YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

LINUX İŞLETİM SİSTEMİNDE INTERNET PROTOKOLÜ İÇİN BİR BANT GENİŞLİĞİ YÖNETİM SİSTEMİ TASARLANMASI VE GERÇEKLENMESİ

Bilgisayar Müh. Ufuk YÜZEREROĞLU

F.B.E Bilgisayar Mühendisliği Anabilim Dalında Hazırlanan

YÜKSEK LİSANS TEZİ

Tez Danışmanı : Yrd. Doç Dr. A. Gökhan YAVUZ

Prof. M. Yahya Karsken

Prof. Dr. Ergun Adalı

İSTANBUL, 2002

İÇİNDEKİLER

KISALTMA LİSTESİ

DSCP	Differentiated Services Code Point
IP	Internet Protocol
Kb	Kilobyte
LKM	Loadable Kernel Module
Mb	Megabyte
QoS	Quality of Service
RSVP	Resource Reservation Protocol
ToS	Type of Service
VoIP	Voice over IP

ŞEKİL LİSTESİ

ÇİZELGE LİSTESİ

ÖNSÖZ

Tüm hayatım boyunca yanımda ve her konuda destek olan aileme, yüksek lisans eğitimi ve tez geliştirme süresince yardımlarını, bilgisini ve zamanını esirgemeyen danışman hocam Yrd. Doç. Dr. A. Gökhan Yavuz'a ve bölüm başkanımız Prof. M. Yahya Karslıgil'e teşekkürü borç bilirim.

Ufuk Yüzereroğlu, 2002

ÖZET

Günümüzde, yazılım ve donanım seviyesinde bütün yönlendiriciler 'en iyiyi sağlama' mantığı ile çalışmaktadırlar. Bu mantıkta yapılan işlem, hiçbir parametre veya değer göz önüne alınmadan, gelen paketlerin, en kısa sürede ve en az işlem gerektirecek şekilde yönlendirilmesidir.

Farklı karmaşıklıkta ve bant genişliği kullanımı birbirinden farklı olan uygulamaların sayısı arttıkça, öncelik tanımını sağlayan, bant genişliği kullanımı kuralları koyan ve standartlar tanımlayan bir mekanizma ihtiyacı doğmuştur. Bugüne kadar, Differentiated Services veya Resource Reservation Protocol gibi birbirinden farklı birçok metod geliştirilmiş ve kullanılıyor olmasına rağmen, halen servis kalitesinin sağlanmasına yönelik bir standart tanımlanmamıştır. Bu alanda çalışan yazılım ve donanım üreticileri de tanımlı yöntemler ile kendi yöntemlerini birleştirerek yeni mekanizmalar geliştirmektedirler.

Servis kalitesinin en yaygın kullanım alanı belirli uygulamalara, bağlantılara veya hostlara, istenilen bant genişliğinin veya bağlantı oranının adanmasıdır. Mevcut Linux kerneli servis kalitesinin sağlanması amacıyla Differentiated Services yöntemini desteklemektedir. B unun yanında, bant genişliğinin ayrılmasını sağlayan 'Traffic Control' adı verilen bir de uygulama geliştirilmiştir. Differentiated Services yavaş yavaş bir standart haline gelmesine rağmen, kaynaktan hedefe gitmekte olan paketin geçtiği her yönlendiricide, paket içinde yer alan bu bilginin değerlendirilip anlam kazanması gerekmektedir. Bu işlem de her ağ yöneticisinin gerçekleştirmesine olanak olmayan bir işlemdir.

Önerilen sistem, Linux yönlendiricisine gelen paketlerin sınıflandırılması ve bu paketlere önceden tanımlanan kurallar baz alınarak servis verilmesi üzerine kuruludur. Sistem, araç sürücüleri veya bazı dosya sistemleri gibi Linux kernelinde modül olarak çalışmaktadır. Bu özellik sayesinde sistem kullanıcı işlemlerinden daha yüksek bir öncelikte çalışmakta ve paketlerin gelmesinden haberdar edilmek zorunda kalmamaktadır. Modülün kullanımındaki en önemli özellik modülün konfigürasyonudur. Servis kalitesinin sağlanması işlemi, Differentiated Services gibi her yönlendiricide değerlendirilmeyecek, yerel ağın çıkışında şekillendirme işlemi yapılmaktadır. Böylece paket üzerinde bilgilerin tekrar tekrar değerlendirilmesi işlemi de gerekmemektedir.

Anahtar Kelimeler: Linux, Servis Kalitesi, Bant genişliği Yönetimi

ABSTRACT

Today most of the hardware and software routers work with a 'best effort' fashion. This method does not involve any parameters and just tries to forward the network packets to the next router with the least overhead and in the shortest time – that is why it is called 'best effort'.

As the applications with different complexities and different needs for bandwidth usage arose, there occured a need for a mechanism that defines priorities, bandwidth usage policies and standards for the implementation. Although several methods have been defined and used so far including Differentiated Service (DiffServ) or Resource Reservaton Protocol (RSVP), there is still no defined standard method for achievement of QoS (Quality of Service) and vendors in the field all build up their own mechanisms with regard to the method used and mentioned above.

The basic and most widely used method of QoS is assigning a dedicated bandwidth or bandwidth usage ratio to specific applications or specific connections from/to predefined host(s). Linux kernel currently supports Differentiated Services as a system service and a software package called 'Traffic Control' has been built for dedicating bandwidth to specified user/host/port information. Even DiffServ is becoming a standard, each router on the way from the destination to the host has to recognize the bits in the IP header for treating the packet according to the Differentiated Services definitions. This is what each administrator does not have the chance to configure.

The proposed system is based on the basic classification of incoming packets to the Linux router box and shaping them according to the predefined policies - such as giving 25% to a www packet from host 192.68.0.1 to network 127.0.0.0 with a netmask of 255.255.255.255 and with priority of 1 - by the system administrator. The system works as a LKM (Loadable Kernel Module) just like any device driver or some file systems in the Linux kernel. This allows the module to use or overwrite standard kernel functions. Being a LKM, the application has the higher priority above the applications running and need no further invocation when the packets arrive. The basic advantage is the usage of the module – that is just a configuration file is enough to achieve all the aimed bandwidth allocation. Because of working at the LAN edge and forwarding packets in the scheduling mechanism implemented, the module needs no further recognization of anything or approving the packet on the other routers on its way. The priorities defined gives the module the ability to treat the policies in a weighted Round Robin fashion and treat the packets in a round-robin/FIFO fashion.

Keywords: Linux, Quality of Service, Bandwidth Management

1. GİRİŞ

Günümüzde ağ kullanımı hayatın her yerinde kendini göstermeye başlamıştır. Basit bankacılık işlemlerinden, telsiz iletişim ile yapılan iletişime kadar ağ kullanımı yavaş yavaş etkin hale gelmektedir. Ağın kullanımının her geçen gün artması, mevcut veri hatları üzerinde aktarılan bilgi miktarını ve çeşitliliğini de beraberinde getirmiştir. Bir şirketin yerel ağında basit veri taşıma işlemleri gerçekleşirken, telekonferans veya veri hatları üzerinden ses bilgisi taşınması gibi işlemler de gerçekleşmektedir. Buna benzer olarak, birbirinden farklı veri tiplerinin aynı veri hatlarının kullanılarak iletilmesi, belirli bant genişliği veya diğer veri tipleri üzerinde iletim önceliği gibi farklı gereksinimlere ihtiyaç duyarlar. Bu tip birbirinden farklı özellikler gösteren ve farklı gereksinimlere ihtiyaç duyan uygulama ve veri tiplerinin artması, ortaya hem yönetilmesi zor, hem de daha yüksek maliyete sahip farklı ağların kullanımı ihtiyacını doğurmuştur.

Bu tip uygulamaların farklı ağlar üzerinde çalıştırılması yerine, tüm verinin tek bir hat üzerinde hareketi, verinin hem yönetimi hem de takibi açısından çok daha elverişlidir. Kullanılan tek ağ, hem kurum içindeki hızlı erişimi gerektiren kritik bilgiyi taşıyacak, hem de gecikmeye toleransı olan basit dosya transferleri gibi uygulamalara ait veriyi taşıyacaktır. Bu noktada ortaya çıkan sorun, az önce belirtildiği gibi, taşınan veri tiplerinin birbirinden farklı gereksinimlerinin karşılanması zorunluluğudur. Böylece her tip verinin, kendisine ait bir ağ üzerinde çalışıyormuş gibi davranması ve kendi gereksinimine göre mümkün olan en kısa süre veya yoldan hedefine varması sağlanabilir.

Tek bir kuruma ait yerel ağ içindeki farklı ihtiyaçlara sahip veri tiplerinin taşınması sorunu, çok daha büyük bir ağ olan Internet'teki veri transferi söz konusu olduğunda da benzer bir öneme sahiptir. Internet kullanımın artması ile karşılaşılan en önemli sorun, mevcut bant genişliğinin verimsiz kullanımı ve bunun sonucunda ortaya çıkan servis kalitesi problemidir. Herhangi bir tip veri, Internet üzerinde, kaynağından hedefine giderken birden fazla ara noktaya uğramaktadır. Verinin uğradığı her ara nokta kendi yerel ağına veya bağlı olduğu geniş alan ağına has bazı özelliklere sahip olduğundan, verinin ihtiyaç duyduğu parametreler, hem bilginin çıkış noktasında hem de ara noktalarda sağlanmak zorundadır.

Bu tez çalışmasında, yerel ağ için yukarıda belirtilen servis kalitesi sorunlarının giderilmesine ve belirli bir orana kadar bant genişliği kullanımının sınırlandırılmasının garanti edilmesine yönelik, farklı veri tiplerine ait bilgi akışlarının ve bağlantıların istenilen oranda sınırlandırılmasına öngören bir sistem tasarlanmış, tasarlanan sisteme ait uygulama

gerçekleştirilmiş ve sistemin test edilmesi ile elde edilen sonuçlar değerlendirilmiştir.

Tez konusu ve seçilen geliştirme platformunun daha iyi anlaşılabilmesi için, 2. ve 3. bölümlerde, servis kalitesi ve Linux işletim sistemi ile ilgili daha ayrıntılı bilgi verilmiştir.

2. SERVİS KALİTESİ

Servis kalitesi, yerel ve genel ağlar üzerinde, belirli kriterler veya kurallara göre, hata oranı, bant genişliği kullanımı ve iletim hızı oranlarının hesaplanabilmesi, iyileştirilebilmesi ve belirli bir orana kadar istenilen değerlere sahip olmasının garanti edilmesidir. Servis kalitesinin günümüzdeki genel kullanım amacı, belirli kurallar ve kriterlere göre tanımlı bilgi akışlarına ve bağlantılara, istenilen önceliğin veya bant genişliğinin verilmesidir. [2]

Servis kalitesi, yüksek bant genişliğinin verimli kullanımının sağlanması yanında, sınırlı bant genişliği olan ortamlarda, bağlantı tiplerinin kullanımının sınırlandırılması için de kullanılabilir. Bu işlem küçük ve ağ kullanımı sınırlı olan işletmelerde bant genişliğinin gereksiz kullanımını engelleme amaçlıdır.

Servis kalitesinin sağlanması, ses ve görüntü uygulamaları gibi yüksek bant genişliği ihtiyacı olan uygulamalar için daha önemlidir. VoIP (Voice Over IP) gibi, veri hatları üzerinden ses ve görüntü gibi, aktarılması zor olan ve zaman alan bilgilerin taşınmasına yönelik uygulamaların sayısı ve kullanımı arttıkça, servis kalitesi kavramı daha çok önem kazanmıştır.

Ağ bağlantıları üzerindeki servis kalitesini iyileştirmek amacı ile farklı teknikler geliştirilmiştir. Bu tekniklerden bir kısmı her yönlendiricide (router) desteklenmek zorunda iken, bir kısmı sadece kullanılan yerel ağın ağ geçidinde (gateway) çalışarak yerel bir servis kalitesi desteği sağlamaktadır.

2.1 Servis Kalitesine Duyulan İhtiyaç

Servis kalitesi uygulamalarının temelde bant genişliği yönetimine yönelik olması, sınırsız bant genişliğine sahip olunması durumunda servis kalitesi kavramının gereksiz olacağı düşüncesini doğurmuştur. Servis kalitesinin bant genişliği yönetiminden daha önemli olan görevi, farklı veri tiplerine sınıflandırılma sonucu farklı öncelik derecelerinde servis verilmesidir.

Servis kalitesinin veri tiplerine öncelik verme özelliği sayesinde, hangi bant genişliğine sahip olunursa olunsun, iletim önceliği bulunan verilerin diğerlerinden önce iletimine olanak sağlanabilir. Böylece servis kalitesi tanımında yer verilen iletim hızı, hata oranı gibi değerlerin belirli bir sınır içinde olması garanti edilebilir.

Servis kalitesi uygulamalarına ihtiyaç duyulan bazı durumlar aşağıdaki gibi sıralanabilir. Bu durumlar servis kalitesine duyulan ihtiyacın önemli sebepleridir. Her kurum yerel veya geniş

alan ağ kullanımında kendi servis kalitesi gereksinimlerini tanımlayabilir. [2]

- Aşırı kullanım: Çok fazla kullanıcının, aynı anda, aynı kaynağa erişim ihtiyacı, ağ tasarımının getirdiği sınırlamalar yüzünden, ağ üzerinde yığılmaya yol açabilir.
- Yetersiz bağlantı kapasitesi: Her ağ yapısı belirli bir iletim kapasitesine sahiptir ve ağ üzerinde aşırı yüklenmenin olmayacağı bir durumun öngörülmesi imkansızdır.
- Yetersiz anahtar (switch)/yönlendirici kaynakları: Aşırı yük gelmesi durumunda, ağ üzerinde anahtarlama ve yönlendirme yapan cihazların kaynakları, yapılan işlemlerden dolayı yetersiz kalabilir.
- Trafik akışındaki yükselmeler: Ağ yapısı ne olursa olsun, ağda her zaman, sabah herkesin e-mailini kontrol etmesi gibi, bir 'yoğun saat' yaşanma olasılığı vardır.
- Trafik akışlarındaki karışıklık: Farklı veri tipleri, birbirinin akışını etkileyebilir. Önceliğe sahip bir video uygulamasının, terminal uygulamasını engellemesi buna bir örnek olarak gösterilebilir.
- Kaynakların verimsiz veya amaç dışı kullanımı: Kurum içindeki ağ kaynakları, işe yönelik amaçlar dışında kullanılabilir. Kaynakların belirli amaçlar dışında kullanılmasını önlemek bunu engelleyebilir.

2.2 Servis Kalitesi Tekniklerinin Sınıflandırılması

Servis kalitesini sağlamak için kullanılan teknikler, farklı kriterlere göre birden fazla şekilde sınıflandırılabilirler. [2]

2.2.1 Açık ve Kapalı Servis Kalitesi (Implicit / Explicit QoS)

Açık servis kalitesi tekniklerinde kullanıcıların servis kalitesi işlemlerinin nasıl ve ne şekilde gerçekleştiğinden haberi yoktur. Bu teknikte, her şeyden sorumlu olan ağ üzerindeki çalışan protokollerdir. Kapalı servis kalitesi tekniklerinde ise, ağın servis kalitesini ne şekilde sağlayacağı bir sistem veya yönetici tarafından belirlenir.

2.2.2 İç ve Dış Servis Kalitesi (Internal / External QoS)

Bu sınıflandırma, servis kalitesi kontrolünün dışarıdan verilmesi veya sistemin kendi kontrollerini yapması üzerine kuruludur. Servis kalitesinden kullanıcının sorumlu olması durumunda, ağ üzerinde çalışan protokoller, sisteme sağlanan verilerin geçerliliğini kontrol

etmeden verilen parametreleri uygular. Diğer durumda ise ağ protokolleri, kullanıcı parametreleri girilmiş olsa bile bunları dikkate almadan işlemleri ve kontrolleri kendileri gerçekleştirir.

2.2.3 Toleranslı ve Toleranssız Servis Kalitesi (Soft / Hard QoS)

Servis kalitesi mekanizmaları devrede iken, belirli hedeflerin garanti edilememesi durumunda oluşan servis kalitesi mekanizması toleranslı kabul edilir. Bu durumda, hedeflenen değerlerin elde edilememesi olasılığı her zaman mevcuttur. Toleranssız servis kalitesi mekanizmalarında ise hedeflenen değerlere ulaşılması her zaman garanti edilir.

2.2.4 Fiziksel ve Mantıksal Tasarım

Servis kalitesi mekanizmaları tasarım olarak da gerçekleştirilebilir. Fiziksel tasarımda kullanıcıların kaynaklara olan erişimi sınırlandırılabilir veya kaynakların kullanımı paylaştırılabilir. Mantıksal tasarımda ise göz önünde bulundurulan parametreler, fiziksel cihazların veya hatların dışında olan bağlantılardır. Birden fazla sanal devre aynı fiziksel bağlantıyı kullanmasına rağmen, birbirinden farklı özellikte ve ihtiyaçta veri tipleri taşıyabilirler.

2.3 Farklı Servis Kalitesi Teknikleri

Servis kalitesini sağlayacak olan mekanizmalar, ağ ucunda ya da ağın içerisinde merkezi bir noktada çalışırılar. Her servis kalitesi mekanizması kendi başına çalışabileceği gibi, gereken durumlarda diğer mekanizmalar ile işbirliği içinde de çalışabilir.

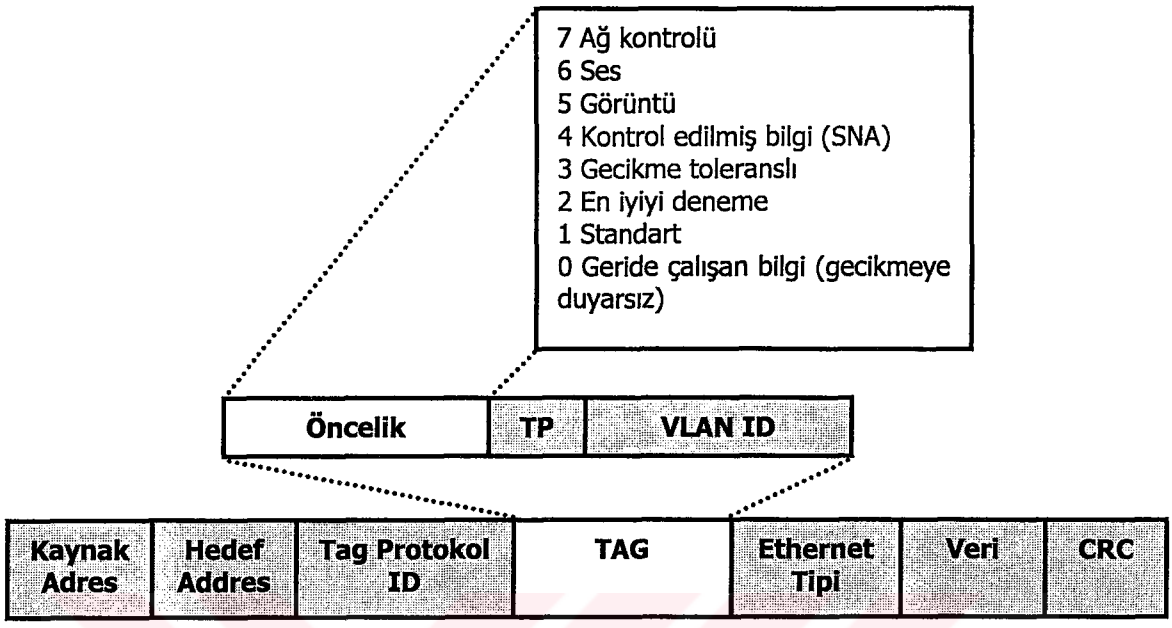
Tanımlanan ve kullanımda olan tekniklerden bir kısmı, mevcut IP bilgisinin içinde rezerve edilmiş bit alanlarını, bazıları ise tamamen kendi yaptıkları tanımları kullanmaktadır. Mevcut tekniklerden bazıları ve çalışma mantıkları aşağıda anlatılmıştır.

2.3.1 Yerel Ağ Tabanlı Öncelik Tanımlama – IEEE 802.1p/Q

Günümüzde yerel ağlarda ulaşılan yüksek hıza rağmen, istenilen servis kalitesinin sağlanması bir problem olmaya devam etmektedir. Paylaşıma açık kaynaklara erişim, bu tip bir problemin ortaya çıkmasındaki başlıca sebep olabilir.

IEEE, 1999'da yerel ağlarda öncelik tanımlamaya yönelik bir standart tanımlaması yapmıştır. IEEE802.1p spesifikasyonunda, MAC başlığındaki 802.1Q alanında 3 bit, öncelik tanımı

yapmak için ayrılmıştır. Bu 3 bitin kullanımı ile 8 ayrı öncelik tanımı yapılabilmektedir. Bu alan ve alabileceği öncelik değerleri Şekil 2.1'de gösterilmiştir.

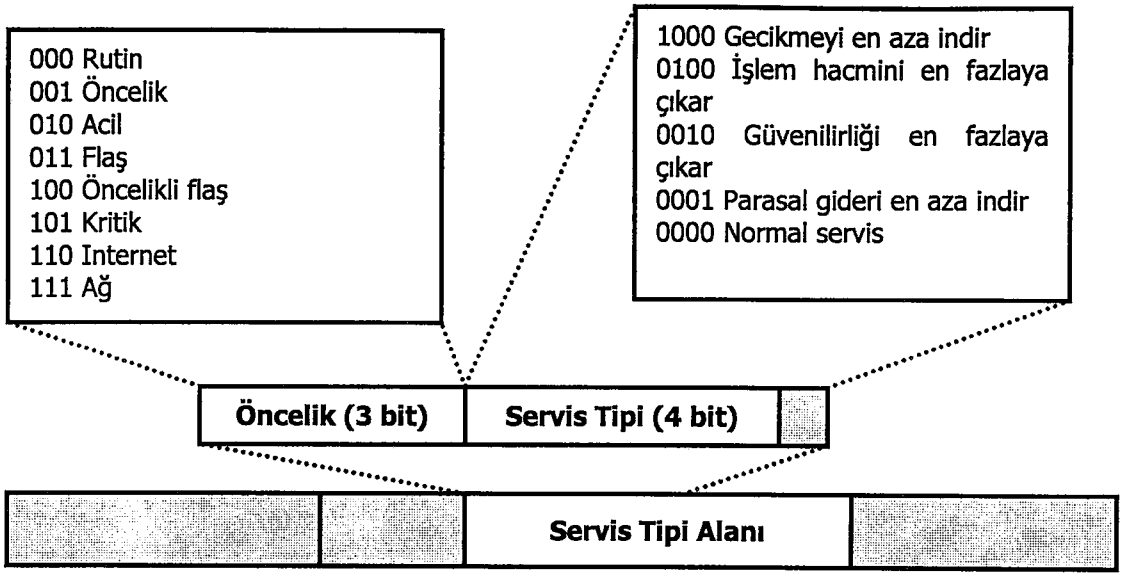


Şekil 2.1 IEEE802.1p/Q Öncelik biti tanımları

Bu yöntem uçtan uca bağlantılarda kullanılamasa bile, yerel ağdaki bir uç sistemin gereksinimlerini tanımlamak için basit ve kullanışlı bir mekanizmadır.

2.3.2 IP Önceliği

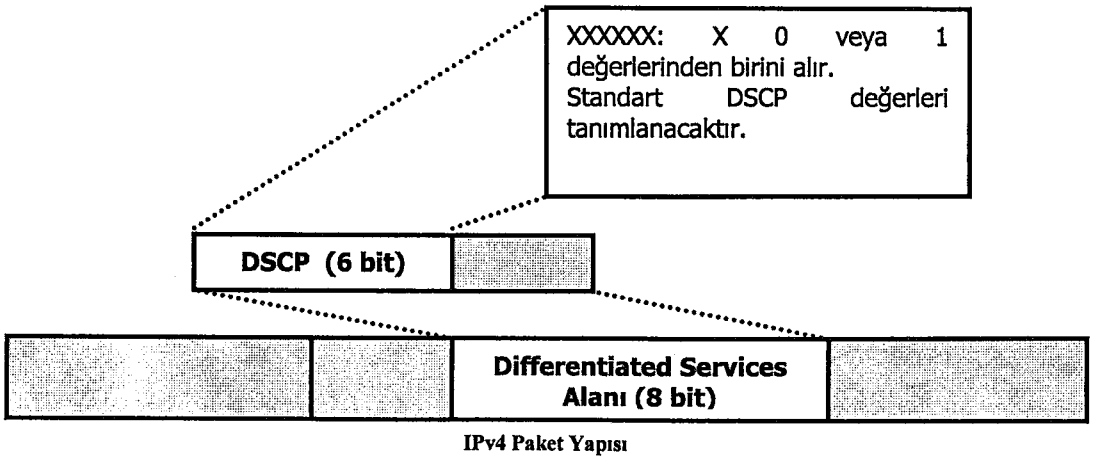
Üçüncü katmanda çalışan servis kalitesi mekanizmaları, uçtan uca servis kalitesi tanımları sağlarlar. IPv4 başlığında, paketler üzerinde öncelik tanımı yapabilmek için 8 bitlik Servis Tipi (Type of Service) alanı tanımlanmıştır. Bu alanın ilk kullanımı, RFC1349 dokümanında yer almıştır. Bu alanın kullanımına ait son bilgi RFC1812'de tanımlanmıştır. 3 bitlik servis tipi alanının yanı sıra, 4 bit ile gecikmeye olan duyarlılık, üretilen iş ve paket kaybı ile ilgili hedeflenen değerler de tanımlanabilmektedir. Bu alanlar ve alabileceği değerler Şekil 2.2' de gösterilmiştir.



