YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

AĞ UYGULAMALARININ SÜREKLİLİĞİNİN SAĞLANMASI İÇİN AĞ VE TAŞIMA KATMANLARININ UYGULAMAYA SAYDAM OLARAK ANAHTARLANMASINI SAĞLAYAN BİR SİSTEMİN TASARLANMASI VE GERÇEKLENMESİ

Bilgisayar Mühendisi Turgut GENÇ

FBE Bilgisayar Mühendisliği Anabilim Dalı Bilgisayar Mühendisliği Programında Hazırlanan

YÜKSEK LİSANS TEZİ

Tez Danışmanı: Yrd. Doç. Dr. A. Gökhan YAVUZ (Y.T.Ü.)

İSTANBUL, 2007

İÇİNDEKİLER

KISALTMA LİSTESİ

ATAFS	Application Transparent Asymmetric Failover System
HA	High Availability
FT	Fault-Tolerance
LKM	Loadable Kernel Module
MAC	Media Access Control
HB	Heart-Beat

TERİMLER LİSTESİ – 1

İngilizce	Türkçe
3 way handshake	3 zamanlı el sıkışma
Active Mode	Aktif çalışma kipi
Authorized user	Yetkili kullanıcı
Availability	Yararlanırlık
Backup	Yedekleme
Base kernel	Ana çekirdek
Batch processing	Toplu İşleme
Boot	Yükleme
Buffer	Ara bellek
Bug	Çapar
Bus	Veri yolu
Cluster	Grup sistem
Cold standby	Elle yedek
Configuration	Yapılandırma
Continuous availability	Kesintisiz yararlanırlık
Data structure	Veri yapısı
Design diversity	Tasarım Çeşitlemesi
Distributed system	Dağıtık sistemler
Downtime	Aksaklık süresi
Dynamic recovery	Dinamik toparlanma
Error	Hata
Failover	Yedeğe geçmelilik
Fail-Safe	Arıza Emniyetli
Failure	Arıza
Fault	Kusur
Fault detection	Kusur saptama
Fault masking	Kusur maskeleme
Fault-tolerance	Kusura dayanıklılık
Forwarding	Yönlendirme
Hardware fault-tolerance	Donanımsal kusura dayanıklılık
Hash	Kıyım
Hashing	Kıyımlı arama
Heart-Beat	Kalp-Atışı
High availability	Yüksek yararlanırlık
Hook	Kanca
Hot standby	Acil yedek
Hub	Göbek
Initialize	İlklendirme
Interrupt	Kesme
Kernel based	Çekirdek temelli
Kernel module	Çekirdek parçası
Loadable kernel module	Yüklenebilen çekirdek parçası
Middleware	Ara katman yazılımı
Mode	Çalışma kipi
Module	Parça
Open source	Açık kaynaklı kod

İngilizce	Türkçe
Operator	İşletmen
Packet handler	Paket İşleyicisi
Patch	Yama
Performance management	Başarım yönetimi
Periodic maintenance	Dönemsel bakım
Point to point	Noktadan noktaya
Recovery	Toparlanma
Redundancy	Yedeklilik
Reliablity	Güvenilirlik
Roll back	Geriye sarma
Routine operation	Olağan etkinlik
Scalability	Ölçeklenebilirlik
Security	Güvenlik
Self-Checking	Kendini denetleme
Session	Oturum
Software fault-tolerance	Yazılımsal kusura dayanıklılık
Stability	Kararlılık
Standby	Yedek
Standby Mode	Bekleme kipi
Stateful failover	Durum bilgili yedeğe geçme
Stateless failover	Durum bilgisiz yedeğe geçme
Storage	Depolama
Timer	Süreölçer
Version	Sürüm
Warm failover	Sıcak yedeğe geçmeli
Warm standby	Sıcak yedek

TERİMLER LİSTESİ – 2

Türkçe	İngilizce
3 zamanlı el sıkışma	3 way handshake
Acil yedek	Hot standby
Açık kaynaklı kod	Open source
Aksaklık süresi	Downtime
Aktif çalışma kipi	Active Mode
Ana çekirdek	Base kernel
Ara bellek	Buffer
Ara katman yazılımı	Middleware
Arıza	Failure
Arıza Emniyetli	Fail-Safe
Başarım yönetimi	Performance management
Bekleme kipi	Standby Mode
Çalışma kipi	Mode
Çapar	Bug
Çekirdek parçası	Kernel module
Çekirdek temelli	Kernel based
Dağıtık sistemler	Distributed system
Depolama	Storage
Dinamik toparlanma	Dynamic recovery
Donanımsal kusura dayanıklılık	Hardware fault-tolerance
Dönemsel bakım	Periodic maintenance
Durum bilgili yedeğe geçme	Stateful failover
Durum bilgisiz yedeğe geçme	Stateless failover
Elle yedek	Cold standby
Geriye sarma	Roll back
Göbek	Hub
Grup sistem	Cluster
Güvenilirlik	Reliablity
Güvenlik	Security
Hata	Error
İlklendirme	Initialize
İşletmen	Operator
Kalp-Atışı	Heart-Beat
Kanca	Hook
Kararlılık	Stability
Kendini denetleme	Self-Checking
Kesintisiz yararlanırlık	Continuous availability
Kesme	Interrupt
Kıyım	Hash
Kıyımlı arama	Hashing
Kusur	Fault
Kusur maskeleme	Fault masking
Kusur saptama	Fault detection
Kusura dayanıklılık	Fault-tolerance
Noktadan noktaya	Point to point
Olağan etkinlik	Routine operation
Oturum	Session
Ölçeklenebilirlik	Scalability
Paket İşleyicisi	Packet handler

Türkçe	İngilizce
Parça	Module
Sıcak yedeğe geçmeli	Warm failover
Sıcak yedek	Warm Standby
Süreölçer	Timer
Sürüm	Version
Tasarım Çeşitlemesi	Design Diversity
Toparlanma	Recovery
Toplu İşleme	Batch processing
Veri yapısı	Data structure
Veri yolu	Bus
Yama	Patch
Yapılandırma	Configuration
Yararlanırlık	Availability
Yazılımsal kusura dayanıklılık	Software fault-tolerance
Yedeğe geçmelilik	Failover
Yedek	Standby
Yedekleme	Backup
Yedeklilik	Redundancy
Yetkili kullanıcı	Authorized user
Yönlendirme	Forwarding
Yükleme	Boot
Yüklenebilen çekirdek parçası	Loadable kernel module
Yüksek yararlanırlık	High availability

ŞEKİL LİSTESİ

ÇİZELGE LİSTESİ

ÖNSÖZ

Bilgisayar mühendisi olacağım sözünü ilk sarf ettiğim ortaokul günlerinden bu yana başta öğretmenlerim, ailem ve dostlarım olmak üzere eğitim ve kişisel gelişimimde bana katkısı olan herkese; hayatımın her anında yanımda olduklarını hissettiğim aileme, özellikle akademik kariyer yapma düşüncesini aşılayan ve aynı zamanda tanıdığım en disiplinli, çalışkan, azimli insan olan babama; yüksek lisans eğitimi ve tez geliştirme süreci boyunca yardımlarını, bilgisini ve vaktini paylaşan Yrd. Doç. Dr. A. Gökhan YAVUZ' a; yüksek mühendis olma yolunda bana katkısı olan tüm Yıldız Teknik Üniversitesi – Bilgisayar Mühendisliği Bölümü akademisyenlerine; ihtiyacım olan her an her konuda yanımda yer alan, bu tez çalışmasının dokümantasyonu sırasında benimle birlikte sabahlayıp benimle birlikte çalışan hayat arkadaşım Melek Gözde MERİÇ ile çok sevgili dostlarım Erdem CAN ve Haluk ÖRNEK'e sevgi, saygı ve şükranlarımı sunarım.

ÖZET

Bilgisayarların hayatımıza girmesi ve bilgisayar ağlarının kurulması bilgi yönetimi ve paylaşımını kolaylaşmıştır. Internet kavramının ortaya çıkışıyla iletişim kavramı yepyeni bir boyut kazanmış, gerçek hayatta karşımıza çıkan ve temelinde iletişim yatan pek çok uygulama artık bilgisayarlar aracılığıyla yürütülür hale gelmiştir.

Günümüzde bilgisayar ağları üzerinde yürütülen bilgi akışının devamlılığının sağlanması ve kesintisiz hizmet veren sistemlerin oluşturulması pek çok uygulamada kritik önem arz etmektedir. Bilgisayar sistemlerinin sürekliliğinin sağlanması ve kesintisiz hizmet verebilmesi için yüksek yararlanırlık, kusura dayanıklılık gibi pek çok kavram ortaya atılmış, bu konular üzerine önemli çalışmalar yapılmıştır.

Bu çalışmada yüksek yararlanırlıklı ve kusura dayanıklı sistemler incelenmiş, var olan çalışmalardaki eksik ve zayıf yönler değerlendirilmiştir. Küçük çaplı sistemlerde kullanılabilecek ve düşük maliyetle yüksek performans gösterecek bir mimari, Application Transparent Asymmetric Failover System (ATAFS) tasarlanmıştır. ATAFS mimarisi Linux işletim sistemi üzerinde C programlama dili ve Yüklenebilen Çekirdek Modülü kullanılacak şekilde tasarlanmıştır. Tasarımın hayata geçirilerek elde edilen sonuçların değerlendirilmesi bu çalışmanın faydasını daha da artıracaktır.

Anahtar Kelimeler: Linux, çekirdek temelli, yüksek yararlanırlıklı, kusura dayanıklı, uygulamaya saydam, sıcak yedeğe geçmeli

ABSTRACT

Letting computers come into our daily lives and building computer networks made it easy to manage and share the information via computer networks. With the help of the Internet the concept of communication has evolved into a new generation and lots of daily applications related to communication are being handled by the computers.

Nowadays, it is very important to build highly available computers and make the data on the network flow continuously. In the idea of Continuously Available Servers, there are some other concepts like High Availability, Fault-Tolerance etc. and there are some researches on these subjects.

In this work, the disadvantages and the incompetent sides of highly available and fault tolerant systems are examined and extracted and then a new system, called Application Transparent Asymmetric Failover System (ATAFS) is designed to work on small systems with a low cost and high performance. ATAFS architecture is planned to be implemented on Linux operating system and is planned to be a loadable kernel module. Implementing this design would be a complement to all these studies on high availability and fault tolerant systems researches.

Keywords: Linux, kernel-based, high availability, fault-tolerant, transparent, warm failover

1. GİRİŞ

Bilgisayarların hayatımıza girmesiyle bilginin yönetimi, bilgisayar ağlarının kurulmaya başlamasıyla da bilginin paylaşımı kolaylaşmıştır. Internet'in ortaya çıkışı iletişim kavramına yepyeni bir boyut kazandırmış, bilgisayar ağlarının günlük hayatımızdaki yeri arttıkça gerçek hayatta karşımıza çıkan ve temelinde iletişim yatan pek çok uygulamayı artık bilgisayarlar aracılığıyla yürütülür hale getirmiştir.

Bilgisayar ağlarının ve bilgisayar destekli sistemlerin kullanım oranları arttıkça üzerinde yaşayan bilginin de önemi artmış, pek çok uygulamada sistemin kararlılığı ve sürekliliği kritik önem arz eder olmuştur. Bilgisayar ağları üzerinden yürütülen bilgi akışının devamlılığının sağlanması ve kesintisiz hizmet veren sistemlerin oluşturulması uzay bilimleri, sağlık gibi sektörlerde hayati, çeşitli ticari uygulamalarda ise maddi açıdan yüksek önem taşımaktadır. Bir uzay aracını veya bir yaşam destek ünitesini kontrol eden bilgisayar sisteminin devre dışı kalması can kaybına sebebiyet verebileceği gibi telafisi mümkün olmayan problemlere de yol açabilir. Sürekli bilgi alışverişi yapılan büyük bir veritabanı uygulamasına ihtiyaç duyan bir firma, veritabanına erişilemeyen her dakika on binlerce dolar kaybedebilir. Günümüzde bilgisayar sistemlerinin pek çoğunun kesintisiz hizmet vermesi zaruri bir hal almıştır.

Kesintisiz hizmet verebilecek bilgisayar sistemleri üzerine çalışmalar 1950'lerde başlamış, ilk olarak yararlanırlık (availability) kavramı irdelenmiş ve mümkün olduğunca yüksek yararlanırlıklı (high availability) sistemler tasarlanmaya çalışılmıştır. İlk zamanlarda kesintilerin temelini donanım kaynaklı sorunlar oluşturduğundan donanım sorunlarının üstesinden gelecek yönde çalışmalar yapılmış, donanım problemleri zaman içinde aşıldıkça yüksek yararlanırlık yolunda ortaya çıkan yazılım, çevresel ve insan kaynaklı diğer sorunların da ortadan kaldırılmasına çalışılmıştır (Gray ve Siewiorek, 1991).

Donanımların mümkün olduğunca sorunsuz çalışması ve işlev bozukluğu ortaya çıktığında mümkün olan en kısa sürede aynı görevi gören bir başka donanıma aktarılması yüksek yararlanırlık yolunda ortaya atılan ilk adımdır. Günümüzde kullanılan yüksek yararlanırlıklı sistemlerde hem donanım hem de yazılım kusurlarının üstesinden gelecek çeşitli düzenekler bir arada kullanılmaktadır (David A.Rennels, 2004).

Yüksek yararlanırlıklı ve kusura dayanıklı (fault tolerant) sistemlerin daha iyi anlaşılabilmesi için gerekli teknik terimler tezin 2. bölümünde, bu alanda bugüne kadar yapılan çalışmalar 3. bölümünde verilmektedir. Geliştirilen sistemin daha iyi anlaşılabilmesi için 4. bölümde Linux işletim sistemi ile ilgili bilgiler, 5. bölümde çekirdek parçaları ile ilgili ayrıntılı bilgi ve 6. bölümde Linux işletim sistemindeki ağ yönetimi hakkında bilgi verilmiştir. 7. bölümde ATAFS sisteminin tasarım aşaması ve ATAFS sisteminin özellikleri yorumlanmıştır. 8. ve 9. bölümlerde ATAFS sisteminin kodlanması sırasında kullanılan fonksiyon ve veri yapıları tanıtılmış, 10. bölümde ATAFS sisteminin karşılaşabileceği senaryolar anlatılmıştır.

Bu tez çalışmasında çekirdek temelli, kusura dayanıklı, sıcak yedeğe geçmeli (warm failover), uygulamaya saydam (transparent) ve yüksek yararlanırlıklı bir sistem tasarlanmış, Linux işletim sistemi üzerinde gerçeklenecek şekilde tasarlanan sistemin detayları anlatılmış ve elde edilen sonuçlar değerlendirilmiştir.

2. BİLGİSAYARDA YARARLANIRLIK

Bilgisayar sistemlerini incelediğimizde yararlanırlık (availability) kavramı kritik önem taşımaktayken, yüksek yararlanırlıklı sistemler oluşturmak maliyet ve verim açısından çeşitli zorluklar taşımaktadır. Son derece yüksek yararlanırlıklı ufak bilgisayar parçaları üretmek mümkün olsa da birçok parçadan oluşan bilgisayar sistemlerini milyonlarca satır koddan oluşan yazılımlar ile desteklemek başlı başına bir sanat olarak değerlendirilebilir (Gray ve Siewiorek, 1991).

Kesintisiz yararlanırlık (Continuous availability) hedefinde yapılan çalışmalar içinde çeşitli kavramlar ve yöntemler ortaya atılmıştır. Günümüz teknolojisi içerisinde her ne kadar sorunsuz çalışan donanım ve yazılımlar üretmek mümkün olabilse da istatistiksel olarak %100 kesintisiz çalışabilen sistemler üretmek mümkün olamamakta, bunun yerine mümkün olduğunca fazla yararlanılabilen, mümkün olan en yüksek yararlanırlık imkânı sağlayan sistemler üretilmeye çalışılmaktadır. Kesintisiz yararlanırlık konusu üzerine yapılan çalışmalarda kullanılan başlıca kavramlar şunlardır:

- *Yararlanırlık*
- *Kesintisiz Yararlanırlık*
- *Yüksek Yararlanırlık*
- *Kusura Dayanıklılık*
- *Arıza Emniyetlilik*
- *Yedeğe geçmelilik*

Kesintisiz yararlanırlık gerçekte var olmayan ve bilgisayar teknolojileri söz konusu olduğunda yararlanırlık kavramı üzerine varılmak istenen son noktadır. Kesintisiz yararlanırlık konusu ele alındığında en fazla ön plana çıkan kavramlar yararlanırlık, yüksek yararlanırlık ve kusura dayanıklılık kavramları olmuştur. Bu kavramlar hakkında detaylı bilgiler ileriki bölümlerde anlatılmıştır. Yüksek yararlanırlık ve kusura dayanıklılık özelliklerine sahip sistemler, bir sonraki bölümde anlatılmıştır.

2.1 Yararlanırlık (Availability)

Bir sistemin verdiği hizmetin devamlılığı ve hizmete ihtiyaç duyulduğu sürece hizmetin kullanıma açık durumda olması yararlanırlık kavramı altında incelenir. Bir sistemin hizmete ihtiyaç duyulduğu bir **t** anında servis verebilir durumda olma ihtimaline yararlanırlık adı verilir [1]. Bir başka deyişle yararlanırlık, işlevsel bir birimin kullanılabildiği ve hizmet verebildiği sürenin kullanılma gereksinimi bulunan toplam süreye oranıdır. Yararlanırlık oranı hesaplanırken Şekil 2.1'de görülen formülden yararlanılır.

