya çıkar.
0010	Güvenilirliği en fazlaya çıkar.
0001	Maliyeti en aza indir.
0000	Normal servis

IETF tarafından RFC 2474’te [Nichols vd., 1998] yapılan yeniden tanımlama sonucunda IPv4 ToS oktet ve IPv6 Traffic Class oktet Differentiated Services (DS) alanı olarak adlandırılmıştır. DS alanının 6 biti, DSCP değerlerini tutmak için ayrılmıştır. RFC 1349’da [Almquist, 1992] tanımlanan yapıda sadece 8 farklı servis sınıfı tanımlanabilmesine karşın DiffServ, 3 ek bit sayesinde, $64 (2^6)$ farklı servis sınıfı tanımlanabilmesine izin vermektedir. DS alanındaki kalan 2 bit, DiffServ tarafından kullanılmamaktadır. Kullanılmayan bu alan RFC 3168’de [Ramakrishnan vd., 2001] Explicit Congestion Notification (ECN) mekanizmaları için ayrılmıştır. DS alanının yapısı Şekil 3.2’de gösterilmiştir.

0	1	2	3	4	5	6	7
DSCP (6bit)						0	0

Şekil 3.2 Differentiated Services alanı

DSCP değerleri Çizelge 3.2’de gösterilen üç gruba ayrılır. Bu gruplardan birincisi standart, diğer ikisi deneysel veya yerel kullanım içindir. DSCP değerleri xxxyy0 şeklinde bir seviye daha ileri gruplandırılabilir. Burada x değerleri sınıf seçici değerler, y değerleri ise reddedilme (drop) önceliğini belirleyen değerlerdir.

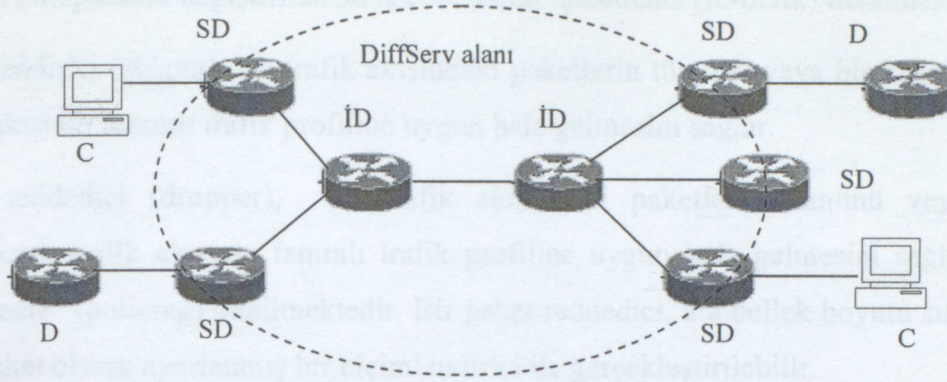
Çizelge 3.3 DSCP değerleri [Manner, 2003]

Grup	DSCP	Atama kuralı
1	xxxxx0	Standart
2	xxxx11	Deneysel veya yerel kullanım
3	xxxx01	Deneysel veya yerel kullanım

3.2 Differentiated Services Yapısı

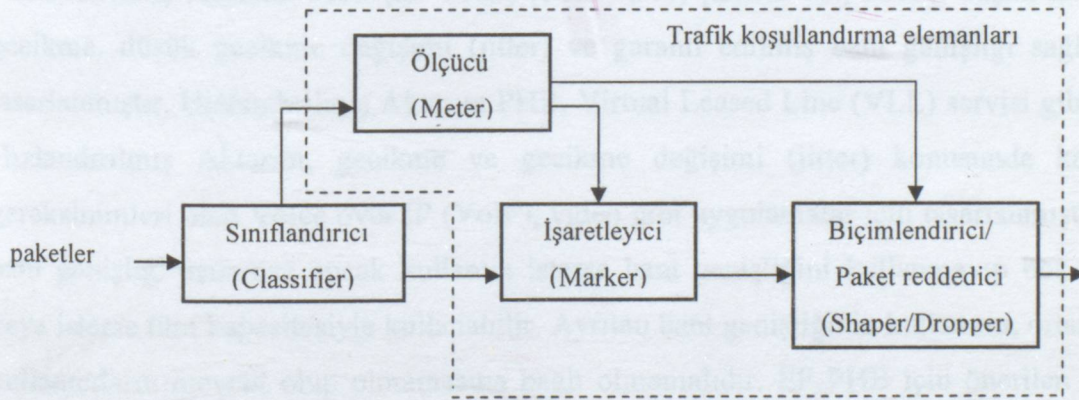
DiffServ modeli kullanılarak servis kalitesi sağlanan ağ alanı, Differentiated Services alanı (Differentiated Services domain - DiffServ domain) olarak isimlendirilir. DiffServ alanını çevreleyen kenar yönlendiriciler (edge router), DiffServ sınır düğümleri (DiffServ boundary node) olarak, iletim servisini veren iç yönlendiriciler (core router), DiffServ iç düğümleri (DiffServ interior node) olarak adlandırılır. Şekil 3.3’te temel DiffServ yapısı bir örnek ile gösterilmektedir. Şekilde, sınır düğümler “SD”, iç düğümler “İD”, DiffServ desteği olmayan düğümler “D”, sunucu, masaüstü bilgisayar gibi diğer cihazlar C sembolü ile gösterilmektedir.

DiffServ sınır düğümleri, DS alanına giren trafiği PHB (Per Hop Behaviour) adı verilen servis gruplarına ayırabilmek için paket başlıklarındaki bilgilere göre paketleri sınıflandırır ve ilgili DSCP değeri ile işaretler. DiffServ iç düğümleri ise, DSCP değerini kontrol ederek DS alanındaki tüm iç düğümler tarafından destekleniyor olan PHB gruplarından birini seçer. [Majalainen, 2002]



Şekil 3.3 DiffServ – Temel Yapısı [Majalainen, 2002]

DiffServ sınır düğümleri, tanımlı trafik kurallarının uygulanması için kontrol mekanizmaları da gerçekleştirebilir. Bu işlem trafik koşullandırma (traffic conditioning) olarak adlandırılmaktadır. Trafik koşullandırma, ölçücü (meter), işaretleyici (marker) biçimlendirici (shaper) ve paket reddedici (dropper) elemanlardan oluşur. Bu elemanların dördünün birlikte uygulanması her zaman gerekemeyebilir. Örneğin, herhangi bir trafik profilinin uygulanmasının gerekmediği durumlarda paketler sadece bir sınıflandırıcı ve bir işaretleyiciden geçebilirler. Paket sınıflandırıcısı ve trafik koşullandırma elemanlarından oluşan DiffServ sınır düğüm yapısı Şekil 3.4'te gösterilmektedir.



Şekil 3.4 Paket sınıflandırıcısı ve trafik koşullandırma elemanları
lojik blok diyagramı (RFC 2475) [Blake vd., 1998]

Ölçücü (meter), sınıflandırıcı tarafından seçilen trafik akışlarının hız vb. geçici özelliklerini ölçer. Bu durum bilgisi diğer koşullandırma elemanlarına geçirilerek tanımlı trafik profillerine uyan ve uymayan trafik akışları için belirli işlemlerin tetiklenmesini sağlar.

İşaretleyici (marker), paketin DS alanını belirli bir DS kod puanı ile işaretler. İşaretleyici, paketin kod puanını değiştirirse bu işleme tekrar işaretleme (re-mark) denilmektedir.

Biçimlendirici (shaper), bir trafik akışındaki paketlerin tümünü veya bir kısmını geciktirerek trafik akışının tanımlı trafik profiline uygun hale gelmesini sağlar.

Paket reddedici (dropper), bir trafik akışındaki paketlerin tümünü veya bir kısmını reddederek trafik akışının tanımlı trafik profiline uygun hale gelmesini sağlar. Bu işleme “limitleme” (policing) denilmektedir. Bir paket reddedici, ara bellek boyutu sıfır ya da birkaç adet paket olarak ayarlanmış bir biçimlendirici ile gerçekleştirilebilir.

3.3 Per-Hop-Behaviour

DiffServ’de paket, izlediği yol üzerindeki ağ düğümlerinde, IP paket başlığındaki DSCP değerine göre belirli bir servis tipine başka bir deyişle per-hop-behaviour’a (PHB) sınıflandırılarak servis alır. IETF tarafından tanımlanmış iki ana PHB mevcuttur: Hızlandırılmış Aktarım (Expedited Forwarding) (EF) PHB (RFC 3246) [Davie vd., 2002] ve Garantili Aktarım (Assured Forwarding) (AF) PHB. (RFC 2597) [Heinanen ve Baker, 1999]

Günümüzde Internet “en iyiyi sağlama” (best effort) mantığı ile çalışmakta, servis kalitesi bakımından bir garanti sağlanmamaktadır. DiffServ’de best effort PHB varsayılan olarak 000000 kod puanı değerine sahiptir.

Hızlandırılmış Aktarım PHB (EF PHB) (RFC 3246) [Davie vd., 2002], düşük kayıp, düşük gecikme, düşük gecikme değişimi (jitter) ve garanti edilmiş bant genişliği sağlamak için tasarlanmıştır. Hızlandırılmış Aktarım PHB, Virtual Leased Line (VLL) servisi gibi davranır. Hızlandırılmış Aktarım, gecikme ve gecikme değişimi (jitter) konusunda katı garanti gereksinimleri olan Voice over IP (VoIP), video gibi uygulamalar için tasarlanmıştır. Ayrılan bant genişliği aşılamaz ancak kullanıcı isterse bant genişliğini kullanmayıp boş bırakabilir veya isterse tüm kapasitesiyle kullanabilir. Ayrılan bant genişliğinin kullanımı, ortamda başka kullanıcıların mevcut olup olmamasına bağlı olmamalıdır. EF PHB için önerilen kod puanı 101110’dır.

Garantili Aktarım PHB (AF PHB) (RFC 2597) [Heinanen ve Baker, 1999], bant genişliği garantisi sağlamaz. AF PHB tipindeki paketlerin ağ ortamındaki iletilme olasılığı, best effort PHB tipindeki paketlerden daha yüksektir. Ağ üzerinde tıkanıklık olduğu durumda, Garantili Aktarım servisi alan kullanıcıların bant genişliği “best effort” kullanıcılara göre daha az etkilenir.

AF sınıfı “i” olan ve reddedilme önceliği “j” olan bir IP paketi, $1 \leq i \leq 4$ ve $1 \leq j \leq 3$ olmak üzere Afij kod puanı ile işaretlenir. RFC 2597’de genel kullanım için dört sınıf ve üç reddedilme seviyesi tanımlanmıştır. Ağ üzerinde tıkanıklık oluştuğunda, WRED (Weighted Random Early Detection) gibi tıkanıklık engelleme teknolojileri kuyruk yapısı tamamen dolmadan önce belirli eşik değerlerine ulaşıldığında daha düşük öncelikli trafiği reddederler. Böyle bir durumda paketlerin reddedilme sırası AF13, AF12, AF11, AF23, AF22, AF21, AF33, AF32, AF31, AF43, AF42, AF41 olacaktır. Bu sıradan da anlaşıldığı üzere AF13 en düşük öncelikli, AF41 en yüksek öncelikli sınıftır. [Majalainen, 2002]

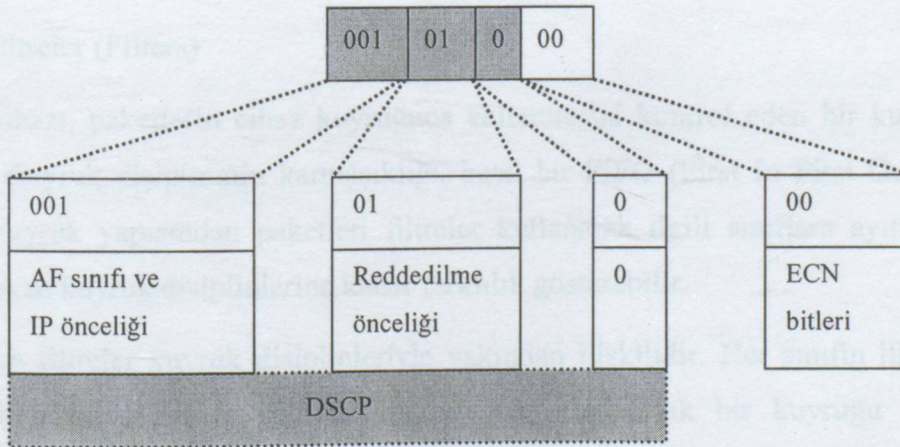
Linux’ta GRED (Generalized Random Early Detection) kuyruk yapısı bulunmaktadır. GRED sayesinde AF sınıflarındaki çoklu paket reddetme seviyelerini gerçekleştirmek mümkündür. Bu tez çalışmasında GRED kullanılmıştır.

AF, EF ve BE sınıflarına ait ToS oktet değerleri, bunlara karşılık gelen DSCP değerleri Çizelge 3.4’te gösterilmiştir.

Çizelge 3.4 ToS ve DSCP değerleri

IP Önceliği (3bit)	ToS oktetini veya DS Alanı (8bit)			DSCP (6bit)				
Bin - (Dec)	Dec	Hex	Bin	Dec	Hex	Bin	Kod Puanı	Reddedilme önceliği
000- (0) Routine	0	0x00	00000000	0	0x00	000000	BE	
001- (1) Priority	40	0x28	00101000	10	0x0a	001010	AF11	1:Düşük
	48	0x30	00110000	12	0x0c	001100	AF12	2:Orta
	56	0x38	00111000	14	0x0e	001110	AF13	3:Yüksek
010 - (2) Immediate	72	0x48	01001000	18	0x12	010010	AF21	1:Düşük
	80	0x50	01010000	20	0x14	010100	AF22	2:Orta
	88	0x58	01011000	22	0x16	010110	AF23	3:Yüksek
011- (3) Flash	104	0x68	01101000	26	0x1a	011010	AF31	1:Düşük
	112	0x70	01110000	28	0x1c	011100	AF32	2:Orta
	120	0x78	01111000	30	0x1e	011110	AF33	3:Yüksek
100 - (4) Flash override	136	0x88	10001000	34	0x22	100010	AF41	1:Düşük
	144	0x90	10010000	36	0x24	100100	AF42	2:Orta
	152	0x98	10011000	38	0x26	100110	AF43	3:Yüksek
101 - (5) Critical	184	0xb8	10111000	46	0x2e	101110	EF	
110 - (6) Internetwork Control	—	—	—	48 - 55	—	—	—	—
111 - (7) Network Control	—	—	—	56 - 63	—	—	—	—

DS alanındaki bitlerin ifade ettikleri anlamlar, Şekil 3.5'te örnek olarak seçilen "00101000" oktet değeri üzerinde gösterilmiştir.

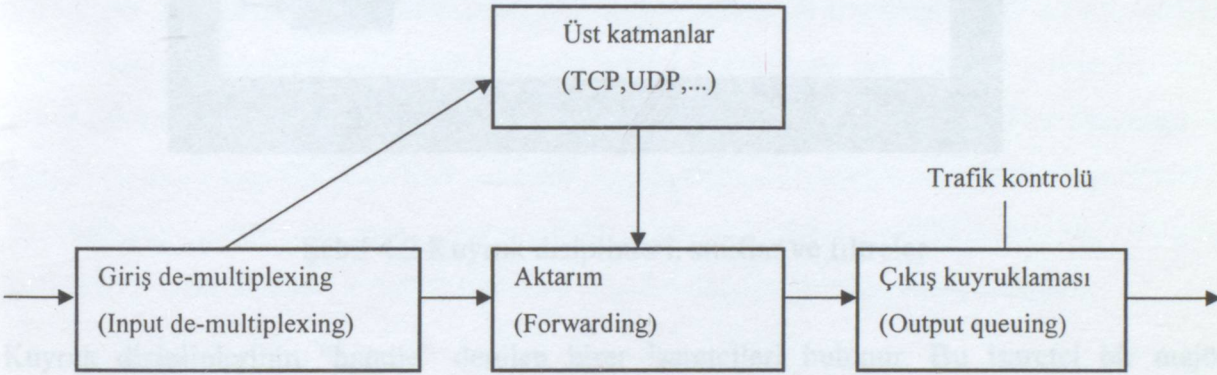


Şekil 3.5 DS alanındaki bitlerin anlamları

4. LINUX DIFFSERV DESTEĞİ

Linux'ta 2.4 kernel versiyonundan itibaren DiffServ desteği bulunmaktadır. Linux'ta trafik kontrolü "iproute2" paketi içinde gelen "tc" programı ile gerçekleştirilmektedir.

Linux'ta sisteme bir IP paketi ulaştığında sistem eğer bir yönlendirici ise kernel, paketi yönlendirir veya eğer paket sistemin kendisine gönderilmişse protokol yığınının üst katmanlarına geçirilir. Üst katmanlar paketleri işler, bunun yanı sıra yeni trafik üretilen için alt katmanlara geçirebilir. Yönlendirilecek paketler belirli bir cihaz kuyruğuna atanır. Bu kuyruk trafik kontrolü tarafından servis kalitesi gereksinimlerine, servis seviyesi anlaşmalarına göre vb. seçilir. Linux'taki paket işleme akışı Şekil 4.1'de gösterilmektedir.



Şekil 4.1 Linux'ta paket işleme akışı [2]

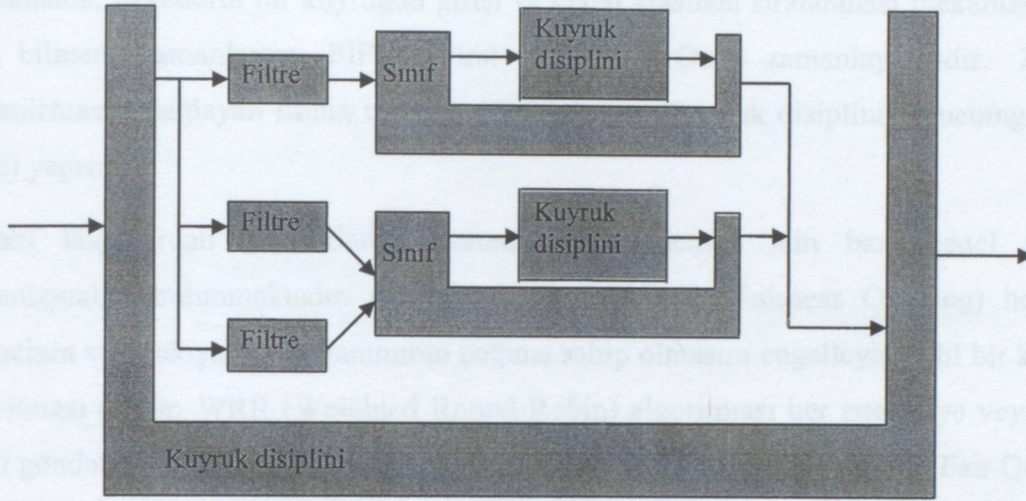
Linux'taki trafik kontrolü aşağıdaki bileşenlerden oluşmaktadır:

- Kuyruk disiplinleri (Queuing disciplines -qdiscs)
- Sınıflar (Classes)
- Filtreler (Filters)

Her ağ cihazı, paketlerin cihaz kuyruğuna eklenmesini kontrol eden bir kuyruk disiplinine sahiptir. Kuyruk disiplininin karmaşıklığı, basit bir FIFO (First In First Out) (ilk giren ilk çıkar) kuyruk yapısından paketleri filtreler kullanarak ilgili sınıflara ayıran ve ona göre işleme sokan kuyruk disiplinlerine kadar farklılık gösterebilir.