Şekil 2.2 IP Öncelik bitleri

2.3.4 Ayrıcalıklı Servisler (Differentiated Services)

IP Önceliği mekanizmasında, paketlerin önceliklerini tanımlamak için kullanılan servis tipi alanının IETF tarafından yeniden tanımlaması sonucunda, bu alan DSCP (Differentiated Services Control Point) adını almıştır. Toleranslı bir servis kalitesi mekanizması olan Differentiated Services'in amacı, tüm trafiği, uçtan uca tanımlı olacak şekilde, sınırlı sayıda sınıf içinde birleştirebilmektir. Differentiated Services desteği şu anda anahtarlama ve yönlendirme birimlerinde desteklenmektedir ve servis kalitesi desteğine tam anlamıyla ihtiyaç duyulduğunda kullanılmaya başlanacaktır.



Şekil 2.3 Ayrıcalıklı Servisler bitleri

2.3.5 Servis Kalitesi Rezervasyonları (RSVP)

Rezervasyon mekanizmasında bilgiyi gönderecek olan taraf, göndereceği uca kadar olan tüm ara noktalardan belirli bir iletim hızını, bant genişliğini veya bunlara ait sınırların rezerve edilmesini ister. Eğer tüm noktalar RSVP protokolü desteğine sahip ise işlem gerçekleştirilir ve gönderen taraf, istenilen kriterlerin sağlandığından emin olarak bağlantıyı gerçekleştirir. RSVP (Resource ReserVation Protocol), sadece isteği yapmakta ve gönderen ucun istediği kriterlerin sağlanmasını garanti etmemektedir.

2.3.6 Servis Kalitesi Tabanlı Paket Yönlendirme

Yukarıda anlatılan servis kalitesi mekanizmalarında, ağ cihazlarının belirli servis kalitesi kriterlerinin yerine getirilmesi gerektiğini bilmeleri gerekmektedir. Bu yüzden her cihazda kuralların yerine getirilmesini ve sınırların belirlenmesini sağlamak amacıyla belirli fonksiyonlar gerçekleştirilmelidir. Bu fonksiyonları yerine getirirken en önemli iki nokta kullanılan sıralama algoritması ve işlemlerin tel hızında (wire speed) gerçekleşmesidir.

İşlemlerin tel hızında gerçekleşmesi, mümkün olan en hızlı şekilde gerçekleşmesi anlamındadır. İşlemler gelen trafiğin birikmesine izin vermeyecek hızda yerine getirilmelidir. Çözüm olarak sunulan servis kalitesi mekanizması, beraberinde yeni sorunlar getirmemelidir.

Sıralama algoritması gelen ve değerlendirilen paketlerin nasıl saklanacağını ve saklanan paketlere hangi sırayla servis verilmesi gerektiğine karar verir. Kullanılan en basit sıralama algoritması FIFO (First In First Out) algoritmasıdır. Bu algoritma servis kalitesi için yetersiz kaldığından, servis kalitesi mekanizmalarında farklı algoritmalar kullanılmaktadır. Bu algoritmalardan birkaçına aşağıda değinilmiştir.

- Öncelik sıralaması: Bu sıralama tipinde, yüksek önceliğe sahip akış ve paket tipleri mevcut bant genişliğini kullanımda, diğer akış ve paket tiplerine göre önceliklidir. Öncelik tanımının yapılabilmesi için IP paketindeki DSCP veya Type of Service (ToS) alanının kontrolü gibi işlemler gerçekleştirilir.

- Ağırlıklı adil sıralama: Etkileşimli uygulamalara ait paketlere ve akışlara öncelik tanıyacak biçimde ve uygulamanın ihtiyacına göre bant genişliğini ayarlar. Bir sınıfa ait uygulamanın, diğer sınıflara ait bant genişliğini kullanmasına izin vermez.

- Sınıf tabanlı sıralama: Trafiğin, servis kalitesi özellikleri ile birbirinden ayrılan hiyerarşik sınıflara ayrılmasını sağlar. Burada sınıflar, diğer sınıflara ait bant

genişliğini kullanabilirler.

2.3.7 Ayrılmış Erişim Oranı (Comitted Access Rate)

Ayrılmış erişim oranı mekanizmasında, ağ kaynaklarına ve hedeflere erişim kurallar tanımlanarak sınırlandırılır. Bu kurallar erişim port bilgisi veya hedef adres gibi bilgileri içerebilir. Tanımlanan kurallara uyan veri akışının istenilen erişim oranını geçmesi engellenir.

2.3.8 Multi-Protokol Etikete Bağlı Anahtarlama

IETF tarafından tanımlanan MPLS mekanizmasında, paketler ağa girdikleri anda kısa ve sabit uzunluğa sahip etiketler ile işaretlenirler. Yönlendirici paketin içindeki bilgiyi kontrol etmeden, sadece etiket bilgisinin kullanımı ile paketin hangi hosta yönlendirileceğine karar verir.

3. LINUX İŞLETİM SİSTEMİ

Linux, Finlandiya, Helsinki Üniversitesi'nde geliştirilen, UNIX işletim sistemine benzeyen fakat UNIX ile birebir aynı olmayan bir işletim sistemidir. Linux, Intel tabanlı platformlardan Digital Alpha, PowerPC veya IBM'in mainframe bilgisayarlarına kadar birbirinden farklı platfomlarda çalışabilmektedir. Linux'un diğer işletim sistemlerinden en önemli farkı serbest dağıtım özelliğidir. Linux'un kaynak kodları ve Linux dağıtımları ücretsiz olarak temin edilebilmektedir.

International Data Corporation'ın (IDC) yaptığı araştırmaya göre, Linux işletim sistemi 2003 yılı içerisinde diğer istemci ve sunucu işletim sistemlerine oranla daha hızlı bir büyüme içerisine girecektir. IDC'nin yaptığı tahminlere göre Linux işletim sisteminin ticari versiyonları, 1999 yılı ile 2003 yılı arasında yıllık ortalama %25 oranında bir büyüme oranına sahip olacaktır. Buna karşın diğer istemci işletim sistemlerinde ortalama yıllık büyüme oranı %10, sunucu işletim sistemlerindeki yıllık büyüme oranı ile %12 olarak tahmin edilmektedir. IDC'nin tahminlerini dayandırdığı noktalar, Linux işletim sisteminin yüksek performansı ve düşük gideridir. [Brody, 1999]

Linux temelde Intel tabanlı bilgisayarlarda çalışmak için tasarlanmış olsa da, modüler ve taşınabilir kodlama sayesinde, IBM'in Watchpad'i ve Samsung'un Yopy'si gibi Personal Data Assistant'lardan (PDA), Los Alamos National Labs.'da kullanılan ve dünyanın en hızlı 500 bilgisayarından biri olarak gösterilen Avalon'a kadar birbirinden farklı platformlarda çalışabilmektedir.

Linux'un dünya çapındaki kullanıcı sayısının gelişim Çizelge 3.1'de gösterilmektedir.

Çizelge 3.1 Linux'un dünya çapında kullanıcı sayısı [Vallopillil, 1998]

Yıl	Kullanıcı Sayısı
1991	1
1992	1000
1993	20,000
1994	100,000
1995	500,000
1996	1,500,000
1997	3,500,000
1998	7,500,000
1999	> 15,000,000

3.1 Linux İşletim Sisteminin Diğer İşletim Sistemleri ile Karşılaştırılması

Güvenilirlik, performans, ölçeklenebilirlik, güvenlik ve toplam sahip olma maliyeti açılarından, Linux ve diğer işletim sistemleri üzerinde yapılan araştırma ve karşılaştırmalar aşağıda belirtilmiştir.

3.1.1 Güvenilirlik Değerlendirmeleri

Wisconsin Üniversitesi'nde yapılan bir işletim sistemi testinde, farklı işletim sistemlerinde çalışan uygulamalara rasgele giriş yapılarak, sistemlerin devre dışı kalma oranları gözlemlenmiştir. Teste tabi tutulan 7 ticari işletim sisteminin ortalama hatalı çalışma olasılığı gözlemler sonucu %23 olarak hesaplanırken Linux'un hatalı çalışma veya devre dışı kalma olasılığı %9 olarak hesaplanmıştır. 1995'te yapılan araştırmanın sonuç kısmında şöyle denilmiştir:

> "...Linux'un güvenilirliği sürpriz şekilde oldukça iyi ve ticari üretilen yazılımların çoğundan açıkça daha iyi çıkmıştır..." [Miller, 2000]

3.1.2 Performans

Linux'un da dahil edildiği performans testlerinden biri SysAdmin dergisi tarafından yapılmıştır. Bu testte Intel tabanlı RedHat Linux 7.0 (Kernel 2.2.16-22), Solaris 2.8, Windows 2000 ve FreeBSD 4.2 UNIX işletim sistemlerinden daha başarılı çıkmıştır. Tüm işletim sistemleri aynı platformda (4-GB IBM model DCAS-34330 SCSI-3 disk birimi, ASUS P3B ana kart, Intel Pentium III 550-MHz işlemci, 384-MB SDRAM hafıza birimi, Adaptec 2940UW SCSI kontrol birimi, ATI Rage Pro 3D ekran kartı, Intel EtherExpress Pro 10/100 Ethernet kartı) çalıştırılmış ve bu test sonucunda en iyi performans gösteren işletim sistemi olan Linux, kendisinden sonra gelen Solaris'e oranla %35 daha yüksek bir performans göstermiştir. İşletim sistemleri 2 ayrı performans testine tabi tutulmuştur:

İlk test, işletim sistemlerinin aynı anda, sorunsuz şekilde kaç adet e-mail gönderebileceğini ölçmek üzerinedir. Aynı anda gönderilen 100 e-maile kadar, işletim sistemleri arasında performans farkı ortaya çıkmaz iken, bu sayı arttıkça ciddi farklar gözlenmiştir. 500 e-mail gönderiminde Linux'un diğer tüm işletim sistemlerinden daha hızlı olduğu ortaya çıkmıştır. Linux haricindeki işletim sistemleri yaklaşık 1,000-1,500 e-mail gönderiminde doyum noktalarına ulaşmışlardır. Aynı anda 1500 e-mail gönderimi ile, Linux saatte yaklaşık 1,3 milyon e-mail gönderebilirken, Solaris'in yaklaşık 1 milyon, Windows 2000 ve FreeBSD ise 0,9 milyon e-mail gönderebildiği tespit edilmiştir.

Disk erişim testinde eşit boyuta sahip 10,000 dosya yaratılıp, diske yazılıp geri okunmuştur. 4 Kb (Kilobyte) ve 64 Kb arasındaki küçük dosya boyutlarında Linux ve Windows 2000, yaklaşık 200 saniyede işini gerçekleştirmekte iken, dosya boyutu 128 Kb'a çıkarıldığında Linux yaklaşık 300 saniyede, Windows 2000 ise yaklaşık 400 saniyede işini gerçekleştirmekte olduğu gözlemlenmiştir. Bu değerler karşısında FreeBSD'nin verdiği değerler yaklaşık 900 saniye, Solaris'in verdiği değerler ise 1,500 saniye olarak hesaplanmıştır. [Rothman, Buckman, 2001]

Yine Linux'un performansının ölçüldüğü ve IBM tarafından yapılan bir karşılaştırmada, RedHat Linux 7.1 (Kernel 2.4.2) işletim sisteminin, pipe kullanımı ve process ve thread yaratılmasında Windows 2000 ve Windows XP'den daha iyi performans gösterdiği açıklanmıştır. Pipe kullanımında Linux'un G/Ç oranı 700MB/saniye olarak belirlenirken, bu değer büyük blok boyutlarında yaklaşık 100 MB/saniyede sabitlenmektedir. Bu değerler Windows 2000'de en çok 500MB/saniye ve ortalama 80 MB/saniye, Windows XP'de ise 120MB/saniye ve 80 MB/saniyedir. Şubat 2002'den yayınlanan diğer bir yazıya göre RedHat Linux 7.2 saniyede 10,000 process yaratabilirken Windows 2000, 5,000 değerine yaklaşmış, Windows XP ise 6,000 process yaratabilmişir. Thread yaratma karşılaştırmasında ise RedHat Linux saniyede 330 thread yaratmış, buna karşın Windows 2000 200, Windows XP ise 160'ın altında thread yaratabilmiştir. [Bradford, 2001]

3.1.3 Ölçeklenebilirlik

Linux'un ölçeklenebilirliği ile ilgili en önemli kanıt desteklediği platformların sayısıdır. Linux, bir PDA'de çalışabileceği gibi, Intel tabanlı bir kişisel bilgisayarda ve bir mainframe'de de çalışabilir. Ayrıca Linux, üniversite ve araştırma merkezlerinde süper bilgisayarlar kurmak ve clustering yapmak için de kullanılmaktadır.

3.1.4 Güvenlik

2002 yılında 400 Linux geliştiricisinin katıldığı bir ankette, Linux işletim sisteminin, dışarıdan gelecek saldırılara diğer işletim sistemlerine göre daha güvenli olduğunun düşünüldüğü ortaya çıkmıştır. Bu geliştiricilerden %78'i sistemlerine istenmeyen bir girişle karşılaşmadığını ve %94'ü de bir virüse maruz kalmadıklarını belirtmişlerdir. Katılımcıların %84'ü de Linux'un AIX ve Solaris gibi yüksek güvenliğe sahip işletim sistemlerine yakın olduğunu ve tüm Windows ailesi işletim sistemlerinden daha güvenli olduğunu düşündüklerini belirtmişlerdir. [EDC, 2002]

Linux'un en büyük rakibi olarak gösterilen Windows işletim sisteminden daha güvenli olduğu ise en az bir sigorta şirketi tarafından kabul edilmiştir. Şirket, müşterilerini sigortalarken, UNIX veya Linux tabanlı işletim sistemleri yerine Windows NT'yi tercih eden müşterilerinden %5-%15 arasında daha fazla ücret almaktadır. [Luening, 2001]

3.1.5 Toplam Sahip Olma Maliyeti

Linux işletim sisteminin yüksek kabul görmesinin en önemli nedenlerinden biri de hem sunucu hem de istemci kurulumu maliyetinin diğer işletim sistemleri ile karşılaştırıldığında oldukça düşük olmasıdır. Linux işletim sistemi Internet'ten elde edilip kullanılabileceği gibi, satın alma durumunda da oldukça düşük bir fiyata mal olmaktadır.

Örneğin, e-mail ve HTTP sunucusu, bir ilişkisel veritabanı sunucusu ve program geliştirme platformu olarak kullanılacak bir sunucu kurulumunda Windows 2000 ile RedHat Linux'un satın alma maliyetleri Çizelge 3.2'de verilmiştir.

Çizelge 3.2 Linux ve Windows toplam sahip olma maliyetleri karşılaştırması

	Microsoft Windows 2000	**RedHat Linux**
İşletim Sistemi	1510 $ – 25 istemci	29 $ Standart, 76 $ Deluxe, 156 $ Professional – Tümü sınırsız istemci
E-mail sunucusu	1300 $ – 10 istemci	Fiyata dahil – Sınırsız istemci
İlişkisel veri tabanı Sunucusu	2100 $ - 10 adet istemci erişim lisansı	Fiyata dahil – Sınırsız istemci
C++ Geliştirme Paketi	500 $	Fiyata Dahil

Tüm bileşenleri içeren bir sunucunun Windows 2000 ile kurulumu 5,410 dolara mal olurken RedHat Linux ile kurulumu Professional edition ile 156 dolara mal olmaktadır.

Consulting Times tarafından yapılan bir araştırmaya göre, 50,000 e-mail kutusu içeren Microsoft Exchange/Intel çözümü 5,4 milyon dolara mal olurken aynı sayıdaki e-mail kutusunun Linux/IBM çözümüyle 3,3 milyon dolara mal olmaktadır. Bu iki çözümde 25,000 kullanıcıya kadar Exchange/Intel çözümü daha makulken 25,000 kullanıcıdan sonra Linux/IBM çözümü çok daha makul bir hal almaktadır. [Harris, 2001]

İşletim sisteminin versiyonun yükseltilmesi işleminde de Linux işletim sistemi diğer işletim

sistemlerine göre daha ucuzdur. Örneğin Windows işletim sisteminin yeni sürümünü almak için, her lisans, işletim sisteminin orijinalinin fiyatının yaklaşık yarısına gelirken, Linux'un yükseltilmesi işlemi ise sınırsız lisans için sadece işletim sistemini tekrar elde etmekle veya tekrar satın almakla çözümlenmektedir.

Linux işletim sistemi oldukça düşük konfigürasyonlar üzerinde çalışabilir. Windows 2000 kurulum için, Pentium-uyumlu en az 133 MHz bir işlemci, tercihen 256 MB olmakla beraber minimum 128 MB hafıza birimi ve en az 1 GB'ı boş olan 2 GB'lık disk birimine ihtiyaç duymaktadır. Buna karşın RedHat Linux 7.1 işletim sistemi kurulum için ihtiyacı en az tercihen Pentium tabanlı olmakla birlikte minimum i486 işlemci, önerileni 64 MB olan 32 MB hafıza birimi ve 650 MB disk birimine ihtiyaç duymaktadır.

Birçok kurum Linux işletim sistemi'ni tercih ederek harcamalarında büyük miktarda kısıtlamalara gitmiştir. Bunlardan Amerika, Florida'da Largo şehrinde Linux ve ince istemci kullanımına geçilmesi ile şehre yıllık yaklaşık 1 milyon dolarlık bir kar getirecek bir yatırım yapılmıştır. [Haber, 2002]

Amazon.com alışveriş sitesi 2001 yılının ilk yarısında Linux işletim sistemine geçerek teknoloji giderlerinde 17 milyon dolarlık bir kazanç elde etmiştir. [Shaklane, kane, lemos, 2001]

3.1.6 Bilişim Dünyasında Yaygınlık

Eylül 2001'de NetCraft tarafından yapılan bir araştırmaya göre Internet üzerindeki sunucular üzerinde kullanılan işletim sistemleri arasında Linux %29,6 oran ile Microsoft Windows ailesinin %49,6 oranından sonra ikinci sırada gelmektedir. Aynı araştırmanın Mart 2001 ayındaki sonuçlarına göre Linux %28,5 kullanım oranına sahiptir. [1]

Idaya şirketinin 1000 Internet Servis Sağlayıcı şirket ile yaptığı bir ankette, Linux'un pazar payının gelişmesi üzerinde yaptığı bir çalışmada Linux'un 2001 yılı içindeki Pazar payı büyüme oranının %154 olarak beklendiği açıklanmıştır. Aynı araştırmada Linux'un 2002 yılı ortalarında ağ sunucusu ortamlarında en çok kullanılan işletim sistemi olacağı öngörülmektedir. [Knipe, 2001]

3.2 Linux İşletim Sisteminde IPv4 Desteği

Linux kerneli, IPv4 protokolünü, birbirine bağlı fonksiyonlar kullanarak gerçekler. IP katmanı, IP işlevlerini yerine getiren koddan oluşur. Bu kod, gelen bilgiye IP bilgilerini ekler

ve gelen paketin TCP veya UDP protokollerinden hangisine gönderileceğine karar verir. IP işlevlerinin gerçeklenmesi, alt katmanda kullanılan ağ cihazları sayesinde olur. Ağ cihazları her zaman fiziksel olmayabilir. Linux'ta kullanılan ve 'loopback' cihazı tamamen yazılım yardımıyla gerçeklenmektedir.

3.2.1 Linux'ta Kullanılan sk_buff Yapısı

Linux çekirdeği, katmanlar arasında bilgi taşınmasını kolaylaştırmak için soket bilgisini tek bir veri yapısında tutmaktadır. Tüm protokollere ait bilgiyi içeren bu yapı *sk_buff* veri yapısıdır. sk_buff, protokollerin kullanımı için 4 ayrı işaretçi tutmaktadır. [Gopinath, 1999]

- head: *sk_buff*'ın tuttuğu verinin başladığı adrestir.
- data: Protokol bilgisinin başladığı adrestir. *sk_buff*'ı kullanan protokole göre değişir.
- tail: Protokol bilgisinin bittiği adrestir. *sk_buff*'ı kullanan protokole göre değişir.
- end: *sk_buff*'ın tuttuğu verinin bitiş adrestir.

Bu işaretçilerle birlikte, *sk_buff* yapısında, transport, network ve data link katmanlarına ait bilgiler de tutulmaktadır.

3.2.2 Linux Kernelinde Ağ Paketinin İzlediği Yol

Bir ağ paketinin, Linux 2.4 kernelinde izlediği yol aşağıda anlatılmıştır. [3]

3.2.2.1 receive Kesmesi

Sistemdeki ağ kartı, sistemin MAC (Medium Access Control) adresine gelen veya broadcast edilen bir paket aldığında bir kesme çağırır. Ağ sürücüsü paketi hafızasına alır. Daha sonra *sk_buff* tipinde bir yapı oluşturulur ve protokolden bağımsız olan cihaz destek rutinleri çağrılır. *sk_buff* yapısına zaman bilgisi konulur ve *sk_buff*, paket tipini işleyecek olan işlemci için sıraya konulur. *sk_buff* sıraya yerleştirildikten, sonra *__cpu_raise_softirq* kesmesi çalıştırılmak üzere, işaretlenir Bundan sonra *receive* kesmesi işini bitirir.

3.2.2.2 Ağ Paket Alma Softirq'su

Ağ paket alma kesmesi, *net_init*, net/core/dev.c dosyasında tanımlanmıştır. Paket üzerindeki diğer işlemler, *do_softirq* tarafından çağrılan ağ paket alma kesmesinde gerçekleşmektedir. *do_softirq*, kernel içinde 3 yerde çağrılmaktadır.

- arch/i386/kernel/irq.c dosyasında tanımlanan *do_IRQ* fonksiyonunda,
- arch/i386/kernel/entry.S dosyasından,
- kernel/sched.c dosyasındaki schedule fonksiyonundan.

Eğer program bu üç noktadan birinden geçerse, *do_softirq* çağrılır ve ağ paket alma kesmesi olan *net_rx_action* fonksiyonunu çağırır. Burada *sk_buff*, işlemcinin bekleme kuyruğundan alınır ve uygun paket işleyici fonksiyonuna gönderilir. Uygun paket işleyici fonksiyonu, kernelde yer alan *ptype_base* hash tablosu kullanılarak belirlenir. Bu tablo, paket tipleri baz alınarak oluşturulmuştur. Gelen paketin tipi bu tablodaki değerlerle eşlenir ve uyan paket tipine ait işleyici fonksiyon çağrılır. Paket birden fazla paket tipine uyuyorsa pakete ait *sk_buff* yapısının kopyası çıkarılır ve her işleyici fonksiyona ayrı ayrı gönderilir.

3.2.2.3 IP Paket İşleyicisi

IP paket işleyici fonksiyonu net/ipv4/ip_input.c dosyasında tanımlanan *ip_rcv* fonksiyonudur. Temel kontroller yapılır ve pakete ait checksum hesaplanır. Kontroller sonucu hatalı olduğuna karar verilen paket bu fonksiyonda düşürülür. Paket kontrolleri geçtiği takdirde, IP paketinin boyutu hesaplanır ve donanımın eklemiş olabileceği bilgiler skb'den çıkarılır.

Tüm işlemlerden sonra, *ip_rcv_finish* fonksiyonuna ait işaretçi parametre olarak verilerek NetFilter hook'u çağrılır. Bu hook *ip_rcv_finish* fonksiyonunu çağırır. Bu fonksiyon içinde, *ip_route_input* çağrılarak paketin hedefi belirlenir. Sonuca göre paket aşağıdaki 4 fonksiyondan birine gönderilir.

- *ip_local_deliver*: Paket sistemin kendisine gelmiştir.
- *ip_forward*: Paket yönlendirilecektir.
- *ip_error*: Paketin yönlendirileceği bir adres bulunamamış ve bir hata oluşmuştur.
- *ip_mr_input*: Paket bir multicast paketidir ve multicast yönlendirme yapılır.

3.2.2.4 Paketin Başka Bir Cihaza Yönlendirilmesi

Yönlendirme fonksiyonu paketin başka bir cihaza yönlendirilmesine karar verdiyse, net/ipv4/core/ip_forward.c dosyasındaki *ip_forward* fonksiyonu çağrılır. Bu fonksiyon ilk olarak paketin TTL (Time To Live) değerini kontrol eder. TTL değeri 1'den küçük veya 1'e eşitse paket düşürülür ve gönderene ICMP kullanılarak zaman aşımı mesajı gönderilir.

Paket düşürülmediyse, *sk_buff* 'da hedef cihazın data link layer katmanı bilgisi için yer olup olmadığı kontrol edilir. Eğer yeterince yer yoksa *sk_buff* yapısının boyutu artırılır ve TTL değeri 1 azaltılır. Yeni oluşan paketin boyutu hedef cihazın MTU (Maximum Transmission Unit) değerinden büyük ve IP bilgisindeki 'parçalara ayırma' biti 1 ise, paket düşürülür ve gönderene 'parçalara ayırma gereği' ICMP mesajı gönderilir.