Mtbf: Arızalar arası ortalama süre (Mean time between failures)

Mttr: Onarıma kadar geçen ortalama süre (Mean time to repair)

$$\text{Yararlanırlık} = \frac{Mtbf}{(Mtbf + Mttr)}$$

Şekil 2.1 Yararlanırlık Hesabının Yapılması [1]

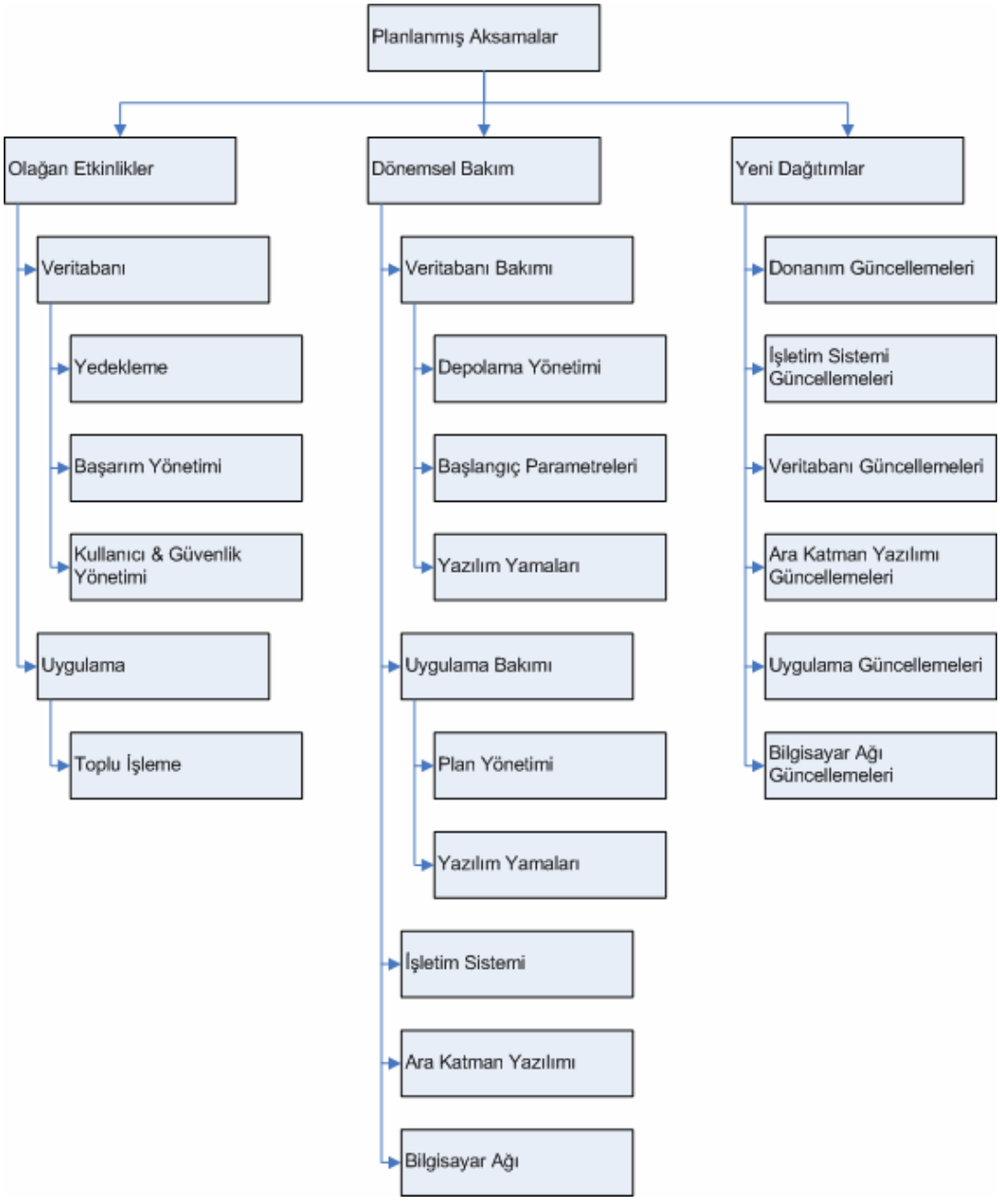
2.2 Aksama Süresi

Yararlanırlık kavramından bahsedildiği zaman aksama süresi (downtime) kavramından da söz etmek gerekir. Bir sistemin veya sistem bileşeninin işlev göremediği veya servis dışı kaldığı süreye aksama süresi denir [3]. Aksama süresi önceden planlanmış olabileceği gibi planlanmamış da olabilir. Her iki türde yer alan aksama süresi de büyük bir firmadaki iş akışını, bilgi akışını ve haberleşme trafiğini, ürün geliştirme aşamalarını ve ürünlerin piyasaya sürülme zamanını etkileyebileceği gibi bunların sonucu olarak müşteri kaybı, iş kaybı veya genel olarak itibar kaybına da sebebiyet verir. Aksama süreleri temel olarak iki farklı grupta incelenmektedir:

- Planlanmış Aksama Süresi
- Planlanmamış Aksama Süresi

2.2.1 Planlanmış Aksama Süresi

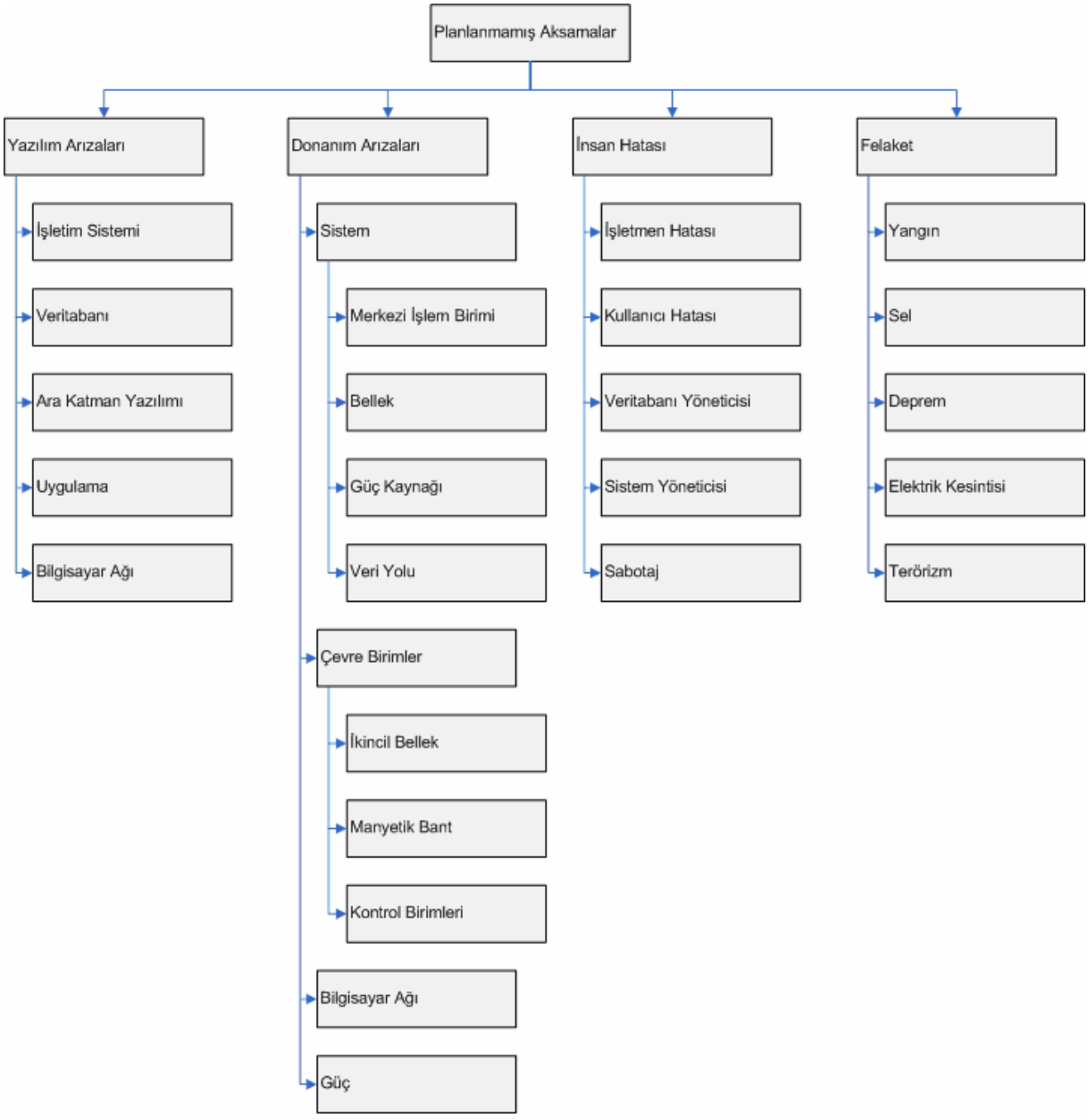
Planlanmış aksama süresi, bir uygulamanın veya bir sistemin normal bakım işlemlerini kapsar. Her ne kadar sistem kullanımını mümkün olduğunca az engellemek üzere planlanmış olsa da planlanmış aksama süresi, planlanmış ve planlanmamış toplam aksama süresinin neredeyse %80 ini oluşturmaktadır [5]. Planlanmış aksama süresinin azaltılmasının en etkin çözümü kesintisiz hizmetler sunan bir alt yapı kullanmaktan geçmektedir. Planlanmış aksama süresinin nedenleri Şekil 2.2'de gösterilmiştir.



Şekil 2.2 Planlanmış Aksama Süresinin Nedenleri [5]

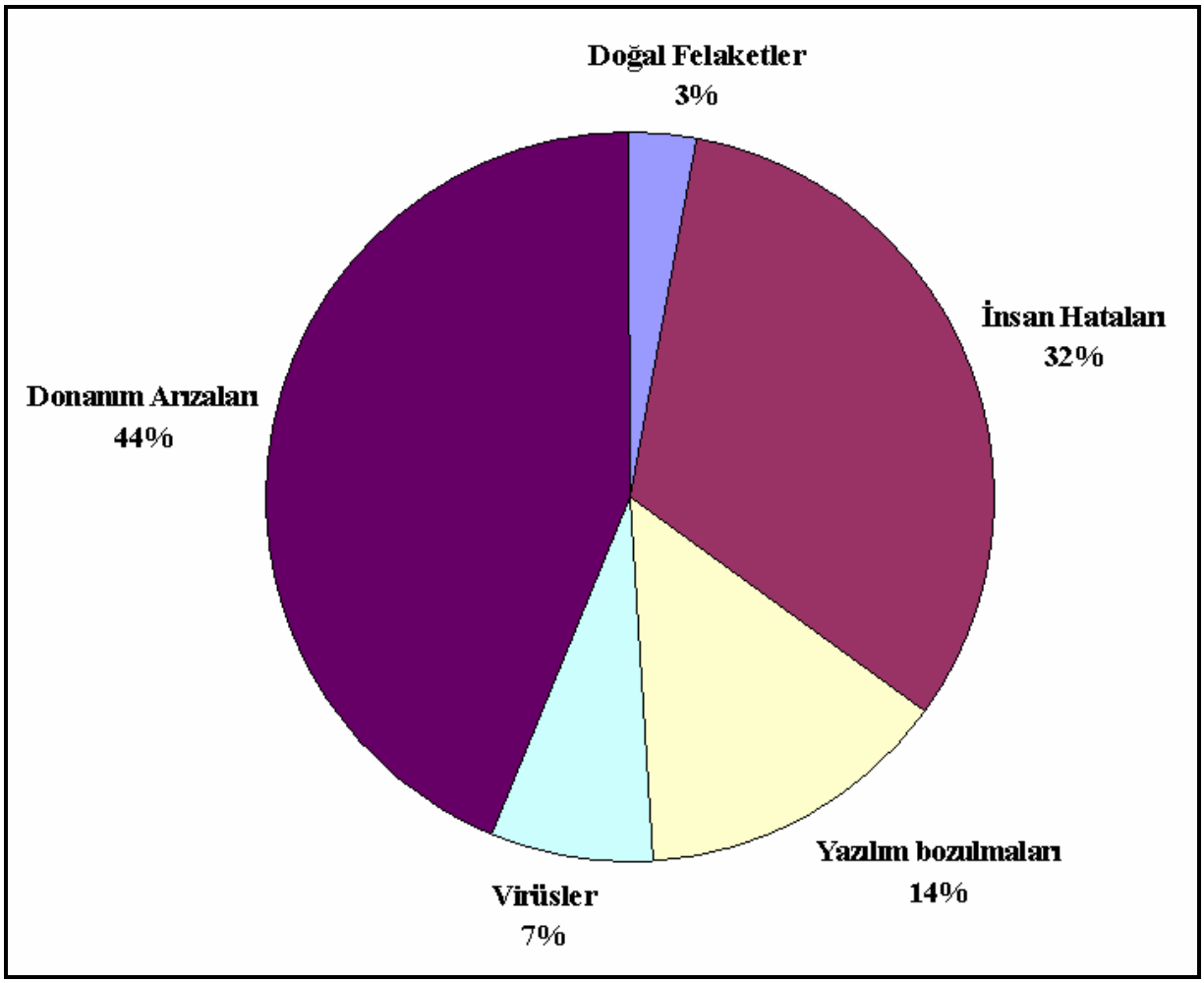
2.2.2 Planlanmamış Aksama Süresi

Planlanmamış aksamalar, donanım arızası, insan hatası, yazılım arızası, doğal felaketler gibi çok çeşitli nedenlerden kaynaklanabilir. Planlanmamış aksamalarla sonuçlanan olayların başlıca nedenleri Şekil 2.3'de gösterilmiştir. Bu tip problemlerin çözümünde başta depolama ve ağ bileşenleri olmak üzere yedekli donanım ve yazılımlar, birden çok ikizleme (mirroring), yedeğe geçmeli sistemler, yerel ve uzaktan yıkım onarımı (disaster recovery) teknolojileri kullanılmaktadır.



Şekil 2.3 Planlanmamış Aksama Süresinin Nedenleri [5]

Planlanmamış aksama süresini en aza indirebilmek için planlanmamış aksamalara yol açan nedenleri daha detaylı incelemek gerekmektedir. Planlanmamış aksama sürelerine yol açan başlıca neden %44 oran ile donanım arızalarıdır. Bunu %32 ile insan kaynaklı hatalar, %14 ile yazılım hataları ve %7 ile virüsler takip etmektedir. Planlanmamış aksamalara sebebiyet veren doğal felaket kaynaklı sorunlar ise tüm sorunların sadece %3 lük kısmını oluşturmaktadır. Planlanmamış aksamalara dair temel sorunların görülme sıklığı Şekil 2.4'te gösterilmiştir.



Şekil 2.4 Planlanmamış Aksamalara Yol Açan Başlıca Nedenler [2]

2.3 Yüksek Yararlanırlık (High Availability)

1950'li yıllarda üretilen bilgisayar sistemleri günlük ortalama 20 saat sorunsuz çalışabilme garantisi sunabiliyorlardı. Bir düzine bilgisayar mühendisinden oluşan bakım personeli arızalanan bir cihazı 8 saatte onarabiliyordu. Bu arızalanma – onarma çevrimi %70

yararlanırlık oranına denk geliyordu. Birkaç ay ömrü olan vakum tüpleri ve röle bileşenleri bu bilgisayarlarda meydana gelen problemlerin temel kaynağını oluşturuyordu. Bu nedenlerle bu bilgisayarlar nadiren bir günden daha uzun süre sorunsuz çalışmaktaydılar (Avizienis *et al.*, 1987).

Günümüzde kullanılan pek çok kusur saptama (fault detection) ve kusur maskeleme (fault masking) tekniği ilk defa bu eski bilgisayarlarda kullanılmıştır. Kendi kendini denetleyen (self-checking) hesaplama teknikleri, hesaplama işlemi devam ederken oluşan kusurları (fault) yakalar. Yazılımlar, durum bilgisini sıklıkla sağlam bölgelere kaydeder. Bir arıza (failure) durumu oluştuğunda, yazılım en son kaydedilen durum bilgisini okur ve hesaplama işlemlerine bu noktadan devam eder. Denetim noktası oluşturma ve kaldığı noktadan yeniden başlama tekniği, sıklıkla arızalanan cihazlarda uzun süreli hesaplamalar yapılabilmesini sağlamıştır.

Donanım gelişmeleri bilgisayar sistemlerinin yararlanırlığını artırmıştır. 1980'li yıllarda iyi çalışan tipik bir bilgisayar %99 yararlanırlık sunmaktaydı (Watanabe, 1985). Bu oran kulağa yeterli gibi gelse de her hafta yaklaşık 100 dakika aksama süresi (downtime) anlamına gelmektedir. Sistemin haftada 100 dakika kesintiye uğraması, bilgi girişini belirli süreler erteleyebilen çeşitli ticari ofis uygulamaları için tahammül edilebilir olsa da yaşamsal önem taşıyan veya çevrimiçi yazılımlar haftalık 100 dakika sistem kesintisiyle işlevsel olamazlar. Bu tip uygulamalar, %99.999 yararlanırlık sunan yüksek yararlanırlıklı sistemlere ihtiyaç duymaktadırlar. Bu, senede azami 5 dakika aksama süresi anlamına gelmektedir.

Yararlanırlık oranları önemlerine göre sınıflandırılabilirler. Düzenli olarak bakım yapılmayan bilgisayar sistemlerinin %90 yararlanırlık oranına sahip olduğu gözlenmiştir. Düzenli olarak bakımı yapılan bilgisayar sistemleri ise bir senelik dönem içerisinde çeşitli sürelerde hizmet veremeyebilir. Bu tip sistemlerde ortaya çıkan sistem arızalarının onarımı ortalama 2 saat sürmektedir. Buna göre normal bakımı yapılan bir sistemin yararlanırlık oranı %99 dur (Watanabe, 1985). 1980'lerin sonunda yer alan kusura dayanıklı bilgisayar sistemleri sadece birkaç senede bir arızalanmakta ve birkaç saat içerisinde onarılmaktaydılar. Bu sistemler %99,99 yararlanırlık oranına sahip idiler (Gray, 1987). Yararlanırlık ve yüksek yararlanırlık konusundan bahsedilirken %99,99 yararlanırlık oranı "dört dokuzluk" olarak da ifade edilebilmektedir. Buna göre "beş dokuzluk" denildiğinde %99,999, "altı dokuzluk" denildiğinde %99,9999 yararlanırlık oranı ifade edilmektedir. Sistemlerin yararlanırlık oranları göz önüne alınarak sınıflandırması Çizelge 2.1'de verilmiştir.

Çizelge 2.1 Sistemlerin Yararlanırlık Oranlarına Göre Sınıflandırılması (Gray ve Siewiorek, 1991)

Sistem Tipi	Yıllık Aksama Süresi	Haftalık Aksama Süresi	Yararlanırlık Oranı	Yararlanırlık Sınıfı
Bakımsız	876 saat	16,9 saat	90. %	1
Bakımlı	5256 dakika	101 dakika	99. %	2
İyi Bakımlı	525 dakika	10,1 dakika	99.9 %	3
Kusura Dayanıklı	52,5 dakika	1 dakika	99.99 %	4
Yüksek Yararlanırlıklı	5,25 dakika	6 saniye	99.999 %	5
Çok Yüksek Yararlanırlıklı	31,5 saniye	0,6 saniye	99.9999 %	6
Azami Yüksek Yararlanırlıklı	3,1 saniye	0,06 saniye	99.99999 %	7

Yararlanırlık sınıfı, yararlanırlık değeri içerisinde yer alan dokuzların sayısı olarak da algılanabilir. Sistemin yararlanırlık sınıfını ifade eden formül Şekil 2.5'de verilmiştir:

Sistemin yararlanırlık sınıfı = $\log_{10}^{(\frac{1}{1-A})}$

A: Yararlanırlık Oranı

Şekil 2.5 Bir Sistemin Yararlanırlık Sınıfının Bulunması (Gray ve Siewiorek, 1991)

Telefon ağları, yararlanırlık sınıfı 5 olan yüksek yararlanırlıklı sistemlere güzel bir örnektir. Bu tür sistemlerin tasarımında 40 yıllık bir zaman diliminde en fazla 2 saat aksama süresi ön görülmektedir. Ancak, aşırı yük binmeleri veya doğal afetler sonucunda oluşan kopmalar bu öngörünün gerçekleşmemesine neden olabilmektedirler. Bu örnek, sistemleri yüksek yararlanırlıklı yapabilmenin zorluğunu ortaya koyar. Bir sistemin yararlanırlığını etkileyen faktörler şunlardır:

- Sistem bileşenlerinin bireysel güvenirlikleri ilk faktörü oluşturmaktadır. Sunucu donanımı, sunucunun işletim sistemi ve uygulamanın kendisi, güvenliğine dikkat edilmesi gereken temel bileşenleri oluşturmaktadır. Sistemdeki diğer bileşenler veri saklama, ağ erişimi gibi çevresel bileşenler olabileceği gibi veritabanları, dosya sistemleri veya bilgi işlem altyapıları da olabilir.