Sınıflar ve filtreler kuyruk disiplinleriyle yakından ilişkilidir. Her sınıfın ilişkili olduğu bir kuyruk bulunur. Bununla birlikte, birden çok sınıf ortak bir kuyruğu da paylaşabilir. Kuyruklar paketlerin ağa aktarılmadan önce depolandıkları yapılardır. Kuyruk disiplinleri paketleri sınıflara ayırmak için filtreler kullanır. Şekil 4.2 kuyruk disiplinleri, filtreler ve herbiri kendine ait kuyruk disiplinlerine sahip iki sınıftan oluşan basit bir yapı göstermektedir.

Kuyruk disiplinine gelen paketler filtreler kullanılarak belirli bir sınıfa ait kuyruğa aktarılırlar. Bu paketler tekrar filtrelenip iç sınıflara ayrılabilirler. Paketler en içteki kuyruğa geldiğinde ağ ortamına iletim için zamanlayıcı (scheduler) devreye girer.



Şekil 4.2 Kuyruk disiplinleri, sınıflar ve filtreler

Kuyruk disiplinlerinin “handle” denilen birer işaretçileri bulunur. Bu işaretçi bir majör numara ve bir minör numara olmak üzere iki parçadan oluşur. Örneğin kök (root) kuyruk disiplini için genellikle 1:0 değeri kullanılır. Kök kuyruk disiplininin minör numarası 0 olmak zorundadır. Sınıflara ait majör numara bağlı oldukları yapının majör numarası ile aynı olmalıdır.

4.1 Geleneksel Trafik Kontrol Mekanizmaları ile Linux Trafik Kontrol Bileşenleri Arasındaki İlişki

Biçimlendirme (shaping), zamanlama (scheduling), sınıflandırma (classifying), paket limitleme (policing), paket reddetme (dropping) ve işaretleme (marking) gibi geleneksel trafik kontrol mekanizmalarını Linux'ta gerçekleştirmek için kullanılabilecek trafik kontrol bileşenleri aşağıda açıklanmıştır. [Brown, 2003]

4.1.1 Biçimlendirme (Shaping)

Biçimlendirme işlemi, paketlerin istenen çıkış hızına düşürülmeleri için çıkış kuyruğuna iletilmeden önce geciktirilmeleridir. Biçimlendirme mekanizmasını sağlayan Linux trafik kontrol bileşeni “sınıf” (class) yapısıdır.

Biçimlendirmenin avantajlarından biri paketlerin gecikmelerinin kontrol edilebilmesidir. Biçimlendirme işlemi için kullanılan mekanizma “jeton kovası” (token bucket)

mekanizmasıdır. TBF (Token Bucket Filter) kuyruk disiplini klasik bir biçimlendirici örneğidir.

4.1.2 Zamanlama (Scheduling)

Zamanlama, paketlerin bir kuyruğun girişi ve çıkışı arasında sıralanması mekanizmasıdır. En çok bilinen zamanlayıcı FIFO (First In First Out) zamanlayıcısıdır. Zamanlama mekanizmasını sağlayan Linux trafik kontrol bileşeni “kuyruk disiplini” (queuing discipline - qdisc) yapısıdır.

Ağıdaki koşullardan kaynaklanan durumları dengelemek için bazı genel zamanlama mekanizmaları bulunmaktadır. Örneğin SFQ (Stochastic Fairness Queuing) herhangi bir istemcinin veya akışın ağ kullanımının çoğuna sahip olmasını engelleyici adil bir kuyruklama algoritması sağlar. WRR (Weighted Round Robin) algoritması her istemciye veya her akışa paket gönderme sırası sağlar. Bunların yanında ESFQ (Extended Stochastic Fair Queuing) ve GRED (Generalized Random Early Detection) gibi daha ileri seviye zamanlama algoritmaları da bulunmaktadır. Bu tez çalışmasında ağırlıklı olarak GRED kuyruk disiplini kullanıldığından bölüm 4.2.1.2’de GRED ile ilgili daha detaylı bilgi bulunmaktadır.

4.1.3 Sınıflandırma (Classifying)

Sınıflandırma, paketlerin farklı servis alabilmeleri ve yoğunlukla da farklı çıkış kuyruklarına iletilmeleri için ayrılması mekanizmasıdır.

Sınıflandırma mekanizmasını sağlayan Linux trafik kontrol bileşeni “sınıflandırıcı” (classifier) vasıtasıyla “filtre” (filter) yapısıdır. Linux sınıflandırıcıları, bir filtre dışında tek başlarına kullanılamayan yapılardır.

4.1.4 Paket Limitleme (Policing)

Paket limitleyici, belirli bir kuyruktaki trafiği ölçer ve limitler. Paket limitleme işlemi genellikle ayrılan bant genişliğinin üzerinde hat kullanımını engellemek için kenar yönlendiricilerde uygulanır. Paket limitleyici, belirli bir hıza kadar olan trafiği kabul eder, bu hızı aşan trafik için belirli bir işlem uygular. Bu işlem, trafiğin reddedilmesi (dropping) veya tekrar sınıflandırılmasıdır. Paket limitleyici, kendi içinde “jeton kovası” mekanizması kullanmasına rağmen biçimlendirme mekanizması gibi paket geciktirme kabiliyeti mevcut değildir.

Paket limitleyici, Linux trafik kontrol uygulamasında, filtre yapısının bir parçası şeklinde bulunmaktadır.

4.1.5 Paket Reddetme (Dropping)

Paket reddetme, paketin göz ardı edilmesi mekanizmasıdır. Trafiğin reddedilmesi için “paket limitleyici” (policer) içeren bir filtre kullanılmalıdır. Paket limitleyicinin belirli hızı aşan trafiğe uyguladığı işlem paket reddetme (drop) işlemi olmalıdır.

4.1.6 İşaretleme (Marking)

İşaretleme, paketin DS alanının belirli bir DS kod puanı ile işaretlenmesidir. Bunun için DSMARK kuyruk disiplini kullanılmaktadır.

4.2 Linux'taki Trafik Kontrol Bileşenleri

Linux'ta desteği olan belli başlı kuyruk disiplinleri ve filtreler aşağıda açıklanmıştır.

4.2.1 Kuyruk Disiplinleri

Linux'taki kuyruk disiplinleri kendilerine filtreler, sınıflar ve diğer kuyruk disiplinleri bağlanabilen tipte olan “sınıflı kuyruk disiplinleri” ve bağlanamayan tipte olan “sınıfsız kuyruk disiplinleri” (yaprak kuyruk disiplinleri) olmak üzere iki gruba ayrılarak incelenebilir. Sınıflı kuyruk disiplinleri aşağıda listelenmiştir.

- PRIO Priority Scheduler
- CBQ Class Based Queueing
- CSZ Clak-Scott-Zhang
- ATM Asynchronous Transfer Model
- DSMARK - DiffServ field marker
- INGRESS
- HTB Hierarchical Token Bucket

Yaprak kuyruk disiplinleri aşağıda listelenmiştir.

- FIFO First In First Out
- TBF Token Bucket Filter
- RED Random Early Detection
- GRED Generalized Random Early Detection
- TEQL Traffic Equaliser
- SFQ Stochastic Fair Queueing

Bölüm 4.2.1.1 ve 4.2.1.2’de temel kuyruk disiplinleri açıklanmıştır.

4.2.1.1 Sınıflı Kuyruk Disiplinleri

Aşağıda CBQ, HTB, DSMARK, PRIO, INGRESS sınıflı kuyruk disiplinleri açıklanmıştır.

CBQ (Class-Based Queueing) : CBQ (Class-Based Queueing) (Sınıf-Tabanlı Kuyruklama) [21]’de tanımlanmıştır. CBQ kuyruk disiplinine farklı sınıflar bağlanıp sınıfların herbirine ayrı bant genişliği payı ayrılabilir. CBQ, başka kuyruklar içerebilen ve hatta sınıfların bant genişliklerini bir kademe daha bölümlendirmek için başka CBQ’lar içerebilen bir ana kuyruk olarak görülebilir. Temel olarak CBQ, WRR (Weighted Round Robin) gerçekleştirir. Genel olarak Stochastic Fairness Queueing (SFQ), Token Bucket Filter (TBF) veya Random Early Detection (RED) gibi diğer kuyruk disiplinleri CBQ için yaprak kuyruk disiplinleri olarak kullanılır. [Floyd ve Jacobson, 1995]

CBQ, iş-korumalı (work conserving), ağ-bağımlı ve öncelik tabanlı bir zamanlayıcıdır. İş-korumalı zamanlayıcılar, yalnızca servis bekleyen paket bulunmadığı zamanlarda çalışmaz. CBQ, servis sınıflarını önceliklendirir, bunu yaparken herhangi bir trafik sınıfının sistem kaynaklarını ve bant genişliğini kaplamasını engelleyerek adil bir servis sağlamaya çalışır. Bunun yanı sıra, CBQ trafik sınıfları arasında kaynak paylaşımını sağlayan hat paylaşımı desteğine sahiptir. CBQ paket zamanlayıcısı, “genel” ve “hat-paylaştırmacı” olmak üzere iki tip zamanlayıcıdan oluşur. Genel zamanlayıcı öncelik tabanlı bir zamanlayıcıdır ve yalnızca tıkanıklık yaşanmadığı durumlarda paketlerin iletim zamanlamasını (scheduling) gerçekleştirir. Hat-paylaştırmacı zamanlayıcı ise tıkanıklık mevcut olduğu durumlarda farklı öncelikli sınıfların paket iletim zamanlamasını kontrol eder. [Markaki vd.]

CBQ’da Biçimlendirme :

CBQ, kullanılan bant genişliğinin istenen bant genişliğine düşürülmesi için gerekli süre kadar hattın boş bırakılması mantığına göre çalışır. Bunun için, paketler arasında geçmesi gereken süreyi hesaplar. Örneğin, 100Mbit/s bant genişliğine sahip bir Ethernet arabirimi üzerinde yapılacak 1Mbit/s sınırlama için, hattın geçen sürenin %99’u gibi bir süresinde boş kalması gerekmektedir. CBQ algoritması bunu sağlamak için paketler arasına gecikme koyar.

Efektif boş kalma süresi (idle time) EWMA (Exponential Weighted Moving Average) yöntemi ile ölçülür. Hesaplanan boş kalma süresinden ölçülen EWMA değeri çıkartılarak “avgidle” değeri bulunur. Hat tam yükte kullanılıyorsa avgidle değeri sıfırdır, paketlerin hesaplanan aralıkta bir geldiği anlamındadır. Hat fazla yükte kullanılıyorsa avgidle değeri

negatif olur, yüksek negatif değerlere ulaşırsa, CBQ bir süre durur ve sonra “limit üstü” (overlimit) durumda tekrar devam eder.

Tam tersi durumda, bir başka deyişle hat uzun süre kullanılmazsa avgidle değeri çok büyür, bu da hattın birkaç saat boş kalması sonucu sonsuz bant genişliklerine yol açabilecektir. Bunun önlenmesi için ortalama boş kalma süresi olan “avgidle” değeri, en yüksek boş kalma süresi olan “maxidle” değeri ile sınırlandırılır.

CBQ Konfigürasyon Parametreleri :

CBQ’da biçimlendirme için kullanılan parametreler aşağıda listelenmiştir:

- avpkt** : Oktet cinsinden ortalama paket boyutu. Boş kalma süresi hesabında kullanılır.
- bandwidth** : Ağ cihazının fiziksel bant genişliği.
- rate** : İstenen bant genişliği.
- cell** : Paketin ağ cihazında iletim süresi “cell” değeri olarak adlandırılan ve paket boyutu cinsinden tanımlanan adımlara bağlı olarak büyür. Genellikle 8 oktet olarak tanımlanır.
- maxburst** : Sınıfın kendine ayrılan bant genişliğinin tamamını kullanmadığı durumlarda ilgili sınıfın gönderebileceği paket cinsinden en yüksek “burst” boyutu.
- minburst** : Sınıfın kendine ayrılan bant genişliğinden daha fazlasını kullanmaya çalıştığı durumlarda ilgili sınıfın gönderebileceği paket cinsinden burst boyutu. Maxburst sınıfa sağlanacak “burstiness” değerini, minburst sınıfın limit değerlerini aştığı durumda sınıfa ait sıradaki paketi göndermek için beklenmesi gereken zaman aralığını hesaplamada kullanılır.
- mpu** : Minimum paket boyutu.

CBQ’da WRR (Weighted Round Robin) işlemi için kullanılan parametreler aşağıda listelenmiştir:

- allot** : Dış CBQ’ya paket göndermek için sıra geldiğinde, iç kuyruk disiplinlerine (sınıflara) önceliklerine göre sıra verilir. Her sınıf sırası geldiğinde sınırlı miktarda veri gönderebilir. “Allot” bu miktar için kullanılan birimdir.
- prio** : Sınıfa ait önceliği gösterir. Trafik mevcutsa iç sınıflar arasında öncelik değeri düşük olanlar ilk gönderilirler.

weight : Her sınıf sırası gelince paket gönderir. Diğer sınıflara göre daha fazla bant genişliğine sahip sınıflara, sıra gelince daha fazla paket gönderebilmeleri mantığıyla ağırlık (weight) atanabilir. Atanan ağırlıkların değerleri değil, aralarındaki oran önemlidir. “Weight” değeri “allot” değeri ile çarpılarak bir turda ilgili sınıfın ne kadar veri gönderileceği hesaplanır.

CBQ’da hat paylaşımı için kullanılan parametreler aşağıda listelenmiştir:

isolated/sharing : “Isolated” olarak ayarlanmış sınıf, kardeş sınıflara bant genişliği vermez. Bunun tersi “sharing” parametresidir.

bounded/borrow : “Bounded” olarak ayarlanmış sınıf, kardeş sınıflardan bant genişliği ödünç almaz. Bunun tersi “borrow” parametresidir.

CBQ’da genel çalışma prensibi :

Bir trafik sınıfı kendine ait bant genişliğinin tümünü kullanmadığı durumda (limit altı sınıf – underlimit class), gelen paket hemen çıkış hattına iletilir ve genel zamanlayıcı tarafından yalnızca yönlendiricilerdekine benzer sakla ve ilet tipinde gecikmelere maruz kalır. Sınıfın kendine ayrılan bant genişliğinden daha fazlasını kullanmaya çalıştığı durumlarda (limit üstü sınıf – overlimit class), hat-paylaştırmacı zamanlayıcı (link-sharing scheduler) devreye girer ve iki durum söz konusu olabilir. Paket ilgili kuyruğa yerleştirilir ve sonra ya *minburst* parametresine göre hızı biçimlendirilerek çıkış hattına iletilir ya da diğer trafik sınıflarına ait boş bant genişliğini borç alması sağlanır. Herhangi bir trafik sınıfının hattaki herhangi bir boş bant genişliğini borç alıp alamayacağı tanımlanabilir. Bunun yanısıra, sınıflar periyodik burst’leri (*maxburst*) karşılayabilmek için kendilerine ait bant genişliğinin üzerine çıkabilir.

HTB (Hierarchical Token Bucket) : HTB, 2001 yılı sonlarında Martin Devera tarafından doğruluk ve kullanım kolaylığı bakımından CBQ kuyruk disiplini yerine alternatif olarak geliştirilmiştir. Floyd ve Jacobson’ın (1995) geliştirdiği formal paylaşım mekanizmaları çalışmasına dayalıdır.

HTB, herhangi bir sınıfa sağlanan servis miktarının en az, arzu edilen miktar ve ayrılan miktar arasında en düşük olanı kadar olmasını sağlar. Bir sınıf kendine ayrılan miktardan daha azını kullanırsa kalan bant genişliği servis isteyen diğer sınıflar arasında dağıtılır. Bu kural, kendilerine başka HTB sınıfları bağlı olmayan sınıflar bir başka deyişle yaprak sınıfları için geçerlidir. Kendilerine başka HTB sınıfları bağlı olan “iç sınıflara” sağlanan servis miktarı ise

en az, kendilerine ayrılan miktar ve kendilerine bağlı alt sınıfların istedikleri toplam miktar arasında en düşük olanı kadardır.

HTB, sınıf-tabanlı sistemin yanında TBF (token bucket filter) algoritmasını kullanır. TBF algoritması bölüm 4.2.1.2'de anlatılmaktadır. HTB algoritması, üst sınır ve söz verilen sınırlamayı sağlamak için iki adet TBF'nin arka arkaya dizilmesinden oluşur. Aşağıda HTB konfigürasyonu için kullanılan temel parametreler açıklanmıştır: [4]

rate : Trafiği limitlemek için kullanılan, istenilen en düşük hız parametresidir.

ceil : Trafiği limitlemek için kullanılan, istenilen en yüksek hız parametresidir. Bir başka deyişle sınıfın diğer sınıflardan ödünç alabileceği miktarı limitler. Varsayılan olarak "rate" değeri ile aynıdır.

burst ve cburst : Herhangi bir sınıfa ait trafik kısa bir süre ağ cihazına ait maksimum hızda gönderilir sonra diğer sınıflara benzer şekilde servis verilir. "Burst" ve "cburst" parametreleri, başka sınıflara servis vermeden donanımın maksimum hızında gönderilecek veri miktarını kontrol eder. "Burst" değeri, hız kovanının ("rate" bucket) boyutudur. "Cburst" değeri tavan kovanının ("ceil" bucket) boyutudur.

prio : Kullanılmayan bant genişliği sınıflar arasındaki "rate" değerinin oranına göre dağıtılır. Ancak kullanılmayan bant genişliği için ilk olarak yüksek öncelikli sınıflara hak tanınır. "Prio" değeri 0 ve 7 arası bir değer olabilir. 0 değeri en yüksek öncelikli değerdir. "Prio" parametresi paketler kuyruktan iletme verilirken kullanılır. Yaprak sınıf olmayan sınıflar paket tutmadıklarından paketi kuyruktan alıp iletemezler, dolayısıyla "prio" parametresine gereksinim yoktur.

DSMARK DiffServ field marker (Diffserv alan işaretleyici) : Paketlerin DSCP alanı, *DiffServ field marker* (dsmark) kuyruk disiplini sayesinde erişilebilir ve paket sınıflandırma için kullanılabilir. *Dsmark* kuyruk disiplini, DSCP alanına erişmenin yanında DSCP alanına atama yapmak için de kullanılır. *Dsmark* konfigürasyonu yaparken *set_tc_index* parametresi kullanıldığında *dsmark* kuyruk disiplini ToS alanındaki değeri *skb->tc_index*'e kopyalar. *skb->tc_index*, */usr/include/linux/skbuff.h*'te tanımlanan *struct sk_buff* paket ara bellek tanımlayıcısındaki *tc_index* alanıdır.

Dsmark kuyruk disiplinine bağlanan sınıflandırıcının (classifier) döndürdüğü değer *skb->tc_index* alanına saklanır. Eğer eşleşen bir filtre mevcut değilse, paket, kök (root) kuyruk disiplini yaratılırken *default_index* parametresi ile tanımlanmış olan sınıfa atanır.