Bundan sonra yeni NetFilter hook'u çağrılır. Bu hook, *ip_forward_finish* fonksiyonunu çağırır. *ip_forward_finish*, IP bilgisi içindeki IP opsiyonlarını atar ve *ip_send*'i çağırır. Paketin parçalara ayrılması gerekirse bu işlem gerçekleştirilir ve *ip_finish_output* çağrılır. Bu fonksiyon da *ip_finish_output2*'yi çağırır. *ip_finish_output2*, donanım bilgisini sk_buff'a ekler ve *ip_output*'u çağırarak paketin gönderilmesini sağlar.

3.3 Linux İşletim Sisteminde Servis Kalitesi Desteği

Linux kerneli versiyon 2.1.90'dan itibaren servis kalitesi desteği sunmaktadır. Bu kernel servis kalitesi için Differentiated Services desteği ve bu desteği kullanabilmek için de Traffic Control (tc) adı verilen arayüzü sunar. Bu desteğin kullanılabilmesi için kernel derlenmesinde servis kalitesi opsiyonu ve alt opsiyonlarının seçilmesi gerekmektedir. [Radhakrishnan, 1999]

Linux'ta servis kalitesi, diğer sistemlerden gelen ve yönlendirilecek paketler ile sistem tarafından üretilen ve gönderilecek olan paketlerin yönlendirilme aşamasında gerçekleştirilir. Linux'taki trafik kontrolü, paketlerin çıkış arayüzünde sıralanması esnasında birden fazla kuyruk kullanımı, sıralama ve servis verme algoritması sağlayarak servis kalitesini sağlar. Linux'taki servis kalitesi desteği, aşağıdaki 3 bloktan oluşur:

- Kuyruk disiplinleri
- Sınıflar
- Filtreler/Kurallar

3.3.1 Kuyruk Disiplinleri

Kernelde tanımlı olan her ağ cihazına ait bir kuyruk tanımlıdır. Kullanılabilen kuyruk disiplinleri aşağıda listelenmiştir.

- Class Based Queue
- Token Bucket Flow

- Clark-Shenker-Zang
- First In First Out
- Priority
- Traffic Equalizer
- Stochastic Fair Queueing
- Asynhcronous Transfer Mode
- Random Early Detection
- Generalized RED
- DiffServ Marker

Her kuyruk kendi içinde farklı bir disipline göre çalışabilir.

3.3.2 Sınıflar

Sınıflar, kuyruk disiplinleri ile beraber çalışırlar. Her sınıfın kendine ait bir kuyruğu vardır ve bu kuyrukların standart disiplini FIFO'dur. Aynı şekilde her kuyruğun, birden fazla sınıfı da olabilir. [Radhakrishnan, 1999]

3.3.3 Filtreler

Filtreler, paketleri belirli özelliklerine göre sınıflandırmak için kullanılır. Bu özellikler, IP adresleri, port numarası ve TOS biti gibi değerler olabilir. Filtreler, kuyruk disiplinleri tarafından gelen paketleri, sınıflardan birine atamak için kullanılır. [Radhakrishnan, 1999]

3.3.4 tc (Traffic Control) Kullanımı

Traffic Control (tc), kullanıcı seviyesinde bir uygulama olup, kernel fonksiyonları ve veri yapıları ile netlink soketleri aracılığı ile haberleşmektedir.

tc, çıkış cihazları için kuyruk yaratmak ve ilişkilendirmek için kullanılır. tc ile, farklı kuyruk çeşitleri oluşturulur ve bu kuyruklar sınıflar ile ilişkilendirilir. tc ile, yönlendirme tablosu, RSVP sınıflayıcıları gibi değerleri kullanarak filtre tanımı da yapılabilir.

4. TASARIM

'Servis Kalitesi' bölümünde anlatılan servis kalitesi tekniklerinden yola çıkıldığında, bir bant genişliği yönetim sisteminin tasarlanması ve gelişmesinde aşağıdaki noktalar göz önüne alınmalıdır:

1. Sistem gelen ağ bağlantılarını kontrol edebilmelidir.
2. Sistemde değerlendirilecek olan kurallar tanımlanabilmeli ve bu kurallar üzerinde değişiklik yapılabilmelidir.
3. Gelen ağ paketleri önceden tanımlanan kurallara uygun şekilde sınıflandırılabilmelidir. Servis verilme oranlarının sınırlandırılması gerektiğinden, sistem toleranssız bir servis kalitesi sağlamalıdır.
4. Sınıflandırma işlemi sonucunda paketler hiyerarşik bir yapıda listelerde tutulmalıdır.
5. Tanımlı kurallar göz önüne alınarak, listelere yerleştirilen paketlere servis verilmelidir.
6. Sistem tanımlanan kurallara göre yönlendirilen paket oranını sınırlandırabilmeli veya paket yönlendirmesini durdurabilmelidir.
7. Paketlere servis verme işlemi kendi içinde adil olmalı ve öncelik tanımlarını değerlendirebilmelidir.
8. Paketlerin değerlendirilmesi ve paketlere servis verilmesi işlemleri sistemin normal çalışmasını etkilememeli ve gecikmelere sebep olmamalıdır.

Yukarıdaki noktaları sağlayacak şekilde bir tasarım temel olarak 3 ayrı modülden oluşmaktadır. Bu 3 ayrı modül aşağıdaki gibidir:

1. Sistemin başlaması
2. Sınıflandırma
3. Şekillendirme

4.1 Sistemin Başlaması

Sistemin başlaması aşamasında, sistemin sınıflandırma ve şekillendirme fonksiyonlarını yerine getirebilmesi için gereken başlangıç işlemeleri gerçekleştirilmelidir. Bu aşamada yapılması gereken işlemler aşağıdaki listelenmiştir.

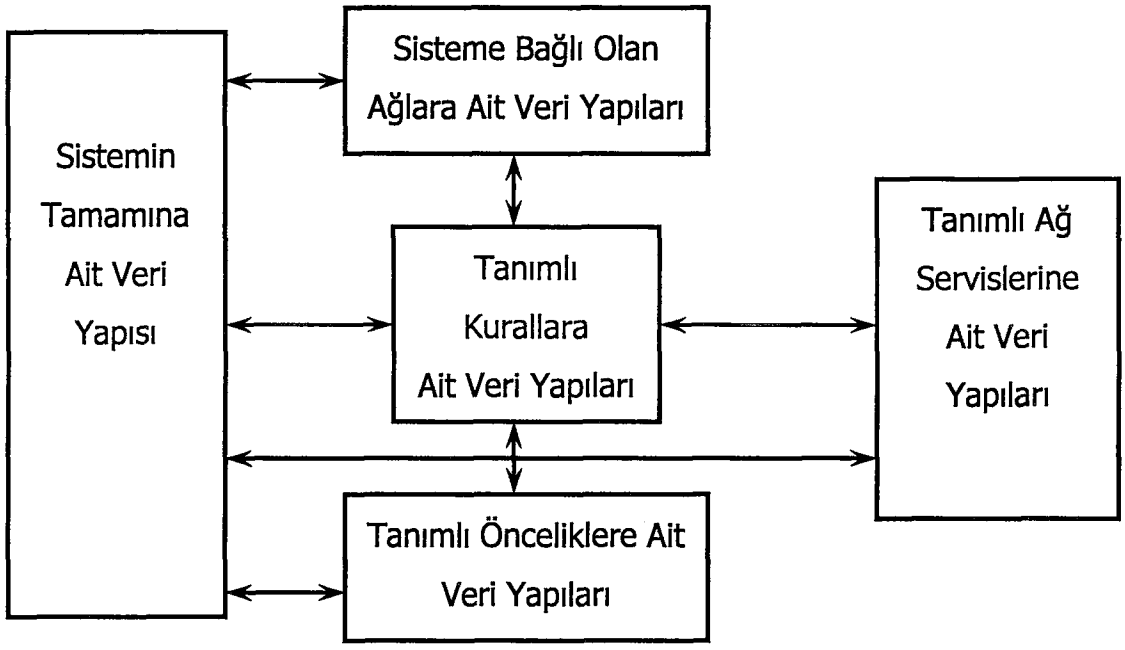
1. Genel kullanıma açık olan veri yapılarının oluşturulması
2. Konfigürasyon dosyalarının okunup, okunan verilerin oluşturulan veri yapılarında saklanması
3. Gelen paketleri alıp değerlendirecek yeni paket tipinin tanımlanması
4. Mevcut paket tipinin yedeklenip yerine tanımlanan yeni paket tipinin kaydedilmesi

Bu işlemlere ait ayrıntılı bilgiler aşağıdaki bölümlerde verilmiştir

4.1.1 Genel Kullanıma Açık Olan Veri Yapılarının Oluşturulması

Sistemin gelen paketler ve tanımlı kurallar üzerinde işlem yapacağı, bu yapıların değerleri üzerinde değişiklik yapacağı göz önüne alınırsa, kullanılacak değerlerin genel veri yapılarında tutulması zorunluluğu ortaya çıkar. Bu zorunluluğun yerine getirilebilmesi için sistemin başlangıcında bu veri yapıları oluşturulmalı ve sistem tamamıyla devreye girmeden bu veri yapılarının doğruluğu sağlanmalıdır.

Sistemde genel kullanıma açık olan veri yapıları, sistemin tamamına, sistemin bağlı olduğu ağların tanımlarına, sistemde tanımlı olan servislere ve sistemin temelini oluşturan kurallara dayanmalıdır. Temel olarak bu beş yapı birbirine bağlı olmalı ve sistemin devreye girebilmesi için tanımlarının doğru yapılması gereklidir. Bu beş yapının birbirleri ile olan durumları Şekil 4.1 de gösterilmiştir.



Şekil 4.1 Sistemde kullanılan veri yapıları ve birbirleri ile olan bağlantıları

Genel kullanıma açık olan veri yapıları ve içermeleri gereken değerlere ait bilgi aşağıdaki bölümlerde belirtilmiştir.

4.1.1.1 Sistemin Geneline Ait Olan Veri Yapısı

Sistemin tamamı tarafından erişilebilecek ve çalışma esnasında her aşamada ihtiyaç duyulabilecek olan bilgileri içermelidir. Bu yapıdaki bilgilerin bir kısmı çalışmanın her esnasında değiştirilebileceği gibi, bir kısmı da sistemin başlaması esnasında aldıkları değerleri sistem devreden çıkarılana kadar koruyabilmelidir.

Çalışma esnasında erişilmesi gereken bilgilerden bazıları sisteme gelen toplam paket sayısı, sistemin konfigürasyon dosyalarının disk üzerindeki yerleri, bant genişliği kullanımının sınırlandırılmasında kullanılacak servis verme algoritmasının işlemesi için gerekli olan değerler olarak özetlenebilir.

4.1.1.2 Tanımlı Önceliklere Ait Veri Yapıları

Kurallar, sistemde tanımlanacak öncelik değerlerine göre servis alabilmeli ve yüksek önceliklere daha fazla servis verilmelidir. Her önceliğe ait yapıda, o öncelik değerine sahip kurallara ait bilgi tutulmalıdır.

4.1.1.3 Sisteme Bağlı Olan Ağlara Ait Veri Yapıları

Sistemin bağlı olduğu ağların tanımlanması, sistemin hangi ağlardan gelen bilgileri yönlendirmesi gerektiğine karar verebilmesi için gereklidir. Bu bilgi sayesinde, kuralların tanımlanması aşamasında, sistem kendine bağlı olmayan ağlara ait kuralları değerlendirmeyebilir.

Sistemin bağlı olduğu ağlara ait bilginin tutulduğu veri yapılarında, ağ adresi ve ağ maskesi değerlerinin tutulması gereklidir.

4.1.1.4 Tanımlı Kurallara Ait Veri Yapıları

Sistemde bant genişliğinin yönetimi için temel alınacak kuralların tanımı, sistemin başlaması esnasında ilgili veri yapılarına alınmalıdır. Bu kurallar, gelen paketlerin şekillendirilmesi aşamasında kullanılan kuralları oluşturmalıdır. Kurallara ait veri yapıları oluşturulurken, sistemin bağlı olduğu ağ bilgileri kullanılmalıdır.

Kuralların tanımında, bir bağlantıyı ayırt edebilecek tüm bilgiler bulunmalıdır. Kural tanımında, bağlantıyı tanımlayabilecek bilgiye ek olarak, kurala verilecek olan bant genişliği yüzdesi ve kuralın sistem içinde hangi öncelik değeri ile servis alacağını belirten bir değer bulunmalıdır.

4.1.1.5 Tanımlı Ağ Servislerine Ait Veri Yapıları

Kurallarda kriter olarak tanımlanmak üzere, sistemde tanımlı olan ağ servislerinin sistem tarafından bilinmesi gerekmektedir. Bu tanımlar, işletim sisteminde tanımlı olmasına rağmen uygulama içinde de tanımlı olmalıdır. Bu yüzden, ağ servislerine ait bilgiye erişebilmek için bu servislerin bilgilerinin veri yapılarında tutulması gerekir. Ağ servislerine ait port, protokol ve isim bilgisini tutmak yeterlidir.

4.2 Sınıflandırma İşlemi

Herhangi bir servis kalitesi yönetim sisteminin çalışmasındaki en önemli iki aşamadan birincisi, sisteme gelen paketlerin, tanımlı kural sınıflarından birisi ile eşleştirilmesidir. Sınıflandırma işlemi sonucu, trafiğin şekillendirilmesi aşamasında, öncelik ve servis verilecek paketler ve sınıfların servis alma sıraları belirlenir.

Sınıflandırma işleminde önem verilecek noktalar aşağıdaki gibidir:

- Bağlantılara ait bilgilerin mevcut kurallarla eşleştirilmesi mümkün olan en kısa sürede olmalıdır.
- En kısa sürede sınıflandırmayı sağlamak amacıyla kurallar, ağ ve host bilgisine göre değerlendirilmelidir.
- Sınıflandırma sonucu bir sınıfa yerleştirilen pakete ait bilgi, pakete daha sonra servis verilmek üzere saklanmalıdır.
- Kurallar sınırlandırılmak istenilen trafiğe ait bilgiyi içermeli, sınıflandırma sonucu bir sınıfa yerleştirilemeyen pakete servis verilmelidir.

Sınıflandırma işleminde, bağlantılara ait bilgilerin mevcut kurallarla karşılaştırılma süresinin mümkün olduğunca düşük olmasına dikkat edilmelidir. Sınıflandırma işleminin yavaş olması sisteme gelen paketlerim birikmesine ve yüksek servis verme sürelerine yol açabilir. Bir paketin ait olduğu kural bulunduktan sonra, pakete servis verilmesini sağlayacak bilgiler kurala ait bir hafıza bölgesinde saklanmalıdır. Böylece servis verilme sırası o kurala geldiğinde, önceden gelmiş, sınıflandırılmış ve kurala ait bölgede saklanan paketlerden birine servis verilebilir. Kurala ait hafıza bölgesinden, servis verilmek üzere paket seçimi işlemi ise şekillendirme aşamasının bir parçasıdır.

4.3 Şekillendirme İşlemi

Şekillendirme işlemi, sınıflandırma işlemi sonucunda, servis verilmek üzere kurallara ait hafıza bloklarında saklanan paketlere, uygulanması karar verilen servis verme ve sıralama algoritmasına göre, servis verilmesi işlemidir. Bant genişliğini ve yönlendirme oranını istenilen seviyede tutan ve sınırlayan işlemler sistemin bu aşamasında gerçekleştirilmelidir.

Şekillendirme işleminin aşamaları aşağıdaki özelliklere sahip olmalıdır.

- Sistemde bekletilen paketlere öncelik tanımlarına göre servis verilmelidir.
- Önceliği yüksek olan paketler, sistemin kullanımını ele geçirip, düşük öncelikli paketlere servis verilmesini engellememelidir. Bunun için adil ve öncelik kullanımına dayanan bir sıralama ve servis verme algoritması kullanılmalıdır.
- Yüksek önceliğe sahip servis verilecek paket yok ise bir sonraki öncelik değerine

sahip paketler değerlendirilmelidir.

- Servis verilen pakete ait bilgi sistemden uzaklaştırılmalıdır.
- Bir pakete servis verildikten sonra, ait olduğu kurala ve genel kullanıma ait veri yapıları güncellenerek, şekillendirme işleminin bir sonraki adımı için kullanılacak bilginin doğruluğu garantilenmelidir.
- Bir pakete servis verirken, yeni gelen bir paketin genel kullanıma açık veri yapısını veya kullanılmakta olan kurala ait yapıyı bozması engellenmelidir.

Yukarıda da belirtildiği gibi şekillendirme işlemi için en önemli nokta seçilecek sıralama ve servis verme algoritmasıdır. Kullanılacak algoritma, öncelik değerlerini değerlendirebilmeli, adil olmalı ve istenilen kullanım kısıtlamalarını gerçekleştirebilmelidir.

5. UYGULAMA

Sisteme ait uygulama, tasarımda ortaya çıkan 3 aşamaya ait fonksiyonların tanımlanıp gerçekleştirilmesi sonucu ortaya çıkmıştır. 3 aşamaya ait uygulama ayrıntıları aşağıda anlatılmıştır. Uygulamada kullanılan fonksiyonlar, veri yapıları ve dosya formatları 6., 7. ve 8. bölümlerde açıklanmıştır.

Sistemin uygulama ayrıntılarından önce, seçilen uygulama geliştirme metodu olan "Yüklenebilir Kernel Modülleri" (Loadable Kernel Modules) anlatılmıştır.

5.1 Seçilen Geliştirme Metodu: LKM'ler

Hedeflenen sistemin, sisteme gelen bütün bağlantıları ele geçirebilmesi, işleyebilmesi ve gerektiği takdirde paketlerin iletilip iletilmemesine karar verebilmesi gerekmektedir. Bu işlemlerin C soket kütüphanesi kullanılarak gerçekleştirilmesi, hem uygulamaya getireceği yük açısından, hem de kernelde çalışmanın getireceği avantajları kullanamama açısından uygun değildir. Bu yüzden kullanılabilecek 2 adet seçenek vardır:

- Kernel kodunun amaca uygun olarak değiştirilmesi,
- Yüklenebilir kernel modüllerinin kullanımı.

Kernel kodunun değiştirilmesinin getireceği dezavantajlar aşağıdadır.

- Geliştirilen kodun derlenmesinin ve hata izlenmesinin zorluğu ve getireceği zaman kaybı,
- Geliştirilen kodun başka platformlarda kullanılması için kernel'ın tamamının değiştirilmesi,
- Yapılacak konfigürasyon değişikliklerinin devreye girmesi için, sistemin tekrar başlatılması,

şeklinde listelenebilir. [pragmatic, 1999]

LKM kullanımı, listelenen dezavantajları ortadan kaldırır. LKM'ler, kernelin işlevselliğini artırmak için kullanılan kernel modülleridir. LKM'ler, kernel kodu üzerinde değişiklik yapmadan dinamik olarak yüklenmeyi sağlar ve kernel seviyesinde çalışır. Bu avantajlarından dolayı sistemin geliştirilmesi için LKM metodu seçilmiştir.

5.2 Sistemin Başlaması

Sistemin başlaması aşamasında, ilk gerçekleşen işlem kullanılacak veri yapılarının oluşturulmasıdır. Kullanılacak veri yapılarına ait bilgi konfigürasyon dosyalarında tanımlanmış olup, bu dosyaların okunup parçalara ayrılması sonucu, tasarlanan veri yapılarına yerleştirilir.

5.2.1 Ana Konfigürasyon Dosyasının Okunması

Fonksiyonların tamamı tarafından kullanılacak olan değerler ana konfigürasyon dosyasından okunur. Ana konfigürasyon dosyası, ağ, servis ve kural dosyalarının yerlerini ve isimlerini ve kullanılabilecek en yüksek öncelik değerini içeren dosyadır. Bu bilgilerin dosyadan okunması için *parseConfFile* ve *parseCLine* fonksiyonları kullanılır. Dosyadan okunan değerler, ana veri yapısı olan, *shaper* tipindeki (Şekil 7.1) *shape* değişkeninde saklanır. Bu bilgilerin dosyadan okunması esnasında oluşan bir hatada veya okunan bilgilerin geçersiz olması durumunda hata kodu üretilir ve modülün yüklenmesi engellenir.

Dosyadan okunan en yüksek öncelik değerinin sonucu, *shape* yapısının *maxPriority* alanına yazılır. Bu değere bağlı olarak, *shape* yapısı üzerinden erişilebilen, her gözü bir öncelik değerine ait olacak şekilde, en yüksek öncelik değeri kadar *priorityElement* (Şekil 7.2) tipinde eleman içeren bir dizi oluşturulur. Oluşturulan dizinin her gözündeki yapının alanlarına ilk değerleri atanır. İlk değerlerin atanmasında dikkat edilmesi gereken nokta, öncelik yapısına ait değerlerin bir değer almaması halinde, atanmış olan ilk değerlerin sistemin işleyişini etkilememesi gereğidir. Bunun için *policyCount* alanı 0, *policyArray*, *lastPAE* ve *lastPolServed* alanları da NULL ilk değerlerini alırlar.

Ana konfigürasyon dosyasının başarılı bir şekilde okunmasından sonra, servis, kural ve ağ konfigürasyon dosyalarının okunma ve değerlendirilme işlemleri, yerleri ve isimleri ana konfigürasyon dosyasında belirlenen dosyalar üzerinden gerçekleşir.

5.2.2 Servis Konfigürasyon Dosyasının Okunması

Servis konfigürasyon dosyası için özel bir format ve değerlendirme fonksiyonu geliştirilmemiş, Linux işletim sisteminde kullanılan ve ağ servislerini tanımlayan, *etc* dizininde yer alan *services* dosyası kullanılmıştır. Bu dosyanın değerlendirilmesi sonucunda okunan servis değerleri *shape* yapısının *serviceEntry* tipindeki (Şekil 7.8) alanı *servicesBase* işaretçisi başlangıç olmak üzere linkli listeye yerleştirilir.

5.2.3 Ağ Konfigürasyon Dosyasının Okunması

Ağ konfigürasyon dosyasından okunan ağ bilgileri *shape*, yapısındaki *netBase* alanı başlangıç değeri kabul edilerek bir linkli listeye yerleştirilir. Bu linkli listedeki verilerin tipi *net* veri yapısıdır. (Şekil 7.3) Dosyadan okunan bilginin geçerli olmaması veya dosyada hiçbir ağ bilgisinin bulunamaması durumunda ağ dosyasını okuyan fonksiyon bir hata üretir ve modül devreye girmez. Okunan ağ bilgisi geçerli ise, bu bilgiye ait veri yapılarına ilk değerlerin atanması gerekmektedir. İlk değer atanması gereken alanlar, ağa ait host bilgisini tutan dizi ve ağa ait kurallar listesidir.

Ağa bağlı olan hostlara ait bilgi, *hostArrayEntry* veri tipindeki (Şekil 7.4) elemanlardan oluşan bir dizide saklanmaktadır. Bu dizideki her göze NULL değeri atanarak, başlangıç anında ağa bağlı hostlardan hiçbiri ile ilgili tanım yapılmadığı varsayılır.

Ağ bilgisinin içerdiği *hostArrayEntry* tipindeki dizi, her gözünde en fazla 4 adet elemanı olan ve basit bir *hashing algoritması* ile ağa bağlı hostlara ait bilgiyi tutmak için kullanılır. Bu dizide hostların eşleştirilmesi için kullanılan hashing fonksiyonu aşağıdadır.

$$i = (host\_adresi\ AND\ ağ\_adresi)\ mod\ ((ağ\_maskesi)' / 4)$$

Hashing fonksiyonu sonucu oluşacak *i* değeri, hosta ait bilgiyi tutmak için kullanılacak dizi elemanının indisini verir. Hashing sonucu çakışma olması durumunda, dizi elemanları, birim boyutu 4 olan bir linkli liste oluştururlar. *hostArray* dizisinin ve bu dizinin elemanlarının gösterdiği *hostArrayEntry* tipindeki yapılarda hostlara ait yer ve kural bilgisi tutulmaktadır. (Şekil 7.4)

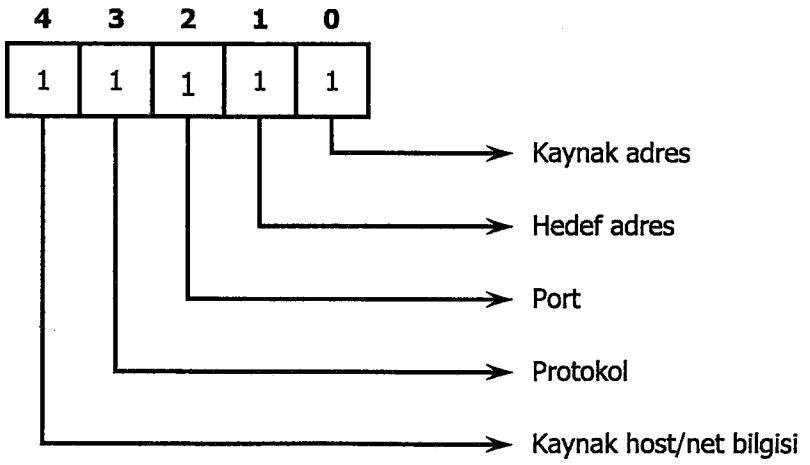
5.2.4 Kural Konfigürasyon Dosyasının Okunması

Kural dosyasının okunması ve değerlendirilmesi işleminden önce, sistemde tanımlanacak kurallar ve kurallarda kullanılabilen kriterler açıklanacaktır. Sisteme gelen paketlerin değerlendirilmesi için, pakete ait TCP ve IP protokollerine ait bilgiden faydalanılmıştır. Bu bilgiler aşağıda listelenmiştir.