- Bir arıza oluşması durumunda, uygulamanın ve sistemi eski haline döndürme işleminin alacağı süre ikinci faktörü oluşturmaktadır. Uygulamanın yeniden işler hale getirilmesi için gereken süre, arızalanan bileşene göre değişkenlik gösterir. Uygulamanın kendisinin arızalanması durumunda uygulamanın kendisini yeniden başlatmak bazen tek çözümdür.

Bir uygulama bir donanım arızası nedeniyle bozulduğunda, sistemin eski haline getirilmesi şu adımları kapsar:

1. Arızalanan bileşenin desteğini sunan hizmet sağlayıcının bilgilendirilmesi
2. Onarım teknisyeninin gelmesinin beklenmesi
3. Arızalanan bileşenin belirlenmesi
4. Arızalanan bileşenin değiştirilmesi
5. İşletim sisteminin yeniden başlatılması
6. Dosya sisteminin onarılması
7. Veritabanının onarılması
8. Ağ yazılımının yeniden başlatılması
9. Uygulamanın yeniden başlatılması

Sistem üreticileri satılan sistemlerde belirli oranlarda yararlanırlık garantisi sunmaktadır. Fakat sundukları yararlanırlık, satılan donanımın çalışması ve üzerinde bir işletim sistemi çalıştırılabilmesine yöneliktir. Kullanıcı tarafından algılanan yararlanırlık her zaman daha düşük olacaktır. Yüksek yararlanırlık, bir sistemin kullanıcı tarafına servis verebilmesi için gerekli tüm bileşenlerin yüksek yararlanırlıklı olmasına bağlıdır. Bazı bileşenlerin sunduğu servis yüksek yararlanırlıklı olsa da sistemde yer alan diğer bileşenler aynı oranda yüksek yararlanırlık sağlayamadığı sürece toplam yararlanırlık azalmış olacaktır.[2]

Yüksek yararlanırlıklı sistemler kusurlara tahammül edebilecek, kusurları tespit ve rapor edebilecek, kusurları maskeleyerek sorunlu bileşenler çevrimdışı onarılırken sistemin hizmet

verebilmesine olanak sağlayacak şekilde tasarlanmış olmalıdır. Yüksek yararlanırlıklı bir sistem olağan dışı kusurlar dışında aşağıdaki problemlerin üstesinden gelebilmelidir:

- Elektrik kesintileri
- Yazılım güncelleştirmeleri ve onarımları
- Veritabanı yeniden yapılandırmaları
- Operatör kusurları

Elektrik Kesintileri: Elektrik sistemlerinin kesintiye uğraması, bilgisayar sisteminin tamamen durmasına sebebiyet vereceğinden bu problemin önüne geçilecek düzenekler kurulmalıdır. Kesintisiz güç kaynağı ve jeneratörler bu sorunu aşmak için kullanılan cihazlardır. Kesintisiz güç kaynakları elektrik şebekesini kendi üzerinden aktararak elektrik kesintisinin bilgisayar sistemleri tarafından hissedilmemesini sağlarlar. Kesintisiz güç kaynağının depo kapasitesine ve kullanılan bilgisayar sistemlerinin ihtiyaç duyduğu elektrik yüküne bağlı olarak bu cihazların elektrik kesintisi süresince dayanabileceği belirli süreler vardır. Bu süre zarfında elektrik şebekesi gelmediği takdirde elektrik kesintisinin bilgisayar sistemlerince hissedilmemesi için jeneratör veya jeneratörlerin devreye alınması gerekir.

Yazılım Güncelleştirmeleri ve Onarımları: Günümüzde kullanılan pek çok bilgisayar sisteminde başta veritabanı uygulamaları olmak üzere çok çeşitli yazılımlar kullanılmakta ve bu yazılımların bazen bakımı veya güncelleştirilmesi gerekmektedir. Bu durum senede en az bir defa ortaya çıkmaktadır. Buna göre, oluşturulan düzenek, yazılım güncelleştirmeleri sırasında sistemin yararlanırlığının devam etmesini sağlamalıdır.

Veritabanı Yeniden Yapılandırmaları: Kullanılan veritabanlarına yeni bilgi veya bilgi tiplerinin eklenmesi gerektiğinde veya veritabanının efektif kullanılabilmesi için yeniden organize edilmesi gerektiği durumlarda bilginin erişilebilir olması ve aynı zamanda değişikliklerin arka planda çalışan depolama alanlarına uygulanması gerekebilmektedir. Bu tip yeniden organize olma durumları senede bir veya birkaç defa tekrarlanabilir. 1990'lardan itibaren bu tip değişikliklere imkân tanıyan çevrimiçi uygulamalar geliştirilmiş ve genel amaçlı olarak hizmete sunulmuştur.

Operatör Kusurları: Sistemleri kullanan veya yöneten operatörlerin hataları sistemlerin yararlanırlığının azalmasına veya erişilememesine neden olabilmektedir. Bu tip hatalar

sistemlerin birkaç saate kadar erişilemez duruma gelmesine sebebiyet verebilir. Yüksek yararlanırlıklı sistemlerin bu tip hatalara karşı da dayanıklı olması gerekmektedir.

Bu kusurlar, sistem aksamalarının büyük kısmını oluşturmaktadır. Bu kusurlara dair önlem alınan bakımlı sistemler, bakımsız sistemlere oranla çok daha yüksek yararlanırlık desteği verirler (Bkz: Şekil 2.1).

2.4 Kusura Dayanıklılık (Fault Tolerance)

Kusura dayanıklılık, kusur durumu ortaya çıktığında kimseyi rahatsız etmeden hizmetini vermeye devam edebilen sistemler oluşturma bilimi ve sanatıdır. Kusura dayanıklı bir sistem şu kusurların birini veya birden fazlasının üstesinden gelebilir olmalıdır:

- Geçici, aralıklı veya kalıcı donanım kusurları
- Yazılım ve donanım tasarım hataları
- Operatör hataları
- Dış kaynaklı bozulma veya fiziksel zararlar

Kusura dayanıklılık konusu üzerine son 30 yılda yapılan çalışmalarda geniş kapsamlı yöntembilimler ortaya konulmuş, çoğunluğu donanım kusurlarının geriye kalan az bir miktarı da yazılım, tasarım ve operatör kusurlarının üstesinden gelen kusura dayanıklı sistemler geliştirilmiştir. Yine bu konu üzerinde çok sayıda araştırma yapılmış ve raporlanmıştır.

Kusura dayanıklı ve güvenilir sistem araştırmaları gerçek zamanlı gömülü sistemler, ticari işlem gören sistemler, taşımacılık sistemleri, askeri ve uzay bilim temelli sistemler gibi çok geniş bir alanda uygulamaya sahiptir. Bu araştırmalar sistem mimarisi, tasarım teknikleri, kodlama teorisi, test etme, doğruluk sağlama, doğruluk ispatı, modelleme, yazılım güvenilirliği, işletim sistemleri, eş zamanlı işleme ve gerçek zamanlı işleme özelliklerini kapsamaktadır. Bu konular genellikle biçimsel mantık (formal logic), rastgele modelleme matematiği (mathematics of stochastic modeling), graf teorisi (graph theory), donanım tasarımı ve yazılım mühendisliği gibi çeşitli can alıcı konulardan yararlanmaktadır (David A.Rennels, 2004).

2.4.1 Donanımsal Kusura Dayanıklılık

Bilgisayar sistemlerinde oluşan problemlerin büyük çoğunluğu donanım kaynaklı kusurlar olduğundan, kusura dayanıklı sistem tasarımlarının büyük çoğunluğu donanımsal bileşenlerde oluşan rastlantısal kusurların kendiliğinden onarıldığı bilgisayarlar üzerine yapılmıştır. Bu sistemlerin temel prensibi, bilgisayarın kusur çıkabilecek bölgelerinin birbirinden ayrı bileşenlere ayrılmasını öngören tekniklerdir. Birbirinden bağımsız her bir bileşen koruyucu yedekler ile desteklenmiştir ve bir bileşen sorun çıkardığında diğer bir yedek onun görevini yürütmeye başlar. Bu tekniklere, hataları tespit edebilen ve toparlanmayı sağlayan özel düzenekler eklenmiştir. Donanımsal kusura dayanıklılık üzerine genel olarak iki yaklaşım söz konusudur:

- Kusur Maskeleme (Fault Masking)
- Dinamik Toparlanma (Dynamic Recovery)

Kusur Maskeleme: Kusurların bir yedek bileşenler seti içerisinde maskelenmesini sağlayan yapısal bir yedeklilik tekniğidir. Birden fazla sayıda özdeş parça tamamen aynı anda aynı görevi yerine getirerek çıktı üretirler. Bu özdeş parçaların ürettiği çıktılar oylanarak kusurlu parçanın ürettiği hatalı bilginin ortadan kaldırılması sağlanır. Üç parçalı yedeklilik (Triple Modular Redundancy)(TMR) kullanılan devrelerin oluşturduğu oylama, kusur maskeleme içinde en çok kullanılan yöntemdir. Bunun yanında oylama düzeneği de üç parçalı yedeklilik ile desteklendiğinde, oylama sonuçlarında ortaya çıkabilecek kusurların da önüne geçilmesi sağlanır. Bir TMR sistemi, ancak üç parçadan ikisinin hatalı sonuç ürettiği zamanlarda hatalı sonuç üretir. Üç parçadan ikisi hatalı sonuç ürettiğinde oylama işleminin sonucu da geçerli olamayacaktır. Karma yedeklilik (Hybrid Redundancy) TMR düzeneğindeki her bir parçanın yedeklerle desteklenerek biraz daha kusura dayanıklı hale getirilmesine yönelik bir yaklaşımdır. Oylama temelli bu sistemler yedekliliği olmayan sistemlere oranla üç kat donanım gerektirmesinin yanında herhangi bir kusur oluştuğunda işlemlerin ve sistemlerin herhangi bir arızaya ve kesintiye uğramadan devam edebilmesi avantajını getirirler.

Dinamik Toparlanma: Dinamik toparlanma, kusur maskeleme tekniğinde olduğu gibi bilgisayarın kusur çıkabilecek bölgelerinin birbirinden ayrı bileşenlere ayrılmasını öngören ve her bir parçanın yedeklerle desteklenmesini sağlayan bir tekniktir. Bu teknik kullanılacağı

zaman kusurları tespit edecek, kusurlu parçayı devre dışı bırakıp yerine yedeklerden birini devreye alacak özel mekanizmalar gereklidir. Bu mekanizmalar aynı zamanda hesaplama işlemlerinin kaldığı yerden devam edebilmesi için gerekli olan geriye sarma, ilk kullanıma hazırlama, işlemi yeniden deneme ve işlemi yeniden başlatma konularında yazılıma ön ayak olmalıdır. Tek bilgisayardan oluşan sistemlerde bu işlemleri yerine getirecek özel yazılım ve donanımlar gerektirirken, çok bilgisayardan oluşan sistemlerde bu düzenek diğer işlemciler ile yürütülür.

Dinamik toparlanma, genel olarak oylama temelli sistemlere oranla donanımı daha verimli kullanır. Bu nedenle düşük güç gerektiren sistemler gibi kaynak sınırlamaları olan alanlarda tercih edilmektedir. Ancak kusur toparlanması sırasında işlemlerin bekletilmesi, kusurların kapatılma başarısının oylamaya dayalı sistemlere oranla düşük olması ve özel işletim sistemi gerektirmesi temel dezavantajlarıdır.

2.4.2 Yazılımsal Kusura Dayanıklılık

Yazılım tasarımlarından doğan kusurlara karşı dayanıklılık sağlanabilmesi için üretilen yazılımlar, donanımsal kusura dayanıklılık için kullanılan statik ve dinamik yedeklilik yaklaşımlarının benzeri özellikler gösterirler. Sürüm programlama (version programming) adı verilen bir yaklaşım, aynı görevi gören ve birbirinden farklı sürüme sahip yazılımların birbirinin yedeği olarak çalışmasını ve bu farklı sürümlerin aynı anda çalıştırılmasıyla elde edilen sonuçların özel denetim noktalarında oylanması yöntemini kullanır. Bu teknik kullanıldığında farklı sürümlerden farklı sonuçlar elde edilmesi durumu, oluşan sonuçlardan birinin hatalı olması anlamına gelebileceği gibi farklı sürümlerin doğal olarak farklı sonuçlar üretmesinden de kaynaklanabilir. Bu nedenle bu teknik kullanıldığında, donanımsal kusura dayanıklılık başlığı altında incelenen oylama tekniğinden farklı olarak bazı kıstasların kullanılması, bu kıstaslar sayesinde kusurlu sürümlerin belirlenmesi, reddedilmesi ve tutarlı değerler oluşturularak tüm sürümlerin bu değerleri kullanması sağlanır.

Dinamik toparlanma olarak incelenebilecek bir diğer teknik ise programların toparlanabilir bloklar halinde işlenerek, işlenen her bloktan sonra sonuçların kabul edilebilirlik testlerinin yapılmasıdır. Kabul edilebilirlik testlerinden birinin kusur bulması durumunda, yedek kod bloğu çalıştırılarak üretilen sonuçlar, kusurlu program bloğunun ürettiği sonuçların yerine kullanılır.

2.4.3 Tasarım Çeşitlemesi (Design Diversity)

Tasarım çeşitlemesi adı verilen bir başka yaklaşım, donanımsal ve yazılımsal kusura dayanıklılık tekniklerinin bir arada kullanıldığı, çeşitli donanım ve yazılımların yedekli kanallar üzerinde kullanılarak kusura dayanıklı sistemler oluşturulmasını öngörür. Tamamen aynı görevi yerine getirmekle yükümlü olarak tasarlanmış kanallar ve diğerlerinden farklı davranan kanal veya kanalları tespit etmek üzere tasarlanmış bir metottan oluşan bu yaklaşım, aynı anda hem donanım hem de yazılım temelli kusurlara dayanıklılık sağlama hedefini taşımaktadır. Bu yaklaşım oldukça yüksek maliyet getirdiğinden her alanda kullanılmamakta, ancak kritik hava aracı kontrol uygulamaları gibi çok yüksek önem taşıyan sistemlerde kullanılmaktadır.

2.4.4 Kusur – Hata – Arıza (Fault – Error – Failure)

Kusur, arıza ve hata, kusura dayanıklılık konusu içerisinde yer alan ve anlamları birbirine çok yakın gibi dursa da birbirilerinden farklı anlam ve kullanım alanı olan kavramlardır. Bir bilgisayar sisteminde birçok parça bulunmaktadır. Her parça için öngörülen bir tanımlanmış davranış biçimi olduğu gibi, her bir parçanın çalıştığında gözlemlenen gerçek davranışları vardır. Bir parçanın gözlemlenen gerçek davranışı, o parçaya dair tanımlanmış davranış biçiminden saptığı zaman bir arıza (failure) durumu ortaya çıkar. Bu arıza durumu, parçanın içerisinde bir hata (error) oluştuğu için meydana gelmiştir. Hatanın nedeni ise bir kusur (fault) tur. IEEE'nin standartlarına [3] göre yapılan tanımlamalar şu şekildedir:

Kusur: Bir bilgisayar programı içerisinde yer alan yanlış bir adım, işlem veya veri tanımı.

Hata: Hesaplanmış, gözlemlenmiş, ölçülmüş değer veya durum ile olması gereken, daha önceden tanımlanmış, teorik olarak doğru değer veya durum arasındaki fark. Örnek: 30 metrelik bir mesafe değeri için hesaplanan ve ölçülen değerler arasındaki fark.

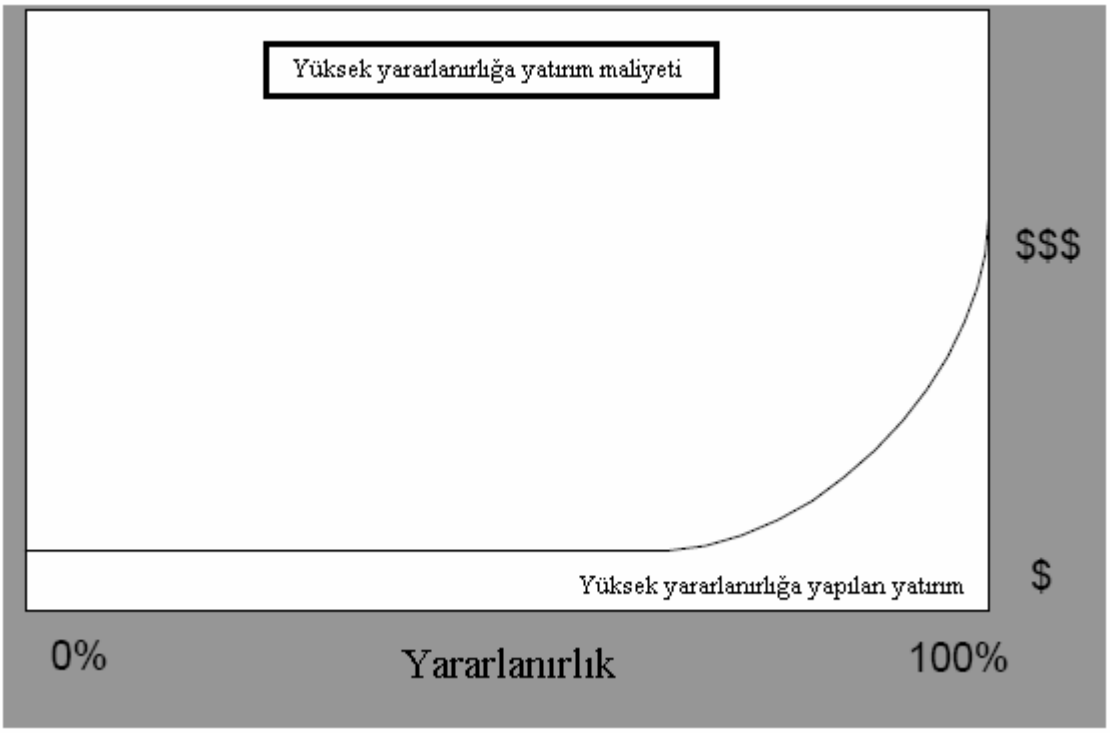
Arıza: Bir sistemin, tanımlanmış performans gereksinimleri içerisinde görevini yerine getirememesi durumu.

Bu üç kavramın kusura dayanıklılık konusu içerisindeki yerini daha iyi anlamak için bir örnekten yararlanılabilir. Bir bilgisayar donanımı içerisinde normal koşullarda olması beklenmeyen bir sorun oluştuğunu varsayalım. Örnek olarak bellek birimi içindeki bir alanda yer alan bir verinin bir biti hatalı yazılmış olsun. Bellekteki hatalı bilgiyi kullanan program parçası, kendisi bir sorun oluşturmadığı halde hatalı bir işlem sonucu üretecektir. Örnek olarak ürettiği değer 0 olsun. Bu hatalı işlem sonucunu kullanan program ise bir bölme işlemi yapıyor ve aldığı bu değeri bölen olarak kullanıyor olsun. Bu örnekte bölen değeri 0 olarak geldiğinden ve 0'a bölme yapılamayacağından, program "sıfıra bölme hatası" vererek yaptığı işlemi yapamaz hale gelecek ve program kapanacaktır.