DSCP değerinin atanabilmesi için “maske” ve “değer” olmak üzere iki parametre gereklidir. Bu parametreler, sınıflandırıcının çıktısının, (maske, değer) çiftlerinden oluşan bir iç tabloya indeks olarak kullanılması ile elde edilirler. Sonra ToS alanına işaretlenecek değer (4.1) bağıntısı ile elde edilir.

$$\text{ToS} = (\text{skb} \rightarrow \text{tc_index} \& \text{mask}) | \text{değer} \quad (4.1)$$

(Maske, değer) çiftlerinden oluşan iç tablonun değerleri, *dsmark* kuyruk disiplinine bağlanan sınıfların parametreleri olarak tanımlanır. Tablonun boyutu ise kuyruk disiplininin *indices* parametresi ile tanımlanır. [3]

Şekil 4.3'te DSMARK ile paket işaretleme işlemi bir örnekle gösterilmektedir. Örnekte, DSMARK kuyruk disiplini numarasının 1:0 olduğu kabulüyle sınıf numarası (classid) 1:3 değerinden 3 değeri indeks olarak belirlenmektedir. İndeks ile iç tabloda maske = 0x3 ve değer = 0x68 seçilmektedir. DSMARK, paketten DS alanının değerini alıp (4.2) işlemini gerçekleştirmektedir:

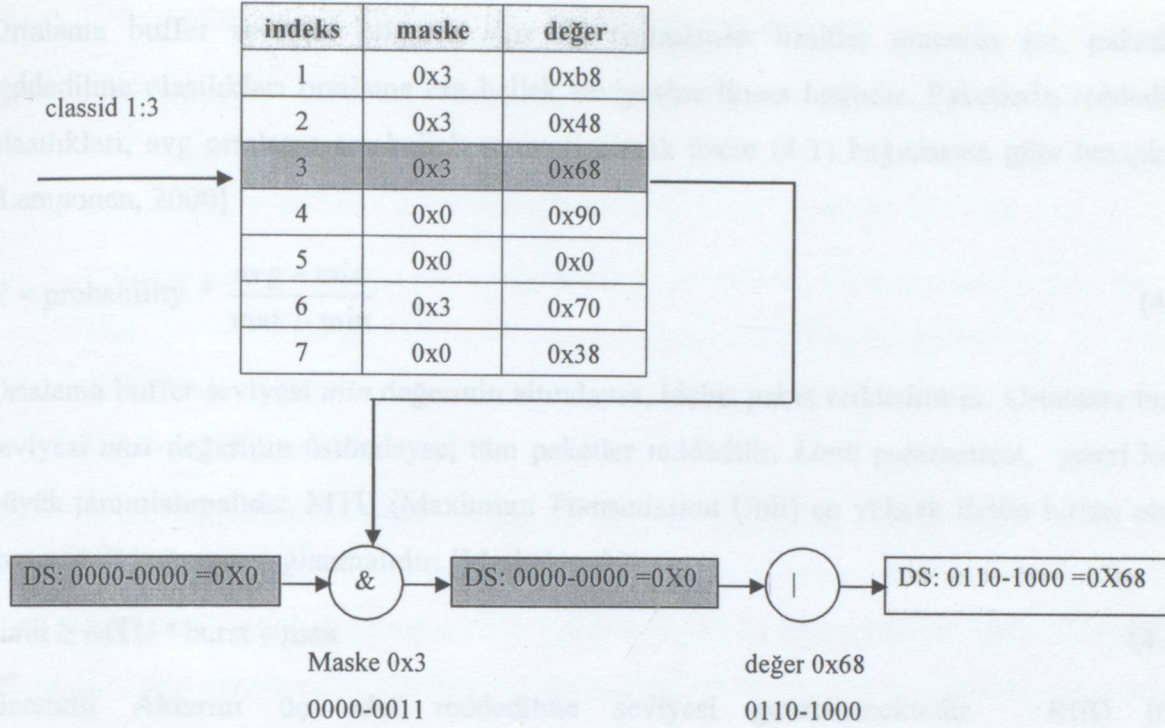
$$\text{Yeni DS değeri} = (\text{Eski DS değeri} \& \text{maske}) | \text{değer} \quad (4.2)$$

Elde edilen yeni DS değeri paketin DS alanına işaretlenir. Paket böylelikle DS alanı 0x68 olarak kuyruktan ayrılacaktır, bir başka deyişle “best effort” olarak gelen paket AF31 servisi alacak şekilde işaretlenmiş olmaktadır.

PRIORİTİZE KUYRUK : PRIORİTİZE basit öncelik kuyruklaması (simple priority queueing) kullanır. Yüksek öncelikli kuyruktaki paketler, her zaman düşük öncelikli kuyruktakilerden önce seçilirler.

Herbiri FIFO kuyruk disiplini kullanan üç adet bant içerir. Sıfırıncı banttaki paketler önce iletilir, bundan sonra sırayla birinci ve ikinci banttaki paketler iletilir.

Ingress (Giriş) kuyruk disiplini : Ingress (Giriş) kuyruk disiplini, belirli bir hızı geçen paketleri reddederek gelen trafiği belirli bir hıza limitlemek için kullanılır. Ingress kuyruk disiplini giriş tarafında işlem gören Linux'taki tek kuyruk disiplini.



Şekil 4.3 DSMARK ile paket işaretleme [Balliache, 2003]

4.2.1.2 Sınıfsız Kuyruk Disiplinleri

Aşağıda sınıfsız kuyruk disiplinleri açıklanmıştır.

FIFO (First In First Out) : FIFO (ilk giren ilk çıkar), en basit kuyruk disiplinidir. Paketler mümkün olan en hızlı şekilde, geldikleri sıra ile ağa iletilirler. Tutabileceği paket adedine göre tanımlanan tipine “pfifo”, tutabileceği oktet adedine göre tanımlanan tipine “bfifo” denilmektedir. FIFO, servis kalitesi uygulanmayan yönlendiricilerin normal operasyon şeklidir ve best effort (en iyiyi sağlama) trafiği için kullanılan kuyruk disiplinidir.

RED (Random Early Detection) : Floyd ve Jacobson (1993) tarafından tanımlanan Random Early Detection (RED) algoritması, kuyruk boyutlarını gözlemleyerek kuyruk tıkanıklığını engellemeye çalışır. Tıkanıklığın yüksek olması durumunda belirli bir olasılık değerine sahip paketleri reddetmeye başlar. Bu durum, gerçek bir tıkanıklığı simüle ederek TCP’nin iletim hızını düşürme yoluyla tepki vermesini sağlar ve tıkanıklık azalır.

Aşağıda RED konfigürasyonu için kullanılan parametreler açıklanmıştır:

limit : Paketlere ayrılan ara bellek boyutudur.

avpkt : Tahmini ortalama paket boyutudur.

burst : Trafik patlaması durumunda ara belleğe sığacak *avpkt* boyutlu paket adedidir.

bandwidth : Kuyruk disiplininin bağlandığı hattın hızıdır.

Ortalama buffer seviyesi *min* ve *max* ile tanımlanan limitler arasında ise, paketlerin reddedilme olasılıkları ortalama ara bellek seviyesine lineer bağlıdır. Paketlerin reddedilme olasılıkları, avg ortalama ara bellek seviyesi olmak üzere (4-1) bağıntısına göre hesaplanır: [Lemponen, 2000]

$$P = \text{probability} * \frac{\text{avg} - \text{min}}{\text{max} - \text{min}} \quad (4.3)$$

Ortalama buffer seviyesi *min* değerinin altındaysa, hiçbir paket reddedilmez. Ortalama buffer seviyesi *max* değerinin üstündeyse, tüm paketler reddedilir. *Limit* parametresi, yeteri kadar büyük tanımlanmalıdır. MTU (Maximum Transmission Unit) en yüksek iletim birimi olmak üzere (4.2) bağıntısı sağlanmalıdır: [Markaki vd.]

$$\text{limit} \geq \text{MTU} * \text{burst} + \text{max} \quad (4.4)$$

Garantili Aktarım üç adet reddedilme seviyesi gerektirmektedir. RED bunu desteklemediğinden bu amaca yönelik olarak Linux'ta Generalized RED (GRED) geliştirilmiştir. GRED zamanlayıcı, 16'ya kadar reddedilme seviyesi tanımlamayı desteklemektedir. GRED'de fiziksel olarak tek bir kuyruk bulunmasına rağmen bu kuyruk sanal kuyruklara ayrılır. Tez çalışmasında Garantili Aktarım PHB için GRED kuyruk disiplinininden yararlanılmıştır.

SFQ (Stochastic Fairness Queueing) : SFQ, hesaplama bakımından WFQ'nun (Weighted Fair Queueing) daha basit bir şeklidir. SFQ, paketleri kaynak ve hedef adreslerine ve protokol numaralarına göre farklı akışlara ayırır. SFQ, her akış için bir FIFO kuyruk kullanır ve paketleri her bir FIFO'dan round-robin yöntemi ile seçer.

TEQL (True Link Equalizer) : True Link Equalizer birden çok fiziksel arabirimin bir sanal arabirim olarak birleştirilmesini sağlar.

TBF (Token Bucket Filter) : TBF yalnızca belirli bir hızı aşmayan paketleri geçiren ancak bu hızın aşıldığı durumda kısa patlamalara da izin verebilen basit bir kuyruk disiplini. Token bucket filter bölüm 2.5'te anlatılmış olan jeton kovası (token bucket) modeline göre çalışır. TBF uygulaması, belirli bir jeton hızı (token rate) ile jeton (token) doldurulan bir ara bellekten (bucket) oluşur. Jeton, paketlerin iletimi için kullanılan sanal bir bilgidir. "Token bucket" boyutu, depolanabilecek jeton miktarını belirler. Jetonlar paket iletimi için kullanılır ve paket iletilindiğinde bir jeton eksilir. TBF algoritmasında aşağıdaki üç senaryo oluşabilir :

1. Gelen her paket için bir jeton mevcutsa paketler gecikme olmadan iletilir. Paketlerin TBF'e gelme hızı jetonların dolma hızına eşit ise gerçekleşir.
2. Paketler jeton hızından daha küçük bir hızla gelirse, jetonların sadece belirli bir bölümü kullanılır ve jeton kovanı dolana kadar jetonlar birikir. Kullanılmayan jetonlar standart jeton hızından daha büyük hızda gelebilecek paketler için kullanılır.
3. Paketler jeton hızından daha büyük bir hızla gelirse, jeton kovanında kısa bir süre sonra jeton kalmaz ve TBF'in bir süre hız kesmesine (throttle) neden olur. Bu duruma "limit üstü durum" (overlimit situation) denilir. Paketler gelmeye devam ederse, paketler reddedilmeye başlanır.

TBF kuyruk disiplini de jeton biriktirme özelliği sayesinde "limit üstü" verinin kısa patlamalar şeklinde, veri kaybı olmadan iletimi sağlanır. Ancak uzun süreli yük oluşumunda paketler sürekli olarak gecikmeye tabi kalıp daha sonra reddedilirler. Linux kernel'ında jetonların paket yerine oktet cinsinden değerlendirildiği de ek olarak göz önüne alınmalıdır. [Rusu vd.]

4.2.2 Filtreler

Filtreler paketleri sınıflandırmak için kullanılmaktadır. Linux'ta desteği olan filtreler aşağıda listelenmiştir. Tez çalışmasında kullanılan filtre tipleri "u32" ve "tcindex" dir.

fw : Sınıflandırma kararı firewall'un paketi nasıl işaretlediğine bağlıdır.

u32 : Sınıflandırma kararı kaynak IP adresi, hedef portu gibi paket başlığındaki alanlara bağlıdır.

route : Sınıflandırma kararı yönlendirme tablolarına göre paketin nasıl yönlendirileceğine bağlıdır.

rsvp : Paketlerin RSVP isteklerine göre sınıflandırılmasını sağlar.

tcindex : DSMARK kuyruk disiplini ile birlikte kullanılır. DSMARK kuyruk disiplini paketin IP başlığındaki DS alanını sk_buff veri yapısındaki skb->tc_index alanına atar. Tcindex filtresi skb->tcindex değerini okuyarak sınıflandırma gerçekleştirir.

5. TASARIM

Bu tez çalışmasında hedeflenen amaç, Differentiated Services modeline göre farklı uygulamalara ait trafiklere farklı servis sağlayacak bir sistemin tasarlanmasıdır. Öncelikle farklılaştırılmış servis sağlamak üzere yönlendiricilerde uygulanması gereken DiffServ konfigürasyonu tasarlanmıştır. Bu konfigürasyonların DiffServ alanındaki yönlendiricilere merkezi bir yerden otomatik olarak gönderilerek tanımlanması ve güncellenmesi gerçekleştirilmiştir. Kansas Üniversitesi tarafından gerçekleştirilmiş olan “DiffSpec” uygulaması temel alınmıştır. [Sundaresan ve Dhandapani, 1999]

Differentiated Services uygulama platformu olarak, açık kaynak kodlu olması ve Differentiated Services desteği bulunması bakımından Linux platformu seçilmiştir. Kenar ve iç yönlendiriciler olarak Linux yüklü makineler kullanılmıştır.

Trafik kontrol konfigürasyonlarını ve yönlendirici bilgilerini merkezi olarak saklamak için dizin sunucusu kullanımı tercih edilmiştir. Konfigürasyon deposu olarak bir dizin sunucusu tercih edilmesinin nedeni saklanan konfigürasyonların dağıtım işleminin daha çok okuma işlemleri gerektirmesidir. Kaynak kodu açık olmasından dolayı dizin sunucusu olarak OpenLDAP seçilmiştir.

Linux'ta mevcut olan “tc” trafik kontrol uygulaması sadece komut satırından konfigürasyona izin vermektedir. Program kodu içinden trafik kontrol konfigürasyonu yapılmasını sağlayacak bir API mevcut değildir. Tez çalışmasında DiffServ konfigürasyonu trafik kontrol bileşenlerinin tanımlanabilmesi ve güncellenebilmesi için gerekli API oluşturulmuştur. Böylelikle trafik kontrol konfigürasyonu program kodu içinden çağrılabilen ve doğrudan Netlink [5] ile haberleşen fonksiyonlarla gerçekleştirilmiştir.

5.1 DiffServ Konfigürasyonu

Tasarım, Garantili Aktarım PHB, Hızlandırılmış Aktarım PHB ve “Best Effort” PHB desteklenecek şekilde gerçekleştirilmiştir. DiffServ konfigürasyonunu gerçekleştirmek için kenar yönlendiricilerde ve iç yönlendiricilerde iki ayrı konfigürasyon gerçekleştirilmiştir. Konfigürasyon detayları 5.1.1 ve 5.1.2 bölümlerinde açıklanmıştır.

İç yönlendirici konfigürasyonunda ilk olarak CBQ kuyruk disiplini kullanılmıştır. Ancak CBQ kuyruk disiplininin kullanıldığı konfigürasyonda, trafik tiplerine atanan bant genişliklerinde sapmalar görülmüştür. Bunun sonucunda iç yönlendiricilerde CBQ yerine CBQ'dan daha doğru değerler verdiği görülen HTB (Hierarchical Token Bucket) kuyruk disiplini kullanılmıştır.

CBQ'da hattın boş kalma süresi (idle time) EWMA (Exponential Weighted Moving Average) yöntemi ile ölçülürken HTB'de TBF (Token Bucket Filter) kullanılmaktadır. HTB, hız tahminindeki (rate estimation) doğruluğu daha yüksektir. [6]

Linux kernel 2.4.20 versiyonundan itibaren HTB desteği mevcuttur. Eski versiyon kernel'lar için <http://luxik.cdi.cz/~devik/qos/htb> adresinde yama (patch) bulunmaktadır.

5.1.1 Kenar Yönlendirici Konfigürasyonu

Kenar yönlendiriciler paketlerin DS alanına giriş noktaları olduğundan DSCP işaretleme ve paket limitleme işlemleri kenar yönlendirici konfigürasyonunda gerçekleştirilmiştir. DSCP işaretleme işlemi için DSMARK kuyruk yapısı kullanılmıştır. İşaretlenecek paketlerin seçilmesi u32 filtresi ile gerçekleştirilmiştir. U32 filtresi kullanılarak hedef IP adresi, hedef TCP/UDP portu, kaynak IP adresi, kaynak TCP/UDP portu, TOS oktet ve protokole göre sınıflandırma yapılabilir. Tez çalışmada uygulama tabanlı servis kalitesi gerçekleştirilmek hedeflendiğinden hedef port'a göre sınıflandırma temel alınmıştır. Filtreler ile istenen trafik profiline uygun olan ve olmayan paketler ölçülmektedir. Profile uygun EF paketleri 0x2e DSCP değeri ile işaretlenmekte, uymayanlar reddedilmektedir. Profile uygun AF paketleri ilgili DSCP ile işaretlenmekte, uymayanlar "best effort" olarak tekrar sınıflandırılmaktadır.

5.1.2 İç Yönlendirici Konfigürasyonu

İç yönlendirici konfigürasyonu için HTB kuyruk disiplini kullanılarak tasarlanmış olan [3]'teki AF PHB ve EF PHB konfigürasyonlarından yararlanılarak AF, EF ve BE olmak üzere üç PHB'yi birlikte destekleyen konfigürasyon hazırlanmıştır. Geliştirilen API kullanılarak program kodu içinden uygulanan iç yönlendirici DiffServ konfigürasyonu kolay anlaşılabilirliği açısından karşılık geldikleri "tc" komutları şeklinde Ek 1'de açıklanmıştır.

İç yönlendirici konfigürasyonunda öncelikle gelen paketlerin DS alanını *tc_index* alanına kopyalamak üzere DSMARK kuyruk disiplini oluşturulmuştur. *tc_index* alanındaki değerden DSCP değerini elde etmek üzere $(DS \text{ alanı} \& 0xfc) \gg 2$ işlemi gerçekleştirilmiştir.

DSMARK kuyruk disiplinine bağlı olarak HTB kuyruk disiplini yaratılmıştır. HTB kuyruk disiplini bant genişliği dağıtımı için bir ana sınıfa ihtiyaç duymaktadır. Bu ana sınıf bant genişliğinin tümünü kullanacak şekilde yaratılmıştır.

EF paketlerine istenen bant genişliğini ayırmak için ana HTB sınıfına bağlı bir HTB sınıfı yaratılmıştır. HTB'nin hat paylaşımı özellikleri bu konfigürasyonda göz önüne alınmadığından "ceil" değeri varsayılan değer olan "rate" değeri ile aynı verilmiştir. EF

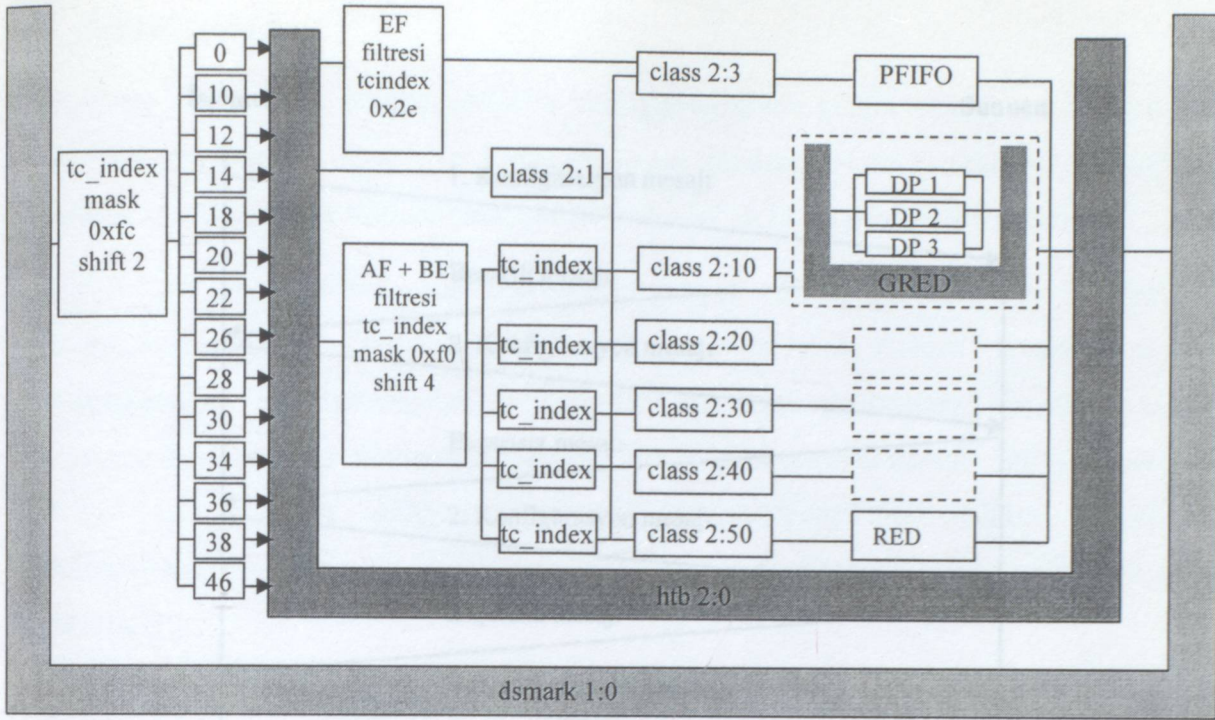
paketlerine ait HTB sınıfına bağlı olarak PFIFO kuyruk disiplini kullanılmıştır. EF paketleri kenar yönlendiricilerde DSCP değerleri 0x2e olacak şekilde işaretlenmektedir. DSCP değeri 0x2e olarak işaretlenmiş olan paketleri EF sınıfına yönlendirmek için “handle” değeri 0x2e olan bir filtre eklenmiştir.