- Kaynak IP adresi: Paketin geldiği kaynak adrese göre sınıflandırma yapmak için kullanılır. Bir host veya ağı belirtebilir.
- Kaynak ağ maskesi: Kaynak adres kriterinin bir ağ bilgisini göstermesi durumunda, tanımlanan kaynak ağa ait ağ maskesi bilgisidir. Aynı ağda bulunan hostlara uygulanacak kurallar için kullanılır.

- Hedef IP adresi: Paketin gideceği hosta ait IP adresi bilgisidir. Bir hosta veya bir ağdaki hostlara erişimi kısıtlamak için kullanılır. Bir host veya ağ belirtebilir.
- Hedef ağ maskesi: Hedef adres kriteri bir ağ bilgisini gösterecek şekilde verildiğinde, hedef ağa ait ağ maskesi bilgisini belirtir. Belirli bir ağdaki hostlara erişimi sınırlandırmak için kullanılır.
- Port: Belirli bir servise ait bilgi akışını sınırlandırmak için kullanılır.
- Protocol: IP paketinde taşınan protokol bilgisine göre akış sınırlaması yapmak için kullanılır.
- Öncelik: Kurala ait öncelik bilgisidir. Kurallara farklı öncelikte servis verilmesi amacıyla kullanılır.
- Kurala ayrılan oran: Kurala ait servis verilen paketlerin, sisteme gelen toplam paketlere oranını belirtir. Bir kural için sağlanması istenen üst sınırdır.

Kurallara ait dosyanın okunmasında yapılan işlem, her satırın okunup satırdaki bilgilerin geçerliliğinin kontrolü ve kuralın mevcut yapılardan hangisine dahil edileceğine karar verilmesidir. Yapılan kontroller sonucu geçerli olduğu doğrulanan her kural *policy* tipindeki bir yapıda saklanır. (Şekil 7.5) Bu yapıdaki *base* alanına tanımlanabilecek kriterlerden hangilerinin kullanıldığı bilgisini göstermesi için bir değer atanır. Bu alandaki her bitin hangi kritere göre değer alacağı aşağıda belirtilmiştir. Kural tanımında, bite karşılık gelen kriter kullanılmışsa, o bit 1 değerini alır.



Şekil 5.1 *policy* veri tipi *base* alanı tanımı

Geçerli olduğu doğrulanan ve *policy* tipindeki değişkene aktarılan bilgiye aşağıdaki kontroller uygulanır. Bu kontroller sonucunda, tanımlanan kaynak adrese göre kuralın hangi veri yapısı içinde yer alacağına karar verilir.

- Kurala ait bir kaynak IP adresi belirlenmemiş ise, kural *shape* yapısındaki *policyBase* yapısında saklanır. Bu işaretçi, kaynak adresi belirtilmeyen kuralların tutulduğu linkli listeyi gösterir.

- Kurala ait bir kaynak adres tanımlanmışsa, bu kaynak adresin sisteme bağlı olan ağlarda yer alıp almadığı kontrol edilir. Eğer kaynak adres tanımlı ağlardan birinde yer alıyorsa, kaynak adresin bir host adresini mi, ağ adresini mi tanımladığı, *base* alanı değerlendirilerek kontrol edilir.

- Eğer kaynak adres sisteme bağlı ağlardan biri ise, kural o ağa ait *net* tipindeki yapının *policy* alanı ile gösterilen linkli listeye eklenir.

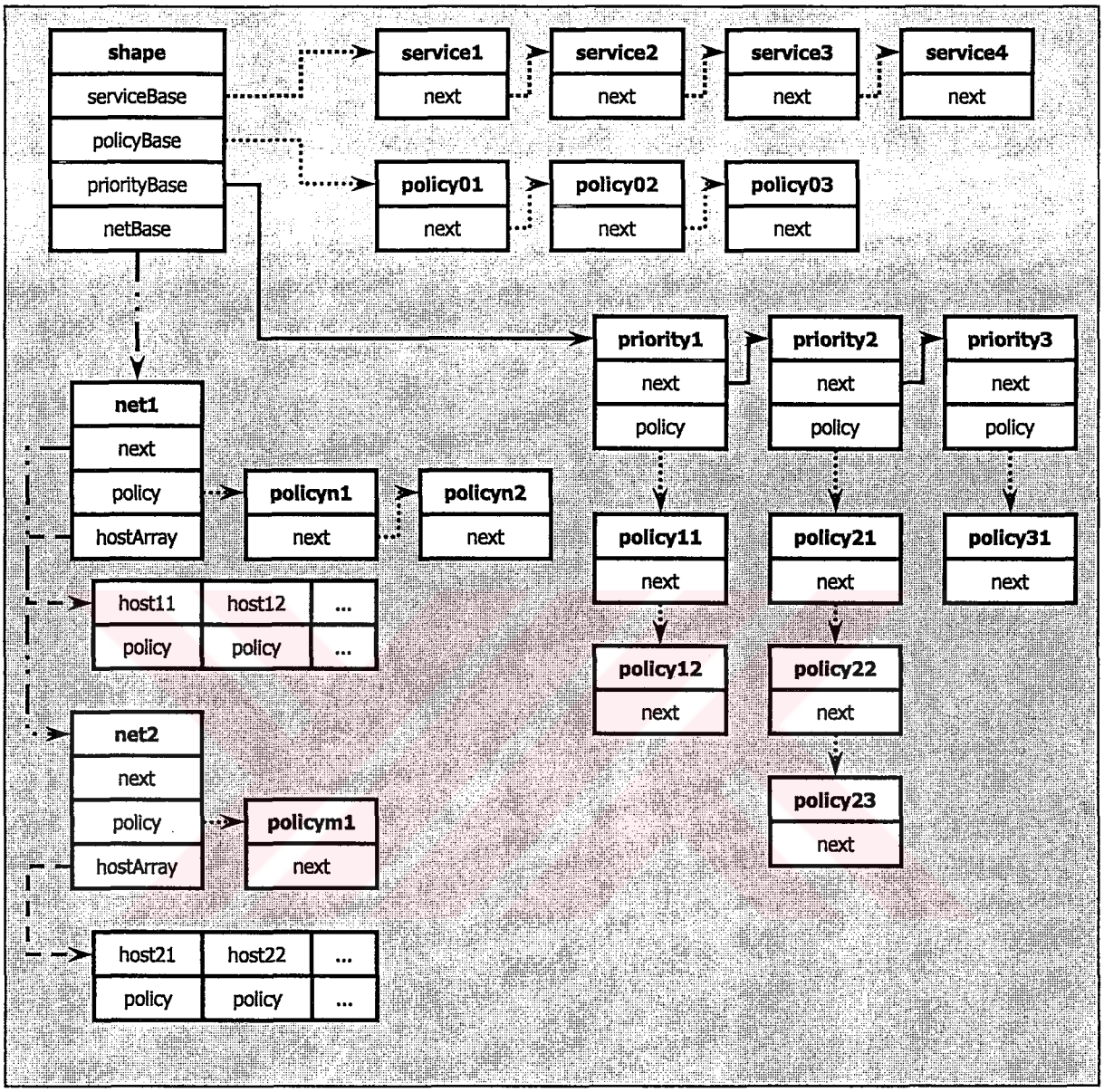
- Kaynak adres bir hostu belirtiyorsa o hostun dahil olduğu ağ bilgisi bulunur. Hosta ait kural bilgisini yerleştirmek için, hostun bağlı olduğu ağa ait *net* yapısının *hostArray* alanında hostun bilgisinin yerleştirileceği indisin hesaplanması gerekir. Bunun için 'Ağ Konfigürasyon Dosyasının Okunması' bölümünde tanımlanan hashing fonksiyonu kullanılır. İndisi hesaplanan eleman, en fazla 4 eleman içerebilecek bir linkli listenin başlangıç adresi olduğundan, bu liste üzerinde boş olan ilk yere kuralda tanımlanan hosta ait *hostArrayEntry* yapısı yerleştirilir. *hostArrayEntry* tipindeki dizi elemanının *address* alanına hostun bağlı olduğu ağdaki sırası değeri atanır. Dosyadan okunan kural da, *policy* alanı ile belirtilen, hosta ait kuralların linkli listesine eklenir.

- Kuralda belirtilen kaynak adres tanımlı ağlardan birinde yer almıyorsa okunan kural bilgisi yok varsayılır.

Veri yapılarından herhangi birine eklenebilen kuralın, *shape* yapısında oluşturulan öncelik dizisinde uygun yere yerleştirilmesi gerekir. Kuralın öncelik değeri, kuralın *priorityBase* dizisinin hangi gözüne yerleştirileceğini belirtir.

priorityElement tipinde yer alan *polArrayElement* tipindeki (Şekil 7.6) *policyArray* elemanı, *net*, *hostArrayEntry* veya *shaper* tipindeki veri yapılarında tutulan kurallara, sadece işaretçi kullanımı ile sıralı bir şekilde erişmeyi sağlamaktadır. Bu yapının kullanılması ile paketlere servis verilmesi esnasında, *net*, *hostArrayEntry* ve *shaper* yapılarına ait kuralların kontrolünün engellenmesi ve kurallara önceliklere göre servis verilmesi sağlanmıştır.

Tüm konfigürasyon dosyalarının okunması sonucunda, oluşturulan yapılar Şekil 5.2'de gösterilmiştir.



Şekil 5.2 Sistemin başlaması işlemi sonrası veri yapılarının bağlantıları

5.2.5 Tanımlı Ağ Arayüzlerinin Bulunması

Sistemde tanımlı ağ arayüzlerinin bulunması ve IP adreslerinin ve ağ maskelerinin bilinmesi, sistemin kendi arayüzlerine ait IP adreslerine gelen paketlerin tanımasına yardımcı olmaktadır.

Linux işletim sisteminde ağ arayüzlerine ait bilgi, kernel kodunda ağ cihazlarına ait bilgileri tutan *dev_base* dizisi kullanılarak elde edilebilmektedir. *net_device* veri tipinde linkli bir liste olan *dev_base*, tanımlı ağ cihazlarına ait bilgileri tutmaktadır. Bu bilgilerden biri de, cihazın

IP protokolünde kullanılabilmesini sağlayan arayüz ve adres bilgisidir. *net_device* veri tipindeki *ip_ptr* alanı, *indevice* tipinde bir yapıyı göstermektedir. *indevice* veri yapısındaki, *ifa_list* alanı cihaza ait arayüzler ile ilgili adres ve maske bilgisini içeren *in_ifaddr* tipindeki listeyi gösteren bir alan içermektedir. *in_ifaddr* yapısındaki *ifa_address* ve *ifa_mask* değerleri kullanılacak ağ arayüzü bilgisi için yeterli bilgilerdir.

Ağ arayüz bilgileri, ilgili yapılardan okunmadan önce, sistemde kaç arayüz tanımlı olduğu bilgisinin bulunması gerekmektedir. Ağda tanımlı arayüz sayısı bulununca, *shape* yapısındaki *interfaces* alanında, tanımlı arayüz sayısı kadar adres ve ağ maskesinin tutmak üzere 2 adet dizi oluşturulur. (Şekil 7.7) Arayüz sayısı bulunduktan sonra, kernelde yer alan yapılar tekrar kontrol edilerek okunan ağ arayüz ve maske değerleri oluşturulan dizilerin gözlerine atanır.

5.2.6 Yeni IP Paket Tipinin Oluşturulması Ve Kaydedilmesi

Sistemin, gelen paketler üzerinde işlem yapabilmesi ve tanımlanmış kuralları paketlere uygulayabilmesi için, paketlerin sistemin bir noktasında ele geçirilmesi gerekmektedir. Bunun için de Linux kerneli tarafından, kernel programlamada kullanılması için dışarı aktarılan fonksiyon işaretçilerinden ve veri yapılarından faydalanılmıştır.

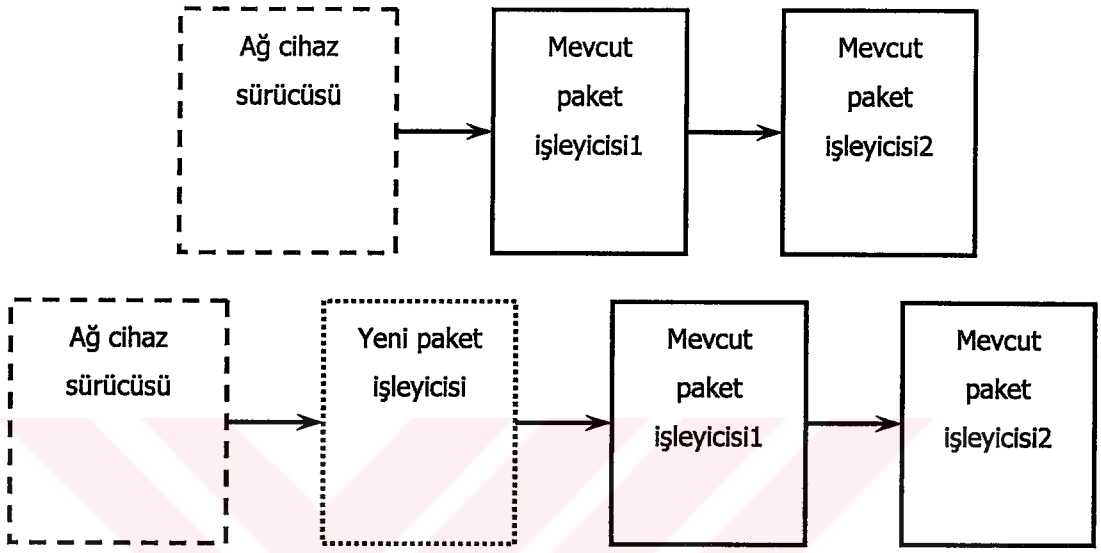
Linux kerneli gelen ağ paketlerine, tiplerine göre servis verebilmek için, paket tiplerinin kaydedildiği bir dizi tutmaktadır. Kayıtlı olan her paket tipinde aşağıdaki bilgiler bulunmaktadır.

- Paketin tipi
- Paket tipinin ilişkilendirileceği ağ cihazına ait işaretçi
- Paket tipinde ağ paketi geldiğinde kullanılacak işleyici fonksiyona ait işaretçi
- Paket tipinin kullanımına ait özel veri
- Bir sonraki paket tipine ait işaretçi

Paketlere, cihaz sürücüsünden çıktıktan hemen sonra müdahale edilebilmeli ve mevcut IP işleyici fonksiyonunun paketler üzerinde işlem yapması engellenmelidir. Sistem bu amaçla kendi IP paket tipini oluşturmalı, bu paket tipi için kendi işleyici fonksiyonunu bu paket tipi ile ilişkilendirip, yeni paket tipini kernele kaydetmelidir. Kernele yeni paket tipi kaydetme işlemi şu şekilde gerçekleşmektedir:

Öncelikle yeni paket tipine ait işleyici fonksiyon yazılmalı ve yeni paket tipi için *packet_type*

tipinde bir data yapısı (Şekil 7.9) oluşturulmalıdır. Daha sonra kernelde tanımlı olan *dev_add_pack* fonksiyonu yardımı ile yeni paket tipi mevcut paket tiplerinin tanımlı olduğu diziye eklenmelidir. Yeni kaydedilen paket tipi, cihaz sürücüsü ile aynı tipte daha önce tanımlanmış olan paket tipinin önüne yerleşir ve yeni paket işleyici fonksiyonu, mevcut paket işleyicisinden daha önce çalışır. Bu yapı Şekil 5.3'te gösterilmiştir.



Şekil 5.3 Yeni paket tipinin eklenmesi işlemi ve paket tiplerinin durumu

Yeni oluşturulan IP paketine ait işleyici fonksiyon *new_ip_rcv* fonksiyonudur. Yeni paket kernele kaydedildikten sonra sisteme gelen IP paketleri bu fonksiyon tarafından işlenecektir.

Gelen paketlere hemen servis verilmesini engellemek ve paketler üzerinde kontrol ve değişiklik yapabilmek için, yeni IP paket tipi tanımlandıktan sonra mevcut IP paket tipinin geçici olarak sistemden kaldırılması gerekmektedir. Yeni paket tipi oluşturulduktan ve *dev_add_pack* fonksiyonu ile kernele kaydedildikten sonra, yeni paket tipinin *next* alanında mevcut IP paket tipine ait işaretçi bulunmaktadır. Bu işaretçinin değeri, modülün devre dışı bırakılması esnasında paket tipini tekrar devreye alabilmek için saklanır.

5.3 Paketlerin Sınıflandırılması

Paketlerin sınıflandırılması işlemi, tasarlanan bant genişliği yönetim sisteminin, modül devreden çıkarılana kadar aktif olarak çalışan iki parçadan biridir. Sınıflandırma işlemi temelde, gelen paketin tanımlı kuralların kontrolü sonucu kriterlerine uyduğu sınıfın bulunması ve sınıfa ait bekleyen paketler listesine yerleştirilmesidir. Sınıflandırma işlemi

aşağıdaki aşamalardan oluşmaktadır:

- Paketin sisteme, ağa veya yerel ağdan bir adrese gelip gelmediğinin kontrolü,
- Paketin tanımlı kurallar ile karşılaştırılıp, paket için kriterleri sağlayan kuralın bulunması,
- Tanımlı kuralın bulunması durumunda paketin bekleyen paketler listesine eklenmesi.

Bu aşamalar ve çalışmaları aşağıda anlatılmıştır.

5.3.1 Paketin Kaynak Adres Kontrolü

Sisteme paket geldiğinde yapılan ilk işlem, sisteme gelen toplam paket sayısını 1 artırmaktır. Daha sonra paketin sistemin kendisine mi, yerel ağda bir hosta mı ait olduğu, gelen paketin tipinin BROADCAST olup olmadığı kontrol edilir. Paketin sisteme ait ağ arayüzlerinden birine gelip gelmediğinin kontrolü, *ifLocalAddress* fonksiyonu kullanılarak yapılır. Bu fonksiyon sistemde tanımlı olan ağ arayüzü adreslerini, pakete ait IP bilgisindeki hedef adres ile karşılaştırılır. Paketin yerel ağdan bir adrese gelip gelmediği *ifInLAN* fonksiyonu kullanılarak kontrol edilir. Bu fonksiyon, bir hostun sisteme bağlı ağlardan birinde yer alıp almadığını, *shape* yapısındaki, bağlı ağlara ait adres ve ağ maskesi bilgileri ile pakete ait IP bilgisinin hedef adresinin kontrolü ile belirlenir.

Paketin tipinin BROADCAST olması, paketin yerel ağ içindeki tüm hostlara iletileceği anlamına gelmektedir. Bunun kontrolü için, pakete ait *sk_buff* tipindeki, paketin protokol bilgisini içeren *skb* değişkeninin, *pkt_type* alanı kontrol edilmelidir. Bu alan ve alabileceği değerler aşağıda sıralanmıştır:

- PACKET_HOST – 0: Paket hosta gelmiştir.
- PACKET_BROADCAST – 1: Paket tüm hostlara iletilecektir.
- PACKET_MULTICAST – 2: Hostun içinde bulunduğu host grubuna iletilecektir.
- PACKET_OTHERHOST – 3: Başka bir hosta iletilecektir.
- PACKET_OUTGOING – 4: Dışarı gönderilecek bir pakettir.
- PACKET_LOOPBACK – 5: MC/BRD frame looped back
- PACKET_FASTROUTE – 6: Fastrouted frame

skb yapısındaki *pkt_type* alanı BROADCAST / 1 değerine sahipse, sistem tarafından pakete servis verilmelidir.

Paket bilgisi yukarıda belirtilen 3 kriterden herhangi birini sağlıyorsa, pakete hemen servis verilir. Pakete servis vermek için, orijinal IP paket tipine ait işleyici fonksiyonu çağırmak gerekir. Orijinal IP paket bilgisi sistemin başlaması aşamasında yedeklendiğinden, bu yapıya ait işleyici fonksiyon işaretçisine ulaşılabilir. Linux kernelinde, orijinal IP paketinin işleyici fonksiyonu, *ip_rcv* fonksiyonudur. Kriterleri sağlayan pakete servis vermek için, *ip_rcv* fonksiyonunu, gerektirdiği parametreler ile çağırmak yeterlidir.

5.3.2 Tanımlı Kurallar İçinde Pakete Uyan Kuralın Aranması

Gelen paket yukarıda tanımlanan 3 kriterden birini sağlamıyorsa, yönlendirilecek bir pakettir. Yönlendirilecek paketin servis alabilme sırasını belirleyebilmek için, pakete ait bilginin, tanımlı olan kurallardan birine uyup uymadığı kontrol edilir. Eğer paket tanımlı kurallardan birine uyuyorsa, daha sonra servis verilmek ve kurala ait bant genişliği kullanımını kontrol edebilmek amacıyla kurala ait paket listesine yerleştirilir. Paketin mevcut kurallarla karşılaştırılması için *findPolicy* fonksiyonu kullanılır. *findPolicy* fonksiyonu sırası ile *hostArrayEntry*, *net* ve *shape* yapılarını kontrol ederek, paket bilgisi ile uyuşan kural bilgisini arar.

Bir pakete ait bilginin, bir kuralda tanımlanan kriterler ile karşılaştırılmasında aşağıdaki değerler ve yapılar kullanılır.

- Kurala ait kaynak adres ve ağ maskesi değeri ile paketin IP bilgisinde yer alan kaynak adres bilgisi,
- Kurala ait hedef adres ve ağ maskesi değeri ile paketin IP bilgisinde yer alan hedef adres bilgisi,
- Kurala ait port bilgisi ile, pakete ait TCP bilgisindeki hedef port bilgisi,
- Kurala ait protokol bilgisi ile, pakete ait IP bilgisindeki protokol değeri karşılaştırılır

Kural kontrolü için her yapıda kontrol edilecek alan ve kontrol mekanizması aşağıda anlatılmıştır.

5.3.2.1 Hosta Ait Kuralların Kontrolü

Hostlara ait kurallar, hostun bulunduğu *net* yapısındaki *hostArray* dizisinin elemanlarında

saklanmaktadır. Hostun bağlı olduğu ağı bulmak için, *findInNets* fonksiyonunda yapılan işlemler yapılır. Hosta ait yer bilgisi hesaplandıktan sonra (*hostIndex*), *hostArray* dizisinde o hosta ait bilgi olup olmadığı kontrol edilir. Bunun için *hostArray* dizisinin *hostIndex* indisli elemanından başlanarak linkli liste üzerinde ilerlenir. Eğer hosta ait bir *hostArrayEntry* yapısına rastlanırsa, yapının *policy* alanı *findInPolicyList* fonksiyonuna parametre olarak verilerek uygun bir kural için kontrol edilir.

5.3.2.2 Ağa Ait Kuralların Kontrolü

Eğer hosta ait bilgi veya bu bilgi içinde hosta uygulanması gereken tanımlı kural bulunamamışsa, hostun bağlı olduğu ağların bilgisi kontrol edilir. Bu işlem *findPolicyInList* fonksiyonu ve parametre olarak hostun bağlı olduğu ağa ait *net* yapısındaki *policy* alanı kullanılarak gerçekleştirilir.

5.3.2.3 shape Yapısındaki Kuralların Kontrolü

Host ve ağa ait yapılarda uygun kural bulunamaması durumunda, uygulanabilecek kural, *findInPolicyInList* fonksiyonuna *shape* yapısındaki *policyBase* alanı parametre olarak verilerek bulunur.

5.3.3 Uygulanabilir Kuralın Bulunması Durumu

Pakete uygulanabilecek bir kural bulunması durumunda, pakete servis verilmesini sağlayacak olan bilgi, kurala ait bekleyen paketler listesine eklenir. Kurala servis verilmesini sağlayacak bilgiyi tutan yapı *queueElement* yapısıdır. (Şekil 7.10) Yapıda bulunan *firstQE* alanı, kurala ait servis verilecek ilk elemanı, *lastQE* alanı ise son elemanı göstermektedir.

Önce yeni pakete ait *queueElement* yapısı oluşturulur. Bu yapıda tutulan paketlere, FIFO (First In First Out) mekanizmasına göre servis verileceğinden, yeni gelen paket listede en sona yerleştirilir. Yeni paketi bekleyen paketler listesine yerleştirme işlemi *insertQE* fonksiyonu tarafından yapılır.

Pakete ait bilgiyi kuraldaki listeye ekledikten sonra, sisteme gelen ve kurala uyan toplam paket sayısı 1 artırılır.

5.3.4 Uygulanabilir Kuralın Bulunamaması Durumu

Host, ağ ve *shape* yapılarının kontrolü sonucunda paketteki bilgi ile eşleşen kural bulunamadığında, pakete servis verilir.