Bu örneği ele aldığımızda, bellekteki bilginin bir bitinin bozulması bir Kusur (fault), bu bilgiden dolayı üretilen hatalı bilginin programın sıfıra bölme hatası yaptırması bir Hata (error) ve programın işlevini yerine getiremez hale gelmesi ise Arıza'dır (failure).

2.5 Yüksek Yararlanırlık ile Kusura Dayanıklılık Karşılaştırması

Yüksek yararlanırlık ve kusura dayanıklılık konuları birbiri ile bağlantılı konular olsalar da temelde birbirinden farklı iki kavramdır. Aradaki en temel farkın altını çizmek gerekirse yüksek yararlanırlık bir sistemin mümkün olduğunca yüksek yararlanırlık sunmasını hedeflerken kusura dayanıklılık bu hedeften çok daha zoru, yani kusurların sistemin akışını kesmesini engellemeyi hedefler. Birçok yüksek yararlanırlıklı sistem, bazı bileşenlerinde kusura dayanıklılık özelliğine sahiptir. Sistemin yararlanırlığını artırmaya yönelik adımlar attıkça sistemin maliyeti de artmaktadır. Şekil 2.6 yüksek yararlanırlık kavramı için maliyet-yarar analizini göstermektedir.



Şekil 2.6 Yüksek Yararlanırlık Maliyet-Yarar Analizi [2]

Tamamıyla kusura dayanıklı bir sistemden bahsedileceği zaman, sistemde var olan her türlü bileşenin en az bir yedeğinin olmasını öngörmektedir. Bunun sonucu olarak kusura dayanıklı sistemlerin maliyeti çok daha yüksektir. Dolayısıyla kusura dayanıklı sistemler, ancak maliyet-yarar dengesinin sağlanabildiği hava kontrol sistemleri, acil durum sistemleri gibi herhangi bir aksaklığın büyük sorunlar yaratabileceği kritik uygulamalarda kullanılmaktadırlar.

3. ÇÖZÜM MİMARİLERİ

Yüksek yararlanırlık yolunda son yıllarda büyük gelişmeler kaydedilmiş, günümüzde kullanılan çözümlerin çoğunda bilgi ikizlemesi (mirroring) teknikleri akıllı sistem yönetimi yazılımları ile desteklenerek büyük bir yüksek yararlanırlık çatısı altında bulunan bilgisayar kümeleri ile depolama alanları birbirinden ayrılmış ve yıkım onarımı olanakları sağlanmıştır. Yüksek yararlanırlık sağlayan sistemleri işleyişine göre ve tasarımına göre sınıflandırmak mümkündür.

- İşleyişine Göre Sistemler
    - Dağıtık Sistemler (Distributed Systems)
    - Paralel Sistemler (Parallel Systems)
    - İhtiyaca Bağlı Sistemler (On Demand Servers)
- Tasarımına Göre Sistemler
    - Yüksek Yararlanırlıklı Grup Sistemler (Clusters)
    - Yedeğe Geçmeli Sistemler (Failover Systems)

3.1 İşleyişine Göre Sistemler

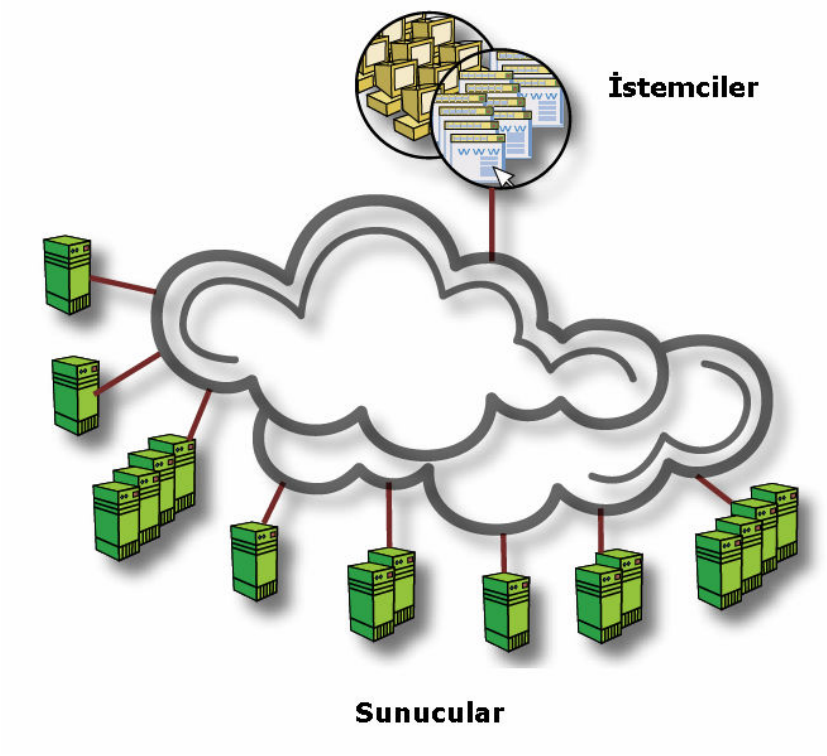
Yüksek yararlanırlık sağlayan sistemler işleyişine göre üç kategoriye ayrılabilirler:

- Dağıtık Sistemler (Distributed Systems)
- Paralel Sistemler (Parallel Systems)
- İhtiyaca Bağlı Sistemler (On Demand Servers)

3.1.1 Dağıtık Sistemler

Dağıtık sistemlerde bilgi aynı grup içerisinde bulunan birden çok bilgisayar arasında dağıtılmıştır. Bilgi üzerindeki değişiklikler normalde yalnızca bir bilgisayar üzerinde yapılır ve bu değişiklikler zaman içerisinde diğer bilgisayarlara da yansıtılır. Bu yapıyı günümüzde

Tucows [14] gibi indirilebilir dosyalar için bilgi ikizlemesi yapan yazılım dağıtıcılarında görmek mümkündür. Şekil 3.1'de dağıtık sistemlerin genel yapısı gösterilmektedir.



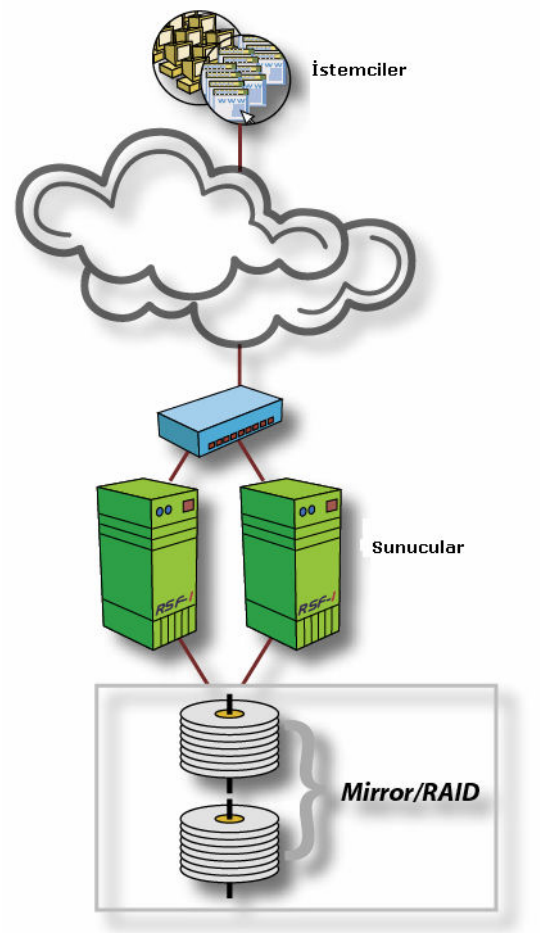
Şekil 3.1 Dağıtık Sistemler [15]

Dağıtık sistemler, düşük maliyetle yüksek yararlanırlık sunan ve büyük çapta bir arızaya karşı esneklik sağlayan sistemlerdir. Bunun yanında, aynı bilgi üzerinde 2 farklı bilgisayar üzerinde eş zamanlı olarak değişiklik yapıldığı takdirde bilgi uyuşmazlığı problemi ortaya çıkar. Bu sorun tipik bir okuyucular/yazıcılar (readers/writers) problemi olarak ele alınarak çözülebilir. Bilgi değişikliği konusunda bir bilgisayar yazma yönünde haklara sahip olurken diğer bilgisayarlar okuma haklarına sahip olurlar. Bu durumda ise bilgi yazma hakkına sahip olan bilgisayarın devre dışı kalması sorunlara yol açabilir.

3.1.2 Paralel Sistemler

Birden çok bilgisayarın oluşturduğu bir grup, bilgi işleme hareketlerini eş zamanlı olarak yürütürler. Bu yapının dağıtık sistemlerden farkı, bilgisayarlar bilgiyi eş zamanlı olarak işler iken koordine edilmiş bir kilitleme mekanizması kullanırlar ve böylelikle bilgi çakışması problemi yaşanmaz. Günümüzde sadece güçlü firmalar gerçek anlamda paralel hesaplama

imkânı sunan sistemler sunmaktadırlar. Şekil 3.2'de paralel sistemlerin yapısı gösterilmektedir.



Şekil 3.2 Paralel Sistemler [15]

Teorik olarak, paralel sistemler gelişmiş ölçeklenebilirlik ve performans faydası sağlarlar. Özel çözümlere bağlı olarak, sunucular yan yana konumlandırılmak ve hatta aynı disk birimlerini paylaşmak zorunluluğundan kurtulabilmektedirler. Bunun yanında sistemin performansının önem taşıdığı çözümlerde sunucular arasındaki bağlantıların yan yana konumlandırılmış kadar hızlı ve düşük bekletme süreli olması gerekmektedir. Bu tip durumlarda özel donanımlar gerektiren paralel sistemler, uygulama veya işletim sistemi seviyesinde veya ara katmanlar vasıtasıyla gerçeklenebilir.

3.1.3 İhtiyaca Bağlı Sistemler

Hazır durumda bulunan sistemlerden sadece bir tanesi işlemlerin tamamını yüklenirken, diğer sistemler aktif sistemde bir problem çıkması durumunda iş yükünü devralmak üzere beklerler.

Bu tip sistemler, aktif sunucu yanında konumlandırılmış acil yedek sunucular ve her sunucuya ait ayrı depolama alanında saklanan ikizlenmiş veriler ile sağlanabilir.

3.2 Tasarımına Göre Sistemler

Yüksek yararlanırlık sağlayan sistemler tasarımlarına göre iki kategoriye ayrılabilirler:

- Yüksek Yararlanırlıklı Grup Sistemler (Clusters)
- Yedeğe Geçmeli Sistemler (Failover Systems)

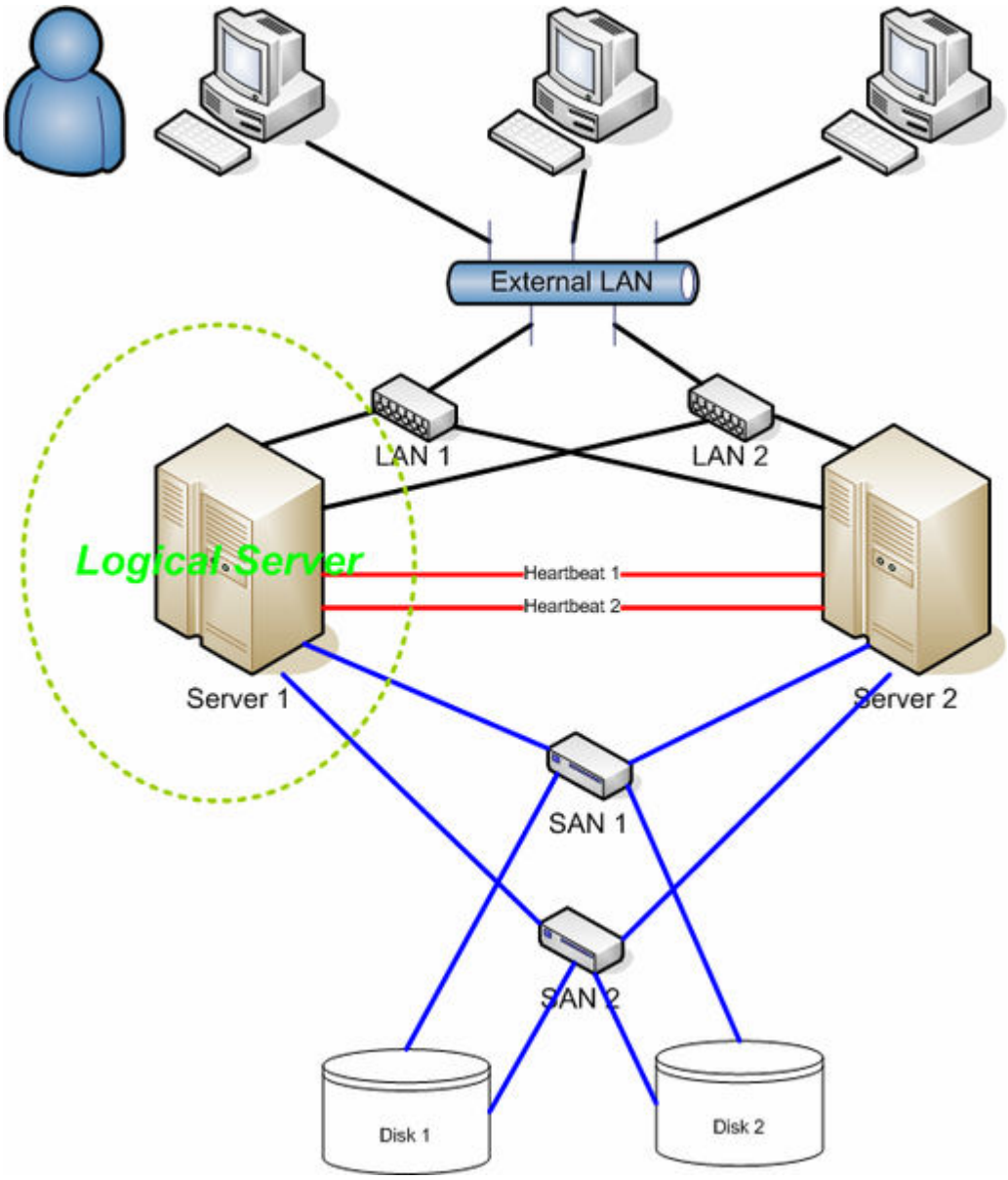
3.2.1 Yüksek Yararlanırlıklı Grup Sistemler

Grup sistemler, birden çok bilgisayarın aynı iş yükü için bir arada bulunduğu ve dışarıdan bakıldığında büyük tek bir bilgisayar gibi algılanan büyük yapılardır. Günümüzde, grup sistemler içerisindeki bilgisayarlar çoğunlukla yerel ağ bağlantısı üzerinden birbirilerine bağlanmaktadırlar. Grup sistemlerin kullanım amacı daha çok, tek bilgisayardan elde edilebilen hız ve güvenilirliği artırmaktır.

Yüksek yararlanırlıklı grup sistemlerin hedefi, grup sistemin sunduğu servisin daha yüksek yararlanırlıklı olabilmesini sağlamaktır. Bu sistemlerde, sistemde bulunan parçaların birer yedekleri bulunmakta ve arızalanan parçanın yerine yedeği geçmektedir. Tipik bir yüksek yararlanırlıklı grup sistem iki adet bilgisayardan oluşur. İki bilgisayar bu tip bir sistem oluşturulacağı zaman gerekli olan minimum sayıdır. Çok sayıda sunucunun aynı anda kontrolünün yapıldığı ve kusur tespit edilmesi durumunda kusurlu birim üzerindeki yükün başka bir sunucuya aktarılmasını sağlayan yazılım temelli mekanizmalara sahiptir.

Yüksek yararlanırlıklı grup sistemlere örnek teşkil eden çok sayıda ticari uygulama mevcuttur. IBM HACMP [18], Veritas Clustering Services [19] bunlardan bazılarıdır. Bunların yanında Linux-HA [20] adı verilen ve Linux işletim sistemini kullanan bilgisayarlar arasında yüksek yararlanırlıklı grup sistemi kurmaya olanak sağlayan ve tamamen açık kaynak kodlu bir proje GPL lisansı altında mevcuttur.

Tipik bir yüksek yararlanırlıklı grup sistem şeması Şekil 3.3'te gösterilmiştir:



Şekil 3.3 Yüksek Yararlanırlıklı Grup Sistemi [16]

Tipik bir yüksek yararlanırlıklı grup sistemi iki adet sunucudan oluşsa da sunucu sayısının çok daha fazla sayıları bulduğu uygulamalar da mevcuttur. Çok sayıda sunucunun bulunduğu grup sistemleri, yapılandırmalarına göre dörde ayrılırlar:

- Aktif / Aktif
- Aktif / Pasif
- N + 1
- N + M

Aktif / Aktif: Bütün sunucular aktif durumda çalışırlar. Bir arıza durumunda, arızası olan sunucunun üzerindeki yük diğer sunuculara aktarılır. Bu yapılandırmayı kullanabilmek için yük dağılımını sağlayan özel bir mekanizmaya ihtiyaç vardır. Yedeğe geçmeli sistemler içerisinde bu yapılandırmanın adı simetrik yedeğe geçmeliliktir.

Aktif / Pasif: Sunucular için tam yedekli bir yapılandırmadır. Aktif durumda çalışan her sunucu için boşta pasif halde bekleyen bir adet yedek bulunmaktadır. Aktif sunucuda arıza olduğunda yedeği görevi devralır. Bu yapılandırma ihtiyaç duyulan donanım miktarını iki katına çıkarır. Yedeğe geçmeli sistemler içerisinde bu yapılandırmanın adı asimetrik yedeğe geçmeliliktir.

N + 1: N adet çalışan aktif sunucuya karşılık 1 adet boşta bekleyen yedek sunucu vardır. Aktif sunuculardan herhangi birinde oluşan bir arıza da yedek sunucu arızalı sunucunun görevini devralır. Bu yapılandırmada, yedek sunucu, aktif haldeki sunucuların herhangi birinin işini yapabilir yetenekte olmalıdır.

N + M: Gerçek hayat uygulamalarında N adet aktif sunucunun yer aldığı bir grup sistemde, 1 adet yedek sunucu yeterli olamadığı için, birden fazla olmak kaydıyla M adet yedek sunucunun konumlandırıldığı bir yapılandırmadır. Yedek sunucu adedi, maliyet-fayda analizine göre belirlenir.