AF konfigürasyonu AF1 sınıfı ile örneklendirme yapılarak aşağıda açıklanmıştır. AF1 sınıfına ait paketlerin DS alanı değerleri AF11, AF12 ve AF13 sınıfları için sırasıyla 0x28, 0x30 ve 0x38’dir. DS alanı değerlerine uygulanan $(DS \text{ alanı} \& 0xfc) \gg 2$ işlemi sonucunda elde edilen DSCP değerleri AF11, AF12 ve AF13 sınıfları için sırasıyla 0xa, 0xc ve 0xe’dir. Bu değerlere karşılık gelen ondalık değerler ise sırasıyla 10, 12 ve 14’tür. Gelen AF1 paketlerinin sınıflandırılması için “handle” değerleri 10, 12 ve 14 olan tcindex filtreleri kullanılmıştır. Filtreler paketlerin yönlendirilecekleri sınıfın minör numarasını döndürmektedir. Bu numara AF11, AF12 ve AF13 için sırasıyla 0x111, 0x112, 0x113’tür.

Yukarıda elde edilen sınıf numaralarına 0xf0 maskesi uygulanıp 4-bit sağa kaydırma işlemi gerçekleştirilmiştir. Bu işlem sonucunda elde edilen 0x1 değeri paketin AF1 sınıfına ait olduğunu göstermektedir. Son olarak paketler “handle” değeri “1” olan filtre sayesinde AF1 paketlerine ait olan HTB sınıfına yönlendirilmektedir. AF sınıfları için GRED kuyruk disiplini kullanılmıştır. Farklı reddedilme önceliklerine (drop priority – DP) sahip AF11, AF12, AF13 sınıfları için GRED kuyruk disiplinine üç sanal kuyruk ilave edilmiştir.

Kenar yönlendiricilerde DSCP değeri 0x0 olarak işaretlenmiş olan BE paketleri için “handle” değeri “0” olan filtre sayesinde paketlerin ilgili HTB sınıfına yönlendirilmesi sağlanmıştır. BE paketleri için RED kuyruk disiplini kullanılmıştır.

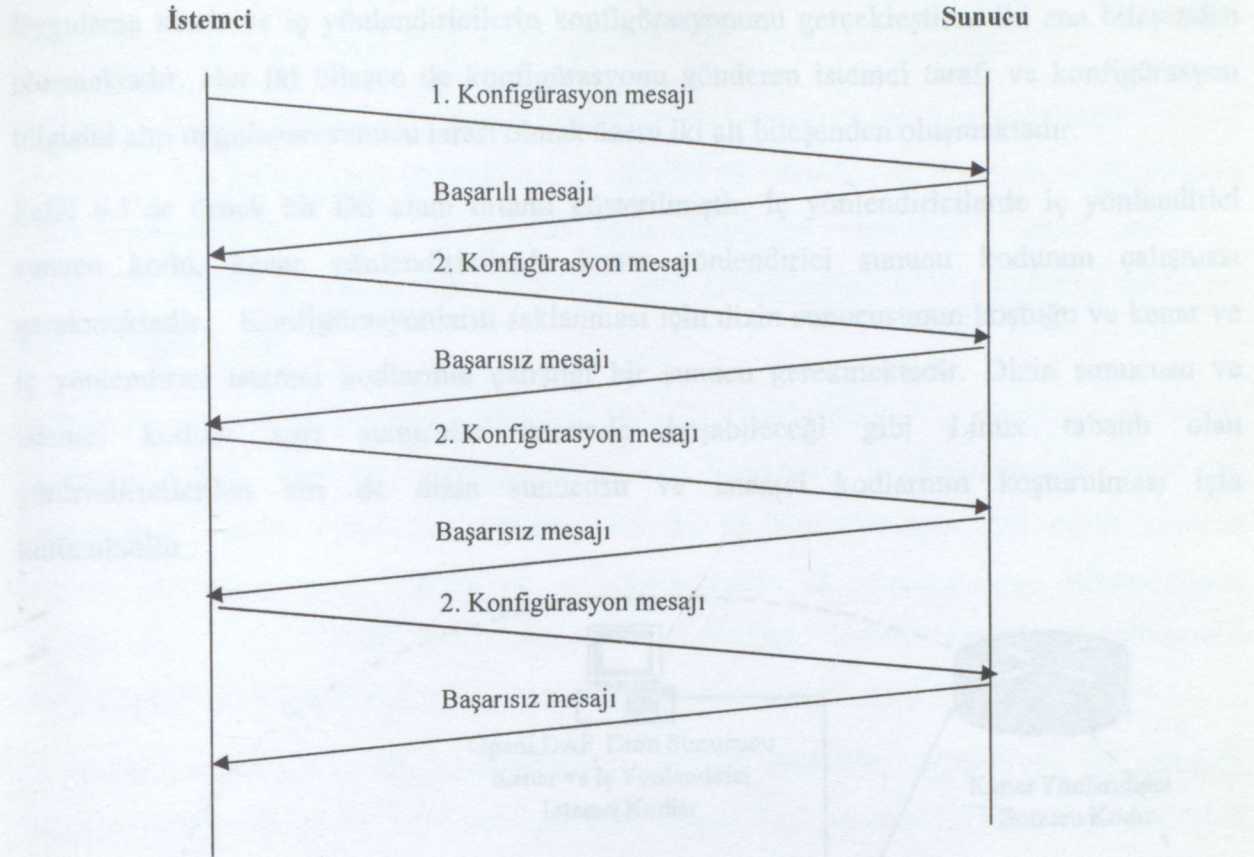
Şekil 5.1’de iç yönlendirici DiffServ konfigürasyonu gösterilmektedir. Şekilde sadece 2:10 sınıfının kendine bağlı bir GRED kuyruk disiplini gösterilmiş olmasına rağmen 2:20, 2:30 ve 2:40 sınıflarının da benzer şekilde kendilerine bağlı GRED kuyruk disiplinleri mevcuttur.



Şekil 5.1 İç yönlendirici DiffServ konfigürasyonu

5.2 İstemci-Sunucu Uygulamaları Arasındaki İletişim

Gerçekleştirilen uygulama istemci sunucu mimarisinde çalışmaktadır. Konfigürasyon bilgilerini dizin sunucusundan okuyup yönlendiricilere dağıtan program kodu istemci bileşenini, istemci programından gönderilen konfigürasyonları alıp uygulayan yönlendirici tarafındaki program kodu sunucu bileşenini oluşturmaktadır. İstemci ve sunucular arasındaki iletişim soket bağlantısı ile gerçekleştirilmiştir. Soket bağlantısı ile gönderilen konfigürasyonun başarı ile uygulanıp uygulanmadığı bilgisinin teyidi için çift taraflı iletişim gerçekleştirilmiştir. Ağda oluşabilecek hata ve gecikmelere karşı, sunucu program kodundan konfigürasyonun başarı ile uygulandığı bilgisi gelmezse, istemci program kodu konfigürasyonu tekrar gönderir. Başarısız mesajı gelmeye devam ederse konfigürasyon gönderme işlemi önceden belirlenen sayıda tekrarlanır. Bu tekrarlar da başarı sağlanamazsa program çıkar. Karşı taraftan konfigürasyonun başarı ile uygulandığı bilgisi gelirse bir sonraki konfigürasyon bilgisi gönderilir. İstemci sunucu arasındaki iletişim Şekil 5.2’de gösterilmiştir.

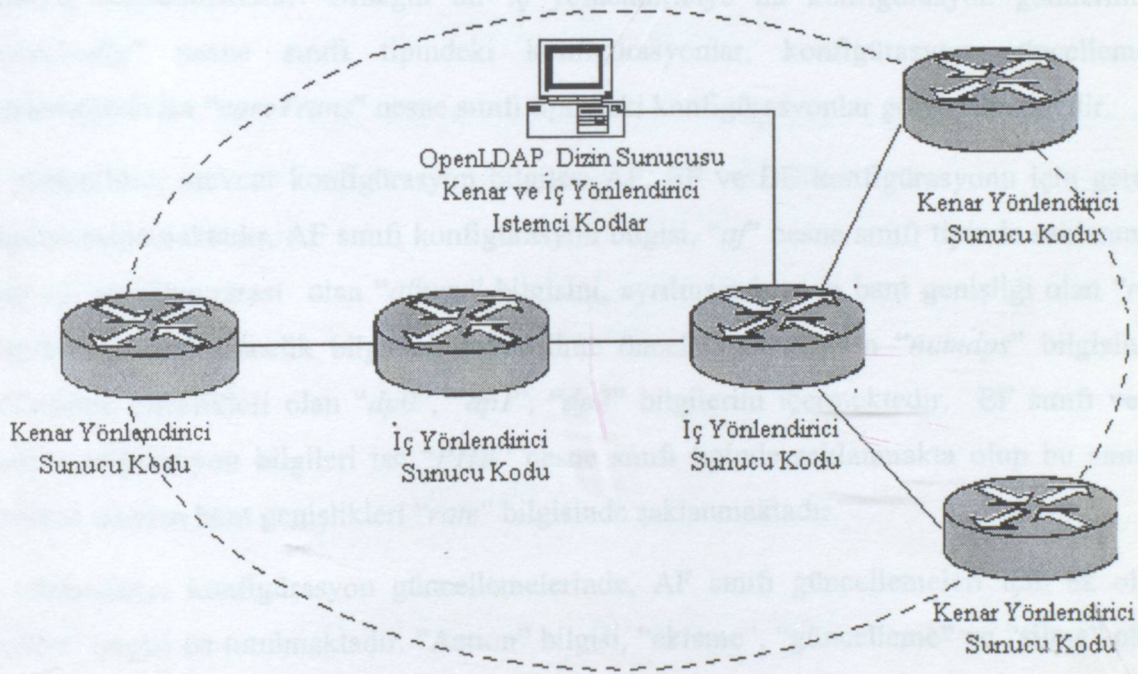


Şekil 5.2 İstemci sunucu arasındaki zamana bağlı iletişim

6. UYGULAMA

Uygulama kenar ve iç yönlendiricilerin konfigürasyonunu gerçekleştiren iki ana bileşenden oluşmaktadır. Her iki bileşen de konfigürasyonu gönderen istemci tarafı ve konfigürasyon bilgisini alıp uygulayan sunucu tarafı olmak üzere iki alt bileşenden oluşmaktadır.

Şekil 6.1’de örnek bir DS alanı ortamı gösterilmiştir. İç yönlendiricilerde iç yönlendirici sunucu kodu, kenar yönlendiricilerde kenar yönlendirici sunucu kodunun çalışması gerekmektedir. Konfigürasyonların saklanması için dizin sunucusunun koştugu ve kenar ve iç yönlendirici istemci kodlarının çalıştığı bir sunucu gerekmektedir. Dizin sunucusu ve istemci kodları ayrı sunucular üzerinde koşabileceği gibi Linux tabanlı olan yönlendiricilerden biri de dizin sunucusu ve istemci kodlarının koşturulması için kullanılabilir.



Şekil 6.1 Örnek DS alanı ortamı

LDAP dizin sunucusunda tutulan bilgiler ve oluşturulan şema bölüm 6.1’de, kenar ve iç yönlendirici konfigürasyonları bölüm 6.2 ve 6.3’te açıklanmıştır.

6.1 LDAP Dizin Sunucusu

Konfigürasyon bilgilerini tutmak için hazırlanmış olan LDAP veri şeması ve LDAP dizin sunucusuna erişim için kullanılmış olan fonksiyonlar aşağıda açıklanmıştır.

6.1.1 Veri Şeması

Yönlendirici bilgilerini ve yönlendiricilere gönderilecek konfigürasyon bilgilerini LDAP dizin sunucusunda saklamak üzere Şekil 6.2’de gösterilen LDAP şeması geliştirilmiştir.

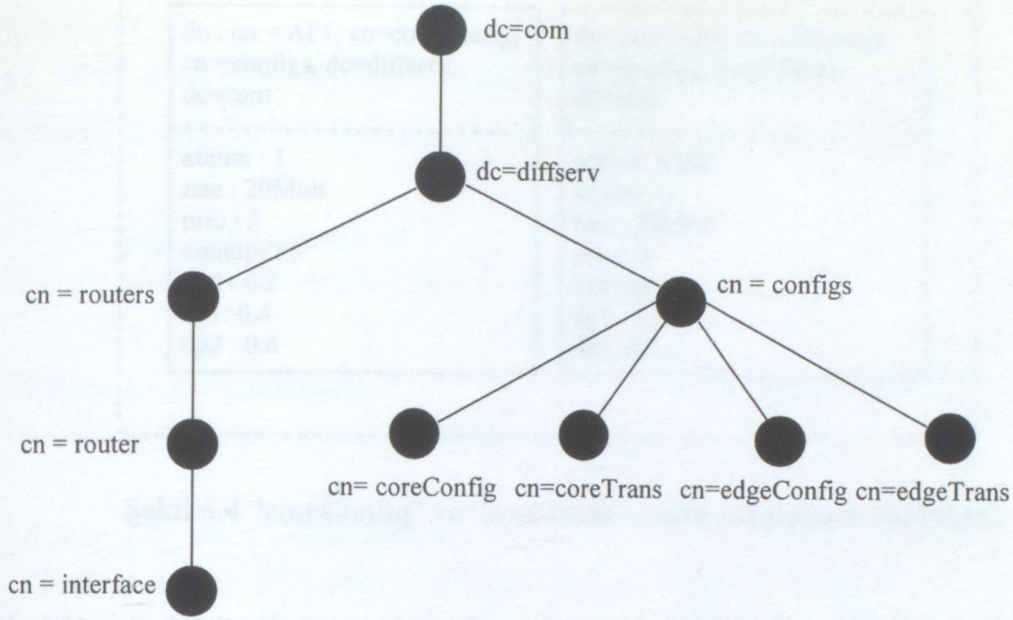
Şemadaki “router” nesne sınıfı, “kenar” (edge) ve “iç” (core) olmak üzere yönlendirici tipini ve yönetim için kullanılacak yönlendirici IP adresini tutmaktadır. Bu IP adresi yönlendiriciye soket bağlantısı ile erişim aşamasında kullanılmaktadır. “Interface” nesne sınıfı ise, yönlendiricilerdeki ağ cihazlarının isimlerini, bant genişliklerini ve IP adreslerini saklamak için kullanılmaktadır.

Şemada kenar ve iç yönlendirici konfigürasyonları ayrı saklanmaktadır. Bu ayrıma ek olarak, konfigürasyonlar “mevcut konfigürasyon” ve “mevcut konfigürasyonda yapılacak değişiklikler” olmak üzere ayrılmaktadır. Bu iki tip konfigürasyon ayrı nesne sınıfları şeklinde saklanmaktadır. Örneğin bir iç yönlendiriciye ilk konfigürasyon gönderiminde “coreConfig” nesne sınıfı tipindeki konfigürasyonlar, konfigürasyon güncellemeleri gönderiminde ise “coreTrans” nesne sınıfı tipindeki konfigürasyonlar gönderilmektedir.

İç yönlendirici mevcut konfigürasyon bilgileri AF, EF ve BE konfigürasyonu için gereken bilgileri saklamaktadır. AF sınıfı konfigürasyon bilgisi, “af” nesne sınıfı tipinde saklanmakta olup AF sınıf numarası olan “afnum” bilgisini, ayrılması istenen bant genişliği olan “rate” bilgisini, “prio” öncelik bilgisini, reddedilme önceliği adedi olan “numdps” bilgisini ve reddedilme öncelikleri olan “dp0”, “dp1”, “dp2” bilgilerini içermektedir. EF sınıfı ve BE sınıfı konfigürasyon bilgileri ise “PHB” nesne sınıfı tipinde saklanmakta olup bu sınıflara ayrılmak istenen bant genişlikleri “rate” bilgisinde saklanmaktadır.

İç yönlendirici konfigürasyon güncellemelerinde, AF sınıfı güncellemeleri için ek olarak “action” bilgisi de tutulmaktadır. “Action” bilgisi, “ekleme”, “güncelleme” ve “silme” olmak üzere gerçekleştirilebilecek işlemler için “ADD”, “CHANGE” ve “DEL” değerlerini almaktadır. EF ve BE konfigürasyonları ilk konfigürasyondan sonra ekleme ve silme gereksinimi olmayacağı kabulü ile yalnızca bant genişliği güncellemesi göz önüne alınmıştır. Bu bakımdan EF ve BE sınıflarının işlem tipi varsayılan olarak “CHANGE” kabul edilmiştir. AF sınıfı güncelleme bilgileri “aftrans” nesne sınıfında, EF ve BE sınıfı güncelleme bilgileri “PHBTrans” nesne sınıfında saklanmaktadır.

Kenar yönlendirici konfigürasyon bilgileri hangi trafik tiplerinin hangi DS alanı değerleri ile işaretlenmesi gerektiği bilgisini tutmaktadır, bu bilgilere göre kenar yönlendiricilerde ilgili sınıf ve filtreler yaratılmaktadır. Mevcut konfigürasyon “edgeConfig” nesne sınıfı tipinde, konfigürasyon güncellemeleri “edgeTrans” nesne sınıfı tipinde saklanmaktadır.