5.4 Şekillendirme İşlemi

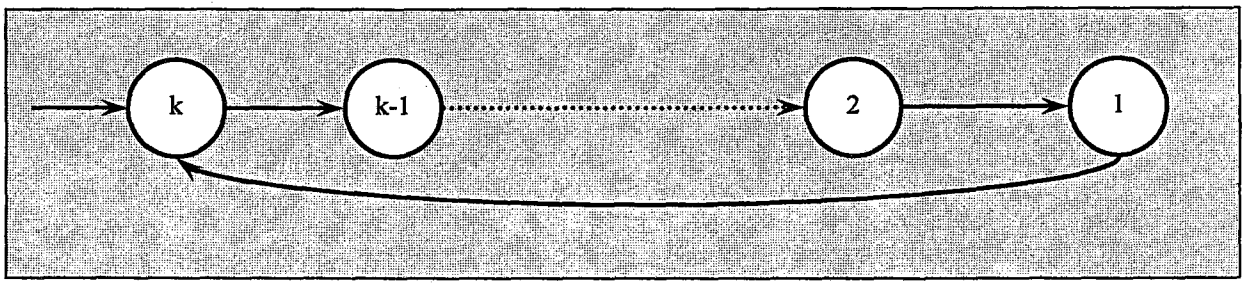
Şekillendirme işleminin gerçekleşmesi için, sisteme en az 1 paket gelmiş ve sınıflandırılmış olmalıdır. Şekillendirme işleminin öncelik tabanlı ve adil bir servis verme mekanizma olabilmesi için aşağıda anlatılan model gerçeklenmiştir.

Sistemde tanımlı olan her kuralın bir önceliği mevcuttur. Paketler kurallara göre sınıflandırılıp, kuralların öncelik değerlerine göre servis almalıdırlar. Yüksek öncelikli kurallara ait bekleyen paketler daha fazla servis alma imkanına sahip olurken, düşük önceliğe sahip kurallara daha az servis verilecektir. Buna göre belirli bir önceliğe ait servis alacak paket sayısı önceliğin yüksekliği ile belirlenmiştir. Örneğin, en yüksek öncelik değeri 4 olarak belirlenen bir sistemde, önceliklere ait kuralların servis alma sayıları, Çizelge 5.1'de verilmiştir.

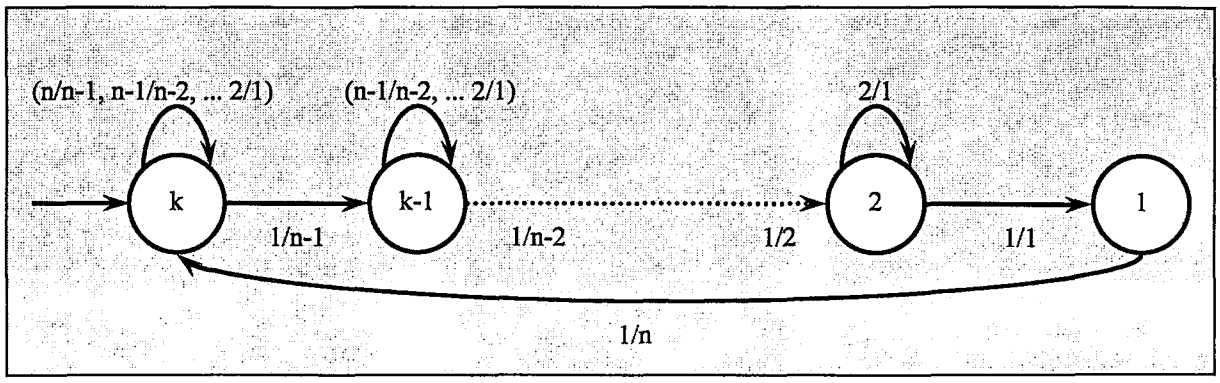
Çizelge 5.1 Örnek öncelik sayılarına göre servis alma sayısını gösteren çizelge

Öncelik değeri	Servis alma sayısı
4	4
3	3
2	2
1	1

Bu mekanizmayı gerçekleştirmek için *shape* yapısında, *prioritySwitch* ve *priorityCounter* adında 2 adet alan kullanılmıştır. Bu alanlardan *prioritySwitch* servis verilen öncelik değerini tutarken, *priorityCounter* servis verilen öncelik değerinden kaç adet kurala servis verildiğini göstermektedir. Bu bilgilere göre sistemdeki *prioritySwitch* ve *priorityCounter* değerlerine ait sonlu durum makineleri Şekil 3.5 ve Şekil 3.6'te verilmiştir.



Şekil 5.4 *prioritySwitch* alanına ait sonlu durum makinesi



Şekil 5.5 *priorityCounter* alanına ait sonlu durum makinesi

Bu durum şemalarına göre k. öncelik değerine n, (k-1). öncelik değerine (n-1) ve 1. öncelik değerine 1 kez servis verilecektir. k ve n, 1 ile en yüksek öncelik değeri arasında değer almaktadır ve her durumda eşittirler.

Sistem her önceliğe ait kural listesindeki kurallara sıralı olarak servis verir. Kuralların bekleyen paket listelerinde de her zaman *firstQE* işaretçisi ile gösterilen paket servis alır. Genel yapı özetlenecek olursa;

- Bekleyen paketler FIFO,
- Servis verilen önceliğe ait kurallar Round Robin,
- Öncelikler kendi aralarında Ağırlıklı Round Robin,

servis verme algoritmalarına göre servis almaktadırlar.

Sistem ilk olarak en yüksek önceliğe sahip kuralları kontrol ederek servis vermeye başlar. Bir önceliğin servis alabilmesi için, o öncelik değerine sahip en az bir kuralın bekleyen paket listesinin boş olmaması ve mevcut servis alma oranının, ayrılan orandan küçük olması gerekmektedir. Bir kuralın servis alma anında mevcut servis alma oranının hesabı aşağıdaki ifade kullanılarak yapılır.

*100 * kurala ait servis verilen paket sayısı / sisteme gelen toplam paket sayısı*

Bu oranın kurala ait *policy* yapısındaki *pctAllocated* alanı ile karşılaştırılması, kurala servis verilip verilmeyeceğini belirler. Servis alma oranı, kendisine ayrılan orana eşit veya bu orandan büyük olan kurallar servis alabilmek için beklemek zorundadır. Bu özelliği ile sistem toleranssız bir servis kalitesi sunmaktadır. Öncelik yapıları arasında servis verilecek kuralın bulunması, *findPolicyTBS* fonksiyonu tarafından gerçekleştirilir.

Öncelik değerinde servis verecek kural ve kurala bağlı bekleyen paket bulunamaması durumunda, sistem servis verilen öncelik değerini belirleyen *priorityS witch* değerini bir sonraki önceliğe eşitler. Bir sonraki adımda, mevcut öncelik değerinden 1 sonraki öncelik değeri servis alır.

Servis almasına karar verilen kuralın, servis alacak elemanı bekleyen paketler listesinin ilk elemanı olan *firstQE* alanının gösterdiği yapıdır. *firstQE* işaretçisinin gösterdiği yapıya servis verilip, bu yapıya servis verme sonrasında ihtiyaç duyulmayacağından, servis vermeden önce, kurala ait *firstQE* değerini değiştirmek gerekmektedir. Paket servis aldıktan sonra *firstQE* değeri listedeki ikinci değeri göstermelidir. Böylece bekleyen paketler listesi FIFO mekanizmasını sağlamış olur.

queueElement veri yapısı, orijinal IP işleyici fonksiyonunu çağırabilmek için gereken parametreler olan *skb*, *dev* ve *pt'yi* içermektedir. *firstQE* elemanına servis vermek için, *skb*, *dev* ve *pt* alanlarını, *ip_rcv* fonksiyonuna parametre olarak verilir ve *ip_rcv* fonksiyonu çağrılır. Pakete servis verdikten sonra, kurala ait toplam servis verilen paket sayısını gösteren değer *totalServed* alanının değeri 1 artırılır.

5.5 Senkronizasyon ve Veri Yapılarının Korunması

Sistemin bekleyen paketlere sürekli servis verebilmesi için, sınıflandırma ve şekillendirme işleminin paralel yürümesi gerekmektedir. Sınıflandırma işleminin sisteme gelen her pakete uygulanması gerektiğinden, bu işlem sisteme yeni paket gelmesi anında yapılmalıdır. Şekillendirme işlemi ise sürekli olmalı ve gerekli yapıları sürekli kontrol etmelidir. Bekleyen paketlere servis vermek için, sisteme yeni paket gelmesi beklenmemelidir. Bunun yanında sistemdeki veri yapıları üzerinde yapılan işlemler b irbirini e tkilememelidir. S istem d evrede iken değeri değişen bir değişkenin, sistemin başka kollarında okunmasını koruma altına almak gerekmektedir.

Sınıflandırma ve şekillendirme adımları arasında paralelliğin ve veri yapılarının korunmasının sağlanması için iki ayrı mekanizmadan yararlanılmıştır. Bunlardan biri veri yapılarını korumak için kullanılan semaphore yapısı, diğeri de iki adımın paralel çalışabilmesi için kullanılan kernel thread kullanımıdır.

5.5.1 Kullanılan Semaphorelar

Sistemin çalışması esnasında üzerinde değişiklik yapılacak ve okunacak yapılar şunlardır.

- *shape* yapısı: Şekillendirme işleminde, öncelikler arasında geçiş ve servis verme için kullanılan *priorityS witch* ve *priorityCounter* değerleri ve sisteme gelen paketlerin sınıflandırılması işleminde paket sayıları sürekli güncellenecektir.

- *priorityBase* dizisi: Şekillendirme işleminde her servis verme işleminden sonra, servis verilen önceliğe ait *lastServedPol* değeri güncellenecektir. Aynı değer bir öncelikte servis verilecek kuralı bulma işleminden önce okunacaktır.

- *policy* yapıları: Şekillendirme işleminde bir kurala ait *firstQE* ve *lastQE* elemanları değişebilir. Bu değerler, sınıflandırma işleminde gelen paketi bekleyen paketler listesine eklerken okunur.

Bu 3 yapının okuma ve yazma işlemlerinde korunabilmesi için, 3 ayrı semaphore kullanılmıştır. Bu semaphorelar, yapıların okunduğu ve değiştirildiği noktalarda kullanılmıştır.

5.5.2 Kernel Thread Yapısı

Şekillendirme işleminin sürekli gerçekleştirilmesi için, bu işlemi bir fonksiyon olarak çağrılması yerine, sürekli çalışan bir thread olmasına karar verilmiştir. Şekillendirme işlemini yapan fonksiyon *shaper*'ı kullanarak 5 ayrı kernel thread oluşturulmuştur. Bu threadler, periyodik olarak devreye girip *shape* yapısına bağlı kalarak bekleyen paketlere servis verirler.

Threadler, kernel modülünün devreye girmesi anında oluşturulur. Her thread *shaper* fonksiyonunu kullanarak oluşturulur. *shaper* fonksiyonu, *findPolicyTBS* ve *serveQE* fonksiyonlarını kullanarak servis vereceği kuralı bulur ve bekleyen ilk pakete servis verir. Her thread oluşturulduktan ve bir pakete servis verdikten sonra, kernel tarafından belirlenen HZ değerinin 1000'de biri kadar sürede tekrar çağrılır. HZ değeri 1 saniyeyi ifade ettiğinden, her thread 1 milisaniyede bir uyanır, servis verme işlemini gerçekleştirir ve tekrar beklemeye başlar.

6. KULLANILAN FONKSİYONLAR

Sistemde gerçeklenen fonksiyonlar, kullanıldıkları aşamalara göre aşağıda sınıflandırılmıştır. Yardımcı fonksiyon olarak kullanılan fonksiyonlar ise ayrı bir başlık altında değerlendirilmiştir.

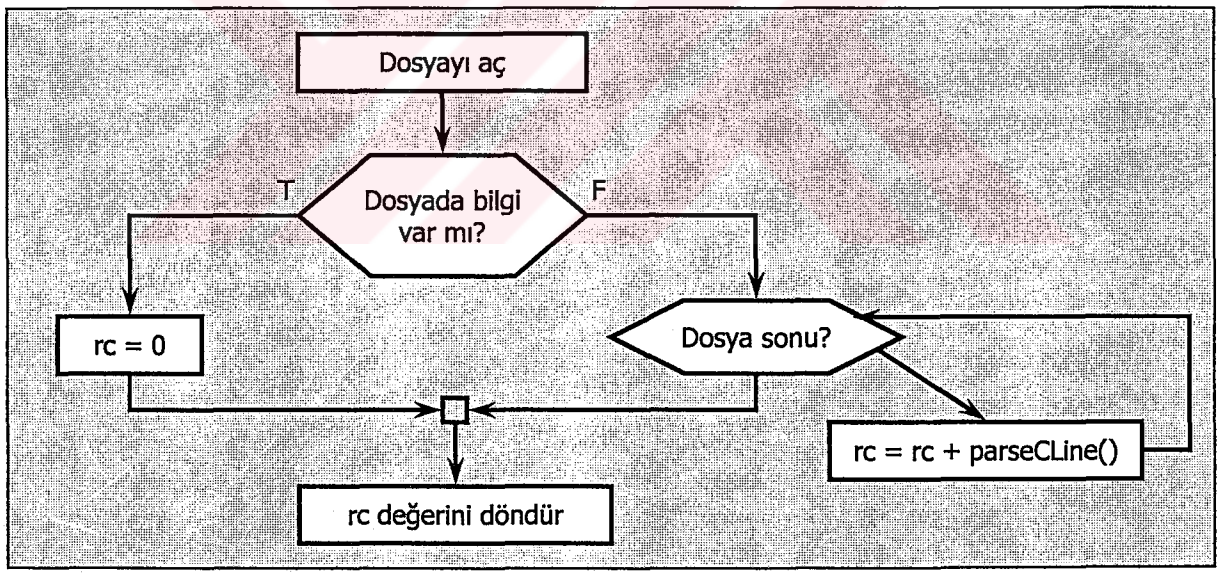
6.1 Sistemin Başlaması İşlemi İçin Kullanılan Fonksiyonlar

Sistemin başlaması aşamasında kullanılan fonksiyonlar aşağıda listelenmiştir.

6.1.1 parseConfFile Fonksiyonu

Ana konfigürasyon dosyasının okunup parçalara ayrılması içi kullanılan fonksiyondur. Yüklenebilir modüle ait *init_module* fonksiyonu tarafından çağrılır. Dosyanın açılıp her satırdaki bilginin okunması ve kontrolünden sorumludur. Fonksiyona ait prototip aşağıdadır. Fonksiyonun akışı Şekil 6.1'de verilmiştir.

int parseConfFile(void);



Şekil 6.1 *parseConfFile* fonksiyonuna ait akış diyagramı

Okunacak dosyayı açmak için kullanılan standart C kütüphanesinden bulunan fonksiyonlar Linux kernelinde kullanılamadığından, dosya işlemleri için kernelde kullanılan file veri yapısı ve *filp_open* fonksiyonu kullanılmıştır. Dosya açıldıktan sonra dosyanın boyutu kontrol edilir. Dosyanın boş olmaması durumunda dosyadaki her satır okunur ve parçalara ayrılır. Satırların parçalara ayrılması işlemi *parseCLine* fonksiyonu tarafından gerçekleştirilir.

6.1.2 parseCLine Fonksiyonu

parseConfFile fonksiyonu tarafından çağrılır ve parametre olarak verilen, ana konfigürasyon dosyasına ait satırlardaki bilgileri parçalara ayırmak için kullanılır. Satırlar ana konfigürasyon değerinde kullanılabilecek direktifler olan

- policyconf
- netsconf
- services
- prioritylevel

değerleri ile karşılaştırılır. Geçerli satırlardaki direktifler değerlendirilir ve *shape* yapısının uygun alanına değer atanır. Okunan satırdaki direktifin ve direktife ait değerin geçerli olması durumunda 1, direktif veya değerin geçerli olmaması durumunda 0 değeri döndürülür. Fonksiyonun prototipi aşağıda verilmiştir.

*int parseCLine(char *);*

6.1.3 parseNetsFile Fonksiyonu

Sistemin başlaması esnasında sistemin bağlı olduğu ağların tanımlandığı dosyanın okunup parçalara ayrılması görevini yerine getirir. Ayrıştırılacak dosyanın adı *parseConfFile* fonksiyonunun çalışması sonrasında *shape* yapısının *ncnf* alanında tutulmaktadır. Çalışma prensibi *parseConfFile* fonksiyonu ile aynıdır. Dosyanın açılmasından sonra dosyadan okunan her satır değerlendirilir. Satırdaki bilginin geçerli bir ağ bilgisine ait olduğu durumda bu ağ bilgisi, *shaper* veri yapısına ait *netBase* alanının gösterdiği linkli listenin sonuna eklenir. Daha sonra okunan ağa ait *net* veri yapısındaki *hostArray* dizisinin elemanlarına NULL değeri atanarak o ağa ait hostlar sıfırlanır. Fonksiyona ait prototip aşağıdadır.

int parseNetsFile(void);

6.1.4 parseNLine Fonksiyonu

parseNetsFile fonksiyonu tarafından çağrılır. Ağ konfigürasyon dosyasından okunan satırların parçaralara ayrılması işlemini yerine getirir. Satırda belirtilen ağ adresi ve ağ maskesi bilgilerinin geçerli olması halinde bu bilgiler oluşturulan *net* veri yapısındaki bir değişkende saklanarak *parseNetsFile* fonksiyonuna döndürülür. Bilginin geçersiz olması durumunda

NULL değerini döndürür. Fonksiyona ait prototip aşağıdadır.

*struct net *parseNLine(char *);*

6.1.5 parseServicesFile Fonksiyonu

Kullanıcı seviyesinde uygulama geliştirildiğinde, ağ servislerine ait bilgiye erişim C kütüphanelerindeki fonksiyonlar sayesinde gerçekleştirilebilmektedir. Kernelde uygulama geliştirildiğinde bu fonksiyonları kullanma imkanı yoktur. Bu yüzden ağ servislerine ait fonksiyonların tamamı tekrar yazılmıştır. Servis bilgilerine erişebilmek için de servislerin tanımlandığı dosyanın parçalara ayrılması gerekmektedir. *parseServicesFile* fonksiyonu servis tanımlarının yapıldığı dosyanın okunmasını ve değerlendirilmesi işlemini gerçekleştirmektedir. Bu fonksiyonun çalışması da diğer parçalara ayırma fonksiyonları ile aynıdır. Dosyadaki her satır *parseLine* fonksiyonuna parametre olarak gönderilir. Satırdaki bilginin geçerli olması durumunda *serviceEntry* veri yapısında saklanan yeni servis bilgisi, *shape* veri yapısındaki *serviceBase* işaretçisinin gösterdiği linkli listeye eklenir. Fonksiyona ait prototip aşağıdadır.

void parseServicesFile(void);

6.1.6 parseLine Fonksiyonu

parseServicesFile fonksiyonu tarafından çağrılır. C kütüphanesinde servis dosyasını parçalara ayırmak için kullanılan fonksiyonun, kernel uygulamalarında kullanılmak üzere değiştirilmiş halidir. Servisin geçerli olabilmesi için gereken isim, port ve protokol bilgilerinin tamamının geçerli olması durumunda o servise ait *serviceEntry* yapısındaki değeri döndürür. Aksi halde NULL değerini döndürür. Fonksiyona ait prototip aşağıdadır.

*struct serviceEntry *parseLine(char *);*

6.1.7 parsePolicyFile Fonksiyonu

Kural dosyasındaki bilgilerin okunup, geçerliliğinin kontrol edilmesi ve geçerli ise uygun veri yapısına eklenmesi işlemini gerçekleştirir. Dosya açıldıktan sonra her satır *parsePLine* fonksiyonuna gönderilir. *parsePLine* fonksiyonu geçerli bir satır üzerinde işlem yapmış ve *policy* veri yapısına ait bir işaretçi gönderdiyse bu işaretçi uygun veri yapısına yerleştirilir.

parsePLine fonksiyonunun geçerli bir kuralla karşılaştığına dair gönderdiği değer üzerinde yapılması gereken kontroller ve işlemler aşağıdadır.

- Kuralda kaynak adres tanımı yapılmamışsa, kural başlangıcı *shape* yapısında yer alan *policyBase* alanıyla gösterilen linkli listeye eklenir.

- Kurala ait bir kaynak adres tanımı yapılmışsa, bu adresin sistemin bağlı olduğu ağlar içinde yer alıp almadığı kontrol edilir. Bu işlem *findInNets* fonksiyonu kullanılarak yapılır. Eğer kaynak adres bağlı ağlardan birinde bulunamazsa okunan kural bilgisi sisteme kaydedilmez.

- Sisteme bağlı ağlarda olduğu doğrulanan kaynak adresin host mu ağ mı olduğu kontrol edilir.

- Kural bir hosta aitse, kuralın o hosta ait veri yapısına yerleştirilmesi için *putInHost* fonksiyonu çağrılır.

- Kural bir ağa ait ise, *putInNetPolicy* fonksiyonu kullanılarak o ağa ait veri yapısındaki *policy* alanıyla gösterilen linkli listesine eklenir.

- Kurala ait *polArrayElement* dizi elamanı yapısı oluşturulur. Bu yapı, kuralın sahip olduğu öncelik değerine ait *priorityElement* yapısındaki dizi elemanının *policyArray* alanı ile gösterilen linkli listeye eklenir. Önceliğe ait yapının *policyCount* alanının değeri artırılır.

- shape yapısındaki *policyCount* değeri artırılır.

Dosyadaki bütün satırlar okunduktan sonra, şekillendirme aşamasında yanlış işaretçi değerleri oluşmasını engellemek için, *shape* yapısındaki öncelik dizisi elamanlarının *lastServedPol* ve *policyAray* elemanlarına ilk değerleri atanır.

Fonksiyonun prototipi aşağıdadır. Fonksiyona ait akış Şekil 6.2'de gösterilmiştir.

int parsePolicyFile(void);

6.1.8 parsePLine Fonksiyonu

parsePolicyFile fonksiyonu tarafından çağrılır ve kural konfigürasyon dosyasındaki satırları parçalara ayırmakla görevlidir. Parametre olarak aldığı satırdaki bilgilerin geçerli olması halinde, bu bilgileri kullanarak *policy* veri yapısında bir değişken oluşturur ve bu değişkene ait işaretçi değerini döndürür. Kuralı oluşturan bilgilerden herhangi birinin geçersiz olması durumunda NULL değerini döndürür. Fonksiyonun prototipi aşağıda verilmiştir.

struct policy *parsePLine(char *);

6.1.9 getServiceByName Fonksiyonu

Kural konfigürasyon dosyasının okunması sırasında, kurala ait port kriterinin port numarası yerine, ağ servisi adı verilerek tanımlanması durumunda çağrılır. Parametre olarak verilen ağ servis adına ait *serviceEntry* (Şekil 7.8) yapısını döndürür. Servis bilgileri *shape* yapısındaki *serviceBase* işaretçisinden başlanarak kontrol edilir. Fonksiyonun prototipi aşağıdaki gibidir.

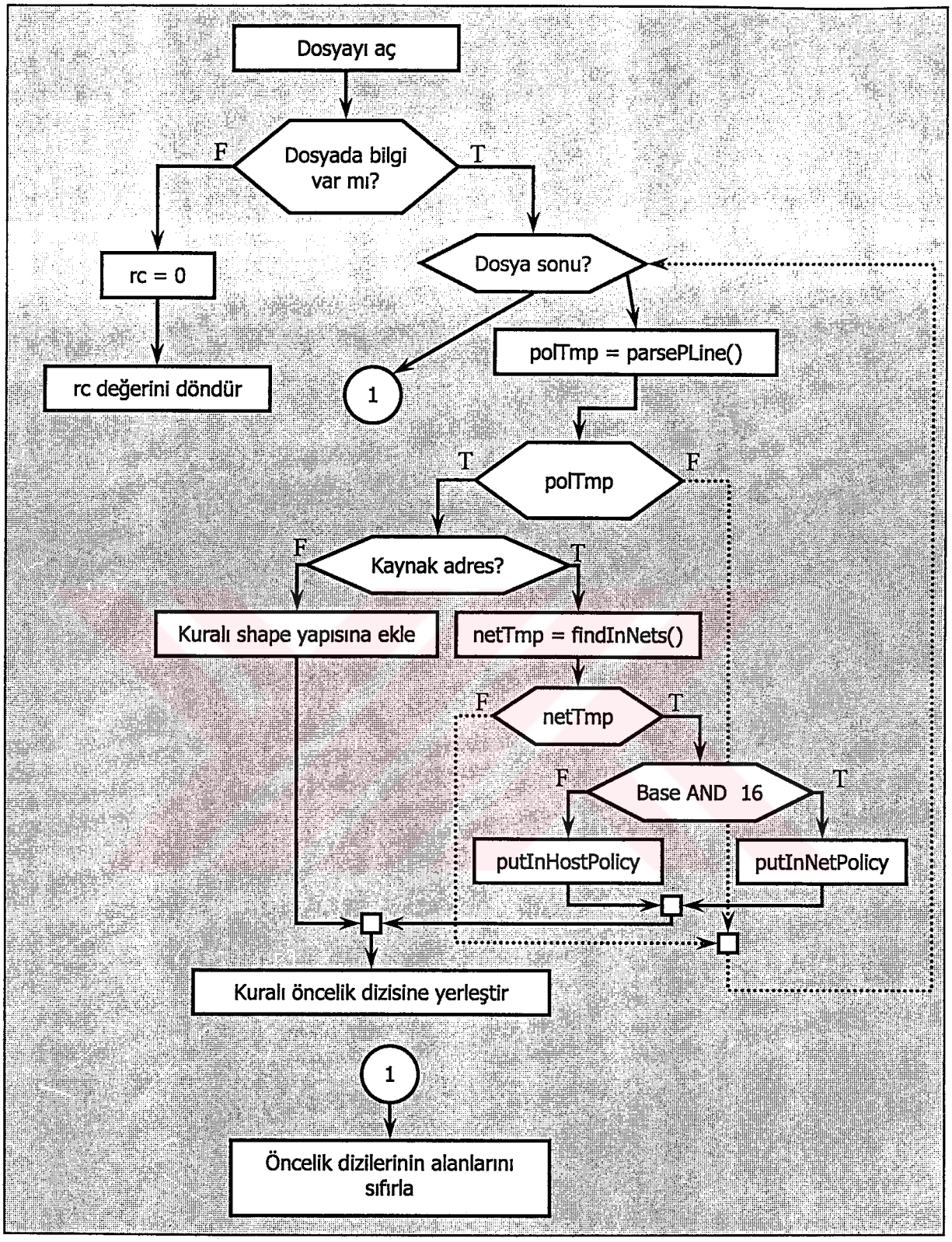
*struct serviceEntry *getServiceByName(char *);*

6.1.10 findInterfaces Fonksiyonu

findInterfaces fonksiyonu sistemde tanımlı olan ağ arayüzü bilgilerinin kernelden okunması için kullanılmaktadır. Sistemin çalışması süresince bir kez ve *init_module* fonksiyonu tarafından çağrılır. Fonksiyonun prototipi aşağıdaki gibidir.

void findInterfaces(void);

Ağ arayüzü bilgileri kerneldeki ağ cihazları bilgileri kontrol edilerek bulunur. Sistemde en azından loopback arayüzü tanımlı olduğundan ve bu arayüze ait bir adres bulunduğundan, arayüz bilgisi bulunamaması veya yanlış bilgi elde edilmesi durumları gözardı edilmiştir.