3.2.2 Yedeğe Geçmeli Sistemler (Failover Systems)

Yedeğe geçmeli sistemler, bir arıza durumunda yedek sisteme geçiş yapılabilen sistemlerdir. Bunun için sistemde bulunan her bir parçanın bir yedeğinin bulunması gerekir. Yedeğe geçmeli sistemler, hedefi ve hızlarına göre dörde ayrılırlar:

- Yedeksiz (No Failover)
- Elle Yedekli (Cold Failover)
- Sıcak Yedekli (Warm Failover)
- Acil Yedekli (Hot Failover)

Çizelge 3.1 Yedeğe Geçmelilik Sınıfları [17]

HEDEF	ONARIM ZAMANI	MASRAF	KULLANICIYA ETKİSİ
Yedeksiz	Tahmin edilemez	Sıfır ya da az	Yüksek
Elle Yedekli	Dakikalar	Orta	Orta
Sıcak Yedekli	Saniyeler	Orta seviye ve yukarı	Düşük
Acil Yedekli	Anında	Orta seviye ve yukarı	Sıfır

Yedeksiz (No Failover): Yedeksiz sistemlerin arızaya karşı savunması yoktur. Arıza oluştuğunda kullanıcılar sistemden yararlanamaz. Sorunun çözüm zamanı tahmin edilemeyeceği gibi kullanıcı tarafına etkisi büyüktür. Maliyeti düşüktür.

Elle Yedekli (Cold Failover): Bu sistemlerde, aktif cihazın arızalanması durumunda görevi devralmak üzere bekleyen bir yedek bulunur. Arıza oluştuğunda yedek cihaz, arızalanan cihazın sunduğu servisleri vermek üzere gerekli uygulamaları devreye alır. Servis uygulamalarının yedek sunucuda yeniden başlatılması işlemleri nedeniyle onarım zamanı dakikalar seviyesindedir ve kullanıcıya etkisi orta seviyededir.

Sıcak Yedekli (Warm Failover): Bu sistemlerde, arızalanan cihazı tespit eden ve yerine yedeği devreye alan otomatik bir mekanizma mevcuttur. Problem, bilgisayar sistemleri ile otomatik çözümlendiğinden işlemler saniyeler mertebesinde çözüme kavuşur, kullanıcı tarafı sorundan çok az etkilenir.

Acil Yedekli (Hot Failover): Bu tip sistemler daha çok donanım seviyesinde oluşan arızaların onarılması şeklindedir. Örneğin işlemciler üretilirken transistor seviyesinde lojik düzenlemeler yapılarak arıza kanalları örtülebilir hale getirilir. Bu tip sistemlerin sorunu çözme süresi kullanıcıya hiçbir şey hissettirmeyecek kadar düşüktür.

Bu özellikler dışında yedeğe geçmeli sistemleri bağlantı durum (state) bilgisine bağlı olarak 2 kategoriye ayırmak mümkündür:

- Durum Bilgili Yedeğe Geçme (Stateful Failover)
- Durum Bilgisiz Yedeğe Geçme (Stateless Failover)

3.2.2.1 Durum Bilgili Yedeğe Geçme (Stateful Failover)

Yedeğe geçmeli sistemlerde hizmet veren birincil cihazda bir arıza meydana geldiğinde, bu aktif cihaz üzerinden hizmet almakta olan kullanıcıların bağlantı ve oturum bilgisi, arızalı birincil cihazın yerine geçecek olan yedek cihaza aktarılıyor ve kullanıcı bu yedeğe geçme işleminden etkilenmiyor ise kullanılan yedeğe geçme sistemine durum bilgili yedeğe geçme adı verilir. Özellikle yedekli güvenlik duvarı sistemlerinde aktif cihazın üzerinden geçen bağlantı bilgisinin yedek cihaza aktarılması, özellikle büyük çaptaki ağlar için çok yarar sağlayan bir özelliktir. Bu tip sistemlere örnek olarak Cisco PIX Firewall cihazları verilebilir. Yazılım sürümü 5.0'dan yukarı olan Cisco PIX cihazlarında durum bilgili yedeğe geçme özelliği desteklenmektedir.

3.2.2.2 Durum Bilgisiz Yedeğe Geçme (Stateless Failover)

Durum bilgili yedeğe geçmeli sistemlerde uygulanan "arıza anında durum ve bağlantı bilgilerinin yedek cihaza aktarılması" işlemi durum bilgisiz yedeğe geçmeli sistemlerde göz ardı edilir. Durum ve bağlantı bilgilerinin yeni cihaza aktarılmaması, aktif cihaz üzerinden geçmekte olan tüm bağlantıların yarım kalması ve kullanıcının yaptığı işleme yeniden başlamasını gerektirebilir. Örnek olarak kullanıcı web sayfası üzerinden oturum açmış ve e-postalarını kontrol ediyor olsun. Kullanıcının haberi olmaksızın güvenlik duvarı bileşenlerinde gerçekleşen bir durum bilgisiz yedeğe geçme işlemi kullanıcının oturum ve bağlantı bilgisinin yok olmasına neden olurken, kullanıcının Internet gezgini programının e-posta sunucusuna yeni bir oturum açmasını gerektirecektir.

4. LINUX İŞLETİM SİSTEMİ

Linux, UNIX işletim sisteminin minik bir uyarlaması olan MINIX işletim sistemi incelenerek ve onun standartları geliştirilerek 1991 yılında Finlandiya Helsinki Üniversitesi'nde bir öğrenci olan Linus Torvalds tarafından geliştirilmiştir. Linus Torvalds, 1991 yılında Linux üzerinde çalışmaya başlamış ve aynı sene içinde 0.02 sürümünü yayımlamıştır. Üç yıl süren çalışmaların ardından 1994 yılında Linux Kernel 'in 1,0 sürümü ortaya çıkmıştır. Linux 'un merkezini oluşturan açık kaynak kodlu Kernel, GNU General Public License altında geliştirilmiş ve piyasaya sürülmüştür. Bu Kernel, Linux işletim sisteminin geliştiği temeli oluşturmaktadır.[6]

International Data Corporation'ının yaptığı araştırmaya göre Linux işletim sistemi 2003 yılında marketin %6'sına sahipken, 2008 yılında %17'sine sahip olacağı tahmin edilmektedir. Araştırma ayrıca Linux'a olan ilginin artışına dikkat çekiyor. Bunun altında yatan neden olarak ise sağlam bir işletim sistemi olan UNIX' e alternatif olacak özelliklere sahip olması gösteriliyor.

Linux başlangıçta Intel tabanlı bilgisayarlarda çalışmak için tasarlanmış olsa da, modüler ve taşınabilir kodlanabilirliği 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.

4.1 Linux İşletim Sisteminin Diğer İşletim Sistemleri ile Karşılaştırması

Linux işletim sisteminin diğer işletim sistemleri ile yapılan karşılaştırmalarında alınan araştırma sonuçları başlıklar halinde incelenmiştir.

4.1.1 Güvenilirlik ve Kararlılık

IBM Linux Teknoloji Merkezi tarafından gerçekleştirilen testte Linux işletim sisteminin güvenilirliği ve kararlılığı ölçülmüştür. Test ortamında donanım olarak IBM pSeries,

sunucular ve işletim sistemi olarak SUSE Linux Enterprise v8 kullanılmıştır. Yapılan 30 günlük stres testinde başarı %97,88 olarak ölçülmüştür. 60 günlük test ise %95,12 başarı oranı ile tamamlanmıştır (Ge *et al.*, 2003).

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.

Yapılan araştırmalar Linux'un aylarca çalıştığında bile hafıza yönetimi nedeniyle yaşanacak kilitlenme sıkıntısı yaşamadığını ve yavaşlama belirtisi göstermediğini ortaya koymuştur.

4.1.2 Performans

InformationWeeks Research tarafından gerçekleştirilen testte Windows Server 2003 DataCenter Edition üzerinde çalışan SQL Server Enterprise Edition ile SUSE Linux Enterprise Server 9 üzerinde çalışan Oracle 10g. Karşılaştırılmıştır. Sonuçta %22 daha ucuz olan Linux-Oracle ikilisi %18 daha hızlı performans ortaya koymuştur.[8]

Linux'un performansının ölçüldüğü ve IBM tarafından yapılan bir karşılaştırmada RedHat 7.1 (Kernel 2.4.2) işletim sisteminin, pipe kullanımı, görev (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 100MB/saniyede sabitlenmektedir. Bu değerler Windows 2000'de en çok 500MB/saniye ve ortalama 80MB/saniye, Windows XP' de ise 120MB/saniye ve 80MB/saniyedir. Şubat 2002'de yayınlanan diğer bir yazıya göre RedHat Linux 7.2 saniyede 10.000 görev yaratabilirken Windows 2000, 5000 değerine yaklaşmış, Windows XP ise 6000 görev yaratabilmiştir. Thread yaratma karşılaştırmasında ise RedHat Linux saniyede 330 thread yaratmış; buna karşın Windows 2000, 200 ve Windows XP, 160'ın altında thread yaratabilmiştir (Bradford, 2001).

Linux iş istasyonlarında ve ağ yapılarında sürekli yüksek performans sağlarken, aynı anda alışılmadık oranda fazla kullanıcıyı sorunsuz olarak yönetebilmektedir.

4.1.3 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 ana çatıda (mainframe) çalışabilir. Ayrıca Linux, üniversite ve araştırma merkezlerinde süper bilgisayarlar kurmak ve gruplama (clustering) yapmak için de kullanılmaktadır.

4.1.4 Güvenlik

Evans Data Summer 2004 Linux Development tarafından yapılan araştırmada görüşülen kullanıcıların %92'sinin Linux işletim sistemlerinde hiç virüs sorunu ile karşılaşmadıkları tespit edilmiştir. %7 civarında kullanıcının da üç veya daha fazla korsan saldırısına uğradığı gözlemlenmiştir. Aynı araştırmada diğer işletim sistemi kullanıcılarının %60'nın güvenlik açıklarından dolayı zarar gördükleri belirtilmiş %32 oranındaki kullanıcının da üç veya daha fazla korsan saldırısına uğradığı ortaya konmuştur [9].

Yapılan bir diğer araştırma ise Windows 2000 kullanıcılarının %12'sinin, Windows 2003 kullanıcılarının bile sadece %18'nin düşüncesi güvenlik konusunda Windows'un Linux'a rakip olacak düzeye geldiğini belirtiyor. Aynı konudaki başka bir araştırmada ise 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 [9].

4.1.5 Maliyet

Linux işletim sistemi ve üzerinde çalışan birçok uygulama GNU General Public License ile birlikte bedava temin edilebilmektedir. İstenirse uygun bir ücret karşılığında farklı üreticiler tarafından geliştirilen standart ve profesyonel sürümleri satın alınabilmektedir. Diğer işletim sistemlerinin yüksek maliyeti incelendiğinde bu maliyet avantajı Linux'un tercih edilmesine neden olmaktadır.

Örneğin; web, mail, veritabanı sunucusu ve program geliştirme platformu olarak kullanılacak bir sunucu kurulumunda Windows 2003 Server Enterprise Edition ve Redhat Enterprise Linux'un satın alma maliyetleri Çizelge 4.1'de verilmiştir.

Çizelge 4.1 Linux ve Windows toplam sahip olma maliyetleri karşılaştırması (Guvensan Amac, 2006)

	Windows 2003 Server EE	**RedHat Enterprise Linux**
İşletim Sistemi	1325 $ - 20 istemci	349 $ Basic, 799 $ Standart Sınırsız İstemci
Mail Sunucusu	1000 $	İşletim Sistemi ile Birlikte - Sınırsız istemci
Web Sunucusu	İşletim Sistemi ile Birlikte	İşletim Sistemi ile Birlikte
Veritabanı Sunucusu	3510 $ - 20 istemci	İşletim Sistemi ile Birlikte - Sınırsız istemci
Yazılım Geliştirme Paketi	975 $	İşletim Sistemi ile Birlikte

Tüm bileşenleri içeren bir sunucunun Windows 2003 Server EE ile kurulumu 5810 dolara mal olurken RedHat Enterprise Linux ile kurulumu Standart Edition ile 799 dolara mal olmaktadır.

İşletim sisteminin sürümünün 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 sisteminin yeni sürümünün yüklenmesi ile çözümlenmektedir.

Linux işletim sistemi oldukça düşük yapılandırmalar üzerinde çalışabilir. Windows 2003 Server kurulum için, Pentium uyumlu en az 133 MHz bir işlemci, tercihen 256 MB olmakla beraber minimum 128 MB hafıza birimi ve en az 1GB'ı boş olan 2GB'lık disk birimine ihtiyaç duymaktadır. Buna karşın RedHat Enterprise Linux işletim sistemi kurulum için en az, tercihen Pentium tabanlı olmakla birlikte, minimum i486 işlemci, önerileni 128 MB olan 64 MB hafıza birimi ve 800 MB disk birimine ihtiyaç duymaktadır.

Sonuç olarak birçok kurum Linux işletim sistemini tercih ederek harcamalarında büyük miktarda kısıtlamalara gitmektedir.

4.1.6 Açık Kaynaklı Kod Özelliği

Linux'un araştırma dünyasında çok tercih edilmesinin en önemli nedeni açık kaynak kodlu oluşudur. Birçok araştırmacı, önemli oranda tercih edilen bir işletim sistemi üzerinde ve/veya işletim sistemine eklenecek parçalar tasarlayarak bilim dünyasına katkılar sağlayıp yeni ufuklar açabilmektedir.

Linux'un açık kaynak kodlu oluşu araştırmacıların özellikle Internet üzerinde sorunlarını paylaşıp çözerek grup çalışmaları yapmalarına olanak vermektedir.

4.1.7 Bilişim Dünyasında Yaygınlık

IBM'in verdiği rakamlara göre yaklaşık 200 devlet; örneğin Fransa, İngiltere, Amerika vb. maliyetleri düşürebilmek, işyükünü dengeleyip verimliliği arttırabilmek için platform olarak Linux kullanmaya başlamıştır. Sadece 2003'te Rusya, İngiltere, Brezilya, Bahreyn, Romanya hükümetleri Linux ile ilgili büyük projelerini açıklamıştır [10].

International Data Corporation (IDC)'nin yaptığı tahminlere göre Linux'un sunucu gelirlerinde %15 market payına sahip olacağı ve %38 paya sahip olacak lider Windows ailesinin en yakın takipçisi olacağı beklenmektedir. Diğer işletim sistemlerini kullanan eski sunucuların da önemli bir bölümünün Linux işletim sistemi kullanacağı hesaplandığında %15'lik payın %28'lere kadar çıkması bekleniyor.[11]

IBM'in yaptığı bir başka tahmine göre 2002-2007 yılları arasında Windows %5,4 ile büyürken Linux %9 oranında büyüme sağlayacaktır [10].

4.2 Linux İşletim Sisteminde Ipv4 Desteği

Linux çekirdeği, 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. Kod, giden paket içinse gerekli IP başlık bilgilerini paketin başına ekleyip paketin alt katmana iletilmesini sağlar. IP işlevlerinin gerçeklenmesi, alt katmanda kullanılan ağ cihazları sayesinde olur. Ağ cihazları her zaman fiziksel olmayabilir. Linux'ta kullanılan 'loopback' cihazı tamamen yazılım yardımıyla gerçeklenmektedir.

5. LINUX ÇEKİRDEK PARÇASI

Linux çekirdek parçası (Linux Kernel Module), istenildiği zaman çekirdeğe yüklenip çekirdekten çıkartılabilen kod parçalarıdır. Çekirdek'in sisteme bağlı olan donanımlara erişmesini sağlayan aygıt sürücüleri, çekirdek parçalarına örnek olarak gösterilebilir [12]. Sistemin ilk kurulumu aşamasında sisteme hangi donanımların bağlı olduğu tespit edilerek gerekli olan sürücüler çekirdeğe yüklenir, gereksiz olanlar ise yüklenmez. Sisteme yeni bir donanım eklenmesi veya var olan bir donanımın değişmesi gerektiği zaman, çekirdeğin derlenmesi ve yeniden başlatılması gerekir. Eğer sistem destekliyorsa çekirdeği derlemeye gerek kalmaz. Sistem yeniden başlatıldığında yeni donanım bulunur, gerekli sürücüsü çekirdeğe yüklenir ve çekirdek otomatik olarak derlenir.

5.1 Yüklenebilen Çekirdek Parçası

Çekirdek parçalarının çekirdek çalışırken bile çekirdeğe eklenebilenleri de vardır. Bu parçalara yüklenebilen çekirdek parçası LKM (Loadable Kernel Module) denir. Bu parçalar, sistemi yeniden başlatmadan çekirdek'in işlevselliğini artırırlar. Eğer bu parçalar olmasaydı, tek parçadan oluşan bir çekirdek kullanılması ve yeni fonksiyonların direk olarak çekirdek kopyasına (image) eklenmesi gerekirdi. Bunun sonucu olarak, çekirdek yapısının normalden daha büyümesinin yanı sıra, her yüklenen yeni fonksiyon için çekirdeğin tekrar derlenip, sistemin yeniden başlatılması gerekirdi.

LKM'lerin çekirdeğin dışında olduğu ve çekirdek ile haberleştiği düşüncesine sahip olanlar yanılırlar. Çünkü LKM'ler yüklendikleri zaman çekirdeğin içinde bulunurlar. Başlatılan sisteme bağlı bulunan çekirdek parçası, yani LKM dışında kalan bütün çekirdek için kullanılan doğru terim ana çekirdek'tir (base kernel). LKM'ler ana çekirdek ile haberleşir. Bazı işletim sistemlerinde LKM'ler çekirdek eklentisi olarak adlandırılır.

5.2 Yüklenebilen Çekirdek Parçasının Avantajları

Bir çekirdek parçasını çekirdeğe eklemek için iki yol vardır. Birincisi, ana çekirdekle (base kernel) birleştirmek, ikincisi ise LKM olarak ana çekirdeğe bağlamak. LKM' nin çekirdekle birleştirmeye nazaran pek çok avantajı vardır. Bu avantajlardan biri çekirdeği tekrar tekrar yeniden yapılandırmanın gerekmemesidir. Böylelikle zaman tasarrufu sağlanmış ve yeniden yapılandırma esnasında oluşacak bir hata yüzünden ana çekirdeği tekrar kurmak zorunda kalmamış olunur.

Başka bir avantaj ise LKM'nin sistem problemlerini teşhis etmede yardımcı olmasıdır. Ana çekirdek ile birleştirilmiş bir aygıt sürücüsünden kaynaklanan bir çapar (bug), sistemin ilk yükleme (boot) işlemini tamamen durdurabilir. Bu durumda ana çekirdeğin hangi parçasında problem olduğunu bulmak çok zor olabilir. Bu sürücü bir LKM ise, ana çekirdek aygıt sürücüsü yüklenmeden önce çalışır hale gelir. Eğer sistem ana çekirdek çalışır hale geldikten sonra durursa, hangi aygıt sürücüsünde problem olduğunu tespit etmek kolaydır ve bu sürücü problem giderilene kadar yüklenmez.

LKM'ler gerçekten ihtiyaç duyulacakları zaman yüklenmesi gerektiği için hafıza tasarrufu sağlarlar. Bunun yanında ana çekirdek her zaman sistemde yüklü bulunmak zorundadır.

LKM'lerin bakımını yapmak ve hatalarını ayıklamak daha hızlıdır. Çekirdek içine gömülmüş olan bir dosya sistemi sürücüsü ile ilgili bir problemi gidermek için sistemi yeniden başlatmak gerekirken, bu işlem LKM'de birkaç komut ile gerçekleştirilebilir. Değişik parametreler kullanarak ve hatta kodu devamlı değiştirerek, sistemin yeniden başlaması beklenmeden gerekli düzeltmeleri yapılabilir.