Şekil 6.2 LDAP şeması

Şekil 6.3'te "interface" nesne sınıfının özellikleri (attribute) örnek veri ile gösterilmiştir. Ağ cihazının ismi "cn" özelliğinde, IP adresi "ipaddr" özelliğinde, bant genişliği "bandwidth" özelliğinde tutulmaktadır.

dn : cn = eth0, cn=power1, cn=routers, dc=diffserv, dc=com

ipaddr = 10.1.1.1 bandwidth : 100Mbit

Şekil 6.3 "interface" nesne sınıfının özellikleri

Şekil 6.4'te "coreConfig" ve "coreTrans" nesne sınıflarının özellikleri örnek veri ile gösterilmiştir. Konfigürasyonu yapılacak sınıfın ismi "cn" özelliğinde, ayrılmak istenen bant genişliği "rate" özelliğinde, öncelik değeri "prio" özelliğinde, reddedilme öncelikleri "dp0", "dp1" ve "dp2" özelliklerinde tutulmaktadır. "numdps" özelliği reddedilme önceliği adedini göstermektedir. "afnum" özelliği ise sınıfın "handle" değerinin atanmasında kullanılmaktadır. "coreTrans" nesne sınıfında ek olarak konfigürasyonla ilgili yapılacak işlemi belirten "action" özelliği mevcuttur.

dn : cn = AF1, cn=coreConfig, cn=configs, dc=diffserv, dc=com	dn : cn = AF1, cn=coreTrans, cn=configs, dc=diffserv, dc=com
afnum : 1 rate : 20Mbit prio : 5 numdps : 3 dp0 : 0.2 dp1 : 0.4 dp2 : 0.6	action: ADD afnum : 1 rate : 20Mbit prio : 5 numdps : 3 dp0 : 0.2 dp1 : 0.4

Şekil 6.4 “coreConfig” ve “coreTrans” nesne sınıflarının özellikleri

Şekil 6.5’te “edgeConfig” ve “edgeTrans” nesne sınıflarının özellikleri örnek veri ile gösterilmiştir. İşaretlenecek DS alanı değeri “cn” özelliğinde tutulmakta, DS alanı değeri ile işaretlenecek trafiğin kaynak IP adresi “srcip”, hedef IP adresi “dstip”, hedef portu “dstport” ve kullandığı protokol “protocol” özelliklerinde tutulmaktadır. “classnum” özelliği yaratılacak filtrenin “handle” değerinin atanmasında kullanılmaktadır. “rate” ve “burst” değerleri ise paket limitleme işlemine ait parametrelerdir. “edgeTrans” nesne sınıfında ek olarak konfigürasyonla ilgili yapılacak işlemi belirten “action” özelliği mevcuttur.

dn : cn = 40, cn=edgeConfig, cn=configs, dc=diffserv, dc=com	dn : cn = 40, cn=edgeTrans, cn=configs, dc=diffserv, dc=com
classnum : 2 rate : 1Mbit dstip : 10.1.2.2 srcip : 10.1.1.2 dstport : 5001 protocol : tcp burst : 500K	action : ADD classnum : 2 rate : 1Mbit dstip : 10.1.2.2 srcip : 10.1.1.2 dstport : 5001 protocol : tcp

Şekil 6.5 “edgeConfig” ve “edgeTrans” nesne sınıflarının özellikleri

6.1.2 LDAP Dizin Sunucusuna Eriřim Fonksiyonları

LDAP dizin sunucusuna bağlantı kurulması, sonlandırılması ve kayıtların okunması için fonksiyonlar geliştirilmiştir. Bu fonksiyonlar aşağıda listelenmiştir :

6.1.2.1 my_ldap_init Fonksiyonu

LDAP dizin sunucusuna bağlantı kurar.

6.1.2.2 ldap_final Fonksiyonu

LDAP dizin sunucusuna kurulan bağlantıyı sonlandırır.

6.1.2.3 ldap_get_attributeFonksiyonu

İlgili LDAP kaydına ait istenilen özelliğin (attribute) değerini döndürür.

6.2 İç Yönlendirici Konfigürasyonu

İç yönlendiriciler için ilk konfigürasyon ve konfigürasyon güncellemelerini gerçekleřtirmek üzere iki uygulama kodu geliştirilmiştir. Aşağıda bu iki uygulama detaylı olarak anlatılmıştır.

6.2.1 İç Yönlendirici İlk Konfigürasyon Kodu

İlk konfigürasyon gönderiminde herbir iç yönlendirici için bir “thread” yaratılmaktadır. Her “thread”, bir iç yönlendiriciye soket bağlantısı kurup ilgili yönlendiriciye ait ağ cihazlarına veri deposundaki mevcut “coreConfig” nesne sınıfı tipindeki konfigürasyonları sırayla göndermektedir.

İletişimde kullanılan “packet_info” veri yapısı ve içerdği alanların anlamları Ek 3 bölümündeki şekillerde gösterilmektedir. İlk olarak “pkt_type” değeri “DEVICE” olan bir “packet_info” veri yapısı kullanılarak konfigürasyonun uygulanacağı ağ cihazının ismi ve bant genişliği bilgisi yönlendiriciye gönderilmektedir.

İkinci olarak “pkt_type” değeri “PHB” olan bir “packet_info” veri yapısı kullanılarak EF sınıfına uygulanması istenen bant genişliği bilgisi yönlendiriciye gönderilmektedir. Gönderilmek istenen AF sınıf bilgileri “pkt_type” değeri “AF_CLASS” olan bir “packet_info” veri yapısı kullanılarak yönlendiriciye gönderilmektedir. Bu işlem gönderilmek istenen AF sınıfları için tekrarlanmaktadır. BE sınıfına ayrılmak istenen bant genişliği ve BE sınıfının reddedilme önceliği bilgisi “pkt_type” değeri “PHB” olan bir “packet_info” veri yapısı kullanılarak yönlendiriciye gönderilmektedir.

Son olarak “pkt_type” değeri “EOF_SETUP” olan bir “packet_info” veri yapısı kullanılarak gönderilmek istenen konfigürasyonların tamamlandığı bilgisi yönlendiriciye gönderilmekte ve

LDAP dizin sunucusuna kurulmuş olan bağlantı sonlandırılmaktadır. Tüm “thread”ler çalışmalarını bitirdiğinde tüm yönlendiricilere konfigürasyonlar gönderilmiş demektir ve uygulama sonlanır.

6.2.2 İç Yönlendirici Konfigürasyon Güncelleme Kodu

Konfigürasyon güncellemeleri kodu, ilk konfigürasyon gönderme koduna benzer şekilde çalışmaktadır. Ana program her bir iç yönlendirici için bir “thread” yaratır. Konfigürasyon güncelleme bilgileri arasında EF ve BE ile ilgili bir güncelleme varsa bu konfigürasyonları “thread”lere parametre olarak verir.

Her “thread”, bir iç yönlendiriciye socket bağlantısı kurup ilgili yönlendiriciye ait ağ cihazlarına dizin sunucusundaki mevcut “coreTrans” nesne sınıfı tipindeki konfigürasyonları göndermektedir. Yönlendiriciye gönderilen konfigürasyon bilgileri için bölüm Ek 3’te anlatılmış olan veri yapıları kullanılmaktadır.

Konfigürasyon güncelleme kodu, ilk konfigürasyon kodundan farklı olarak her beş dakikada bir LDAP dizin sunucusuna yeni güncelleme kaydı eklenip eklenmediğini kontrol ederek konfigürasyon güncelleme bilgisi mevcutsa yönlendiriciye göndermekte, mevcut değilse tekrar beklemeye geçmektedir.

Eğer EF sınıfının bant genişliğinde güncelleme varsa öncelikle ilgili konfigürasyon gönderilir. EF konfigürasyonundan sonra “thread”ler LDAP dizin sunucusuna bağlanarak AF sınıfları ile ilgili güncellemeleri okuyup yönlendiricilere gönderirler. Güncelleme tipinin ekleme, değiştirme veya silme olmasına göre yönlendirici tarafındaki sunucu kod ilgili işlemi gerçekleştirir.

IETF tarafından tanımlanmış olan dört AF sınıfı bulunmaktadır. Yerel ağda kullanılabilecek sınıf adedini arttırabilmek için ek olarak iki AF sınıfı daha tanımlamasına olanak sağlanmıştır. Bu sınıflar diğer standart AF sınıfları gibi kendi içlerinde 3 adet reddedilme önceliği içerecek şekilde uygulanmıştır. Ek AF sınıfları için Çizelge 6.1’deki DSCP değerleri kullanılmıştır. AF52 sınıfına karşılık gelecek “101100” DSCP değeri gerçekte EF sınıfı için kullanıldığından AF5 sınıfına karşılık gelecek DSCP değerleri kullanılmamıştır.

Çizelge 6.1 Ek AF Sınıfları

SINIF	DSCP
AF61	110 010
AF62	110 100
AF63	110 110
AF71	111 010
AF72	111 100
AF73	111 110

Son olarak eğer BE sınıfının bant genişliğinde güncelleme mevcutsa ilgili konfigürasyon yönlendiriciye gönderilir. Yönlendirici tarafındaki sunucu koda “pkt_type” değeri “EOF_SETUP” olan bir “packet_info” veri yapısı gönderilerek soket bağlantısı sonlandırılır.

Ana program, tüm “thread”lerin çalışmasını bitirmesini bekledikten sonra mevcut konfigürasyon bilgilerini konfigürasyon güncelleme kayıtlarındaki bilgilere göre güncelleyip bir sonraki güncelleme kontrol zamanını bekler.

6.2.3 İç Yönlendirici Sunucu Kodu

İç yönlendirici sunucu kodu, iç yönlendirici üzerinde çalışır ve gelen konfigürasyon bilgilerini uygular. Sonsuz döngü içerisinde istemcilerden gelecek bağlantı isteklerini bekler. Konfigürasyon bilgisi geldiğinde konfigürasyonu uygulamak için bir alt işlem tetikler ve ana uygulama gelecek yeni bağlantı isteklerini beklemeye devam eder.

İç yönlendirici sunucu kodu, istemci kodundan gelen paket tipine göre aşağıda açıklanmış olan fonksiyonlardan uygun olanını çağırır. Bu fonksiyonlar, bölüm 5.1.2’de detaylı olarak anlatılmış olan iç yönlendirici DiffServ konfigürasyonu için gerekli olan kuyruk disiplinleri, sınıf ve filtreleri yaratır.

Aşağıdaki fonksiyonlar, trafik kontrol elemanlarını eklemek, güncellemek ve silmek için bölüm 6.4’te anlatılan trafik kontrol fonksiyonlarını çağırırlar.

6.2.3.1 set_dsmark Fonksiyonu

DSMARK kuyruk disiplinini yaratır.

6.2.3.2 set_phb Fonksiyonu

EF ve BE sınıfları için gerekli olan trafik kontrol elemanlarını yaratır.

6.2.3.3 modify_phb Fonksiyonu

EF ve BE sınıfları için ayrılan bant genişliklerini günceller.

6.2.3.4 set_af_class Fonksiyonu

AF sınıfı için gerekli olan trafik kontrol elemanlarını yaratır.

6.2.3.5 delete_af_class Fonksiyonu

AF sınıfı için yaratılmış olan trafik kontrol elemanlarını siler.

6.2.3.6 modify_af_class Fonksiyonu

AF sınıflarına ayrılan bant genişliklerini ve öncelik değerlerini günceller.

6.3 Kenar Yönlendirici Konfigürasyonu

Kenar yönlendirici istemci kodu, kenar yönlendirici sunucu koda hangi trafik tiplerinin hangi DS alanı değerleri ile işaretlenmesi gerektiği bilgisini gönderir. Sunucu kod bu bilgilere göre ilgili sınıf ve filtreleri yaratmaktadır.

Kenar yönlendirici konfigürasyonu iç yönlendirici konfigürasyonuna göre basittir. İç yönlendirici konfigürasyonunu gerçekleştiren uygulamaya benzer şekilde kenar yönlendirici konfigürasyon dağıtımı ve güncellemesini gerçekleştirmektedir. “Mevcut konfigürasyon” ve “mevcut konfigürasyonda yapılacak değişiklikler” LDAP dizin sunucusunda “edgeConfig” ve “edgeTrans” nesne sınıfları şeklinde saklanmaktadır.

6.4 Trafik Kontrol Fonksiyonları

Linux'ta kernel modülleri ile kullanıcı uzayı işlemleri arasında veri iletişimi için Netlink kullanılmaktadır. Netlink ailelerinden “Rtnetlink”, yönlendirme tablolarının okunup değiştirilmesi yanında kuyruk disiplinleri, sınıflar ve filtreler olmak üzere trafik kontrol elemanlarının kontrolü için kullanılmaktadır. Rtnetlink ile ilgili detaylı bilgi [3]'teki rtnetlink kullanım sayfasında bulunmaktadır.

Rtnetlink ailesindeki trafik kontrolü ile ilgili mesaj tipleri aşağıdaki gibidir:

RTM_NEWQDISC, RTM_DELQDISC, RTM_GETQDISC : Kuyruk disiplini eklemek, silmek ve elde etmek için kullanılır.

RTM_NEWTCLASS, RTM_DELTCLASS, RTM_GETTCLASS : Sınıf eklemek, silmek ve elde etmek için kullanılır.

RTM_NEWTFILTER, RTM_DELTFILTER, RTM_GETTFILTER : Filtre eklemek, silmek ve elde etmek için kullanılır.

Yeni trafik kontrol elemanı eklemek ile ilgili ek parametreler ve anlamları aşağıda listelenmiştir :

NLM_F_REPLACE	:	Mevcut nesnenin üzerine yaz.
NLM_F_EXCL	:	Nesne mevcutsa nesneyi değiştirme.
NLM_F_CREATE	:	Nesne mevcut değilse yarat.
NLM_F_APPEND	:	Nesne listesinin sonuna ekle.

Kuyruk disiplinleri, sınıflar ve filtreler olmak üzere trafik kontrol elemanlarını eklemek, güncellemek ve silmek üzere netlink ile haberleşen fonksiyonlar kullanılmıştır. Bu fonksiyonların parametre yapısı benzer tasarlanmıştır. Fonksiyonlara verilen “cmd” parametresi, yukarıda bahsi geçen mesaj tiplerinden birini, “flags” parametresi ise eğer gerekli ise yukarıda bahsi geçen ek parametrelerden uygun olanlarını içermektedir. Bu sayede trafik kontrol elemanlarının ekleme, güncelleme ve silme işlemleri parametrik olarak gerçekleştirilmiştir. Kullanılan fonksiyonlar aşağıda listelenmiş olup fonksiyonlar hakkında daha detaylı bilgi Ek 4’te verilmiştir.

6.4.1 modify_queue Fonksiyonu

Bu fonksiyon kuyruk disiplini eklemek, güncellemek ve silmek için kullanılmaktadır. İstenen kuyruk disiplininin tipine göre DSMARK kuyruk disiplini için add_ds_queue, GRED kuyruk disiplini için modify_gred_queue, CBQ kuyruk disiplini için modify_cbq_queue, RED kuyruk disiplini için modify_red_queue, PFIFO kuyruk disiplini için modify_pfifo_queue, HTB kuyruk disiplini için modify_htb_root alt fonksiyonlarını çağırılmaktadır.

6.4.2 modify_filter Fonksiyonu

Bu fonksiyon filtre eklemek ve silmek için kullanılmaktadır.

6.4.3 modify_cbq_class Fonksiyonu

Bu fonksiyon CBQ sınıfı eklemek, güncellemek ve silmek için kullanılmaktadır.

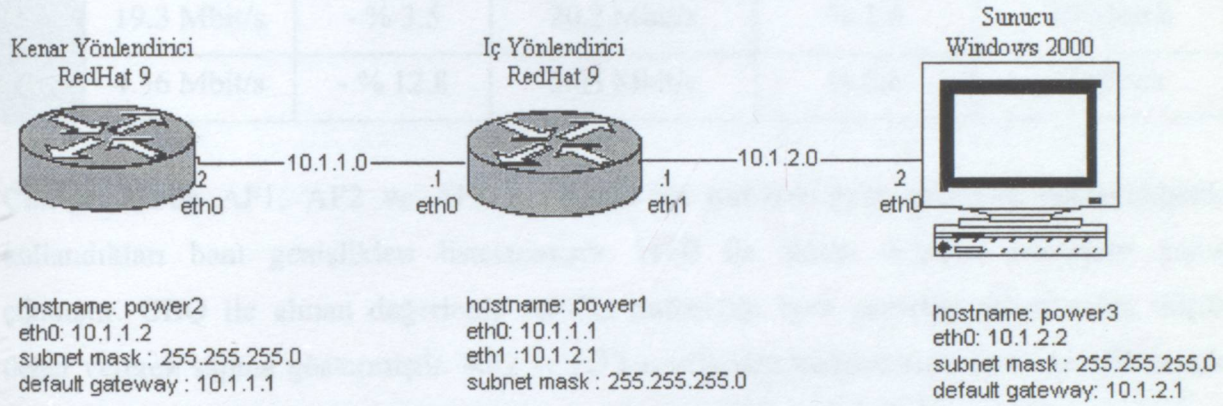
6.4.4 modify_htb_class Fonksiyonu

Bu fonksiyon HTB sınıfı eklemek, güncellemek ve silmek için kullanılmaktadır.

7. TEST

Gerçekleştirilen sistemin test edilmesi için bir yerel ağ oluşturulmuştur. Yerel ağda kenar yönlendirici ve iç yönlendirici olarak iki adet Red Hat 9 yüklü PC, üretilen trafiği alacak taraf olarak kullanılmak üzere bir adet Windows 2000 yüklü PC kullanılmıştır.

Yerel ağın yapısı ve ağ ortamının oluşturulabilmesi için kullanılan ayarlar Şekil 7.1'de gösterilmiştir. İç yönlendiricide `/proc/sys/net/ipv4/ip_forward` dosyasındaki parametre 1 değerine değiştirilerek ip forwarding açılmıştır.



Şekil 7.1 Test Yerel Ağı (I)

Testten önce yönlendiricilerin trafik kontrol konfigürasyonu gerçekleştirilmiştir. Kenar yönlendiricide AF, EF ve BE sınıflarını işaretleyen konfigürasyon uygulanmıştır. İç yönlendiricide ise AF, EF ve BE sınıflarına istenilen bant genişlik değerleri uygulanmıştır.

Trafik üretmek için Iperf [7] yazılımı kullanılmıştır. Iperf sunucu/istemci mimarisinde çalışan bir yazılımdır. Iperf kullanılarak kenar yönlendiriciden yerel ağdaki Windows PC'ye doğru trafik üretilmiştir. Kenar yönlendiricilerde belirli port'lara giden trafik farklı sınıflara ayrıldığından herhangi bir sınıf tipinden trafik üretmek için ilgili porta doğru trafik üretilmiştir. Testlerde 20 saniye boyunca trafik gönderilip kullanılan bant genişlikleri izlenmiştir. Aşağıda test sonuçları verilmiş olup sapma gösteren değerler kalın harflerle yazılmıştır.

Çizelge 7.1'de farklı sınıflara ait trafikler ayrı ayrı olarak gönderildiklerinde kullandıkları bant genişlikleri listelenmiştir. Paketler ayrı ayrı gönderildiklerinde CBQ ve HTB'de alınan sonuçların istenilen sonuçlara yakın çıktığı görülmüştür.

Çizelge 7.1 Test 1.1 sonuçları

	CBQ		HTB		İstenilen Bant Genişliği
	Bant Genişliği	Sapma Yüzdesi	Bant Genişliği	Sapma Yüzdesi	
AF1	20.9 Mbit/s	% 4.5	20.2 Mbit/s	% 1.0	20 Mbit/s
AF2	31.3 Mbit/s	% 4.3	30.2 Mbit/s	% 0.7	30 Mbit/s
AF3	10.4 Mbit/s	% 4.0	10.1 Mbit/s	% 1.0	10 Mbit/s
AF4	15.4 Mbit/s	% 2.7	15.1 Mbit/s	% 0.7	15 Mbit/s
EF	19.3 Mbit/s	- % 3.5	20.2 Mbit/s	% 1.0	20 Mbit/s
BE	4.36 Mbit/s	- % 12.8	5.03 Mbit/s	% 0.6	5 Mbit/s

Çizelge 7.2’de AF1, AF2 ve AF3 sınıflarına ait trafikler aynı zamanda başlatıldığında kullandıkları bant genişlikleri listelenmiştir. HTB ile alınan değerler istenilene yakın çıkmıştır. CBQ ile alınan değerlerde AF1’in kullandığı bant genişliği istenilenden düşük değer vererek sapma göstermiştir. AF2 ve AF3 sınıflarının kullandıkları bant genişliklerinde ise artma görülmüştür.