Şekil 6.2 *parsePolicyFile* fonksiyonuna ait akış diyagramı

6.1.11 findInNets Fonksiyonu

Parametre olarak verilen *policy* yapısında tanımlanan kuraldaki kaynak IP adresinin, bir ağ

bilgisi ise, sisteme bağlı olan ağlardan biri ile aynı olup olmadığını; eğer bir host bilgisi ise sisteme bağlı olan ağlardan birinde yer alıp almadığını verir.

Bir hostun tanımlı ağlardan birinde olup olmadığının anlaşılabilmesi için, host adresiyle karşılaştırılan ağa ait ağ maskesi değerinin VE işlemine sokulması gerekir. Bu işlemin sonucu, ağa ait başlangıç adresi olmalıdır. Örneğin 193.140.4.59 IP adresine sahip hostun, 193.140.4.0 adresi ve 255.255.255.0 ağ maskesine sahip ağ içinde yer alıp almadığının kontrolü için;

Host adresi	11000001 10001100 00000100 00111011 (193.140.4.59)
Ağ maskesinin tersi	11111111 11111111 11111111 00000000 (255.255.255.0)
VE sonucu	11000001 10001100 00000100 00000000 (193.140.4.0)

işlemi yapılmaktadır.

Kaynak adres olarak bir ağ adresi ve ağ maskesi verilmişse, bu ağa ait başlangıç adresinin ve ağ maskesinin, sistemde tanımlı olan ağlardan biri ile aynı olması gerekmektedir.

Eğer kontrol sonucu sistemdeki ağlardan biri bulunmuşsa, bu ağ bilgisine ait *net* yapısına ait işaretçi dönüş değeri olarak kullanılır. Fonksiyona ait prototip aşağıdadır.

```
struct net *findInNets(struct policy *);
```

6.1.12 putInHostPolicy Fonksiyonu

Ayrıştırılan satırdaki kural bilgisi bir hosta ait ise, bu kuralı, hosta ait *hostArrayEntry* tipindeki yapının policy alanındaki adres ile başlayan linkli listeye ekler. Hostun ait olduğu ağ bilgisi, *parsePolicyFile* fonksiyonunda *findInNets* fonksiyonun çağrılması sonucu bulunmuştur. Yapılması gereken işlem, kuralın yerleştirileceği host bilgisinin, *net* yapısındaki *hostArray* dizisinde hangi gözde bulunduğunu hesaplamaktır.

Hostun ağ içindeki, yeri host adresi ve ağ maskesinin değerinin tersinin VE işlemine sokulması sonucu hesaplanır. Örneğin 193.140.4.59 IP adresine sahip hostun, 193.140.4.0 adresi ve 255.255.255.0 ağ maskesine sahip ağ içindeki yerini hesaplamak için;

Host adresi	11000001 10001100 00000100 00111011 (193.140.4.59)
Ağ maskesinin tersi	00000000 00000000 00000000 11111111 (0.0.0.255)
VE sonucu	00000000 00000000 00000000 00111011 (0.0.0.59)

işlemi gerçekleştirilir.

Fonksiyona ait prototip aşağıdadır.

*void putInHostPolicy(struct net *, struct policy *);*

6.1.13 putInNetPolicy Fonksiyonu

Kural dosyasında değerlendirilen satırdaki bilgi içinde yer alan kaynak adres bilgisi bir ağa aitse, kuralın ağa ait *net* yapısına eklenmesi işlemini yerine getirir. Okunan kural bilgisini *parsePolicyFile* fonksiyonu tarafından çağrılan *findInNets* fonksiyonunun döndürdüğü *net* yapısının *policy* alanı ile gösterilen linkli listeye ekler. Fonksiyona ait prototip aşağıdadır.

void putInNetPolicy(struct net*, struct policy *);

6.2 Sınıflandırma İşlemine Ait Fonksiyonlar

Sınıflandırma aşamasında kullanılan fonksiyonlar ve fonksiyonlara ait bilgi aşağıda listelenmiştir.

6.2.1 ifLocalAddress Fonksiyonu

Kendisine parametre olarak verilen IP adresinin, sistemin yerel ağ arayüzlerindeki adresler ile karşılaştırmasını yapar. *shape* yapısındaki *interfaces* alanındaki adres bilgilerini, parametre olarak verilen değer ile karşılaştırır. Aynı olan 2 adres bulunması durumunda 1, aynı 2 adres bulunamaması durumuna 0 değerini döndürür. Fonksiyona ait prototip aşağıdadır.

int ifLocalAddress(u32);

6.2.2 ifInLAN Fonksiyonu

Parametre olarak verilen IP adresinin, sisteme bağlı ağlardan birinde olup olmadığını kontrol eder. Adres ve ağ kontrolü *findInNets* fonksiyonundaki karşılaştırmanın aynısıdır. Fonksiyona ait prototip aşağıdadır.

int ifInLAN(u32);

6.2.3 findPolicy Fonksiyonu

Parametre olarak verilen *sk_buff* veri yapısının, tanımlı kurallardan birine uyup uymadığını kontrol eder. Paketin kurallardan biri ile uyması halinde o kurala ait işaretçiyi, aksi halde

NULL değerini döndürür.

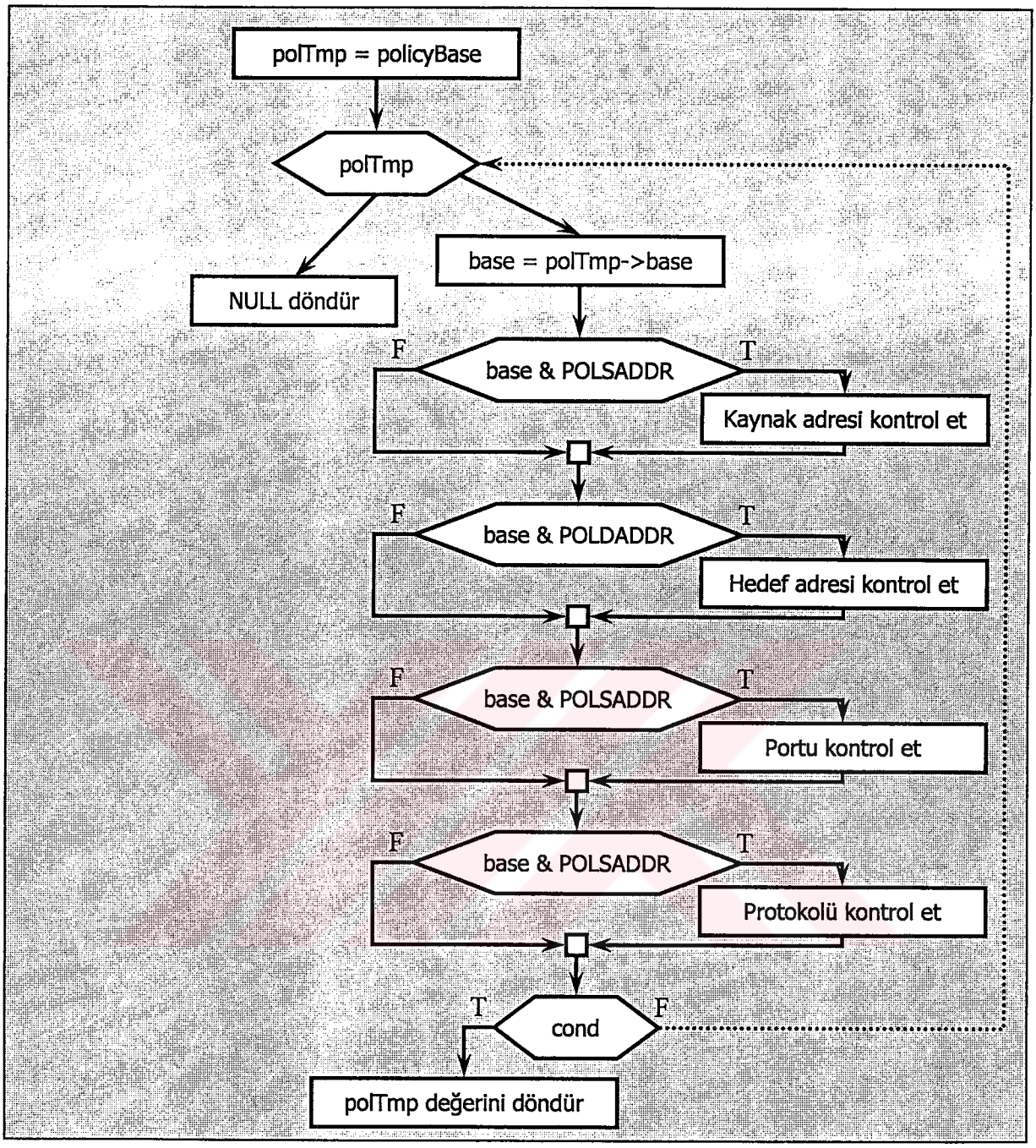
hostArrayEntry, *net* ve *shape* yapılarında saklanılan kuralların kontrolü, belirtilen sırada yapılmaktadır. Bunun amacı, kural bilgisine sıralı erişmek yerine daha hızlı erişimi sağlamaktır. Her yapıdaki *polciyArray* tipindeki dizi içinde kural kontrolü yapılır. Dizi üzerindeki kural kontrolünü yapmak için *findPolicyInList* fonksiyonu kullanılmaktadır. Fonksiyona ait prototip aşağıdadır.

*struct policy *findPolicy(struct sk_buff *);*

6.2.4 findPolicyInList Fonksiyonu

Paket bilgisini, *hostArrayEntry*, net ve shape yapılarındaki kural listesi elemanları ile karşılaştırmak için kullanılır. Paket bilgisinin kuralla karşılaştırılmasında her alan tek tek karşılaştırılmaz. Bunun yerine kurala ait *policy* tipindeki yapıda *base* alanı kullanılarak, kuralda hangi kriterlerin tanımlandığı kontrol edilir. Bu bilgiye göre, kuralın bütün kriterlerinin kontrolü yerine sadece tanımlı ve karşılaştırılması gereken kriterler ve paket bilgileri kullanılır. Kuralların kontrolüne, verilen parametreyi başlangıç adresi alarak başlanır. Fonksiyonun akışı Şekil 6.3'de, prototipi aşağıda verilmiştir.

*struct policy *findPolicyInList(struct policy *, struct sk_buff *);*

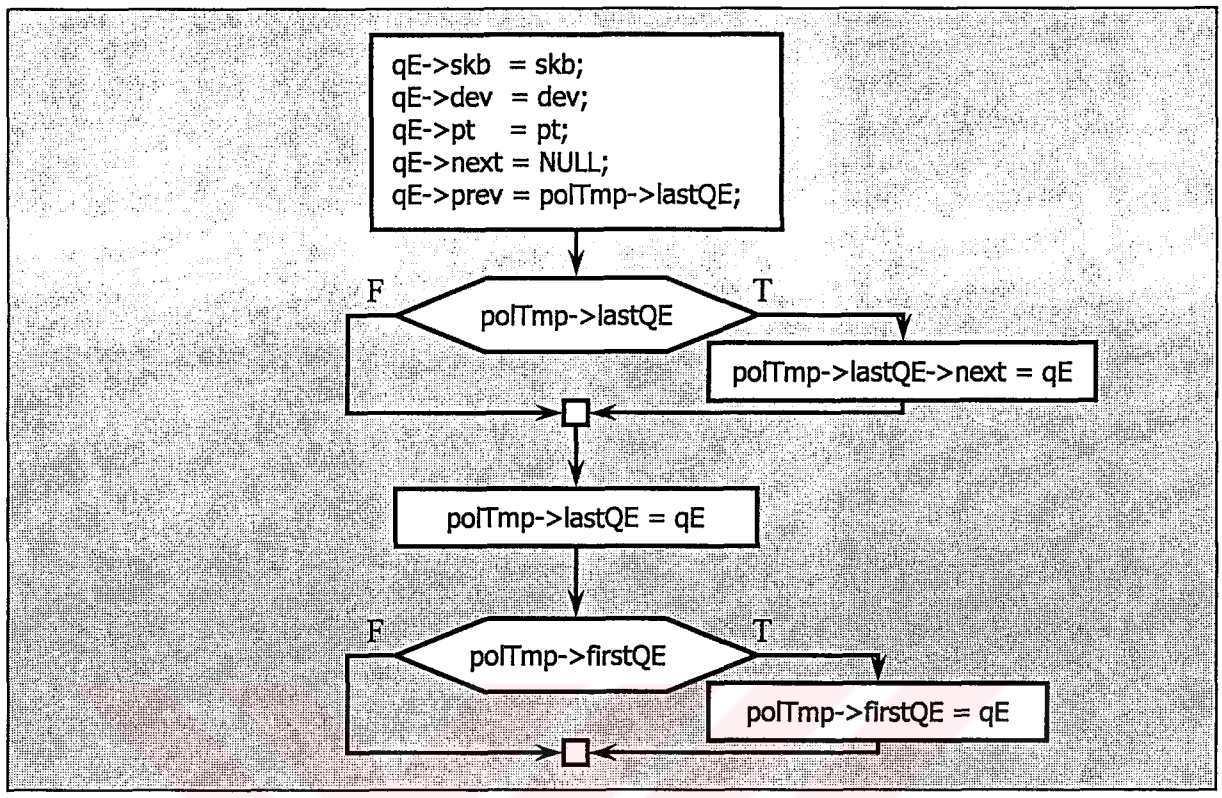


Şekil 6.3 *findPolicyInList* fonksiyonuna ait akış diyagramı

6.2.5 insertQE Fonksiyonu

Bir kurala göre sınıflandırılan pakete ait bilgileri, *queueElement* yapısına yerleştirir ve bu yapıyı kurala ait bekleyen paketler listesine ekler. Yeni elemanın diziye yerleştirilme işleminde kural yapısının *firstQE* ve *lastQE* elemanları kontrol edilmek zorundadır. Fonksiyonun akışı Şekil 6.4'te gösterilmiştir. Fonksiyonun prototipi aşağıdaki gibidir.

*void insertQE(struct policy *, struct sk_buff *,struct net_device *, struct packet_type *);*



Şekil 6.4 *insertQE* fonksiyonuna ait akış diyagramı

6.3 Şekillendirme İşlemine Ait Fonksiyonlar

Şekillendirme işlemine ait fonksiyonların açıklamaları aşağıdadır.

6.3.1 findPolicyTBS Fonksiyonu

Servis verme aşamasında, öncelik yapısı içinde hangi kurala ait bekleyen pakete servis verileceğini belirler. Kurallar kendi aralarında Round Robin servis verme algoritmasına göre servis alırlar. Servis verilecek kuralı ve paketi belirlemek için, öncelik yapısına ait kurallar üzerinde sıra ile kontrol yapılır. Bir kurala servis verilebilmesi için aşağıdaki iki koşulun gerçekleştiğinin doğrulanması gerekir.

- Kurala ait bekleyen paket listesinde en az 1 paket bulunması,
- Kurala ait mevcut servis alma oranının, kuralda tanımı yapılan istenilen servis alma oranından küçük olması.

Kurallar arasında bu iki şartı sağlayan kuralı bulana veya bu şartları sağlayan kural

bulunamayıp bütün kurallar kontrol edilene kadar kontrole devam edilir.

6.3.2 serveQE Fonksiyonu

findPolicyTBS fonksiyonunun servis verilecek bir kural bulması durumunda, servis verilmesi gereken paket, bulunan kurala ait *firstQE* alanının gösterdiği yapıdır. *serveQE* fonksiyonu, bu yapıdaki *skb*, *dev* ve *pt* alanlarını kullanarak orijinal IP işleyici fonksiyonu olan *ip_rcv* fonksiyonunu çağırır. Böylece kurala ait *firstQE* elemanı servis almış olur. Servis alan pakete ait yapı sistemden uzaklaştırılır. Fonksiyona ait prototip aşağıdadır.

*int serveQE(struct policy *);*

6.4 Değer İzleme Fonksiyonları

Kullanıcı seviyesinde program geliştirirken, değer izlemek için standart çıkış fonksiyonları kullanılabilmektedir. Kernel seviyesinde program geliştirirken değerleri izleme işlemi için sadece *printk* fonksiyonu bulunmaktadır. *printk* fonksiyonu verilen mesajları standart çıkışa değil, syslog servisi tarafından belirlenen dosyalara yazmaktadır. Aşağıda açıklanan fonksiyonlar, *printk* fonksiyonunu kullanarak sistemin çalışması anında değerleri izlemek için kullanılmaktadır.

6.4.1 printPacketCount Fonksiyonu

Sisteme gelen toplam, sınıflandırılabilen ve sınıflandırılamayan paket sayılarını yazmak için kullanılan fonksiyondur. Fonksiyonun prototipi aşağıdadır.

void printPacketCount(void);

6.4.2 printFiles Fonksiyonu

Ana konfigürasyon dosyasından okunan, ağ, kural ve servis konfigürasyon dosyalarının, *shape* yapısına doğru aktarılıp aktarılmadığını kontrol etmek için kullanılmıştır. Fonksiyonun prototipi aşağıdadır.

void printFiles(void);

6.4.3 printPolicy Fonksiyonu

İstenilen kurala ait kriterleri yazdırmak ve kontrol etmek için kullanılır. Fonksiyonun prototipi aşağıdadır.

*void printPolicy(struct policy *);*

6.4.4 printAllPolicies Fonksiyonu

Öncelik değerlerine ait dizi üzerinde, tüm kurallara ait toplam kabul edilen ve toplam servis verilen paket sayısını yazdırır. *printPolStat* fonksiyonunu çağırarak kurala ait paket sayılarının, sistemin tamamına gelen paket sayılarıyla olan oranını yazdırır. Kural tanımında, istenilen oranın sağlanıp sağlanmadığının kontrolü için kullanılır. Fonksiyonun prototipi aşağıdadır.

void printAllPolicies(void);

6.4.5 printPolStat Fonksiyonu

Parametre olarak verilen kurala ait istatistik değerlerini yazdırır. Pakete ait toplam servis verilen paket sayısının, sisteme gelen toplam paket sayısına oranını verir. Fonksiyonun prototipi aşağıdadır.

*void printPolStat(struct policy *);*

6.4.6 printNet Fonksiyonu

Parametre olarak verilen ağ bilgisinin içeriğini yazdırır. Fonksiyonun prototipi aşağıdadır.

*void printNet(struct net *);*

6.4.7 printSKB Fonksiyonu

Parametre olarak verilen sokete ait TCP/IP bilgisini yazdırmak için kullanılır. Sisteme gelen veya servis verilmek üzere olan paketleri kontrol etmek amacıyla oluşturulmuştur. Fonksiyona ait prototip aşağıdadır.

*void printSKB(struct sk_buff *, int);*

6.4.8 printInterfaces Fonksiyonu

Sistemde tanımlı olan ağ arayüzlerine ait bilgiyi, shape yapısının *interfaces* alanını kullanarak yazdırmak için kullanılır. Fonksiyona ait prototip aşağıdadır.

void printInterfaces(void);

6.4.9 printPolicyList Fonksiyonu

Başlangıç noktası verilen kural linkli listesini, *printPolicy* fonksiyonunu kullanarak yazdırır. Bir hosta veya ağa ait kural listesinin kontrolü için kullanılır. Fonksiyona ait prototip aşağıdadır.

*void printPolicyList(struct policy *);*

6.5 Dosyaların Ayrıştırılmasında Kullanılan Yardımcı String Fonksiyonları

Konfigürasyon dosyalarının parçalara ayrılmasında ihtiyaç duyulan string fonksiyonları, kernel seviyesinde kullanılamadığı için, bu fonksiyonlar tekrar gerçeklenmiştir. Yeniden gerçeklenen fonksiyonlara ait prototipler, dönüş değerleri ve kullanımları, standart C kütüphanelerinde aynı amaç için kullanılan fonksiyonlar ile aynıdır. Bu fonksiyonları isimleri ve görevleri Çizelge 6.1'te belirtilmiştir.

Çizelge 6.1 Kullanılan yardımcı fonksiyonlar ve görevleri

int isSpace(char)	Karakterin boşluk karakteri olup olmadığını döndürür.
int isDigit(char)	Karakterin nümerik olup olmadığını döndürür.
int isAlpha(char)	Karakterin alfabetik olup olmadığını döndürür.
int isUpper(char)	Karakterin büyük harf olup olmadığını döndürür.
void strNcpy(char *,char *,int)	Bir stringden diğerine verilen sayı kadar karakteri kopyalar.
int strLen(char *)	Parametre olarak verilen stringin karakter cinsinden uzunluğunu verir.
int strCmp(char *, char *)	Verilen iki stringi karşılaştırır.

6.6 Kullanılan Kernel Fonksiyonları

Sistemin kullandığı, kernel'da tanımlı olan fonksiyonlara ait bilgi aşağıda belirtilmiştir.

6.6.1 ip_rcv Fonksiyonu

Sistemde tanımlanan orijinal IP işleyici fonksiyonudur. Aldığı parametreler pakete ait soket ve protokol bilgisini içeren *sk_buff* yapısı, cihazın geldiği cihaza ait işaretçi ve paketin tipine ait yapıdır. Şekillendirme aşamasında sırada bulunan paketlere servis vermek için kullanılmıştır. Fonksiyona ait prototip aşağıdadır.

*int ip_rcv(struct sk_buff *, struct net_device *, struct packet_type *);*

6.6.2 simple_strtol Fonksiyonu

Standart C kütüphanesinde kullanılan *strtol* fonksiyonun, kernelde kullanılmak üzere değiştirilmiş halidir. Parametre olarak verilen karakter dizisini sayısal değerine çevirir. Konfigürasyon dosyalarının parçalara ayrılmasında, karakter olarak yazılan değerlerin sayısal karşılıklarını elde etmek için kullanılır. Fonksiyona ait prototip aşağıdadır.

*int simple_strtol(char *, char **, int);*

6.7 Yüklenebilir Kernel Modülüne Ait Standart Fonksiyonlar

Sistem, kernele dinamik olarak yüklenme imkanı veren yüklenebilir kernel modülü olarak gerçeklendiğinden, bu modüllerde bulunması gereken 2 adet fonksiyonun gerçeklenme zorunluluğu vardır. Bu iki fonksiyona ait bilgiler aşağıdadır.