Bazı durumlarda ise çekirdek parçasının LKM olarak yazılması değil, ana çekirdeğe gömülmesi gerekir. Sistemin LKM'leri yükleyecek seviyeye gelmesi için gereken her şeyin ana çekirdekte bulunma gerekliliği aşikârdır. Örneğin, root dosya sistemini içeren diskin sürücüsü ana çekirdek içine gömülmek zorundadır.

Pek çok özellikleri aynı olmasına rağmen LKM'ler kullanıcı alanı programları (user space) değildirler. LKM'ler çekirdeğin parçasıdırlar. Bu yüzden, sistem üzerinde istedikleri serbestliğe sahip olabilirler ve hata durumunda sistemi kolaylıkla çökertebilirler.

5.3 LKM'lerin Kullanım Alanları

LKM'ler 6 ana amaç için kullanılabilirler: [13]

1. **Aygıt sürücüsü:** Aygıt sürücüsü, belirli bir donanım için yazılmış programdır. Çekirdek bu program aracılığı ile donanımın nasıl çalıştığı ile ilgili hiçbir detay ile ilgilenmeden o donanım ile haberleşebilir. Bir donanımı kullanmak için, çekirdekte o donanımla ilgili bir aygıt sürücüsü bulunmak zorundadır.

2. **Dosya sistemi sürücüsü:** Dosya sistemi sürücüsü, bir dosya sistemi içinde bulunan dosyaları, klasörleri vs. yorumlamaya yarar. Dosyaları ve klasörleri disk sürücüsünde, ağ sunucularında ve farklı birimlerde saklamak için pek çok yöntem vardır. Her yöntem için bir dosya sistemi sürücüsü gerekir.

3. **Sistem çağrısı:** Kullanıcı alanı programları, çekirdekten hizmet alabilmek için sistem çağrısı kullanırlar. Örnek vermek gerekirse, dosya okumak için, yeni bir işlem oluşturabilmek için veya sistemi kapatmak için sistem çağrıları vardır. Çoğu sistem çağrısı sistemle bütünleşiktir ve ana çekirdekte bulunması gerekir (LKM seçeneği olmaksızın). Fakat yeni bir sistem çağrısı oluşturulup bir LKM olarak yüklenebilir. Ya da Linux'un yaptığı bir işlemi değiştirmek için, önceki sistem çağrısının üzerine yazacak şekilde bir LKM yazılabilir.

4. **Ağ sürücüsü:** Ağ sürücüsü, bir ağ protokolünü yorumlamaya yararlar. Çekirdeğin ağ fonksiyonlarının değişik katmanlarında bulunan veri akısını (data stream) besler ve kullanır. Örneğin, ağda IPX bağlantısı kullanılıyorsa, IPX sürücüsü kullanılmalıdır.

5. **TTY bağlantı disiplinleri:** Terminal aygıtlarının aygıt sürücüleri için geliştirmeler sağlar.

6. **Çalıştırılabilen yorumlayıcı:** Bir çalıştırılabilen yorumlayıcı, çalıştırılabilir dosyaları yükler ve çalıştırır. Linux her biri kendine ait bir çalıştırabilen

yorumlayıcıya sahip olduğu sürece çeşitli tipteki çalıştırılabilir dosyaları çalıştırabilmek için tasarlanmıştır.

5.4 LKM'lerin Kodlanması ve Derlenmesi

LKM'ler belirli yazım kuralları bulunmaktadır. Her LKM'nin bir adet giriş ve bir adet çıkış fonksiyonu tanımlanır. En basit haliyle standart bir LKM'ye dair kod parçası Şekil 5.1'de verilmiştir.

```
/* hello.c
 *
 * "Hello, world" - the loadable kernel module version.
 *
 * Compile this with
 *
 *          gcc -c hello.c -Wall
 */

/* Declare what kind of code we want from the header files */
#define __KERNEL__          /* We're part of the kernel */
#define MODULE              /* Not a permanent part, though. */

/* Standard headers for LKMs */
#include <linux/modversions.h>
#include <linux/module.h>

#include <linux/tty.h>      /* console_print() interface */

/* Initialize the LKM */
int init_module()
{
  console_print("Hello, world - this is the kernel speaking\n");
  /* More normal is printk(), but there's less that can go wrong with
     console_print(), so let's start simple.
  */

  /* If we return a non zero value, it means that
   * init_module failed and the LKM can't be loaded
   */
  return 0;
}


/* Cleanup - undo whatever init_module did */
void cleanup_module()
{
  console_print("Short is the life of an LKM\n");
}
```

Şekil 5.1 En basit haliyle LKM kodu (hello.c)

Yukarıdaki kod parçasında yer alan *init_module* fonksiyonu LKM çekirdeğe yükleneceği zaman çalıştırılan işlemleri içermektedir. Bunun tersi olarak *cleanup_module* fonksiyonu ise

LKM çekirdekten çıkarılacağı zaman çağrılmaktadır. Temel olarak bu iki fonksiyon birbirinin tam tersi işlemleri içermektedirler. *init_module* fonksiyonunda çağrılan veya çalıştırılan komutlar, *cleanup_module* fonksiyonu tarafından geri alınır.

LKM'ler komut satırından çağrılan bir derleyicinin çağrılmasıyla derlenir ve oluşturulurlar. En yaygın derleyici olan *gcc* ile derleme işlemi: *gcc -c hello.c –Wall* komutunun çalıştırılmasıyla yapılabilir. Bunun yanında LKM derlemede çok kullanılan bir başka derleme komutu ise *make* komutudur. Make komutuna çok çeşitli parametreler verilebilmektedir. Her derleme işlemi için komut satırında birçok parametreyi yazmak yerine, içinde derleme tanımlamaların bulunduğu ve *make* komutuna rehberlik eden *Makefile* dosyası tanımlanabilmektedir. *Makefile* kullanmak uzun süreli LKM geliştirme süreçlerinde kullanıcının işini oldukça kolaylaştırmaktadır. Bir LKM derlemek için gerekli basit bir *Makefile* içeriği Şekil 5.2'de verilmiştir:

```
obj-m += atafs.o

all:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
~
~
"Makefile" 8L, 156C                                   8,0-1         All
```

Şekil 5.2 Makefile

5.5 LKM'ler için Yardımcı Uygulamalar

LKM derlendikten sonra sistem çekirdeğine yüklenebilecek haldeki ko uzantılı nesne dosyası oluşur. Bu dosya sistem çekirdeğine yüklenecek olan parçadır. Bu parçanın çekirdeğe yüklenmesi, çıkarılması, sorgulanması ve benzeri görevleri üstlenen sistem komutları aşağıda sıralanmıştır.

insmod LKM'yi çekirdeğe yükler

rmmod LKM'yi çekirdekten ayırır

depmod LKM'ler arasındaki bağımlılıkları belirler

ksyms Çekirdek tarafından LKM'lerin kullanımı için aktarılan sembolleri listeler

lsmod	Çekirdeğe yüklü olan aktif LKM'leri listeler
modinfo	LKM içindeki *.modinfo* bölümünde yer alan içeriği görüntüler
modprobe	LKM'lerin otomatik olarak yüklenmesini ve çekirdekten ayrılmasını sağlar

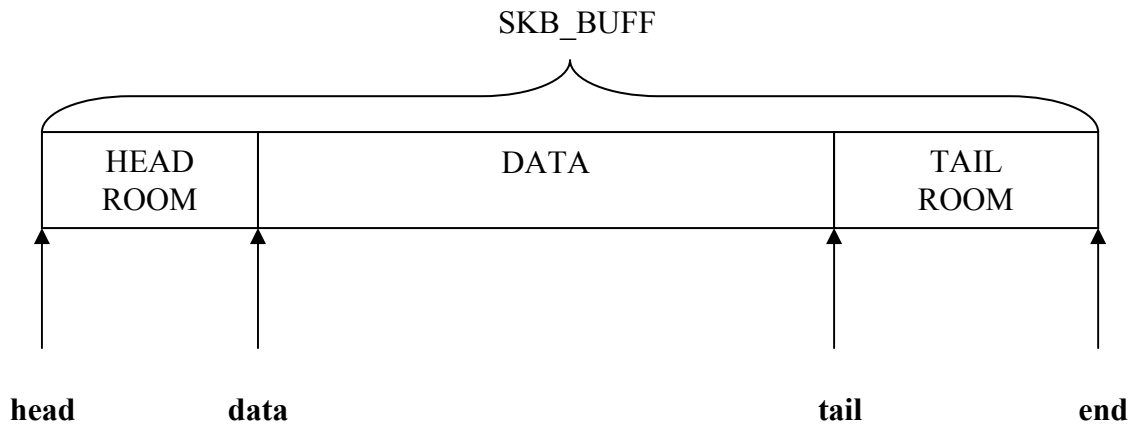
6. LINUX İŞLETİM SİSTEMİNDE AĞ YÖNETİMİ

Linux işletim sistemi 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. Gönderilen ve alınan her paket bu veri yapısı kullanılarak yönetilir. *Sk_buff* dört temel işaretçi tutmaktadır.

- **head:** *skb_buff*'ın tuttuğu verinin başlangıç adresidir.
- **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 adresini gösterir.

Yukarıda belirtilen işaretçilerin dışında, *sk_buff* yapısında, taşıma, ağ ve veri bağı katmanlarına ait bilgiler de tutulmaktadır [7]. Bunlardan bazıları aşağıda belirtilmiştir. Sk_buff'ın genel yapısı Şekil 6.1'de gösterilmiştir.

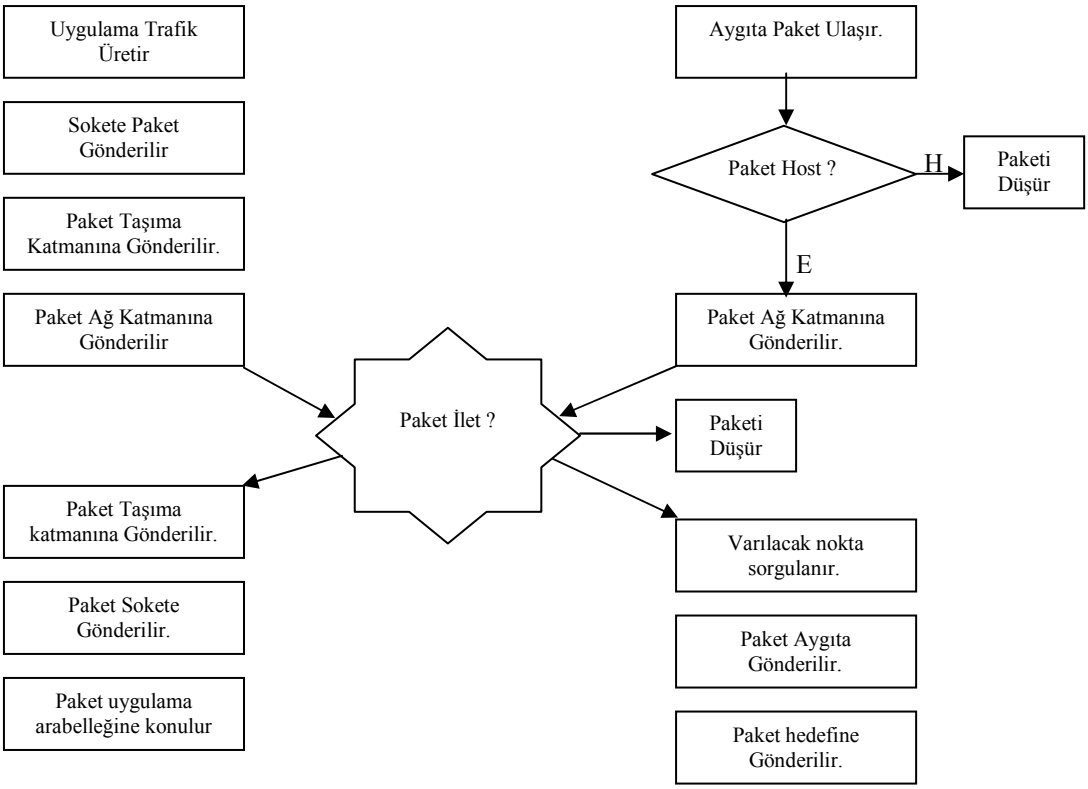
- **stamp:** paketin ulaştığı zamandır.
- **dev:** Alan ya da gönderen aygıtı işaret eder.
- **h:** Taşıma katmanı başlık işaret eder.
- **nh:** Ağ katmanı başlık bilgisini işaret eder.
- **mac:** Veri bağı katmanını gösterir.
- **len:** Gerçek veri uzunluğunu gösterir.
- **csum:** Veriye ait kontrol bilgisidir.
- **protocol:** Verinin hangi protokole ait olduğunu tutar.



Şekil 6.1 sk_buff veri yapısı [7]

6.1 Linux Çekirdeğinde Ağ Paketinin İzlediği Yol

Bir ağ paketinin Linux 2.4 çekirdeğindeki adımları Şekil 6.2'de gösterilmiştir (Kossak, 1999).



Şekil 6.2 Gelen ve giden paketlerin izlediği yol (Guvensan Amac, 2006)

Linux çekirdeğinde gelen ve giden paketler için gerçekleşen adımlar aşağıdaki bölümlerde anlatılmaktadır.

6.1.1 receive Kesmesi

Sistemdeki ağ kartı, sistemin Medium Access Control (MAC) adresine gelen veya yayınlanan (broadcast) bir paket alındığında 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.

6.1.2 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. Eğer program paket alma noktalarından birinden geçerse, hem *do_softirq* hem de ağ paket alma kesmesi olan *net_rx_action* fonksiyonu çağrılır. Burada *sk_buff*, işlemcinin bekleme kuyruğundan alınır ve uygun paket işleyici fonksiyonuna gönderilir. Uygun paket işleyici fonksiyonu, çekirdekte yer alan *ptype_base* hash tablosu kullanılarak belirlenir. Bu tablo, paket tipleri temel 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.

6.1.3 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 başarı ile 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 dört 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 birçoğa gönderim (multicast) paketidir ve çoğa gönderim yönlendirme yapılır.

6.1.4 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 Time To Live (TTL) değerini kontrol eder. TTL değeri "1" değerinden küçük veya "1" değerine eşitse paket düşürülür ve gönderene ICMP kullanılarak zaman aşımı mesajı gönderilir.

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

Bundan sonra yeni NetFilter hook'u çağrılır. Bu hook, *ip_forward_finish* fonksiyonunu çağırır. *İp_forward_finish*, IP bilgisi içindeki IP seçeneklerini 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. *İp_finish_output2*, donanım bilgisini *sk_buff*'a ekler ve *ip_output*'u çağırarak paketin gönderilmesini sağlar.

6.1.5 send Kesmesi

ip_build_xmit() fonksiyonu IP başlık bilgilerini oluşturur. *İp_finish_output* ve *ip_finish_output2* donanım bilgisini *sk_buff*'a ekler ve *ip_output* çağrılarak paketin gönderilmesi sağlanır. *Dev_queue_xmit* gönderilecek paketleri işlemcinin bekleme kuyruğuna yerleştirir. Send kesmesi sonucunda da paket ağ kartı tarafından gönderilir.

6.2 Linux 2.4 Netfilter Mimarisi

Linux v2.4 paket filtreleme fonksiyonunu iki büyük bloğa böler: netfilter kancaları Linux çekirdeğindeki yollarında farklı pozisyonlarda olan işlenmiş IP paketlerini yakalamak ve değiştirmek için rahat bir yol sağlar. Bu artalana dayanarak ip tablosu modülü gelen, iletilen

ve giden IP paketlerini filtrelemek için üç kural listesi gerçekleştirir. Bu listeler yaklaşık olarak ip zincirleri tarafından kullanılan kurallar listelerine ile uyumludur. Buna ek olarak, benzer modüller diğer ağ protokolleri için mevcuttur.

6.2.1 Linux Çekirdeğinde Netfilter Kancaları

Netfilter mimarisi, yeni fonksiyonlar gerçekleştirmekle ilgili maliyeti azaltan bir tekdüzen ara yüz içerir. Buna netfilter kancası adı verilir. Netfilter kancası paket filtre kodu için bir kanca sağlar. Bu bölümde bu mimarinin içerikleri ve Linux kernel içindeki uygulamaları anlatılmaktadır.

Netfilter modüller Linux kernele çalışma anında yüklenebilirler, dolayısıyla fonksiyonların dinamik kancalanmasını sağlamak için mevcut yönlendirme kodu içinde kancalara ihtiyacımız vardır. Bu netfilter kancalarının her birine bir tam sayı tanımlayıcı tahsis edilir. Bütün kancaların tanımlayıcıları her desteklenen protokol için protokol özel başlık dosyasında tanımlanmışlardır (<linux/netfilter_ipv4.h> veya <linux/netfilter_ipv6.h>). Aşağıdaki beş kanca <linux/netfilter_ipv4.h> içinde IP sürüm 4 için tanımlanmışlardır.

- **NF_IP_PRE_ROUTING (0):** Gelen paketler yönlendirme kodu ile işlenmeden önce bu kancayı ip_rcv() fonksiyonunda geçerler. Bundan önce, IP başlığında versiyon, uzunluk ve sağlama alanlarında sadece birkaç basit kontrol gerçekleştirilir.

- **NF_IP_LOCAL_IN (1):** Yerel bilgisayara yönelen bütün gelen paketler ip_local_deliver() fonksiyonunda bu kancayı geçerler. Bu noktada, ip tablosu parçası gelen veri paketlerini filtrelemek yerine giriş kuralları listesini kancalar. Bu ip zincirlerindeki giriş kuralları ile uyumludur.