Çizelge 7.2 Test 1.2 sonuçları

	CBQ		HTB		İstenilen Bant Genişliği
	Bant Genişliği	Sapma Yüzdesi	Bant Genişliği	Sapma Yüzdesi	
AF1	18.7 Mbit/s	-% 6.5	20.1 Mbit/s	% 0.5	20 Mbit/s
AF2	46.2 Mbit/s	% 54.0	30.2 Mbit/s	% 0.7	30 Mbit/s
AF3	17.2 Mbit/s	% 72.0	10.1 Mbit/s	% 1.0	10 Mbit/s

Çizelge 7.3’te AF1, AF2, AF3 ve AF4 sınıflarına ait trafikler aynı zamanda başlatıldığında kullandıkları bant genişlikleri listelenmiştir. HTB ile alınan değerler istenilene yakın çıkmıştır. CBQ ile alınan değerlerde reddedilme önceliği daha yüksek olan AF1 sınıfının bant genişliği kullanımında oldukça büyük bir sapma görülmüştür. Diğer sınıfların kullandıkları bant genişliklerinde artma gözlemlenmiştir.

Çizelge 7.3 Test 1.3 sonuçları

	CBQ		HTB		İstenilen Bant Genişliği
	Bant Genişliği	Sapma Yüzdesi	Bant Genişliği	Sapma Yüzdesi	
AF1	6.5 Mbit/s	- % 67.5	20.1 Mbit/s	% 0.5	20 Mbit/s
AF2	36.9 Mbit/s	% 23.0	30.2 Mbit/s	% 0.7	30 Mbit/s
AF3	17.6 Mbit/s	% 76.0	10.1 Mbit/s	% 1.0	10 Mbit/s
AF4	24.8 Mbit/s	% 65.3	15.1 Mbit/s	% 0.7	15 Mbit/s

Çizelge 7.4'te EF ve BE sınıflarına ait trafikler aynı zamanda başlatıldığında kullandıkları bant genişlikleri listelenmiştir. CBQ ve HTB küçük sapmalarla istenilen bant genişliklerine yakın değerler vermiştir.

Çizelge 7.4 Test 1.4 sonuçları

	CBQ		HTB		İstenilen Bant Genişliği
	Bant Genişliği	Sapma Yüzdesi	Bant Genişliği	Sapma Yüzdesi	
EF	20.2 Mbit/s	% 1.0	19.4 Mbit/s	- % 3.0	20 Mbit/s
BE	4.36 Mbit/s	- % 12.8	5.03 Mbit/s	% 0.6	5 Mbit/s

Çizelge 7.5'te AF1, EF ve BE sınıflarına ait trafikler aynı zamanda başlatıldığında kullandıkları bant genişlikleri listelenmiştir. CBQ ve HTB istenilen bant genişliklerine yakın değerler vermiştir. CBQ değerlerindeki sapma daha fazladır.

Çizelge 7.5 Test 1.5 sonuçları

	CBQ		HTB		İstenilen Bant Genişliği
	Bant Genişliği	Sapma Yüzdesi	Bant Genişliği	Sapma Yüzdesi	
AF1	22.4 Mbit/s	% 12.0	20.2 Mbit/s	% 1.0	20 Mbit/s
EF	22.1 Mbit/s	% 10.5	20.2 Mbit/s	% 1.0	20 Mbit/s
BE	5.3 Mbit/s	% 6.0	5.03 Mbit/s	% 0.6	5 Mbit/s

Çizelge 7.6'de AF1, AF2, EF ve BE sınıflarına ait trafikler aynı zamanda başlatıldığında kullandıkları bant genişlikleri listelenmiştir. CBQ'da BE ve AF1 sınıflarının kullandıkları değerler istenilenden az çıkmıştır. Azalma görülen BE ve AF1 sınıfları, reddilme önceliği bakımından ilk iki sıradaki sınıftır. Trafik yükü arttıkça CBQ'nun BE sınıfının bant genişliğini oldukça fazla düşürdüğü görülmüştür. Diğer sınıfların kullandıkları bant genişliklerinin de arttığı görülmüştür. HTB'de ise genel olarak istenilen bant genişliklerine yakın değerler elde edilmekle beraber EF sınıfının kullandığı bant genişliğinde azalma görülmüştür.

Çizelge 7.6 Test 1.6 sonuçları

	CBQ		HTB		İstenilen Bant Genişliği
	Bant Genişliği	Sapma Yüzdesi	Bant Genişliği	Sapma Yüzdesi	
AF1	19.8 Mbit/s	- % 1.0	20.2 Mbit/s	% 1.0	20 Mbit/s
AF2	41.5 Mbit/s	% 38.3	30.2 Mbit/s	% 0.7	30 Mbit/s
EF	23.7 Mbit/s	% 18.5	18.3 Mbit/s	- % 8.5	20 Mbit/s
BE	0.93 Mbit/s	- % 81.4	5.03 Mbit/s	% 0.6	5 Mbit/s

Çizelge 7.7'de AF1, AF2, AF3, AF4, EF ve BE sınıflarına ait trafikler aynı zamanda başlatıldığında kullandıkları bant genişlikleri listelenmiştir. Bu testte tüm sınıflardan trafik birlikte üretilmiştir. Trafik yükünün fazlalığından dolayı CBQ ve HTB'de reddedilme önceliği bakımından ilk iki sıradaki AF1 ve BE sınıfının bant genişliklerinin istenilenin altına düşerek sapma gösterdiği görülmüştür. CBQ'da diğer sınıfların kullandıkları bant genişliklerinde artma gerçekleşmiştir. HTB'de EF sınıfının kullandığı bant genişliğinde ise azalma görülmüştür.

Çizelge 7.7 Test 1.7 sonuçları

	CBQ		HTB		İstenilen Bant Genişliği
	Bant Genişliği	Sapma Yüzdesi	Bant Genişliği	Sapma Yüzdesi	
AF1	9.9 Mbit/s	- % 50.5	9.71 Mbit/s	- % 51.5	20 Mbit/s
AF2	31.7 Mbit/s	% 5.7	30.1 Mbit/s	% 0.3	30 Mbit/s
AF3	15.7 Mbit/s	% 57.0	10.1 Mbit/s	% 1.0	10 Mbit/s
AF4	22.4 Mbit/s	% 49.3	15.1 Mbit/s	% 0.7	15 Mbit/s
EF	23.7 Mbit/s	% 18.5	18.6 Mbit/s	- % 7.0	20 Mbit/s
BE	1.93 Mbit/s	- % 61.4	1.17 Mbit/s	- % 76.6	5 Mbit/s

Yukarıda açıklanan testler trafik sınıflarına ayrılan bant genişlikleri daha düşük tanımlanarak ve eşit büyüklükte seçilerek tekrar gerçekleştirilmiştir. Tüm sınıflara 10 Mbit'lik bant genişliği ayrılarak ağ cihazının bant genişliğinin tamamının kullanılmasını gerektirmeyecek bir ortam oluşturulmuştur. Ortamdaki paket trafiği miktarı azaldığında CBQ'da gözlenen sapmaların da azaldığı görülmüştür.

Çizelge 7.8'de farklı sınıflara ait trafikler ayrı ayrı gönderildiklerinde kullandıkları bant genişlikleri listelenmiştir. Aşağıdaki diğer testlerde ise farklı trafik sınıflarına ait trafikler aynı zamanda başlatılarak ölçüm yapılmıştır. Trafik sınıfları ayrı ayrı gönderildiklerinde CBQ ve HTB'de alınan sonuçların istenilen sonuçlara oldukça yakın çıktığı görülmüştür. Trafik sınıfları birlikte gönderildiklerinde HTB istenilen değerlere yakın sonuçlar vermeye devam ederken CBQ'da istenilen değerlerden sapmalar görülmüştür.

Çizelge 7.8 Test 2.1 sonuçları

	CBQ		HTB		İstenilen Bant Genişliği
	Bant Genişliği	Sapma Yüzdesi	Bant Genişliği	Sapma Yüzdesi	
AF1	10.4 Mbit/s	% 4.0	10.1 Mbit/s	% 1.0	10 Mbit/s
AF2	10.4 Mbit/s	% 4.0	10.1 Mbit/s	% 1.0	10 Mbit/s
AF3	10.4 Mbit/s	% 4.0	10.1 Mbit/s	% 1.0	10 Mbit/s
AF4	10.4 Mbit/s	% 4.0	10.1 Mbit/s	% 1.0	10 Mbit/s
EF	10.1 Mbit/s	% 1.0	10.1 Mbit/s	% 1.0	10 Mbit/s
BE	10.4 Mbit/s	% 4.0	10.1 Mbit/s	% 1.0	10 Mbit/s

Çizelge 7.9 Test 2.2 sonuçları

	CBQ		HTB		İstenilen Bant Genişliği
	Bant Genişliği	Sapma Yüzdesi	Bant Genişliği	Sapma Yüzdesi	
AF1	12.9 Mbit/s	% 29	10.1 Mbit/s	% 1.0	10 Mbit/s
AF2	13.3 Mbit/s	% 33	10.1 Mbit/s	% 1.0	10 Mbit/s
AF3	13.5 Mbit/s	% 35	10.1 Mbit/s	% 1.0	10 Mbit/s

Çizelge 7.10 Test 2.3 sonuçları

	CBQ		HTB		İstenilen Bant Genişliği
	Bant Genişliği	Sapma Yüzdesi	Bant Genişliği	Sapma Yüzdesi	
AF1	14.7 Mbit/s	% 47	10.1 Mbit/s	% 1.0	10 Mbit/s
AF2	15.5 Mbit/s	% 55	10.1 Mbit/s	% 1.0	10 Mbit/s
AF3	15.9 Mbit/s	% 59	10.1 Mbit/s	% 1.0	10 Mbit/s
AF4	16.0 Mbit/s	% 60	10.1 Mbit/s	% 1.0	10 Mbit/s

Çizelge 7.11 Test 2.4 sonuçları

	CBQ		HTB		İstenilen Bant Genişliği
	Bant Genişliği	Sapma Yüzdesi	Bant Genişliği	Sapma Yüzdesi	
EF	10.6 Mbit/s	% 6.0	10.1 Mbit/s	% 1.0	10 Mbit/s
BE	10.4 Mbit/s	% 4.0	10.1 Mbit/s	% 1.0	10 Mbit/s

Çizelge 7.12 Test 2.5 sonuçları

	CBQ		HTB		İstenilen Bant Genişliği
	Bant Genişliği	Sapma Yüzdesi	Bant Genişliği	Sapma Yüzdesi	
AF1	11.6 Mbit/s	% 16	10.1 Mbit/s	% 1.0	10 Mbit/s
EF	12.2 Mbit/s	% 22	10.1 Mbit/s	% 1.0	10 Mbit/s
BE	11.4 Mbit/s	% 14	10.1 Mbit/s	% 1.0	10 Mbit/s

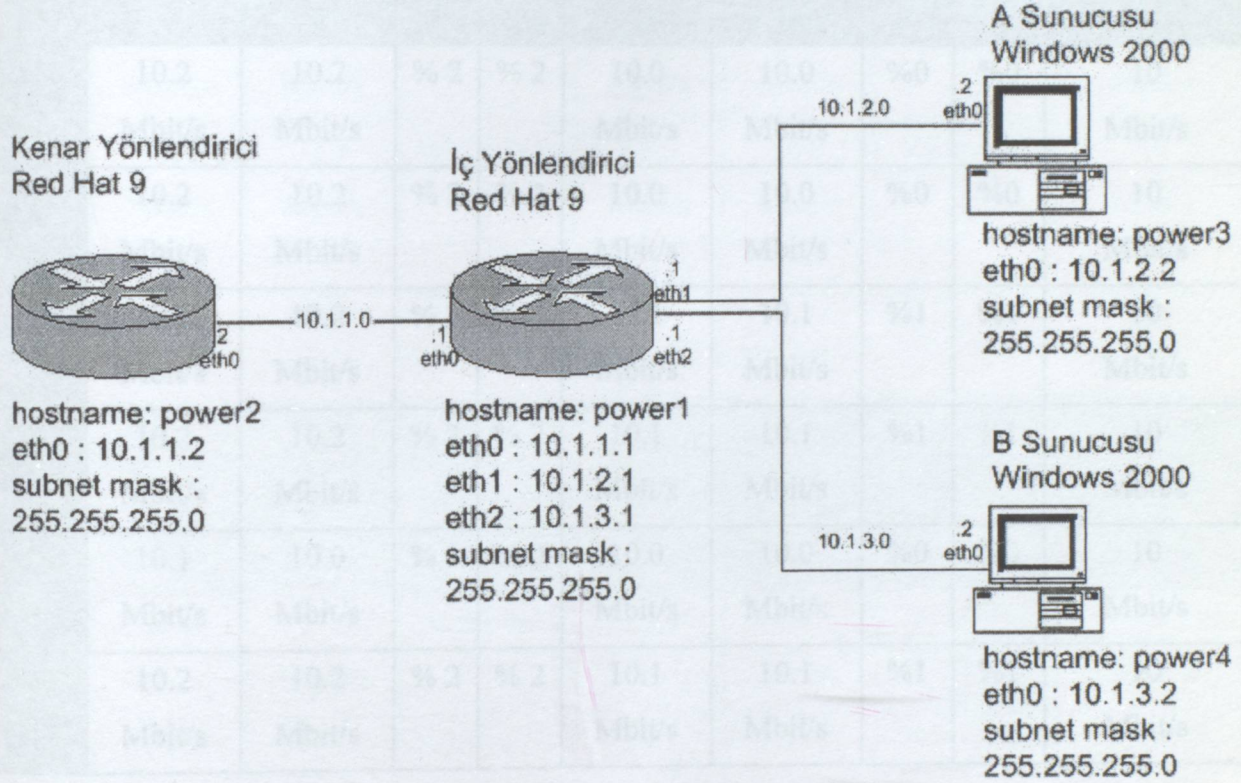
Çizelge 7.13 Test 2.6 sonuçları

	CBQ		HTB		İstenilen Bant Genişliği
	Bant Genişliği	Sapma Yüzdesi	Bant Genişliği	Sapma Yüzdesi	
AF1	13.0 Mbit/s	% 30	10.1 Mbit/s	% 1.0	10 Mbit/s
AF2	13.4 Mbit/s	% 34	10.1 Mbit/s	% 1.0	10 Mbit/s
EF	13.2 Mbit/s	% 32	10.1 Mbit/s	% 1.0	10 Mbit/s
BE	12.7 Mbit/s	% 25	9.74 Mbit/s	-%2.6	10 Mbit/s

Çizelge 7.14 Test 2.7 sonuçları

	CBQ		HTB		İstenilen Bant Genişliği
	Bant Genişliği	Sapma Yüzdesi	Bant Genişliği	Sapma Yüzdesi	
AF1	14.4 Mbit/s	% 44	10.1 Mbit/s	% 1.0	10 Mbit/s
AF2	15.6 Mbit/s	% 56	10.1 Mbit/s	% 1.0	10 Mbit/s
AF3	16.2 Mbit/s	% 62	10.1 Mbit/s	% 1.0	10 Mbit/s
AF4	16.4 Mbit/s	% 64	10.1 Mbit/s	% 1.0	10 Mbit/s
EF	15.6 Mbit/s	%56	10.1 Mbit/s	% 1.0	10 Mbit/s
BE	5.26 Mbit/s	-% 47.4	10.1 Mbit/s	% 1.0	10 Mbit/s

Yukarıdaki testlerin gerçekleştirildiği test ortamına, üretilen trafiği alacak ikinci bir sistem daha ilave edilerek Şekil 7-2'deki test ortamı oluşturulmuştur. "Power1" iç yönlendiricisindeki eth1 ve eth2 olmak üzere iki ethernet arabiriminde trafik kontrol konfigürasyonu gerçekleştirilmiştir. "Power2" kenar yönlendiricisinden "A" ve "B" sunucularına eş zamanlı olarak trafik üretilmiştir. Gerçekleştirilen testlerde tüm trafik sınıflarına 10Mbit/s'lik bant genişlikleri ayrılmıştır.



Şekil 7.2 Test Yerel Ağı (II)

Çizelge 7.15'te açıklanan testte trafik sınıfları ayrı zamanlarda ancak her trafik sınıfına ait trafik hem A hem de B sunucularına aynı anda gönderilmiştir.

Çizelge 7.15 Test 3.1 sonuçları

	CBQ				HTB				İstenilen Bant Genişliği
	Bant Genişliği	Bant Genişliği	Sapma Yüzdesi		Bant Genişliği	Bant Genişliği	Sapma Yüzdesi		
	A	B	A	B	A	B	A	B	
AF1	10.2 Mbit/s	10.2 Mbit/s	% 2	% 2	10.0 Mbit/s	10.0 Mbit/s	%0	%0	10 Mbit/s
AF2	10.2 Mbit/s	10.2 Mbit/s	% 2	% 2	10.0 Mbit/s	10.0 Mbit/s	%0	%0	10 Mbit/s
AF3	10.2 Mbit/s	10.2 Mbit/s	% 2	% 2	10.1 Mbit/s	10.1 Mbit/s	%1	%1	10 Mbit/s
AF4	10.2 Mbit/s	10.2 Mbit/s	% 2	% 2	10.1 Mbit/s	10.1 Mbit/s	%1	%1	10 Mbit/s
EF	10.1 Mbit/s	10.0 Mbit/s	% 1	% 0	10.0 Mbit/s	10.0 Mbit/s	%0	%0	10 Mbit/s
BE	10.2 Mbit/s	10.2 Mbit/s	% 2	% 2	10.1 Mbit/s	10.1 Mbit/s	%1	%1	10 Mbit/s

Aşağıdaki çizelgelerde gösterilen testlerde, belirtilen trafik sınıfları eş zamanlı olarak A ve B sunucularına gönderilmiştir.

Çizelge 7.16 Test 3.2 sonuçları

	CBQ				HTB				İstenilen Bant Genişliği
	Bant Genişliği	Bant Genişliği	Sapma Yüzdesi		Bant Genişliği	Bant Genişliği	Sapma Yüzdesi		
	A	B	A	B	A	B	A	B	
AF1	11.2	11.2	%12	%12	10.0	10.0	%0	%0	10
	Mbit/s	Mbit/s			Mbit/s	Mbit/s			Mbit/s
AF2	11.5	11.5	%15	%15	10.0	10.0	%0	%0	10
	Mbit/s	Mbit/s			Mbit/s	Mbit/s			Mbit/s

Çizelge 7.17 Test 3.3 sonuçları

	CBQ				HTB				İstenilen Bant Genişliği
	Bant Genişliği	Bant Genişliği	Sapma Yüzdesi		Bant Genişliği	Bant Genişliği	Sapma Yüzdesi		
	A	B	A	B	A	B	A	B	
AF1	12.8 Mbit/s	11.3 Mbit/s	%28	%13	10.1 Mbit/s	10.1 Mbit/s	1	%1	10 Mbit/s
AF2	13.3 Mbit/s	11.6 Mbit/s	%33	%16	10.1 Mbit/s	10.1 Mbit/s	%1	%1	10 Mbit/s
AF3	13.4 Mbit/s	10.4 Mbit/s	%34	%4	10.1	10	%1	%0	10 Mbit/s
AF4	12.6 Mbit/s	10.4 Mbit/s	%26	%4	10.1	10.1	%1	%1	10 Mbit/s

Çizelge 7.18 Test 3.4 sonuçları

	CBQ				HTB				İstenilen Bant Genişliği
	Bant Genişliği	Bant Genişliği	Sapma Yüzdesi		Bant Genişliği	Bant Genişliği	Sapma Yüzdesi		
	A	B	A	B	A	B	A	B	
AF1	11.5 Mbit/s	11.4 Mbit/s	%15	%14	10.1 Mbit/s	10.1 Mbit/s	%1	%1	10 Mbit/s
EF	9.8 Mbit/s	10 Mbit/s	-%2	%0	10 Mbit/s	9.9 Mbit/s	%0	-%1	10 Mbit/s
BE	11.2 Mbit/s	11.1 Mbit/s	%12	%11	10 Mbit/s	10 Mbit/s	%0	%0	10 Mbit/s

Çizelge 7.19’da açıklanan testte tüm trafik sınıfları eş zamanlı olarak A ve B sunucuları olmak üzere her iki sunucuya birlikte gönderilmiştir. Ortamdaki trafiğin yüksek olduğu bu testte hem CBQ hem de HTB’de kullanılan bant genişliklerinde azalmalar görülmüştür.