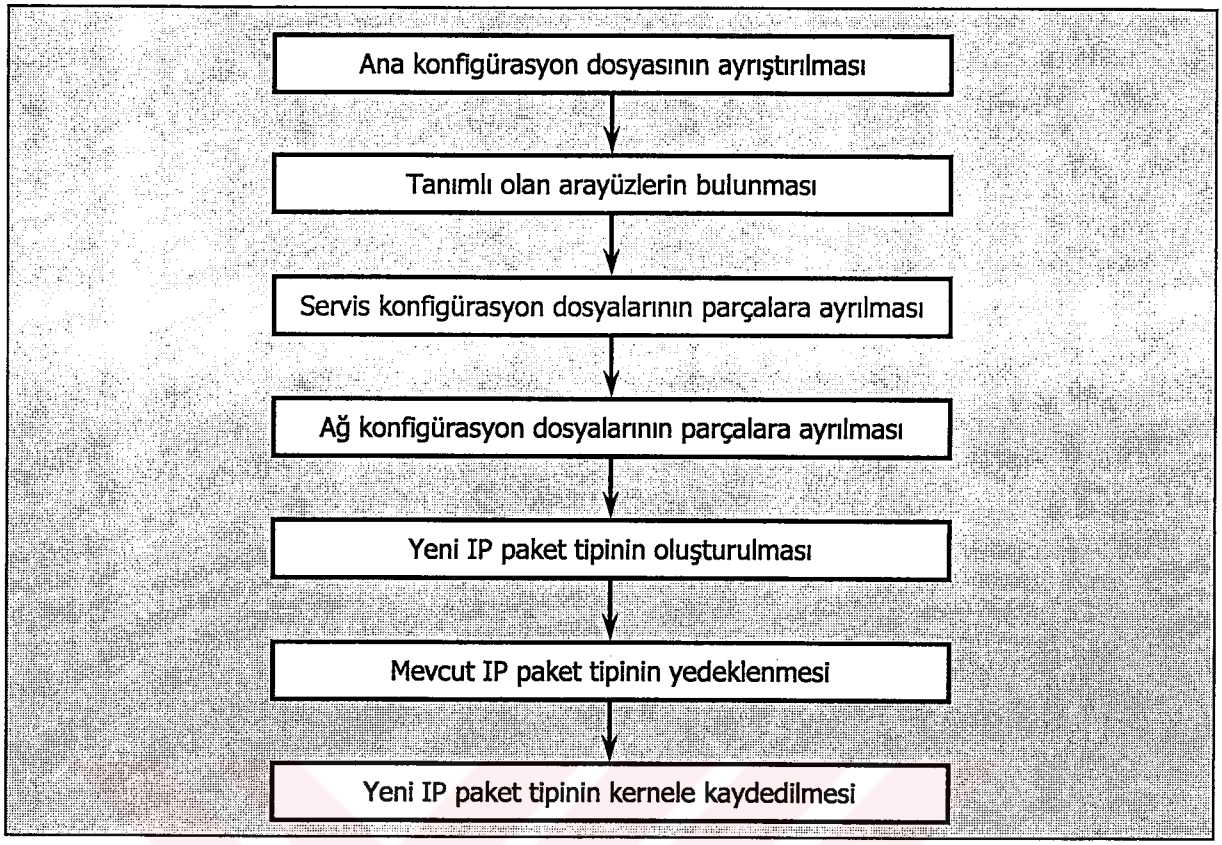
6.7.1 init_module Fonksiyonu

init_module fonksiyonu, sistemin çalışması için kullanılan modülün kernele yüklenmesi anında ilk çalışan fonksiyondur. Bu fonksiyonda modül devreye girmeden önce yapılması gereken işlemler gerçekleştirilir. Gerçeklenen sistemde *init_module* fonksiyonu, sistemin başlaması için gerekli olan fonksiyonların çağrılması için kullanılmıştır. Fonksiyonun prototipi aşağıdaki gibidir.

int init_module(void);

Fonksiyonun akışı Şekil 6.5'te verilmiştir.

init_module fonksiyonu LKM'nin çalışması gereğince, işlemlerin başarılı olması durumunda, modülün yüklenebilmesi için 0 değerini döndürmelidir. *init_module* sistemin başlaması için gerekli olan bütün fonksiyonları çağırır. Tüm fonksiyonları başarılı olması durumunda modülün yüklenmesi işlemini onaylayacak olan 0 değerini döndürür. Çağrılan fonksiyonlardan birinin hatalı değer döndürmesi, *init_module* fonksiyonunun 0'dan farklı bir değer döndürmesini sağlar ve modülün yüklenmesini engeller.



Şekil 6.5 *init_module* fonksiyonunun akış diyagramı

6.7.2 cleanup_module Fonksiyonu

Modülün devre dışı bırakılması esnasında çağrılan fonksiyondur. Kullanım amacı modülde oluşturulan veri yapılarının temizlenmesidir.

Gerçeklenen sistemde *cleanup_module* fonksiyonu, kurallara ait listelerde beklemekte olan paketlere servis verir. Bekleyen bütün paketler servis aldıktan sonra, yedeklenmiş olan orijinal IP paket yapısı devreye alınır ve sistem tarafından tanımlanmış olan yeni IP paketi tipi devreden çıkarılır. Böylece sistemin orijinal IP işlevselliği korunmuş olur. Fonksiyona ait prototip aşağıdadır.

void cleanup_module(void);

7. KULLANILAN VERİ YAPILARI

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

7.1 shape Veri Yapısı

Sistemde tanımlı olan bütün fonksiyonların erişebildiği yapı, sistemin tamamının kullanacağı değerleri içermelidir. Bu değerlerin atanması için yapılması gereken işlem mevcut konfigürasyon dosyasının okunup değerlendirilmesidir. Veri yapısına ait alanlar ve alanların anlamları Şekil 7.1'de gösterilmiştir.

7.2 priorityElement Veri Yapısı

priorityElement veri yapısı, şekillendirme işleminde servis verirken farklı öncelikler arasında geçiş yapabilmek amacı ile kullanılmaktadır. Yapısı ve içerdiği alanların anlamları Şekil 7.2'de belirtilmiştir.

```
struct shaper
{
  char *pcnf;                    // Kural konf. dosyasının adı ve yeri
  char *ncnf;                    // Ağ konf. dosyasının adı ve yeri
  char *scnf;                    // Servis konf. dosyasının adı ve yeri
  unsigned long totalReceived;   // Sisteme gelen toplam paket sayısı
  unsigned long totalLocal;      // Sistem kendisine veya yerel ağa gelen toplam paket
                                 sayısı
  unsigned long totalPolicy;     // Sınıflandırılabilen toplam paket sayısı
  unsigned long totalUndefined;  // Sınıflandırılamayan toplam paket sayısı
  unsigned long totalRejected;   // Sınıflandırılamayan toplam paket sayısı
  struct interface interfaces;   // İşletim sisteminde tanımlı ağ arayüzleri
  struct serviceEntry *serviceBase; // Tanımlı servislerin bilgisini tutan linkli liste
  struct net *netBase;           // Sisteme bağlı ağların bilgisini tutan linkli liste
  struct policy *policyBase;     // Kaynak adresi olmayan kuralların linkli listesi
  struct priorityElement *priorityBase; // Öncelik bilgilerini tutan diziye ait işaretçi
  int prioritySwitch;            // Öncelikler arasında geçiş için kullanılan sayaç
  int priorityCounter;           // Öncelik içinde servis vermek için kullanılan sayaç
  int policyCount;               // Toplam kural sayısı
  int maxPriority;               // En yüksek öncelik değeri
  unsigned long timeLocal;       // Paketin sisteme geldiğine karar verilme süresi
  unsigned long timePolicy;      // Paketin sınıflandırılabileceğine karar verme süresi
  unsigned long timeUndefined;   // Paketin sınıflandırılamayacağına karar verme süresi
  unsigned long timeWait;        // Toplam kuyrukta bekleme süresi
};
```

Şekil 7.1 *shaper* veri yapısı

```
struct priorityElement
{
  struct polArrayElement *policyArray;   // Önceliğe ait kurallar listesi
  struct polArrayElement *lastServedPol; // Önceliğe ait son servis verilen kurala ait işaretçi
  struct polArrayElement *lastPAE;       // Kurallar listesindeki son kurala ait işaretçi
  int policyCount;                       // Önceliğe ait toplam kural sayısı
};
```

Şekil 7.2 *priorityElement* veri yapısı

7.3 net Veri Yapısı

net veri yapısı, sistemin bağlı olduğu ağa ait bilgileri tutmak için kullanılır. Yapısı ve içerdiği alanların anlamları Şekil 7.3'de belirtilmiştir.

```
struct net
{
  u32 base;                          // Ağın başlangıç adresi
  u32 mask;                          // Ağa ait maske değeri
  struct hostEntryArray *hostArray[64]; // Ağa bağlı olan hostlara ait bilgilerin tutulduğu dizi
  struct policy *policy;             // Kaynak adresi olarak mevcut ağ verilen kural listesi
  struct net *next;                  // Bir sonraki ağ bilgisinin yerini gösteren işaretçi
};
```

Şekil 7.3 *net* veri yapısı

7.4 hostArrayEntry Veri Yapısı

hostArrayEntry veri yapısı, sistemin bağlı olduğu ağlar üzerindeki hostlarla ilgili kural bilgilerini tutmak için tasarlanmıştır. *hostArrayEntry* yapısındaki *next* alanı, *hostArray* dizisinin hashing ile çalışan ve birim boyutu 4 olan linkli listeyi sağlamak için kullanılır. Veri yapısı ve içerdiği alanların anlamları Şekil 7.4'de belirtilmiştir.

```
struct hostArrayEntry
{
  short address;                     // Hostun ağ içindeki yeri
  struct policy *policy;             // Kaynak adres olarak host verilen kural listesi
  struct hostArrayEntry *next;       // Aynı hash değerine sahip bir sonraki host bilgisinin
                                     yerini gösteren işaretçi
};
```

Şekil 7.4 *hostArrayEntry* veri yapısı

7.5 policy Veri Yapısı

policy veri yapısı, sistemde tanımlı olan kurallara ait bilgileri tutmak için kullanılır. *hostArrayEntry*, *net* ve *shape* yapılarındaki, *policy* ve *policyBase* alanları, *policy* tipindeki

linkli listede tutulan kural bilgilerini göstermektedir. Veri yapısı ve içerdiği alanların anlamları Şekil 7.5'te belirtilmiştir.

```
struct policy
{
  u8 base;                          // Kuralda hangi kriterlerin tanımlı olduğu
  u32 sAddr;                        // Kaynak adres bilgisi
  u32 sMask;                        //Kaynak adres ağ maskesi
  u32 dAddr;                        //Hedef adres bilgisi
  u32 dMask;                        //Hedef adres ağ maskesi
  u16 port;                         // Port numarası
  u8 prot;                          // Protokol değeri
  float pctAllocated;               // Kurala ayrılan bant genişliği yüzdesi
  u8 priority;                      // Kuralın öncelik değeri
  unsigned long totalReceived;      // Kurala uyan toplam paket sayısı
  unsigned long totalServed;        // Kurala ait servis verilen paket sayısı
  struct policy *next               // Bir sonraki kuralı gösteren işaretçi
  struct queueElement *firstQE;     // Kurala ait ilk paketin işaretçisi
  struct queueElement *lastQE;      // Kurala ait son paketin işaretçisi
};
```

Şekil 7.5 *policy* veri yapısı

7.6 polArrayEntry Veri Yapısı

net, *hostArrayEntry* ve *shaper* yapılarında linkli liste olarak tutulan kural listelerine, önceliğe göre sıralı servis verilebilmesi için, sıralı erişimi sağlamak amacı ile kullanılır. Veri yapısı ve içerdiği alanların anlamları Şekil 7.6'da belirtilmiştir.

```
struct polArrayElement
{
  struct policy *policy;            // polArrayElement'in gösterdiği policy değişkeni
  struct polArrayElement *next;     // Bir sonraki polArrayElement tipine ait işaretçi
};
```

Şekil 7.6 *hostArrayEntry* veri yapısı

7.7 interface Veri Yapısı

Sistemde tanımlı olan arayüzlere ait bilgiler, *interface* yapısındaki tek bir değişkende tutulur. Bu değişken shaper veri yapısının bir alanı olan *interfaces* alanıdır. Veri yapısı ve içerdiği alanların anlamları Şekil 7.7'de belirtilmiştir.

```
struct interface
{
  int ifCount;                    // Tanımlı toplam arayüz sayısı
  u32 *ifAddress;                 // Tanımlı arayüzlere ait adres dizisi
  u32 *ifMask;                    // Tanımlı arayüzlere ait ağ maskesi dizisi
};
```

Şekil 7.7 *interface* veri yapısı

7.8 serviceEntry Veri Yapısı

Sistemde tanımlı olan ağ servislerine ait bilgiler *serviceEntry* tipindeki bir linkli liste olan, *shaper* veri yapısının *serviceBase* işaretçisinde saklanır. Veri yapısı ve içerdiği alanların anlamları Şekil 7.8'de belirtilmiştir.

```
struct serviceEntry
{
  char *s_name;                   // Ağ servisinin adı
   int s_port;                    // Ağ servisinin kullandığı port
  char *s_proto;                  // Ağ servisinin protokolü
  struct serviceEntry *next;      // Bir sonraki ağ servisini gösteren işaretçi
};
```

Şekil 7.8 *serviceEntry* veri yapısı

7.9 packet_type Veri Yapısı

Linux kernelinde paket tiplerinin tanımlandığı veri yapısıdır. Kernel tarafından tanımlanmış olup veri yapısı ve içerdiği alanların anlamları Şekil 7.9'da belirtilmiştir.

```
struct packet_type
{
 unsgined short type;                    // Paket tipini belirler
 struct net_device *dev                  // Paket tipinin ilişkilendirileceği cihaza ait işaretçi
 int (*func) (struct sk_buff *, struct net_device *,  struct packet_type *);
                                         // Paket tipine ait işleyici fonksiyonun işaretçisi
 void data;                              // Paket tipinin kullanımına özel bilgi
 struct packet_type *next;               // Aynı tipteki bir sonraki paket tipini gösteren işaretçi
};
```

Şekil 7.9 *packet_type* veri yapısı

7.10 queueElement Veri Yapısı

Sisteme gelen ve sınıflandırılan paketlerin, servis almak için sıralanmasında kullanılan ve servis verilebilmesi için gereken değişkenlerin ve veri yapılarına ait işaretçilerin saklandığı yapıdır. Kural yapılarındaki *firstQE* ve *lastQE* alanları bu veri yapısına sahiptir. Veri yapısının alanları Şekil 7.10'da gösterilmiştir.

```
struct queueElement
{
 struct sk_buff *skb;                    // Pakete ait soket/protokol bilgilerinin saklandığı yapı
 struct net_device *dev                  // Paketin geldiği cihaza ait işaretçi
 struct packet_type *pkt;                // Gelen paketin tipi
 struct queueElement *nex;;              // Bir sonraki pakete ait bilgiyi gösteren işaretçi
 struct queueElement *prev;              // Bir önceki pakete ait bilgiyi gösteren işaretçi
};
```

Şekil 7.10 *queueElement* veri yapısı

8. KULLANILAN DOSYA YAPILARI

Sistemde kullanılan konfigürasyon dosyalarına ve yapılarına dair bilgi aşağıda verilmiştir. Dosyaların kullanımında bu yapılar değiştirilemez.

8.1 Ana Konfigürasyon Dosyası

Sistemin kullanacağı ağ, kural ve servis konfigürasyon dosyalarının ve kullanılabilecek en yüksek öncelik değerinin tanımlandığı dosyadır. Dosyanın modülün çalıştırılacağı dizinin altındaki *'conf'* dizininin altında durması ve adının *'shape.conf'* olması zorunluluğu vardır.

Bu dosyada yer alabilecek konfigürasyon direktifleri ve alabilecekleri değerler aşağıdaki gibidir. Aşağıda listelenmemiş olan bir direktif kullanıldığında bu direktif ve aldığı değer göz ardı edilir.

8.1.1 policyconf Direktifi

Sistemin kullanacağı kural bilgilerini okuduğu dosyanın adının ve dizinin belirtildiği parametredir. Kullanımı ve alabileceği değerler aşağıdadır.

policyconf (TAB | SP) dizin/dosya_adi*

8.1.2 serviceconf Direktifi

Tanımlı olan ağ servislerinin tanımlı olduğu dosyanın adıdır. Bu dosyanın UNIX veya Windows tabanlı işletim sistemlerinde kullanılan temel servis tanımlama dosyası formatında olduğu varsayılır. *serviceconf* direktifinin kullanımı ve alabileceği değerler aşağıdadır.

serviceconf (TAB | SP) dizin/dosya_adi*

8.1.3 netsconf Direktifi

Sistemin bağlı olduğu ağların tanımlarının yapıldığı dosyanın tanımını yapan direktiftir. Direktifin kullanımı aşağıdaki gibidir.

netsconf (TAB | SP) dizin/dosya_adi*

8.1.4 maxpriority Direktifi

Kullanılabilecek en yüksek öncelik değerini belirten direktiftir. Değer olarak bir sayı almalıdır.

maxpriority (TAB | SP) öncelik_değeri*

8.2 Ağ Konfigürasyon Dosyası

Sistemin bağlı olduğu ağlara ait IP adresleri ve her bir ağa ait ağ maskesinin tanımlandığı dosyadır. Dosya her satırda bir ağ tanımlanacak şekilde oluşturulmalıdır. Her satırda yer alacak bilginin formatı aşağıdadır.

ağ_IP_adresi/ağ_maskesi

8.3 Servis Konfigürasyon Dosyası

Tanımlı olan ağ servislerinin portlarının, protokollerinin ve isimlerinin tanımladığı dosyadır. Bu dosya her işletim sisteminde farklı dizinlerde ve formatlarda saklanmaktadır. Tezde göz önüne alınan format UNIX platformlarında kullanılan formattır.

service_adı port/protokol [aliases ...] [# comment]

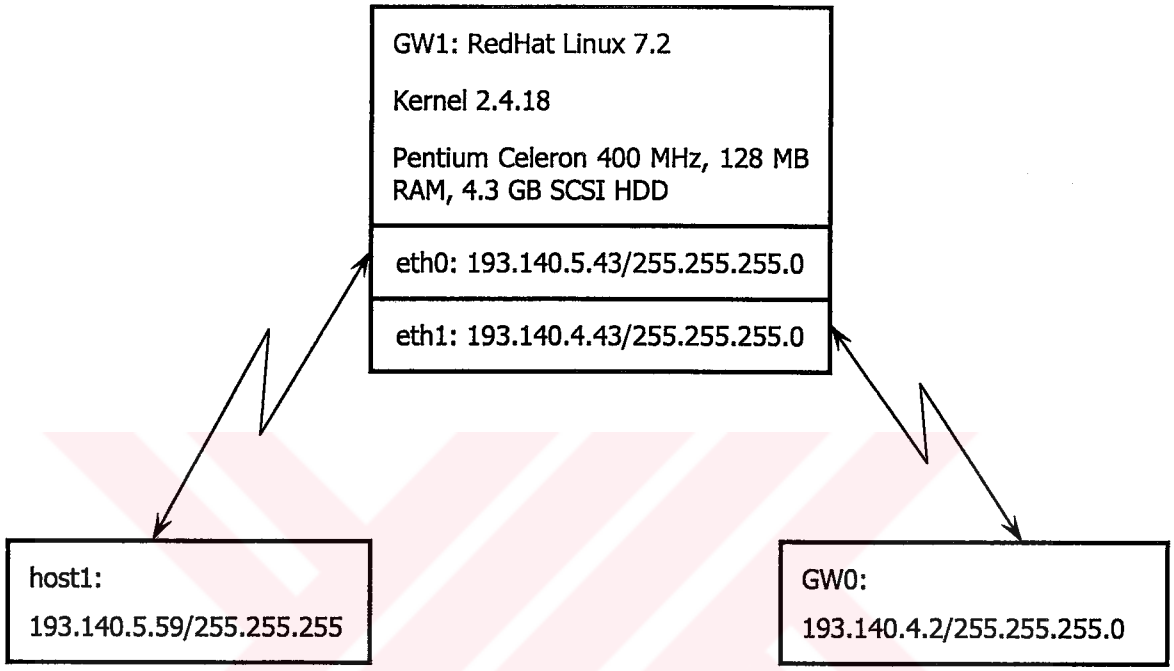
8.4 Kural Konfigürasyon Dosyası

Sistemde tanımlı olan kuralların tanımlandığı dosyadır. Kullanılan format aşağıdadır.

kaynak_adres/ağ_maskesi hedef_adres/ağ_maskesi servis/port protokol yüzde öncelik

9. SİSTEMİN TEST EDİLMESİ

Gerçeklenen sistemin test edilmesi için, bir yerel ağ oluşturulmuştur. Test yerel ağında, ağ geçidi olarak görev yapan ve gerçeklenen modülün üzerinde çalıştığı bir adet host ve bu host ile aynı ağda, trafik üretimi yapmak için kullanılacak bir başka host bulunmaktadır. Yerel ağın yapısı Şekil 9.1'de verilmiştir.



Şekil 9.1 Test altyapısı

Şekilde gösterilen GW1 isimli host, 193.140.5.0 ağı için ağ geçidi olarak davranıp, bu ağdan gelen bilgileri 193.140.4.43 arayüzü üzerinden 193.140.4.2 IP adresine sahip GW0 ağ geçidine göndermektedir. GW1'e bağlı olan 193.140.5.59 IP adresine sahip host1'de ise, sistemin test edilmesi için kullanılan, ZTI şirketine ait trafik üreticisi uygulama ve sistemin testi için diğer ağ uygulamalarına ait istemciler çalıştırılmıştır.

Sistem devreye alınıp, sisteme bağlı hostta çalıştırılan trafik üreticisi uygulama ve diğer istemciler kullanılarak ağ bağlantıları gerçekleştirilmiştir. Uygulanan testler sonucu aşağıdaki değişkenlere ait değerler elde edilmiştir:

- Paketin sisteme geldiğine karar verme süresi (T1),
- Paketin bir kurala ait olup olmadığına karar verme süresi (T2),
- Paketin sınıflandırılamayacağına karar verme süresi (T3),

- Bekleme listesine alınan bir paketin ortalama bekleme süresi (T4),
- Kuralların sistemin çalışması sonucu oluşan servis alma yüzdeleri (P).

9.1 Test Aşamasında Kullanılan Kurallar

Yukarıdaki değerlerin hesaplanması için sistemde 6 adet kural tanımlanmıştır. Bu kurallara ait kriterler Çizelge 9.1'de verilmiştir.

Çizelge 9.1 Test aşaması için tanımlanan kurallar

Kural No	Kaynak Adres	Hedef Adres	Port/ Servis	Protokol	Yüzde	Öncelik
1	193.140.5.0/255.255.255.0	*/*	http	*	15	1
2	193.140.5.59/*	*/*	8080	6	15	1
3	193.140.5.0/255.255.255.0	*/*	21	17	14	2
4	*/*	*/*	*	1	10	2
5	193.140.5.59/*	*/*	22	*	25	3
6	193.140.5.59/*	*/*	21	*	15	4

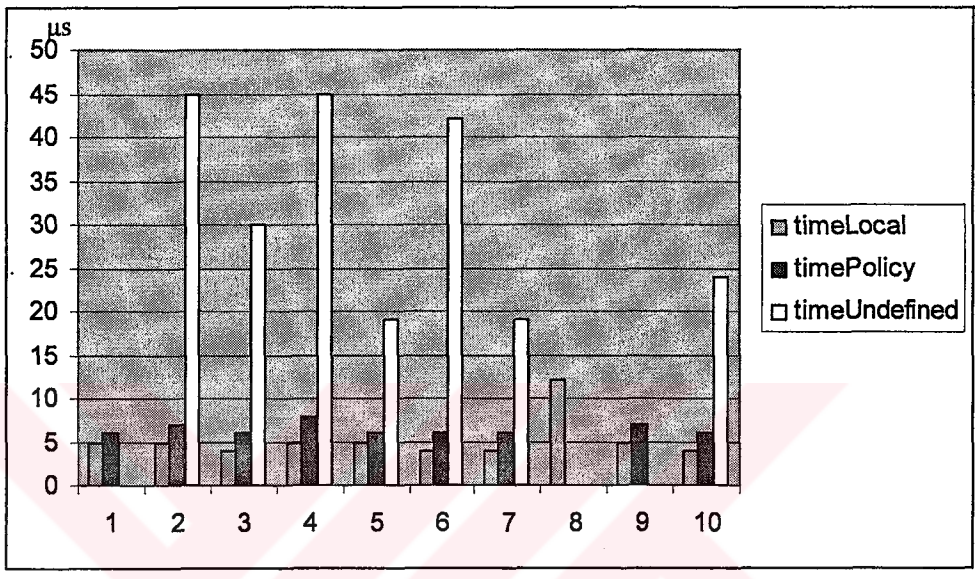
9.2 Değerlerin Hesaplanması

Hesaplanması istenilen her değer için, *shaper* yapısında için bir alan kullanılmıştır. (Şekil 7.1) Bu alanların içerdiği değerler aşağıdadır.

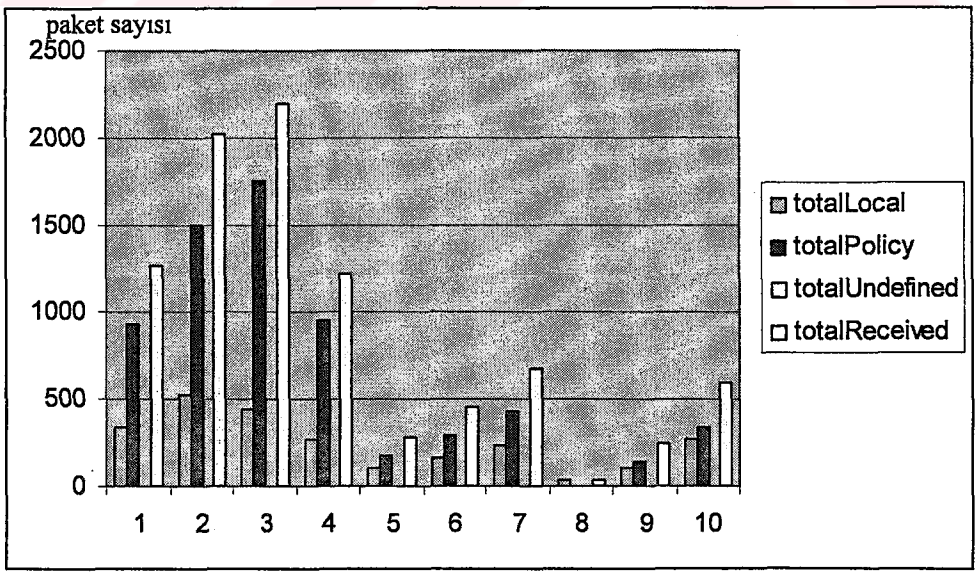
- *timeLocal*: Paketin sisteme geldiği andan, paketin hedefinin sistemin kendisi veya yerel ağda bir host olduğuna karar verildiği ana kadar geçen süre.
- *timePolicy*: Paketin sisteme geldiği andan, paketin sınıflandırılabildiğine karar verildiği ana kadar geçen süre.
- *timeUndefined*: Paketin sisteme geldiği andan, paketin sınıflandırılamayacağına karar verildiği ana kadar geçen süre.
- *timeWait*: Paketin sisteme geldiği andan, bekleme sırasından alınıp servis verilmesine kadar geçen süre.