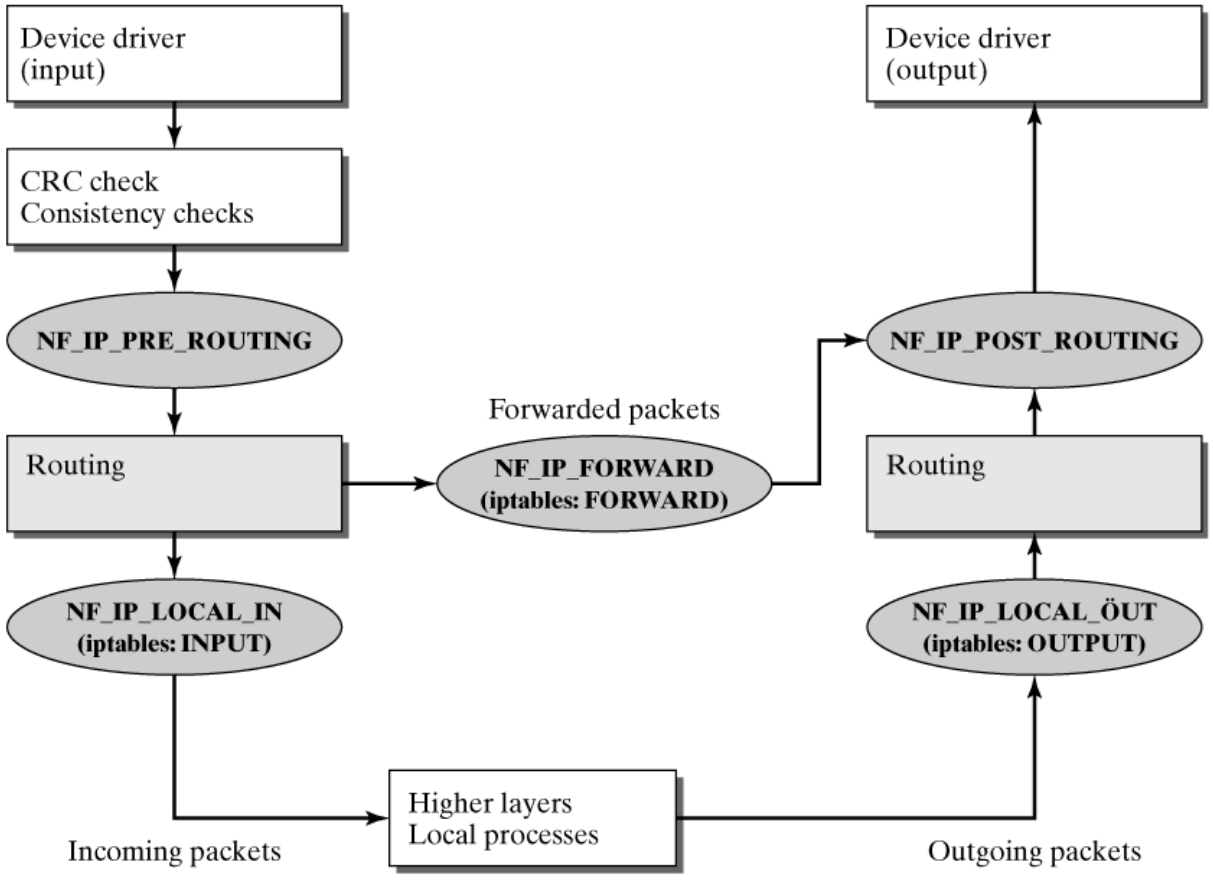
- **NF_IP_FORWARD (2):** Yerel bilgisayara yönelmeyen bütün gelen veriler ip_forward() fonksiyonunda bu kancadan geçerler, paketler farklı ağ ara yüzü üzerinden iletilecek ve bilgisayardan çıkacaklardır.

Bu adresi NAT tarafından değiştirilmiş olan herhangi bir paketi içeriri. Bu noktada, ip tablosu parçası iletilen veri paketlerini filtrelemek yerine iletim kuralları listesini kancalar. Bu ip zincirlerindeki iletim kuralları ile uyumludur.

- **NF_IP_LOCAL_OUT (3):** Yerel bilgisayarda yaratılmış olan giden bütün paketler ip_build_and_send_pkt() fonksiyonunda bu kancadan geçerler. Bu noktada, ip tablosu parçası giden veri paketlerini filtrelemek yerine çıkış kuralları listesini kancalar. Bu ip zincirlerindeki çıkış kuralları listesi ile uyumludur.

- **NF_IP_POST_ROUTING (4):** ip_finish_output() fonksiyonundaki bu kanca giden (iletilmiş veya yerel olarak yaratılmış) tüm paketlere, bilgisayardan bir ağ aygıtı üzerinden ayrılmalarından önce son erişim şansını belirtir. Aynı, NF_IP_PRE_ROUTING kancası gibi, burası hesaplama fonksiyonlarını yerleştirmek için iyi bir yerdir.

Netfilter kancalarının Linux çekirdek mimarisindeki yeri Şekil 6.3'te verilmiştir:



Şekil 6.3 Netfilter paket filtreleme mimarisi [24]

NF_HOOK makro aşağıdaki argümanlara sahiptir:

- pf (protokol ailesi): Bu protokol ailesinin tanımlayıcısıdır: IP Versiyon 4 için PF_INET, IP Versiyon 6 için PF_INET6.
- Kanca: Bu kancanın tanımlayıcısıdır. Her protokol ailesi için tüm geçerli tanımlayıcılar bir başlık dosyasında tanımlanmışlardır (örn: <linux/netfilter_ipv4.h>).
- skb: Bu işlenecek paket ile sk_buff yapısına olan bir göstergedir.
- indev (giriş aygıtı): Bu paketi alan ağ aygıtının net_device yapısına olan bir göstergedir.
- outdev (çıkış aygıtı): Bu yerel bilgisayardan ayrılmak için paket tarafından kullanılacak ağ aygıtının net_device yapısına olan bir göstergedir.
- okfn() (okay fonksiyonu): Bu fonksiyon bu kancada kayıtlı olan bütün filtre fonksiyonları NF_ACCEPT olarak dönerse çalıştırılır, böylece paketin geçişi onaylanır.

6.2.2 Paket Filtreleme Fonksiyonlarını Kaydetmek Ve Kayıtlarını Silmek

Genellikle netfilter kancaları içinde kancalanmış olan paket filtre fonksiyonları nf_hookfn türü kanca fonksiyonlarıdır. Bir kanca fonksiyonun imzası <linux/netfilter.h> içinde aşağıdaki gibi tanımlanmışlardır:

```
Typedef unsigned int nf_hookfn(unsigned int hooknum,
                               struct sk_buff**skb,
                               const struct net_device*in,
                               const struct net_device*out,
                               int (*okfn) (struct sk_buff*));
```

Protokol aile tanımlayıcı haricindeki parametreler NF_HOOK makrosu ile tam olarak uygunluk içerisindedirler ve bu makro aracılığıyla paket filtre fonksiyonlarına geçerler. Bir paket filtre fonksiyonunun geri dönüş değeri pakete ne olması gerektiğini belirtir. Bu imzasız int türüdür ve <linux/netfilter.h>'de tanımlanan aşağıdaki değerlerden herhangi birini alabilir:

- NF_DROP (0): Aktif kural listesi işlemi durdurulur ve paket düşürülür.
- NF_ACCEPT (1): Paket kurallar listesinde sonraki paket filtre fonksiyonuna geçer. Listenin sonuna ulaşıldığında, paket okfn tarafından ileri işlemler için bırakılır.
- NF_STOLEN (2): Paket filtre fonksiyonu pakedi ileri işlemler için tutar, bu nedenle aktif kurallar listesi işlemi durur. NF_DROP'a karşıt olarak paketin açık bir şekilde düşürülmesine gerek yoktur.
- NF_QUEUE (3): nf_queue() (net/core/netfilter.c) fonksiyonu pakedi içinden kaldırılabileceği ve işlenebileceği bir kuyruğa koyar(örn: bir kullanıcı alan programı). Sonra, nf_reinject() netfilter tarafından yapılacak ileri işlemler için pakedi Linux kernele geri göndermek için çalıştırılmalıdır.
- NF_REPEAT (4): NF_ACCEPT'e karşıt olarak, sonraki paket filtre fonksiyonunda bir işlem sürdürme yerine, mevcut filtre fonksiyonu yeniden çalıştırılır.

7. TASARIM

Yüksek yararlanırlık ve kusura dayanıklılık konuları üzerine yapılan çalışmalar ve geliştirilen uygulamalar 2. ve 3. bölümlerde dile getirilmiştir. Bu tez çalışmasında daha önceki uygulamalardaki eksik ve zayıf noktalar tespit edilerek yeni bir tasarım ortaya konulmuştur. Elde edilen bulgular aşağıdaki listede sıralanmıştır.

1. Simetrik yedeğe geçmeli sistemler için yük dengesini dağıtacak ekstra bir programa ihtiyaç duyulduğundan ve asimetrik yedeğe geçmeli sistemin yedekliliği (Redundancy) daha yüksek olduğundan asimetrik yedeğe geçmeli bir yapı kullanılmıştır.
2. Açık kaynak kod, güvenilirlik ve kararlılık özelliklerindeki üstünlüğü nedeniyle uygulama Linux işletim sistemi üzerinde gerçeklenmiştir.
3. Ağ paketinin kullanıcı seviyesine çıkması için harcanacak sistem kaynaklarının getireceği yükün ortadan kaldırılması için uygulama çekirdek seviyesinde çalışacak şekilde düzenlenmiştir.
4. Kolay uygulanabilirliği ve yüksek taşınabilirlik özellikleri nedeniyle uygulama yüklenebilen çekirdek parçası olarak tasarlanmıştır.
5. Sistemde birbirine eşlenik iki bilgisayar kullanılmıştır.
6. İki bilgisayar arasında ağ bağdaştırıcıları üzerinden noktadan noktaya sabit bir bağlantı kurularak, sistemin haberleşmesi bu hat üzerinden sağlanmıştır.
7. Bu hat kullanılarak bilgisayarlar birbirilerine kalp atışı (heart-beat) adı verilen özel sinyaller yollamakta ve bu yolla sistemin işlerliğini kontrol altında tutmaktadırlar.
8. Kalp atışının taşınacağı hat bağlantısının Seri, Paralel bağlantı noktası veya ağ bağdaştırıcısı üzerinden sağlanmasının getireceği avantaj ve dezavantajlar değerlendirilmiş ve kolay uygulanabilirliği nedeniyle 100Mbit hızındaki eşlenik ağ bağdaştırıcıları kullanılmıştır.
9. Düğümlerden (node) biri Birincil (Primary), diğeri İkincil (Secondary) olarak isimlendirilmiş ve sistemdeki aktif cihaz değiştiğinde cihazların Birincil veya İkincil olduğu bilgisi değiştirilmemektedir.
10. Düğümlerden biri bekleme pozisyonundan aktif konuma geçmesi gerektiğine karar verdiğinde, diğer düğümün pasif duruma geçtiğini temin etmektedir. Aksi takdirde iki

düğümün aynı anda aktif olması gerçekleşecek ve bu, ağ üzerinde çakışmaya ve servisin kesilmesine yol açacaktır.

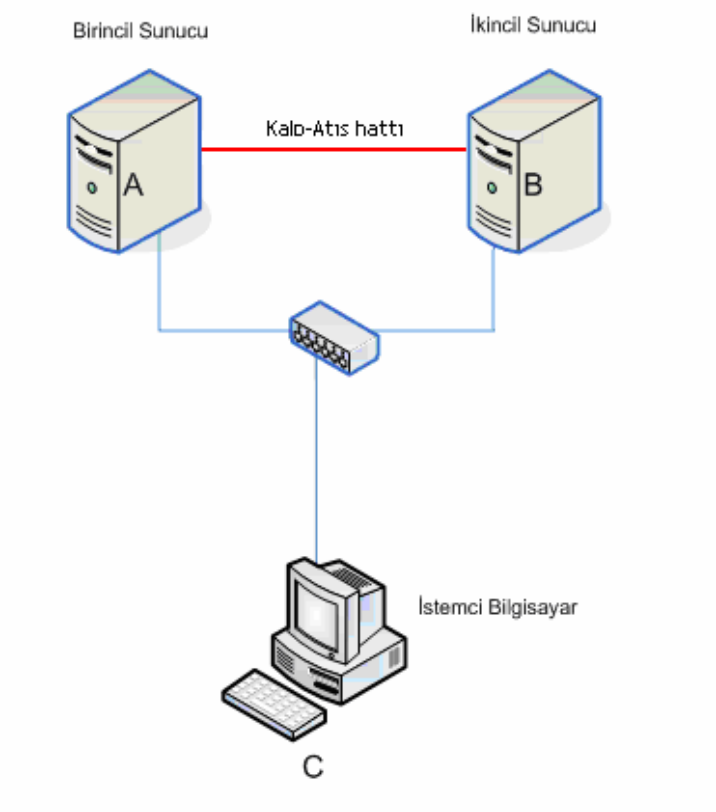
11. Sistemde bir arıza tespiti yapıldığında, haberleşme problemi yaşadığında veya yedeğe geçme işlemi yapıldığında yapılan işlemler sistem günlüğüne kaydedilir.

7.1 ATAFS Mimarisi

ATAFS mimarisi iki adet eşlenik sunucu, bu iki sunucu arasında kalp-atışı işaretleşmesi için kullanılacak olan kalp-atışı hattı, sunucuları istemci tarafına bağlayacak olan ağ bağdaştırıcıları ve bu ağ bağdaştırıcılarının bağlı olduğu göbek cihazı elemanlarından oluşmaktadır. Tasarlanan sistemde kullanılan temel donanım elemanları şunlardır:

1. 2 adet eşlenik sunucu
2. Her sunucu üzerinde servis arayüzünün sağlandığı ağ bağdaştırıcısı
3. Her iki sunucunun birbirine noktadan noktaya bağlandığı kalp-atışı arabirimi
4. İki sunucunun kalp-atış arabirimini birbirine bağlayan kablo
5. Sunucuların servis ayağını oluşturan ağ bağdaştırıcılarının bağlandığı göbek cihazı

Bu donanımlardan oluşan mimarinin görüntüsü Şekil 7.1'de gösterilmiştir. Bu sistemde yer alan iki sunucudan biri aktif görev alırken diğeri aktif sunucunun yerini devralmak üzere yedekte beklemektedir. Sistemdeki sunucular önyükleme yapıldığındaki servis arabirimlerinin IP ve MAC adres yapılandırması, ATAFS sistemi aktif hale getirildiğinde kullanılacak olan IP ve MAC adres yapılandırmasından farklı olmalıdır. ATAFS sistemine ait LKM sunuculara yüklendiğinde, servis arabirimine ait IP ve MAC adres yapılandırması her iki sunucuda da ATAFS sisteminin servis IP ve MAC adres bilgileriyle değiştirilir. Buna göre aynı göbek cihazına bağlı iki adet ağ bağdaştırıcısı, aynı IP ve MAC adres yapılandırmasına sahip olurlar. Ağ üzerinde çakışma yaşanmaması için gereken önlemler, ATAFS paket işleyicisinin kontrolündedir. ATAFS paket işleyicisi, sunucunun çalışma kipine bağlı olarak servis arabiriminden fiziksel ağ katmanına çıkmak üzere olan paketleri yok eder veya paketlerin fiziksel ağ katmanına inmesine izin verir. Bu sayede aynı IP ve MAC adres yapılandırmasına sahip iki sunucu çakışma yaşamadığı gibi, her iki sunucunun da istemci tarafından gönderilen isteklere dair paketleri görmesi ve işleyebilmesi sağlanır.



Şekil 7.1 ATAFS Mimarisi

Bu yapıda iki düğüm arasında durum bilgisinin değiş-tokuş edildiği kalp-atışı hattı önem arz etmektedir. ATAFS LKM'si kalp-atış hattı üzerinden yapılan işaretleşmeye göre çalışma kipine karar verir.

7.1.1 Kalp-Atışı Arabirimi

Kalp-atışı işaretleşme ağı temelde 3 farklı donanım üzerinden sağlanabilir:

1. Seri bağlantı noktası
2. Paralel bağlantı noktası
3. Ağ bağdaştırıcısı

ATAFS sistemi, kalp-atışı hattı üzerinden basit bir işaretleşme ağı kurmaktadır. İki düğüm arasında gönderilen veri, sadece iki düğümün birbiri hakkında temel durum bilgisinin sorgulanmasından ibarettir. Buna göre bu bağlantı tipi için kullanılan hattın hızı, *hb_interval* parametresi ile belirlenen zaman değerinin aşılmamasını temin ettiği sürece temelde bir önem arz etmemektedir. Tasarımın değiştirilmesi ve iki düğüm arasında kalp-atış hattı üzerinden

paket taşınması işlemleri yapılması durumunda hattın hızı önem arz edecektir. ATAFS sisteminde iki düğüm arasında kalp-atışı hattı üzerinden paket transferi gerekmediğinden hız bir kıstas değildir.

Bunların yanında, Linux çekirdeğinde seri ve paralel bağlantı noktalarının yönetimi ve bu donanımlara ait LKM kodlaması çeşitli zorluklar taşımaktadır. Linux çekirdeğinde seri ve paralel bağlantı noktaları birer karakter donanım (character device) olarak yönetildiğinden, bu bağlantı noktaları üzerinden veri transferinin tüm yönetimi kullanıcıya bırakılmaktadır. [27] Bu bağlantı noktaları işaret gönderme işlemleri için donanım kesmelere ihtiyaç duyduklarından, akış kontrolünün sağlanması ve LKM içerisine bütünleşik hale getirilmesi ek bir kodlama yükü getirmektedir.

Bunların yanında ağ bağdaştırıcıları ve bu donanımlar üzerinden akan trafiğin tüm kontrolünün ATAFS paket işleyicisinin elinde bulunması, kalp-atış hattı olarak ek bir ağ bağdaştırıcısı kullanılması fikrini ön plana çıkarmış, sonuç olarak kalp-atış hattı için 100Mbit hızında 2 adet eşlenik ağ bağdaştırıcısı kullanılmıştır.

7.1.2 Çalışma Kipleri

Tasarlanan sistemde yer alan bir düğümün bulunabileceği iki farklı çalışma kipi bulunmaktadır:

1. Aktif çalışma kipi
2. Bekleme kipi

7.1.2.1 Aktif Çalışma Kipi

Bu kipte çalışan düğüm, ATAFS sisteminin sunacağı servisten sorumludur. Sistemde gerçek anlamda hizmet veren ve istemci tarafına cevap döndüren düğüm aktif çalışma kipinde bulunan düğümdür.

7.1.2.2 Bekleme Kipi

Aktif hizmet vermeyen ve yedekte bekleyen düğüm, bu kipte yer alır ve aktif kipte çalışan düğüm ile haberleşme trafiğini yönetir. Bekleme kipindeki düğüm, aktif düğüm üzerinden geçen bağlantıları izler, gerektiği takdirde oturum bilgilerini kendi karşılaştırma tablosuna işler. Bu kipte çalışan düğüm, aktif düğümün durumunu düzenli olarak kontrol eder ve sistemin sağlıklı çalışmasını temin eder.