Çizelge 7.19 Test 3.5 sonuçları

	CBQ				HTB				İstenilen Bant Genişliği
	Bant Genişliği	Bant Genişliği	Sapma Yüzdesi		Bant Genişliği	Bant Genişliği	Sapma Yüzdesi		
	A	B	A	B	A	B	A	B	
AF1	10.7 Mbit/s	7.8 Mbit/s	%7	-%22	8.8 Mbit/s	7.8 Mbit/s	-%12	-%22	10 Mbit/s
AF2	10.3 Mbit/s	8.1 Mbit/s	%3	-%19	9.7 Mbit/s	7.8 Mbit/s	-%3	-%22	10 Mbit/s
AF3	10.3 Mbit/s	7.8 Mbit/s	%3	-%22	9.4 Mbit/s	7.9 Mbit/s	-%6	-%21	10 Mbit/s
AF4	9.8 Mbit/s	6.9 Mbit/s	-%2	-%31	9.1 Mbit/s	8.5 Mbit/s	-%9	-%15	10 Mbit/s
EF	9.7 Mbit/s	7.2 Mbit/s	-%3	-%28	6.5 Mbit/s	6.4 Mbit/s	-%36	-%35	10 Mbit/s
BE	8.8 Mbit/s	8.4 Mbit/s	-%12	-%16	8.4 Mbit/s	7.7 Mbit/s	-%16	-%23	10 Mbit/s

8. SONUÇ

Differentiated Services modeline göre farklı trafik tiplerine farklı servis kalitesi sağlayacak bir sistem tasarlanıp gerçekleştirilmiştir. Merkezi olarak tutulan trafik konfigürasyon bilgilerinin DiffServ alanındaki Linux yönlendiricilere uzaktan dağıtımı sağlanmıştır. Linux'taki mevcut "tc" trafik kontrol arayüz programı, trafik kontrol elamanlarının konfigürasyon işlemlerinin komut satırından gerçekleştirilmesini sağlamaktadır. Trafik kontrol elemanlarının konfigürasyonunun doğrudan uygulama içinden yapılabilmesi için bir programlama arabirimi geliştirilmiştir.

DiffServ tasarımı, Garantili Aktarım (Assured Forwarding) PHB, Hızlandırılmış Aktarım (Expedited Forwarding) PHB ve Best Effort PHB desteklenecek şekilde gerçekleştirilmiştir. Kenar ve iç yönlendiricilerde farklı DiffServ konfigürasyonları tasarlanmıştır. Kenar yönlendiriciler paketlerin DiffServ alanına giriş noktaları olduğundan DSCP işaretleme ve paket limitleme işlemleri kenar yönlendirici konfigürasyonunda gerçekleştirilmiştir. İç yönlendiricilerde AF, EF ve BE olmak üzere üç PHB'yi destekleyen bir DiffServ konfigürasyonu tasarlanmıştır.

Gerçekleştirilen uygulama istemci sunucu mimarisinde çalışmaktadır. Konfigürasyon bilgilerini dizin sunucusundan okuyup yönlendiricilere dağıtan program kodu istemci bileşenini, istemci programından gönderilen konfigürasyonları alıp uygulayan yönlendirici tarafındaki program kodu sunucu bileşenini oluşturmaktadır. Kenar ve iç yönlendirici DiffServ yapıları farklı olduğundan kenar ve iç yönlendiriciler için ayrı istemci ve sunucu bileşenleri geliştirilmiştir.

Trafik konfigürasyon bilgileri merkezi olarak bir OpenLDAP dizin sunucusunda tutulmuştur. Konfigürasyon bilgilerini saklamak için dizin sunucusu tercih edilmesinin nedeni saklanan konfigürasyonların dağıtım işleminin daha çok okuma işlemleri gerektirmesidir.

Yönlendirici bilgilerini ve yönlendiricilere gönderilecek trafik konfigürasyon bilgilerini LDAP dizin sunucusunda saklamak üzere bir veri şeması geliştirilmiştir. Şemada kenar ve iç yönlendiricilere ait konfigürasyonlar ayrı saklanmaktadır. Yönlendiricilerin ilk konfigürasyonu yapıldıktan sonra mevcut konfigürasyonlar üzerinde gerçekleştirilmesi istenen güncellemeler dizin sunucusundan okunarak Linux yönlendiricilere otomatik olarak uzaktan dağıtılabilmektedir.

İstemci kodu ve sunucu kodu arasındaki iletişim soket bağlantısı ile gerçekleştirilmiştir. İstemciden sunucuya doğru gönderilen konfigürasyon bilgisinin yanında konfigürasyonun başarılı veya başarısız olduğuna dair sunucudan istemciye doğru da bilgi aktarımı

gerçekleştirilmiştir. Konfigürasyonun başarı ile uygulanmadığı durumda konfigürasyon bilgisi istemci tarafından sunucuya tekrar gönderilmektedir.

DiffServ konfigürasyon tasarlanırken CBQ ve HTB algoritmaları göz önüne alınmıştır. CBQ algoritmasının kullanıldığı konfigürasyonda, trafik tiplerine atanan bant genişliklerinde sapmalar görülmüştür. Bunun sonucunda CBQ yerine CBQ algoritmasından daha doğru değerler verdiği görülen HTB algoritması kullanımı tercih edilmiştir. CBQ'da hattın boş kalma süresi EWMA (Exponential Weighted Moving Average) yöntemi ile ölçülürken HTB'de TBF (Token Bucket Filter) yöntemi kullanılmaktadır. TBF, hız tahminindeki (rate estimation) doğruluğu daha yüksektir.

Gerçekleştirilen tez çalışmasına ek olarak ağ ortamındaki trafiğin durumuna göre sisteme geri besleme verilmesi, böylelikle merkezi tutulup dağıtılan konfigürasyon bilgilerinin ağ ortamından toplanabilecek bilgilere göre değiştirilmesi ek çalışma olarak gerçekleştirilebilir.

Braden, R., Zhang, L. vd., (1997), "Resource ReSerVation Protocol (RSVP) - Version 1 Functional Specification", Network Working Group Request for Comments: 2205.

Brown, M. A., (2003), "Traffic Control - HOWTO", <http://www.linux.org/docs/HOWTO/other-domains/pdf/Traffic-Control-HOWTO.pdf>.

Davis, B., Chavv, A., Benson, E., Courtney, W., Davari, S., Frenk, V. ve Stiller, D., (2002), "An Expedited Forwarding PHB (Per-Hop Behavior)", Network Working Group Request for Comments: 2470.

Floyd, S. ve Jacobson, V., (1983), "Random early detection gateways for congestion avoidance", IEEE/ACM Transactions on Networking, volume 1.

Floyd, S. ve Jacobson, V., (1993), "Link sharing and resource management algorithms for packet networks", IEEE/ACM Transactions on Networking, volume 1, <http://www.ietf.org/ietf/papers/1993/>.

Heimann, J., Balas, J., Weiss, W. ve Wroclawski, J., (1999), "Assured Forwarding PHB Group", Network Working Group Request for Comments: 2597.

Lempoczn, J., (2004), "Implementation of Differentiated Services Policy Information Base on Linux", Bitume Tezi, Isopore University of Technology Department of Information Technology.

KAYNAKLAR

Almquist, P., (1992), "Type of Service in the Internet Protocol Suite", Network Working Group Request for Comments: 1349.

Bai, H., Atiquzzaman, M. ve Ivancic, W., "Running Integrated Services over Differentiated Service Networks: Quantitative Performance Measurements".

Balliache, L., (2003), "Differentiated Service on Linux HOWTO", <http://www.opalsoft.net/qos/DS.htm>.

Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z. ve Weiss, W., (1998), "An Architecture for Differentiated Services", Network Working Group Request for Comments: 2475.

Braden R., Clark, D. ve Shenker, S., (1994), "Integrated Services in the Internet Architecture: an Overview", Network Working Group Request for Comments : 1633.

Braden, R., Zhang, L. vd., (1997), "Resource ReSerVation Protocol (RSVP) -- Version 1 Functional Specification", Network Working Group Request for Comments: 2205.

Brown, M. A. , (2003), "Traffic Control HOWTO", <http://www.ibiblio.org/pub/Linux/docs/HOWTO/other-formats/pdf/Traffic-Control-HOWTO.pdf>.

Davie, B., Charny, A., Benson, K., Courtney, W., Davari, S., Firoiu, V. ve Stiliadis, D., (2002), "An Expedited Forwarding PHB (Per-Hop Behavior)", Network Working Group Request for Comments: 3246.

Floyd, S. ve Jacobson V., (1993), "Random early detection gateways for congestion avoidance", IEEE/ACM Transactions on Networking, volume 1.

Floyd, S. ve Jacobson V., (1995), "Link-sharing and resource management models for packet networks", IEEE/ACM Transactions on Networking, volume 3, <http://www.aciri.org/floyd/papers/link.pdf>.

Heinanen, J., Baker, F., Weiss, W. ve Wroclawski, J., (1999), "Assured Forwarding PHB Group", Network Working Group Request for Comments: 2597.

Lemponen, J., (2000), "Implementation of Differentiated Services Policy Information Base on Linux", Bitirme Tezi, Tampere University of Technology Department of Information Technology.

Majalainen, J., (2002), "Implementation of Policy Management Tool for Bandwidth Provisioning", Bitirme Tezi, Tampere University of Technology Department of Information Technology.

Manner, J., (2003), "Short Overview of Differentiated Services".

Markaki, M., Saltouros, M. P., ve Venieris I. S., "Comparison of Packet Schedulers for Differentiated Services in the Internet", National Technical University of Athens, School of Electrical and Computer Engineering Telecommunications Laboratory.

Nichols, K., Blake, S., Baker, F. ve Black, D., (1998), "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", Network Working Group Request for Comments: 2474.

Ramakrishnan, K., Floyd, S., ve Black, D., (2001), "The Addition of Explicit Congestion Notification (ECN) to IP", Network Working Group Request for Comments: 3168.

Rexhepi, V. ve Heijenck, G., (2000), "Interoperability of Integrated Services and Differentiated Services Architectures".

Rusu, O., Subredu, M., Sparlea, I. ve Vraciu, V., "Implementing Real Time Packet Forwarding Policies using HTB".

Schwantag, U., (1997), "An Analysis of the Applicability of RSVP", Bitirme Projesi, Institute of Telematics Fakult. at f. ur Informatik Universit. at Karlsruhe (TH).

Shenker, S., Partridge, C. ve Guerin, R., (1997), "Specification of Guaranteed Quality of Service", Network Working Group Request for Comments: 2212.

Stanisic, V. ve Devetsikiotis M., (2001), "Providing Quality of Service in the Internet", North Carolina State University Department of Electrical and Computer Engineering.

Sundaresan, A. ve Dhandapani, G., (1999), DiffSpec - A Differentiated Services tool, Information and Telecommunications Technology Center Department of Electrical Engineering & Computer Science The University of Kansas.

Tanenbaum, A. S., Computer Networks, Prentice Hall.

Wroclawski, J., (1997), "Specification of the Controlled-Load Network Element Service", Network Working Group Request For Comments: 2211.

INTERNET KAYNAKLARI

- [1] Integrated Services on the Internet, http://www.cse.ohio-state.edu/~jain/cis788-97/ftp/integrated_services/#parekh.
- [2] QoS support in Linux, http://www.ittc.ukans.edu/~pramodh/courses/linux_qos/qos_qcf.html.
- [3] Differentiated Service on Linux HOWTO ,<http://www.opalsoft.net/qos/DS.htm>.
- [4] Devera, M., HTB Linux queuing discipline manual - user guide, <http://luxik.cdi.cz/~devik/qos/htb/manual/userg.htm>.
- [5] RTNETLINK Manual Page, <http://netadmintools.com/html/7rtnetlink.man.html>.
- [6] Devera, M.,<http://luxik.cdi.cz/~devik/qos/htb/old/htbmeas1.htm>.
- [7] Iperf, <http://dast.nlanr.net/Projects/Iperf>.

EKLER

Ek 1 “TC” komutları ile iç yönlendirici konfigürasyon komutları

Ek 2 Dizin sunucusu fonksiyonları

Ek 3 Veri yapıları

Ek 4 Trafik kontrol fonksiyonları

Ek 1 “TC” komutları ile iç yönlendirici konfigürasyon komutları

İç yönlendirici konfigürasyonlarının daha kolay anlaşılabilmesi için, program kodu içinden uygulanan bu konfigürasyonlara karşılık gelen “tc” komutları hazırlanmış ve açıklamaları ile aşağıda listelenmiştir. Gerçekleştirilen uygulama bu komutları kullanmamakta, bunun yerine geliştirilen programlama arabirimini kullanmaktadır.

Gerçekleştirilen uygulamada parametre olarak verilebilen değerler aşağıdaki komutlarda n parametre numarasını göstermek üzere “param n ” şeklinde isimlendirilmiştir.

1 --- DSMARK

Gelen paketlerin DS alanını tc_index alanına kopyalamak üzere DSMARK kuyruk disiplini oluşturmaktadır.

```
tc qdisc add dev eth1 root handle 1:0 dsmark indices 64 set_tc_index;
```

2 --- DSMARK sınıflandırıcısı

tc_index alanındaki değerden DSCP değerini elde etmek üzere (DS alanı & 0xfc) >> 2 işlemi gerçekleştirmektedir.

```
tc filter add dev eth1 parent 1:0 protocol ip prio 1 tcindex mask 0xfc shift 2;
```

3 --- Ana HTB kuyruk disiplini

DSMARK kuyruk disiplinine bağlı olarak HTB kuyruk disiplini yaratılmaktadır.

```
tc qdisc add dev eth1 parent 1:0 handle 2:0 htb;
```

4 --- Ana HTB sınıfı

HTB kuyruk disiplini bant genişliği dağıtımı ve hat paylaşımı için bir ana sınıfa ihtiyaç duymaktadır. Bu ana sınıf hattın bant genişliğinin tümünü kullanacak şekilde yaratılmıştır, bant genişliği parametresi “param1” değeri ile verilmektedir.

```
tc class add dev eth1 parent 2:0 classid 2:1 htb rate param1 ceil param1;
```


5 --- EF Sınıfı

EF paketlerine istenen bant genişliğini ayırmak için ana HTB sınıfına bağlı bir HTB sınıfı yaratılmaktadır. HTB'nin hat paylaşımı özellikleri bu konfigürasyonda göz önüne alınmadığından "ceil" değeri, "param2" ile gösterilen "rate" değeri ile aynı verilmiştir

```
tc class add dev eth1 parent 2:1 classid 2:3 htb rate param2 ceil param2;
```

6 --- EF kuyruk disiplini

EF paketlerine ait HTB sınıfına bağlı olarak PFIFO kuyruk disiplini yaratılmaktadır.

```
tc qdisc add dev eth1 parent 2:3 pfifo limit 5;
```

7 --- EF kuyruk sınıflandırıcısı

DSCP değeri 0x2e olarak işaretlenmiş olan paketleri EF sınıfına yönlendirmek için "handle" değeri 0x2e olan filtre eklenmektedir.

```
tc filter add dev eth1 parent 2:0 protocol ip prio 1 handle 0x2e tcindex mask 0x2e classid 2:3 pass_on;
```

8 --- AF

Filtreler paketlerin yönlendirecekleri sınıfın minör numarasını döndürürler. Aşağıda, sınıfın minör numarasını 0xf0 değeri ile maskeleyip 4-bit sağa kaydırma işlemi gerçekleştirecek bir filtre yaratılmaktadır. Bu işlemler sonucunda paketin ait olduğu sınıfın numarası elde edilir. Örneğin AF1 sınıfına ait bir paket için bu değer 0x1 olacaktır.

```
tc filter add dev eth1 parent 2:0 protocol ip prio 2 tcindex mask 0xf0 shift 4 pass_on;
```

AF1, AF2, AF3 ve AF4 sınıfları için ana HTB sınıfına bağlı olarak alt HTB sınıfları yaratılmaktadır. "Param4", "param5", "param6" ve "param7" sırasıyla AF1, AF2, AF3 ve AF4 sınıflarına ayrılmak istenen bant genişliklerini göstermektedir.

```
tc class add dev eth1 parent 2:1 classid 2:10 htb rate param4 ceil param3;
```

```
tc class add dev eth1 parent 2:1 classid 2:20 htb rate param5 ceil param4;
```

```
tc class add dev eth1 parent 2:1 classid 2:30 htb rate param6 ceil param5;
```

```
tc class add dev eth1 parent 2:1 classid 2:40 htb rate param7 ceil param6;
```


BE sınıfı için ana HTB sınıfına bağlı olarak bir alt HTB sınıfı yaratılmaktadır. “Param8” BE sınıfına ayrılmak istenen bant genişliğini göstermektedir.

```
tc class add dev eth1 parent 2:1 classid 2:50 htb rate param8 ceil param8;
```

Handle değerleri 1, 2, 3 ve 4 olan filtreler sırasıyla AF1, AF2, AF3 ve AF4 sınıflarına ait paketleri ilgili HTB sınıfına yönlendirmektedir. “Param9”, “param10”, “param11” ve “param12” sırasıyla AF1, AF2, AF3 ve AF4 sınıflarına ait öncelikleri göstermektedir.

```
tc filter add dev eth1 parent 2:0 protocol ip prio param9 handle 1 tcindex classid 2:10;
```

```
tc filter add dev eth1 parent 2:0 protocol ip prio param10 handle 2 tcindex classid 2:20;
```

```
tc filter add dev eth1 parent 2:0 protocol ip prio param11 handle 3 tcindex classid 2:30;
```

```
tc filter add dev eth1 parent 2:0 protocol ip prio param12 handle 4 tcindex classid 2:40;
```

9 --- BE

BE sınıfına ait paketleri ilgili HTB sınıfına yönlendirmek için handle değeri 0 olan filtre yaratılmaktadır. “Param13” BE sınıfının önceliğini göstermektedir.

```
tc filter add dev eth1 parent 2:0 protocol ip param13 handle 0 tcindex classid 2:50;
```

10 --- AF1

Aşağıda AF11, AF12, AF13 sınıfları için GRED kuyruk disiplini konfigürasyonu yapılmaktadır. “Param14”, “param15” ve “param16” sırasıyla AF11, AF12 ve AF13 sınıflarının reddedilme öncelik değerlerini göstermektedir.

```
tc qdisc add dev eth1 parent 2:10 gred setup DPs 3 default 2 prio;
```

```
tc qdisc change dev eth1 parent 2:10 gred limit 60KB min 15KB max 45KB burst 20 avpkt  
1000 bandwidth 100Mbit DP 1 probability param14 prio 2;
```

```
tc qdisc change dev eth1 parent 2:10 gred limit 60KB min 15KB max 45KB burst 20 avpkt  
1000 bandwidth 100Mbit DP 2 probability param15 prio 3;
```

```
tc qdisc change dev eth1 parent 2:10 gred limit 60KB min 15KB max 45KB burst 20 avpkt  
1000 bandwidth 100Mbit DP 3 probability param16 prio 4;
```