9.3 Alınan Değerler

Trafik üreticisinin farklı sürelerde çalıştırılması sonucunda hesaplanan zamanlar Şekil 9.2'de gösterilmiştir. Şekil 9.2'deki değerler, paket sayma işleminin semaphore kullanılmadan yapılması durumundaki değerlerdir. 10 ayrı durum için toplam üretilen paket sayılarına ait bilgi Şekil 9.3'te verilmiştir. Alınan zaman değerleri mikro saniye olarak belirtilmiştir.

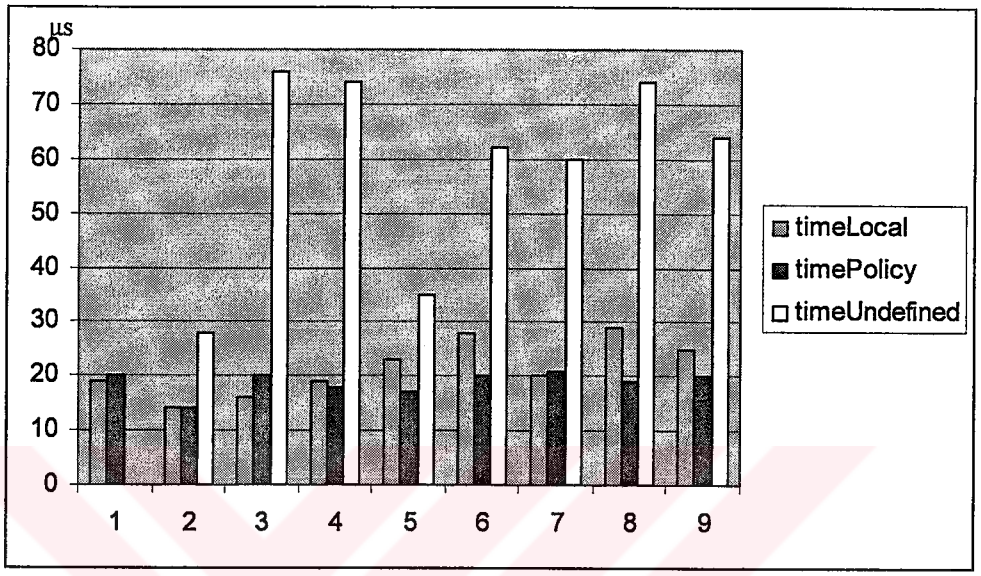


Şekil 9.2 Semaphore kullanılmaması durumu için zaman ortalamaları

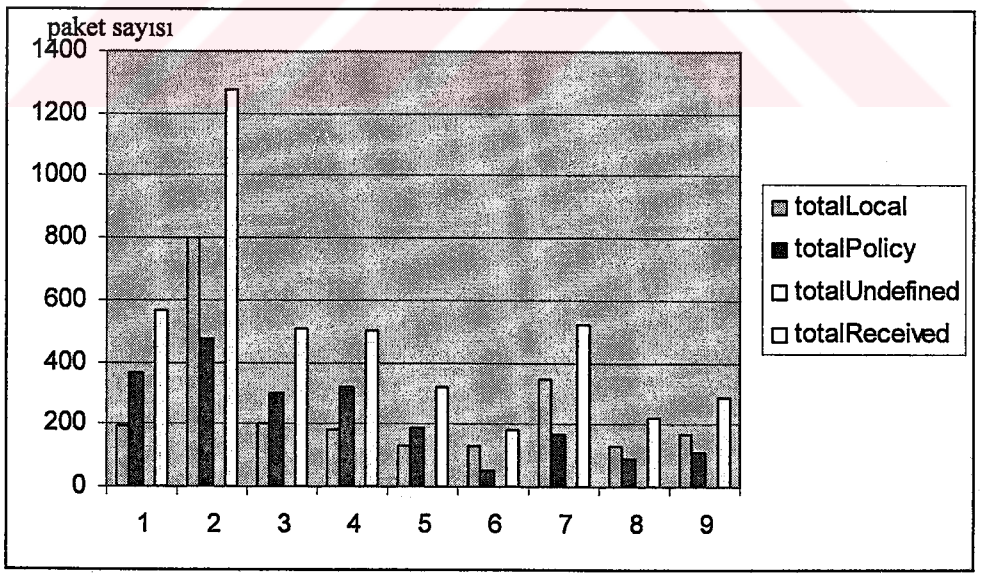


Şekil 9.3 Semaphore kullanılmaması durumu için üretilen toplam paket sayıları

Paket sayımında kullanılan değer artırma işlemi makine kodu seviyesinde atomic olarak gerçekleşmediği için bu değerlerin korunması gereği doğmuştur. Bunun için paket sayımında semaphore kullanma yolu seçilmiş ve semaphore kullanılması durumunda, zaman ve paket hesapları tekrarlanmıştır. Bu durumda elde edilen değerler Şekil 9.4 ve Şekil 9.5'te gösterilmiştir. Alınan zaman değerleri mikro saniye olarak belirtilmiştir.



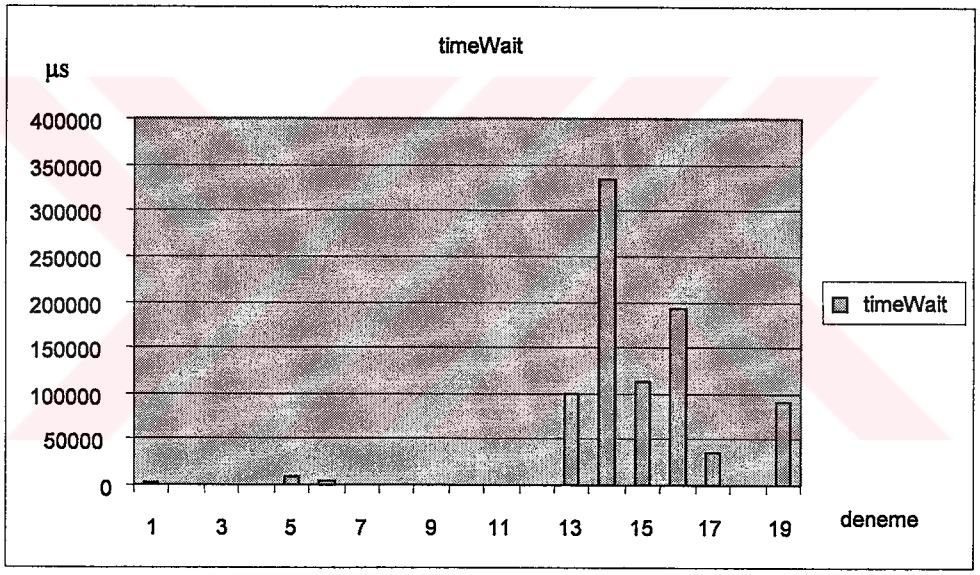
Şekil 9.4 Semaphore kullanılması durumu için zaman ortalamaları



Şekil 9.5 Semaphore kullanılması durumu için üretilen toplam paket sayıları

Semaphore kullanımının sisteme getirdiği yük, her *up* ve *down* işlemi için ortalama olarak 6 mikro saniye olarak hesaplanabilir. İkinci testte ortalama zaman değerlerinin artmasındaki ikinci sebep, paket kontrollerinde kullanılan birer *printk* fonksiyonudur. Yapılan zaman testlerinde, *printk* fonksiyonunun 2 mikro saniyede gerçekleştiği hesaplanmıştır.

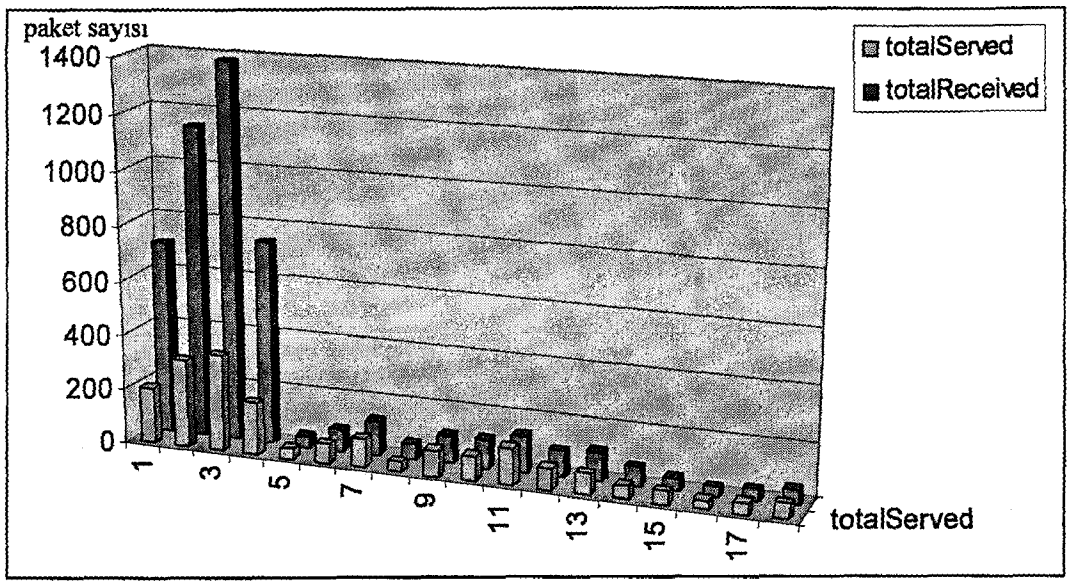
Yukarıda hesaplanan zaman değerlerinin yanında, sınıflandırılan bir paketin, servis almadan önce bekleme kuyruğunda ortalama bekleme süresinin hesaplanması da gerekmektedir. Yukarıda her iki durum için de yapılan testlerde bekleme süreleri hesaplanmıştır. Belirli bir kurala, istenilen servis verme oranının çok üstünde paket geldiğinde, paketlerin bekleme süresi çok yüksek çıkmaktadır. Paketlerin kurallara göre dağılımının homojen olması durumunda ise paketlerin bekleme süreleri düşmektedir. Bekleme sürelerine ait değerleri Şekil 9.6'da gösterilmiştir. Alınan zaman değerleri mikro saniye olarak belirtilmiştir.



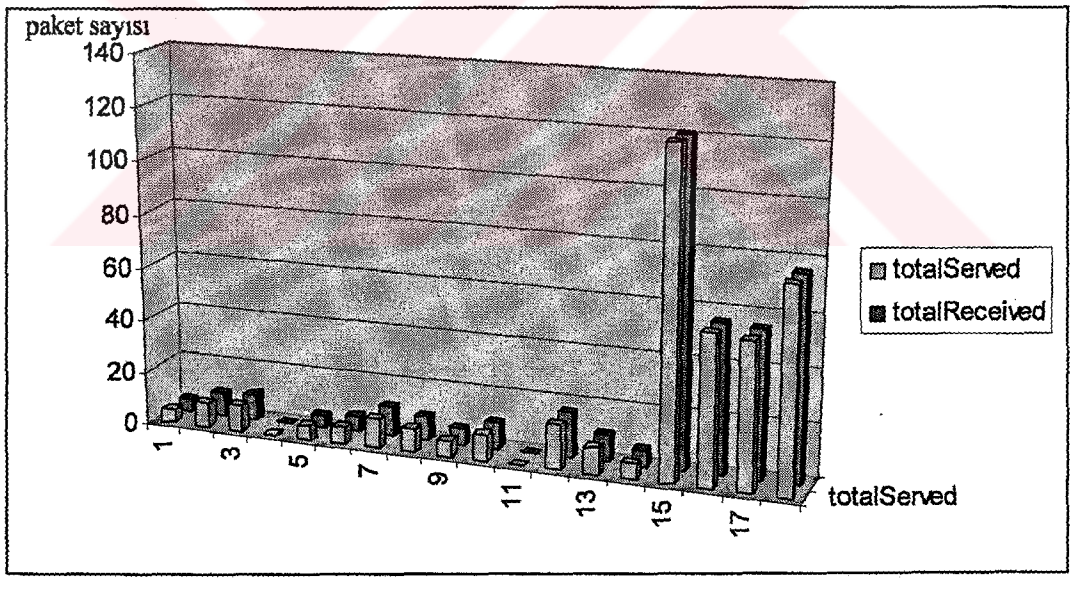
Şekil 9.6 Paketlerin servis almak için ortalama bekleme süresi

Şekil 9.6'da görüldüğü gibi bekleme süresi 0,2 milisaniyeden, 30 milisaniyeye kadar değişebilmektedir.

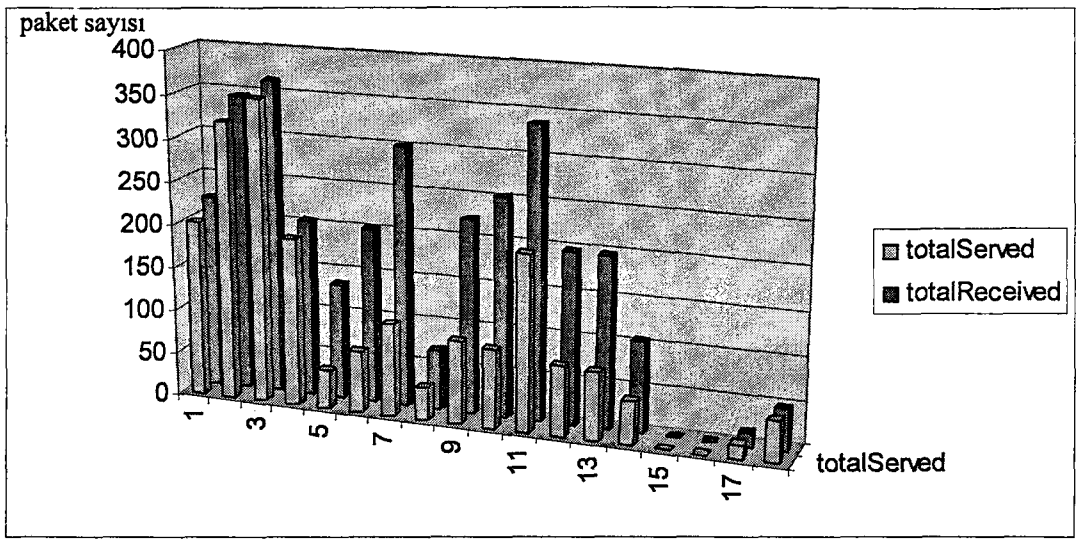
Tanımlanan kurallara göre servis verme oranları sistemin devreden çıkarılması sonucu hesaplanmıştır. Sistemin devreden çıkarıldığı anda kurallara ait toplam alınan, toplam servis verilen paket sayıları hesaplanmıştır. 1., 5. ve 6. kurallara ait değerler Şekil 9.7, 9.8, 9.9, 9.10'da gösterilmiştir.



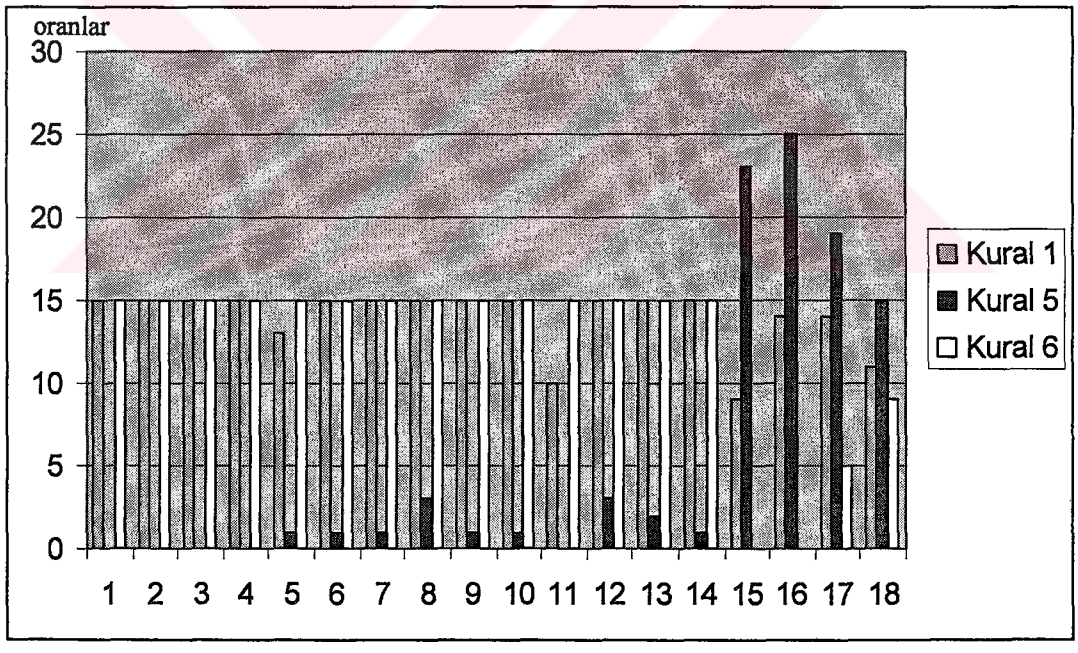
Şekil 9.7 1 numaralı kurala ait gelen/servis verilen paket sayıları



Şekil 9.8 5 numaralı kurala ait gelen/servis verilen paket sayıları



Şekil 9.9 6 numaralı kurala ait gelen/servis verilen paket sayıları



Şekil 9.10 1, 5 ve 6 numaralı kurallara ait servis alma oranları

10. SONUÇ ve ÖNERİLER

Yerel ağda belirli servislerin ve bağlantıların sınırlandırılması, bant genişliğini daha verimli ve istenilen seviyede kullanma imkanı sunmaktadır. Yapılan çalışmada, bu amaca yönelik bir sistem tasarlanmıştır. Daha sonra Linux işletim sistemi üzerinde, C programlama dili kullanılarak bu tasarım gerçeklenmiştir. Oluşturulan sistem, belirli koşullar altında test edilmiş, alınan sonuçlar değerlendirilmiş ve aşağıdaki sonuçlara varılmıştır:

- Gerçeklenen sistem kernele dinamik olarak yüklenebilen bir modül şeklinde geliştirilmiştir. Bu sistemin çalışmasını kernelden bağımsız hale getirmektedir.

- Sistemdeki kuralların, ağların ve servislerin konfigürasyon dosyaları kullanılarak tanımlanması sistemin yönetimini kolaylaştırmıştır. Bu bilgilerin dosyalardan okunması sonucunda, sistemin devreye alınmasında oluşabilecek parametre karmaşası ortadan kaldırılmıştır.

- Sistemin gerçeklenmesi sonucu oluşan servis kalitesi sistemi, toleranssız bir servis kalitesi sistemidir. Böylece kurallarla tanımlanan bağlantı tiplerinin ağı kullanımı, belirli bir seviyede tutulabilmektedir. Yapılan testlerde görüldüğü gibi, bağlantı tiplerinin bu şekilde kısıtlanması, cevap süresini artırmasına rağmen, kurallar ile belirtilmemiş bağlantı tipleri sürekli servis görmektedir.

- Modülün yerel ağdaki ağ geçidi sunucusunda çalışması, servis kalitesinin her adımda sağlanması zorunluluğunu ortadan kaldırmıştır. Bant genişliğinin sınırlandırılması sadece yerel ağ ucunda yapılmış, kaynaktan hedefe kadar olan ara hostlarda bu servisin sağlanması zorunluluğu ortadan kalkmıştır. Bu da daha basit bir servis kalitesi mekanizması demektir.

- Tasarımın gerçeklenmesinde kullanılan arama ve servis verme algoritmaları ile, paketin kurallar içinde bulunması, yerel ağ bilgisi kontrolü işlemleri en kısa sürede gerçekleşmektedir. Bu işlemlerin kısa sürede gerçekleşmesi hem servis verme hem de paketlerin kuyruğa yerleştirilme ve kuyrukta bekleme sürelerini azaltmıştır.

- Seçilen geliştirme yöntemi ve Linux kernelinde kullanılabilen fonksiyonların sınırlılığı, sınıflandırma işleminin paketlerin geldiği arayüzde gerçeklenmesini zorunlu kılmıştır. Kontrollerin bu noktada yapılması, paketin sisteme ait olup olmadığının kontrolü gibi ekstra kontroller yapma zorunluluğunu getirmiştir. Bu da sınıflandırma aşamasının dezavantajıdır.

Sistemin üzerinde yapılan testler ve sistemin çalışması göz önüne alındığında, sistem üzerinde aşağıdaki değişikliklerin yapılabileceği sonucuna varılmıştır.

Kernelin değiştirilmesi ve paket yönlendirme fonksiyonun kontrolüne sahip olunması durumunda, uygulanan yöntem tekrar kullanılarak, sistemin paket yönlendirme aşamasında kullanılması sağlanabilir. Birden fazla kuyruk, sıralama ve servis verme algoritması kullanılarak, sistem yöneticisinin servis verme aşamasında hangi algoritmayı kullanacağını seçmesine olanak tanınabilir. Kural tanımında kullanılan port numarası alanında tek port numarası kullanmak yerine, 2 port arasındaki tüm port numaraları veya birden fazla servis kriter olarak verilebilir.

KAYNAKLAR

Bradford Dr. E. G., "High-performance programming techniques on Linux and Windows – Pipes in Linux, Windows 2000 and Windows XP", IBM DeveloperWorks, Ekim 2001

Bradford Dr. E. G., "High-performance programming techniques on Linux and Windows – Managing Processes and Threads", IBM DeveloperWorks, Şubat 2002

Brody S., (1999), "IDC Says Linux Likely To Lead OS Growth", SunWorld, 31 Mart 1999

Evans Data Corp., (2002), "Spring 2002 Linux Developer Survey", 2002

Gopinath K. N., (1999), "Kernel Support For Building Network Firewalls Based On The Paradigm of Selective Inspection of Packets At Applicaiton Level", Department of Computer Science and Engineering, Indian Institute of Technology, 15 Nisan 1999

Haber L., (2002), "City Saves With Linux/Thin Clients", ZDNet TechUpdate, 10 Nisan 2002

Harris S. E., (2001), "The Truth Behind The Great Server Heist", consultingtimes.com, 2001

Knipe Z., (2001), "Linux Market To Grow 154% in 2001", Idaya, Ocak-Mart 2001

Kossak, (1999), "Building Into The Linux Network Layer", Phrack Magazine, 9 Eylül 1999

Luening E., (2001), "Windows Users Pay For Hacker Insurance", CNET news.com, 29 Mayıs 2001

Miller P.B. et. al., "Fuzz Revisited: A Re-examination of the Reliability of UNIX Utilities and Services", Computer Sciences Department, University of Wisconsin, 18 Şubat 2000

Pragmatic, (1999), "Nearly Complete Guide to Loadable Kernel Modules", Mart, 1999

Radhakrishnan S., (1999) "Linux – Advanced Networking Overview", Information and Telecommunications Technology Center, Dept. Of Electrical Engineering & Computer Science, University of Kansas, 22 Ağustos 1999

Rothman J. B., Buckman J., (2001), "Which OS is Fastest for High-Performance Network Applications?", SysAdmin Magazine, Temmuz 2001

Shakland S., Kane M., Lemos R., (2001), "How Linux Saved Amazon Millions", CNET news.com, 30 Ekim 2001

Vallopillil V., (1998), "Linux Operating System", Microsoft, 11 Ağustos 1998

INTERNET KAYNAKLARI

[1] NetCraft, (2001), "Computers Running On The Web", http://www.netcraft.com/Survey/index-200109.html#computers, Eylül 2001

[2] Ryan J., (2001), "QoS In The Enterprise", http://www.techguide.com

[3] Welte H., (2000) "The Journey of A Packet Through The Linux 2.4 Kernel", http://www.gnumonks.org/ftp/pub/doc/packet-journey-2.4.html, 14 Ekim 2000

ÖZGEÇMİŞ

Doğum tarihi	09.05.1977	
Doğum yeri	Adana	
Lise	1992-1995	Gaziantep Fen Lisesi
Lisans	1995-1999	Yıldız Teknik Üniversitesi Elektrik-Elektronik Fak. Bilgisayar Mühendisliği Bölümü

Çalıştığı kurumlar

1997-1998	Setra Uluslararası Ticaret ve Ltd. Şti.
1999-Devam ediyor	YTÜ Fen Bilimleri Enstitüsü Araştırma Görevlisi