AF11, AF12, AF13 sınıfları tipindeki paketlerin DSCP değerlerine göre sınıflandırılabilirmeleri için handle değerleri sırasıyla 10, 12 ve 14 olan filtreler yaratılmaktadır.

```
tc filter add dev eth1 parent 1:0 protocol ip prio 1 handle 10 tcindex classid 1:111;
```

```
tc filter add dev eth1 parent 1:0 protocol ip prio 1 handle 12 tcindex classid 1:112;
```

```
tc filter add dev eth1 parent 1:0 protocol ip prio 1 handle 14 tcindex classid 1:113;
```

11 --- AF2

Aşağıda AF21, AF22, AF23 sınıfları için GRED kuyruk disiplini konfigürasyonu yapılmaktadır. "Param17", "param18" ve "param19" sırasıyla AF21, AF22 ve AF23 sınıflarının reddedilme öncelik değerlerini göstermektedir.

```
tc qdisc add dev eth1 parent 2:20 gred setup DPs 3 default 2 prio;
```

```
tc qdisc change dev eth1 parent 2:20 gred limit 60KB min 15KB max 45KB burst 20 avpkt  
1000 bandwidth 100Mbit DP 1 probability param17 prio 2;
```

```
tc qdisc change dev eth1 parent 2:20 gred limit 60KB min 15KB max 45KB burst 20 avpkt  
1000 bandwidth 100Mbit DP 2 probability param18 prio 3;
```

```
tc qdisc change dev eth1 parent 2:20 gred limit 60KB min 15KB max 45KB burst 20 avpkt  
1000 bandwidth 100Mbit DP 3 probability param19 prio 4;
```

AF21, AF22, AF23 sınıfları tipindeki paketlerin DSCP değerlerine göre sınıflandırılabilirmeleri için handle değerleri sırasıyla 18, 20 ve 22 olan filtreler yaratılmaktadır.

```
tc filter add dev eth1 parent 1:0 protocol ip prio 1 handle 18 tcindex classid 1:121;
```

```
tc filter add dev eth1 parent 1:0 protocol ip prio 1 handle 20 tcindex classid 1:122;
```

```
tc filter add dev eth1 parent 1:0 protocol ip prio 1 handle 22 tcindex classid 1:123;
```

12 --- AF3

Aşağıda AF31, AF32, AF33 sınıfları için GRED kuyruk disiplini konfigürasyonu yapılmaktadır. "Param20", "param21" ve "param22" sırasıyla AF31, AF32 ve AF33 sınıflarının reddedilme öncelik değerlerini göstermektedir.


```
tc qdisc add dev eth1 parent 2:30 gred setup DPs 3 default 2 prio;
tc qdisc change dev eth1 parent 2:30 gred limit 60KB min 15KB max 45KB burst 20 avpkt
1000 bandwidth 100Mbit DP 1 probability param20 prio 2;
tc qdisc change dev eth1 parent 2:30 gred limit 60KB min 15KB max 45KB burst 20 avpkt
1000 bandwidth 100Mbit DP 2 probability param21 prio 3;
tc qdisc change dev eth1 parent 2:30 gred limit 60KB min 15KB max 45KB burst 20 avpkt
1000 bandwidth 100Mbit DP 3 probability param22 prio 4;
```

AF31, AF32, AF33 sınıfları tipindeki paketlerin DSCP değerlerine göre sınıflandırılabilirmeleri için handle değerleri sırasıyla 26, 28 ve 30 olan filtreler yaratılmaktadır.

```
tc filter add dev eth1 parent 1:0 protocol ip prio 1 handle 26 tcindex classid 1:131;
tc filter add dev eth1 parent 1:0 protocol ip prio 1 handle 28 tcindex classid 1:132;
tc filter add dev eth1 parent 1:0 protocol ip prio 1 handle 30 tcindex classid 1:133;
```

13 --- AF4

Aşağıda AF41, AF42, AF43 sınıfları için GRED kuyruk disiplini konfigürasyonu yapılmaktadır. “Param23”, “param24” ve “param25” sırasıyla AF41, AF42 ve AF43 sınıflarının reddedilme öncelik değerlerini göstermektedir.

```
tc qdisc add dev eth1 parent 2:40 gred setup DPs 3 default 2 prio;
tc qdisc change dev eth1 parent 2:40 gred limit 60KB min 15KB max 45KB burst 20 avpkt
1000 bandwidth 100Mbit DP 1 probability param23 prio 2;
tc qdisc change dev eth1 parent 2:40 gred limit 60KB min 15KB max 45KB burst 20 avpkt
1000 bandwidth 100Mbit DP 2 probability param24 prio 3;
tc qdisc change dev eth1 parent 2:40 gred limit 60KB min 15KB max 45KB burst 20 avpkt
1000 bandwidth 100Mbit DP 3 probability param25 prio 4;
```

AF41, AF42, AF43 sınıfları tipindeki paketlerin DSCP değerlerine göre sınıflandırılabilirmeleri için handle değerleri sırasıyla 34, 36 ve 38 olan filtreler yaratılmaktadır.

```
tc filter add dev eth1 parent 1:0 protocol ip prio 1 handle 34 tcindex classid 1:141;
```



```
tc filter add dev eth1 parent 1:0 protocol ip prio 1 handle 36 tcindex classid 1:142;
```

```
tc filter add dev eth1 parent 1:0 protocol ip prio 1 handle 38 tcindex classid 1:143;
```

14 --- BE

BE sınıfı için RED kuyruk disiplinini ve paketleri ilgili sınıfa yönlendirmek için ilgili filtreyi yaratır. "Param26" hattın bant genişliğini göstermektedir.

```
tc qdisc add dev eth1 parent 2:50 red limit 60KB min 15KB max 45KB burst 20 avpkt 1000
bandwidth param26 probability 0.4;
```

```
tc filter add dev eth1 parent 1:0 protocol ip prio 2 handle 0 tcindex mask 0 classid 2:5;
```

```
void my_ldap_get_attr(LDAP **ldp)
```

3. ldap_get_attrbetsFonksiyonu

İlgili LDAP kaydının ait istenen özelliğin (atribut) değerini döndürür. Fonksiyon parametre olarak hangi özelliğin istendiğini, LDAP bağlantı işaretçisini ve ilgili LDAP kaydına ait işaretçiyi alır. İstenen özelliğin değerini döndürür. Fonksiyon prototipi aşağıdaki gibidir :

```
char* ldap_get_attrbets(char *atribut,LDAP *ld,LDAPMessage *entry)
```


Ek 2 Dizin sunucusu fonksiyonları

1. my_ldap_init Fonksiyonu

LDAP dizin sunucusuna bağlantı kurar. Fonksiyon parametre olarak bağlantı için kullanılacak olan kullanıcı ismi ve şifresi ve kurulacak bağlantının çağırın fonksiyona geri döndürülebilmesi için bir bağlantı işaretçisi alır. Bağlantı başarılı olursa 0 değeri döndürür.

Fonksiyon prototipi aşağıdaki gibidir:

```
int my_ldap_init(char *user, char *credentials, LDAP **ld);
```

2. ldap_final Fonksiyonu

LDAP dizin sunucusuna kurulan bağlantıyı sonlandırır. Parametre olarak bağlantı işaretçisini alır. Fonksiyon prototipi aşağıdaki gibidir:

```
void my_ldap_final (LDAP **ld);
```

3. ldap_get_attributeFonksiyonu

İlgili LDAP kaydına ait istenilen özelliğin (attribute) değerini döndürür. Fonksiyon parametre olarak hangi özelliğin istendiğini, LDAP bağlantı işaretçisini ve ilgili LDAP kaydına ait işaretçiyi alır. İstenen özelliğin değerini döndürür. Fonksiyon prototipi aşağıdaki gibidir :

```
char* ldap_get_attribute(char *attribute, LDAP *ld, LDAPMessage *entry);
```


Ek 3 Veri yapıları

Sistemde kullanılan ana veri yapıları, kullanım amaçları, içerdikleri alanlar ve alanların amaçları aşağıda belirtilmiştir.

1. packet_info Veri Yapısı

packet_info veri yapısı, iç yönlendiricilere konfigürasyon göndermek amacı ile kullanılmaktadır. Yapısı ve içerdği alanların anlamları Şekil Ek3.1'de belirtilmiştir.

```
typedef struct _packet {
    enum pkt_type cmd;           // Gönderilen verinin tipi
    union {
        device_info device;      // Ağ cihazına ait parametreler
        phb_info phb;           // PHB ile ilgili parametreler
        af_class_info af_class;  // AF sınıfı ile ilgili parametreler
    } data;
} packet_info;
```

Şekil Ek3.1 packet_info veri yapısı

```
enum pkt_type {
    DEVICE,      // Ağ cihazına ait parametrelerin gönderildiği paket tipi
    PHB,         // EF ve BE PHB ile ilgili parametrelerin gönderildiği paket tipi
    AF_CLASS,    // AF sınıfı ile ilgili parametrelerin gönderildiği paket tipi
    EOF_SETUP,   // Gönderilecek konfigürasyonların tamamlandığını gösteren paket tipi
};
```

Şekil Ek3.2 pkt_type veri yapısı

```
typedef struct _dev_info {
    char device_name[16];          //Ağ cihazının ismi
    char device_bndw[MAX_ALLOC_SIZE]; //Ağ cihazının bant genişliği
} device_info;
```

Şekil Ek3.3 device_info veri yapısı


```

typedef struct _phb_info {
    enum phb_type cmd;                // PHB tipi
    request_type type;                // Yapılacak işlem
    union {
        char phb_bndw[MAX_ALLOC_SIZE]; //Atanacak bant genişliği
        be_info be_class;                // BE sınıfına ait parametreler
    } phb_params;
} phb_info;

```

Şekil Ek3.4 phb_info veri yapısı

```

enum phb_type {
    AF,                                // AF PHB
    EF,                                // EF PHB
    BE,                                // BE PHB
};

```

Şekil Ek3.5 phb_type veri yapısı

```

typedef enum _request_type{
    ADD_CMD,                          // Konfigürasyon ekleme işlemi
    DEL_CMD,                          // Konfigürasyon silme işlemi
    CHANGE_CMD                        // Konfigürasyon değiştirme işlemi
} request_type;

```

Şekil Ek3.6 request_type veri yapısı

```

typedef struct _be_info {
    char priority[MAX_ALLOC_SIZE];     // BE sınıfı önceliği
    char class_bndw[MAX_ALLOC_SIZE];  // BE sınıfına ayrılacak bant genişliği
    double drop_prob;                 // BE sınıfının reddedilme önceliği
} be_info;

```

Şekil Ek3.7 be_info veri yapısı


```

typedef struct _af_class_info {
    request_type type; // AF sınıfı için yapılacak işlem
    int af_class_num; // AF sınıfı numarası
    char class_bndw[MAX_ALLOC_SIZE]; // AF sınıfına ayrılacak bant genişliği
    char priority[MAX_ALLOC_SIZE]; // AF sınıfının önceliği
    int num_of_dps; // Reddedilme önceliği adedi
    double drop_prob[3]; // Reddedilme önceliği değerleri
} af_class_info;

```

Şekil Ek3.8 af_class_info veri yapısı

2. router_info Veri Yapısı

router_info veri yapısı, konfigürasyonları yönlendiricilere gönderen “thread”lere parametre olarak verilir. Yönlendirici ismi, IP adresi, istenilen EF ve BE bant genişliklerini içermektedir.

```

typedef struct _router_info {
    int thread_id; // Thread numarası
    char name[MAX_ALLOC_SIZE]; // Yönlendirici ismi
    char ip_addr[MAX_ALLOC_SIZE]; // Yönlendirici IP adresi
    char EFBW[MAX_ALLOC_SIZE]; // İstenilen EF bant genişliği
    char BEBW[MAX_ALLOC_SIZE]; // İstenilen BE bant genişliği
} router_info;

```

Şekil Ek3.9 router_info veri yapısı

Ek 4 Trafik kontrol fonksiyonları

Kansas Üniversitesinin geliştirdiği “DiffSpec” çalışmasında kullanılan fonksiyonlar tez çalışması kapsamında oluşturulan yeni DiffServ konfigürasyonuna göre gerektiği gibi değiştirilmiş, “güncelleme” ve “silme” gibi ek işlemler ve “DiffSpec” çalışmasında desteklenmeyen HTB için gerekli ek fonksiyonlar geliştirilmiştir. Ana fonksiyonlar ile ilgili bilgi aşağıda verilmiştir.

1. set_dsmark Fonksiyonu

DSMARK kuyruk disiplinini, DSCP değerini elde etmek üzere (DS alanı & 0xfc) >> 2 işlemini uygulayacak filtreyi, DSMARK kuyruk disiplinine bağlı ana HTB kuyruk disiplini ve sınıfını ve (DSCP & 0xf0)>>4 işlemini uygulayacak filtreyi yaratır. Fonksiyona ait prototip aşağıdadır :

```
int set_dsmark(packet_info packet);
```

2. set_phb Fonksiyonu

Parametre olarak gelen paket tipi EF ise, EF sınıfı için gerekli olan HTB sınıfını, 0x2e maskeleyişini gerçekleştiren filtreyi ve PFIFO kuyruk disiplinini yaratır. Parametre olarak gelen paket tipi BE ise, BE sınıfı için gerekli olan HTB sınıfını, RED kuyruk disiplinini ve ilgili filtreyi yaratır. Fonksiyona ait prototip aşağıdadır :

```
int set_phb(packet_info packet);
```

3. modify_phb Fonksiyonu

Parametre olarak gelen paket tipi EF ise EF sınıfına ait bant genişliğini, paket tipi BE ise BE sınıfına ait bant genişliğini değiştirir. Bunun için ilgili HTB sınıflarını günceller. Fonksiyona ait prototip aşağıdadır :

```
int modify_phb(packet_info packet);
```

4. set_af_class Fonksiyonu

AF sınıfı için gerekli olan trafik kontrol elemanlarını yaratır. GRED kuyruk disiplinini ekler, farklı reddedilme öncelikleri için GRED kuyruk disiplinine sanal kuyrukları ilave eder, ilgili filtreleri yaratır. Fonksiyona ait prototip aşağıdadır :

```
int set_af_class(packet_info packet);
```


5. delete_af_class Fonksiyonu

AF sınıfı için yaratılmış olan trafik kontrol elemelerini ters sırada siler. Fonksiyona ait prototip aşağıdadır:

```
int delete_af_class(packet_info packet);
```

6. modify_af_class Fonksiyonu

AF sınıflarına ayrılan bant genişliklerini ve öncelik değerlerini değiştirmek için ilgili HTB sınıflarını günceller. Fonksiyona ait prototip aşağıdadır :

```
int modify_af_class(packet_info packet);
```

7. modify_queue Fonksiyonu

Bu fonksiyon kuyruk disiplini eklemek, güncellemek ve silmek için kullanılmaktadır. Fonksiyon prototipi aşağıdaki gibidir. Fonksiyon başarılı olursa 0 değerini döndürür.

```
int modify_queue(int cmd, unsigned flags, char* dev, int root, struct qdisc *qdisc_info);
```

“cmd” parametresi : Uygulanacak işlemin tipini gösterir.

“flags” parametresi : Uygulanacak işlem ekleme ise “NLM_F_EXCL|NLM_F_CREATE”, güncelleme ise “NLM_F_CREATE”, silme ise “0” değerini alır.

“dev” parametresi : İşlemin hangi ağ cihazına uygulanacağını gösterir.

“root” parametresi : Kuyruk disiplini kök kuyruk disiplini ise 1, değilse 0 değerini alır.

“qdisc_info” parametresi.: Kuyruk disiplini ile ilgili bilgileri içeren veri yapısıdır.

“modify_queue” fonksiyonu kuyruk disiplininin tipine göre aşağıdaki alt fonksiyonlarını çağırır.

7.1. add_ds_queue Fonksiyonu

DSMARK kuyruk disiplini eklemek için kullanılır. Fonksiyon prototipi aşağıdaki gibidir. Fonksiyon başarılı olursa 0 değerini döndürür.

```
int add_ds_queue(char* dev, int root, struct qdisc *qdisc_info);
```

7.2. modify_gred_queue Fonksiyonu

GRED kuyruk disiplini eklemek, güncelleme ve silmek için kullanılır. Fonksiyon prototipi aşağıdaki gibidir. Fonksiyon başarılı olursa 0 değerini döndürür.

```
int modify_gred_queue(int cmd, unsigned flags, char* dev, int root, struct qdisc *qdisc_info);
```


7.3. modify_red_queue Fonksiyonu

RED kuyruk disiplini eklemek, güncelleme ve silmek için kullanılır. Fonksiyon prototipi aşağıdaki gibidir. Fonksiyon başarılı olursa 0 değerini döndürür.

```
int modify_red_queue(int cmd, unsigned flags, char* dev, int root, struct qdisc *qdisc_info);
```

7.4. modify_cbq_queue Fonksiyonu

CBQ kuyruk disiplini eklemek, güncelleme ve silmek için kullanılır. Fonksiyon prototipi aşağıdaki gibidir. Fonksiyon başarılı olursa 0 değerini döndürür.

```
int modify_cbq_queue(int cmd, unsigned flags, char* dev, int root, struct qdisc *qdisc_info);
```

7.5. modify_htb_root Fonksiyonu

HTB kuyruk disiplini eklemek için kullanılır. Fonksiyon prototipi aşağıdaki gibidir. Fonksiyon başarılı olursa 0 değerini döndürür.

```
int modify_htb_root(int cmd, unsigned flags, char* dev, int root, struct qdisc *htb_info_root);
```

8. modify_filter Fonksiyonu

Bu fonksiyon filtre eklemek ve silmek için kullanılmaktadır. Fonksiyon prototipi aşağıdaki gibidir. Fonksiyon başarılı olursa 0 değerini döndürür.

```
int modify_filter(int cmd, unsigned flags, char* dev, filter_ip *filter_info);
```

9. modify_cbq_class Fonksiyonu

Bu fonksiyon CBQ sınıfı eklemek, güncelleme ve silmek için kullanılmaktadır. Fonksiyon prototipi aşağıdaki gibidir. Fonksiyon başarılı olursa 0 değerini döndürür.

```
int modify_cbq_class(int cmd, unsigned flags, char* dev, int root, struct cbq_class *cbq_info);
```

10. modify_htb_class Fonksiyonu

Bu fonksiyon HTB sınıfı eklemek, güncelleme ve silmek için kullanılmaktadır. Fonksiyon prototipi aşağıdaki gibidir. Fonksiyon başarılı olursa 0 değerini döndürür.

```
int modify_htb_class(int cmd, unsigned flags, char* dev, int root, struct htb_class *htb_info_class);
```


ÖZGEÇMİŞ

Ad Soyad : Güneş Eda Ağar
Doğum Tarih : 17/07/1979
Doğum Yeri : İstanbul
Lise : 1990 -1997 Beyoğlu Anadolu Lisesi
Lisans : 1997-2001 İstanbul Üniversitesi Mühendislik Fakültesi
Bilgisayar Bilimleri Mühendisliği Bölümü
Çalıştığı Kurumlar : 2000 - 2001 Coca-Cola
2000 - 2002 ExecuTrain Corporation
2002 - 2004 SPD İletişim Sistemleri
2004- Devam ediyor Gantek Teknoloji