7.2 ATAFS sisteminin İş Akışı

Oluşturulan tasarım temelde altı ayrı parçadan oluşmuştur:

1. Sistemin başlatılması
2. Kalp-Atışı süreölçeri
3. TCP oturum bilgisinin yönetilmesi
4. Sıcak yedeğe geçiş
5. Yetkili kullanıcı etkileşimi
6. Sistemin durdurulması

7.2.1 Sistemin Başlatılması

Sistemin başlatılması derlenmiş olan LKM'nin *insmod* komutuyla çekirdeğe eklenmesiyle sağlanır. LKM çekirdeğe yüklendiğinde sırasıyla şu aşamalardan geçmektedir:

1. Yapılandırma dosyasının okunması
2. İlklendirme işlemlerinin yapılması ve çalışma kipleri
3. Paket işleyicisinin çekirdeğe kayıt edilmesi

7.2.1.1 Yapılandırma Dosyasının Okunması

LKM yüklendiği anda *main.h* içinde CONFIGFILE tanımıyla belirlenen adresteki atafs.conf dosyasını okur. Yapılandırma dosyasında tanımlı olan bilgileri gerekli değişkenlere kaydeder, bulduğu hataları kullanıcıya iletir. Sistemin çalışmasını engelleyecek oranda kritik bir hata

bulduğunda LKM'nin yüklenmesini durdurur. Yapılandırma dosyasının varsayılan değeri *main.h* dosyasında aşağıdaki tanım satırıyla belirlenmiştir.

```
#define CONFIGFILE      "/etc/atafs.conf"
```

Yapılandırma dosyası Şekil 7.2'de gösterilmiştir.

```
#
#       ATAFS Kernel Module Configuration File
#
#       Should be placed in /etc directory
#
#       ALL OPTIONS ARE CASE-SENSITIVE !!
#
#       Turgut GENC <turgut@ce.yildiz.edu.tr>
#
#       2007.02.03
#

# First part of this configuration file defines
# This Server's and Peer's Interface informations

# Defines whether this server Primary or Secondary

this_is_primary=YES

# Service IP and Mac Address Definitions
# IPv4 Address  : service_ipv4=a.b.c.d (193.140.1.55)
# Mac Address   : service_mac=00:11:25:F1:47:71
# Gateway Addr  : service_gw=193.140.1.62

service_ipv4=193.140.1.55
service_mac=00:11:25:F1:47:7F
service_gw=193.140.1.62

# This Server's serving Interface
# Device Name   : servif_name=eth0
# Ipv4 Address  : servif_ipv4=a.b.c.d (193.140.1.57)
# Mac Address   : servif_mac=aa:bb:cc:dd:ee:ff

servif_name=eth0
servif_ipv4=193.140.1.57
servif_mac=00:11:25:F1:47:71

# This Server's HeartBeat Interface
# Device Name   : hbif_name=eth1
# Ipv4 Address  : hbif_ipv4=a.b.c.d (192.168.1.1)
# Mac Address   : hbif_mac=aa:bb:cc:dd:ee:ff

hbif_name=eth1
hbif_ipv4=192.168.1.1
hbif_mac=00:60:08:66:53:FB

# Peer 's serving Interface
# Ipv4 Address  : pri_servif_ipv4=a.b.c.d (193.140.1.57)
# Mac Address   : pri_servif_mac=aa:bb:cc:dd:ee:ff

peer_servif_ipv4=193.140.1.56
peer_servif_mac=00:11:25:B8:08:A4

# Peer 's HeartBeat Interface
# Ipv4 Address  : pri_hbif_ipv4=a.b.c.d (192.168.1.1)
# Mac Address   : pri_hbif_mac=aa:bb:cc:dd:ee:ff

peer_hbif_ipv4=192.168.1.2
peer_hbif_mac=00:A0:24:E1:6C:97

# Second part of this config file defines
# Other config parameters

# Heart-Beat interval (secs)

hb_interval=5
```

Şekil 7.2 Yapılandırma dosyası

Sistemin başlatıldığı sırada yapılandırma dosyasından değeri okunan ve sistem devreden çıkana kadar değerini koruyan değişkenler şunlardır:

	PARAMETRE	ÖRNEK DEĞER	AÇIKLAMA
1.	this_is_primary	YES / NO	Düğüm birincil / ikincil bilgisi
2.	service_ipv4	193.140.1.55	ATAFS Sistem IP'si
3.	service_mac	11:22:33:44:55:66	ATAFS Sistem Fiziksel Adresi (MAC)
4.	service_gw	193.140.1.62	ATAFS Sistem varsayılan ağ geçidi IP'si
5.	servif_name	eth0	Servis arayüzü adı
6.	servif_ipv4	193.140.1.57	Servis arayüz IP'si
7.	servif_mac	11:22:33:44:55:66	Servis arayüz Fiziksel Adresi
8.	hbif_name	eth1	Kalp-Atışı arayüzü adı
9.	hbif_ipv4	192.168.1.1	Kalp-Atışı arayüzü IP'si
10.	hbif_mac	11:22:33:44:55:66	Kalp-Atışı arayüzü MAC
11.	peer_servif_ipv4	192.168.1.2	Yedek cihaz servis arayüz IP'si
12.	peer_servif_mac	11:22:33:44:55:66	Yedek cihaz servis arayüz MAC
13.	peer_hbif_ipv4	255.255.255.255	Yedek cihaz kalp-atışı arayüz IP'si
14.	peer_hbif_mac	11:22:33:44:55:66	Yedek cihaz kalp-atışı arayüzü MAC
15.	hb_interval	5	Kalp-Atış aralığı
16.	hb_err_limit	3	Ardışık Kalp-Atış hatası sınırı

7.2.1.2 İlklendirme İşlemlerinin Yapılması ve Çalışma Kipinin Belirlenmesi

Bu aşamada yapılandırma dosyasından okunan servis ve kalp-atışı arayüzlerinin çekirdek içindeki işaretçileri tespit edilir. Arayüz bilgileri kontrol edilir. Bir hata tespit edilmesi durumunda sistem günlüğü dosyasında hataya dair bilgiler yazdırılarak LKM'nin çıkış işlemleri başlatılır. Servis ve Kalp-Atışı arayüzlerinde bir sorun tespit edilmediği takdirde, sistemin birincil / ikincil düğüm olma durumu ve yedekteki düğümün aktif çalışma kipinde olup olmadığı kontrolleri yapılır. LKM bu aşamada aktif veya bekleme pozisyonu alacağına karar verir.

İlklendirme işlemleri sırasında sistem, servis arayüzünün IP, fiziksel adres, ağ maskesi ve ağ geçidi IP'si bilgilerini kaydeder ve arayüzün bilgilerini ATAFS sistem IP'si, fiziksel adres, ağ maskesi ve ağ geçidi IP'si bilgileriyle değiştirir. Sistemin ilklendirilmesi sırasında bir hata

tespit edilmesi durumunda ilklendirme (init) fonksiyonu ana programa hata kodunu döndürür. Hata kodunun değerine göre LKM çalışmasını durdurabilir veya başlangıç işlemlerine devam edilir.

7.2.1.3 Paket İşleyicisinin Kayıt Edilmesi

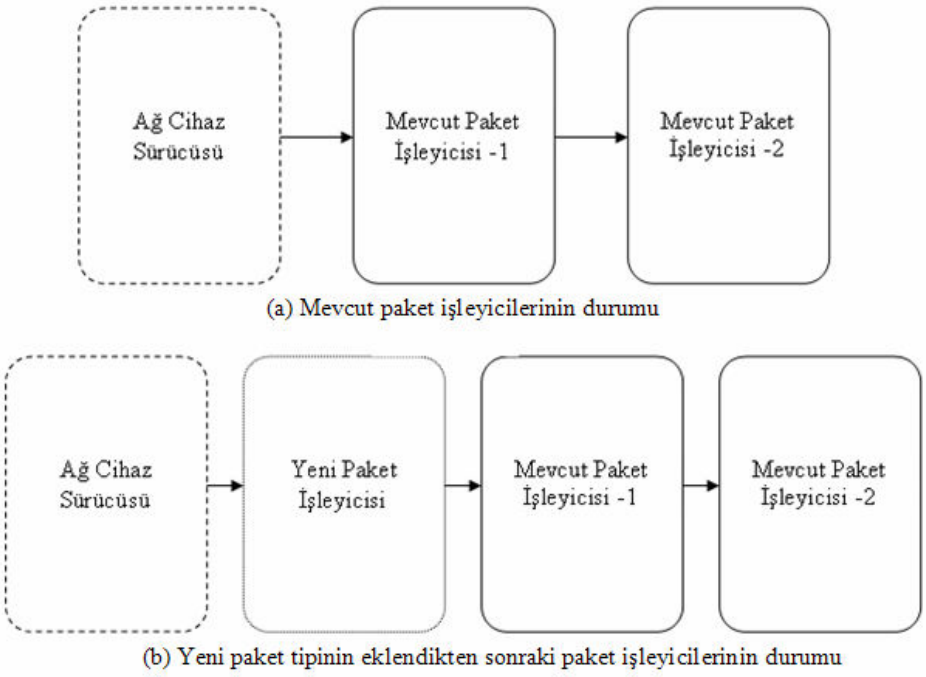
Sistemin gelen ve giden paketleri denetlemesi ve çalışma kipine bağlı olarak paketlerin düşürülebilinmesi için paketlerin sistemin bir noktasında ele geçirilmesi gerekmektedir. Bunun için de Linux çekirdeği tarafından çekirdek programlamada kullanılması için dışarı aktarılan fonksiyon işaretçilerinden ve veri yapılarından faydalanılmıştır.

Linux çekirdeği, gelen ve giden ağ paketlerine tiplerine göre servis verebilmek için paket tiplerinin kaydedildiği bir dizi tutmaktadır. Bu diziler sistemin önyüklemesi (boot) sırasında çekirdek içindeki fonksiyonlar yardımıyla çekirdeğe kaydedilirler ve çekirdeğe herhangi bir LKM müdahalesi olmadığı sürece her paket tipi için karşılık gelen bir adet paket işleyicisi çekirdekte kayıtlı olarak bulunur. LKM'ler yardımıyla, sistemin aktif çalışması sırasında paket işleyicisi dizilerine yeni paket işleyicileri eklenebilir veya var olan paket işleyicilerinden biri veya birileri çekirdekten silinebilir. Ağ desteği olan her Linux dağıtımında temelde IP, TCP, UDP, ARP gibi genel kabul gören paket tiplerine uygun karşılık gelen IP paket işleyicisi, TCP paket işleyicisi, UDP paket işleyicisi, ARP paket işleyicisi gibi paket işleyicilerinin kaydı sistem içerisinde yer alır. Kayıtlı olan her paket tipinde aşağıdaki bilgiler bulunmaktadır.

- Paketin tipi
- Paket tipinin ilişkilendirileceği ağ cihazına ait işaretçi
- Tanımlı paket tipinde ağ paketi geldiğinde çağrılacak işleyici fonksiyona ait işaretçi

Paketlere, işlemci kuyruğunda 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 çekirdeğe kaydetmelidir. Çekirdeğe 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 veri yapısı oluşturulmalıdır. Daha sonra çekirdekte 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 yerleştirilmeli ve yeni paket işleyici fonksiyonu, mevcut paket işleyicisinden daha önce çalıştırılmalıdır. Bu yapı Şekil 7.3'de gösterilmiştir.



Şekil 7.3 Paket işleyicisinin eklenmesi

ATAFS başlangıç işlemleri arasında paket işleyicisinin tanımlamaları ve çekirdeğe eklenmesine dair kod parçacığı aşağıda verilmiştir:

```
/* Creating New Packet Handler */
AtafsPacket.type = htons(ETH_P_ALL);
AtafsPacket.dev  = NULL;
AtafsPacket.func = AtafsPacketHandler;

/* Adding New Packet Handler */
dev_add_pack(&AtafsPacket);
printk("ATAFS: Packet Handler Added!\n");
```

7.2.2 Kalp-Atış Süreölçeri

Kalp-Atış süreölçeri ATAFS sistemi içinde yer alan iki düğüm arasında kurulacak olan işaretleşme ağının başlatılmasını sağlar. ATAFS paket işleyicisinin çekirdeğe kaydedilmesinin hemen ardından *StartHBTimer* fonksiyonu çağrılarak kalp-atışı süreölçerine dair kesmenin kaydı çekirdeğe yapılır. Yapılandırma dosyasından okunan *hb_interval* değeriyle belirlenen aralıklarda yedek düğüm aktif düğüme Kalp-Atışı paketi yollar ve aktif düğümün sağlıklı olup olmadığını kontrol eder. *hb_interval* süresi içerisinde aktif düğümden yanıt alamayan yedek düğüm, *hb_err_count* ve *hb_err_seq* değerlerini bir artırır. *hb_err_count*, iki düğüm arasındaki Kalp-Atış hattında oluşan toplam hata sayısını tutarken, *hb_err_seq* ardışık olarak Kalp-Atışı hattında oluşan hataları sayar. Bir hatanın ardından Kalp-Atış hattından tekrar haber alındığında, *hb_err_seq* değeri 0'a eşitlenir. Kalp-Atış hattında ardışık olarak belirli miktarda hata tekrarlandığında, yedek sunucu aktif sunucudan haber alamadığından dolayı yedeğe geçme kontrol işlemlerini başlatır. Ardışık olarak yapılabilecek en fazla hata değeri yapılandırma dosyasından okunan *hb_err_limit* parametresiyle belirlenmiştir.

StartHBTimer() fonksiyonunun temel görevi, HBTimer olarak tanımlanan Kalp-Atışı süreölçerini ilklendirmek ve süreölçer kesmesini ilk kez çekirdeğe kayıt etmektir. HBTimer çekirdeğe bir kez kayıt edildikten sonra, *hb_interval* süresi dolduğunda HBTimer veri yapısına bağlı olan SendHB fonksiyonunu çalıştırır. SendHB fonksiyonu kendi içinden tekrar HBTimer veri yapısını çekirdeğe kayıt edecek ve böylelikle Kalp-Atış süreölçeri HBTimer'ın özyinelenmesi sağlanmıştır.

7.2.3 Tcp Oturum Bilgisinin Yönetilmesi

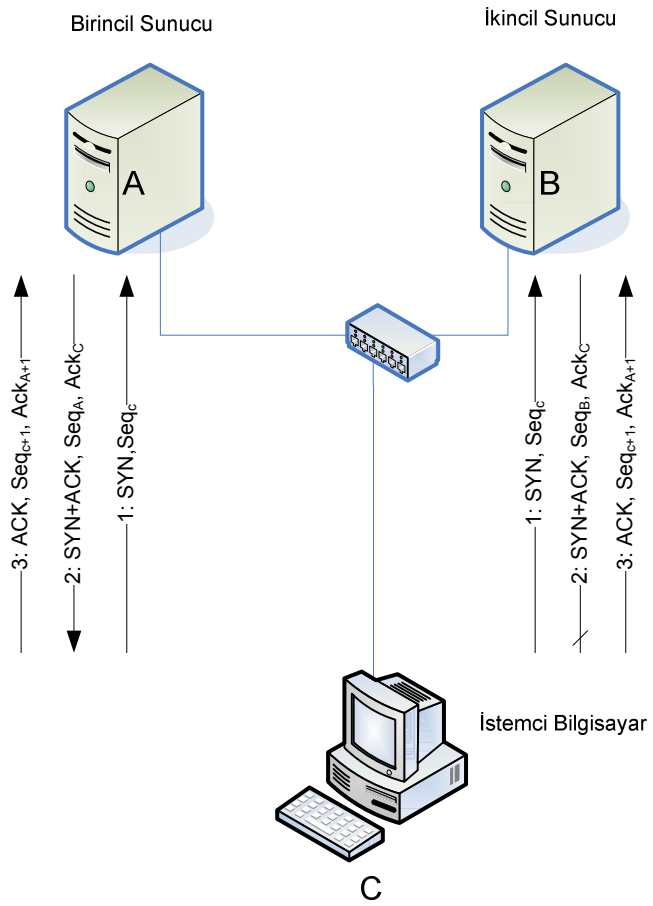
ATAFS sisteminin tasarımında iki adet eşlenik sunucu, istemci tarafının isteklerinin geldiği ağa bağlıdırlar. Bu iki sunucudan biri aktif çalışma diğeri bekleme kipinde bulunur. Bu iki sunucunun istemci tarafına bağlı olan ağ bağdaştırıcıları, ATAFS LKM'sinin sisteme yüklenmesi sırasında, *atafs.conf* yapılandırma dosyası içinde verilen bilgilere bağlı olarak aynı ATAFS servis IP'si ve MAC adresini kullanacak şekilde düzenlenir. Buna göre servisin sağlanacağı hat üzerinde iki farklı sunucu aynı IP ve MAC adresini paylaşmaktadırlar. Aynı hat üzerinde iki adet aynı ağ bağdaştırıcısı bilgisinin bulunması normal şartlarda çakışma

problemi yaşatması gerekirken, ATAFS paket işleyicisinin aktif halde olması nedeniyle bir çakışma ve sorun yaşanmamaktadır. ATAFS paket işleyicisi, düğümün çalışma kipine bağlı olarak servis ağ bağdaştırıcısının çıkış yönündeki kapısını açık tutar veya kapatır. Aktif çalışma kipindeki düğümde yüklü olan ATAFS paket işleyicisi, paketlerin ağ arayüzü üzerinden sisteme giriş yapmasına ve çıkış yapmasına sonuna kadar izin verir. Bekleme kipindeki düğümde çalışan ATAFS paket işleyicisi ise, paketlerin ağ arayüzü üzerinden sisteme giriş yapmasına izin verir, sistemden ağ bağdaştırıcısına ve servis fiziksel hattına paket çıkarılmasını engeller. Uygulama seviyesindeki katmanlardan gelen paketler, sistemin en alt katmanında sistemin kendisinden bağımsız olarak yok edildiklerinden, uygulama katmanı paketlerin yok edildiğini algılayamaz ve haberleşmesinin devam ettiğini varsayar. Bu durumda servis ağı üzerinde herhangi bir çakışma yaşanmaz ve aynı zamanda her iki düğümde de uygulama katmanının istemci tarafıyla haberleşmesi devam eder.

Tasarlanan sistemde, TCP protokolünün sürekliliğinin sağlanması için bir mekanizma oluşturulmuştur. Aktif haldeki TCP oturum bilgileri yedek makine tarafından bir karşılaştırma tablosunda tutulmaktadır. Giden ve gelen paketler incelenerek karşılaştırma tablosu sürekli güncellenmektedir. Her iki makineye de fiziksel ağ katmanından aynı ağ verisi iletildiği için TCP oturumlarının eş zamanlaması yedek makine tarafından yapılabilmektedir.

TCP oturumları, üç yönlü el sıkışma (3-way handshaking) mekanizması ile başlatılırlar. Bu aşamada her iki bilgisayar da oturumun akış kontrolünü sağlamak için birer sıra sayısı (sequence number) üretirler. Bu sıra numaralarının başlangıç değeri her bilgisayar için rasgele seçilir. Şekil 7.4'de tasarlanan sistemde bir TCP oturumunun başlangıç aşaması gösterilmiştir. İstemci bilgisayar gönderdiği SYN paketi ile bir TCP oturumu açma isteğini sunucu bilgisayara iletir. Bu pakete rasgele ürettiği bir sıra numarası (Seq_c) ekler. Tasarlanan sistemin önünde bulunan göbek (hub) cihazı paketi kopyalayarak ikincil sunucuya da aktarmaktadır. Böylece her iki sunucuya da aynı SYN paketi gitmektedir. SYN paketini alan sunucular kendi içlerinde rasgele birer sıra numarası üretirler. İstemciye yanıt olarak her iki sunucu da bir SYN+ACK paketi oluşturur. Bu durumda iki pakette de farklı sıra numaraları bulunmaktadır (Seq_A ve Seq_B). Paketin alındı (ACK) alanına ise istemcinin sıra numarası eklenmektedir. Eğer bu iki paket de istemciye ulaşırsa, istemci iki farklı sıra numarası ile cevap göreceğinden oturum başarısız olacaktır. Bu nedenle istemci her zaman sunucu tarafında tek bir sıra numarası görmelidir. Tasarlanan mekanizma, ister birincil ister ikincil sunucu aktif halde

olsun istemci bilgisayarın aradaki geçişi hissetmeksizin aynı sıra numarası ile oturuma devam edebilmesini sağlamaktadır.

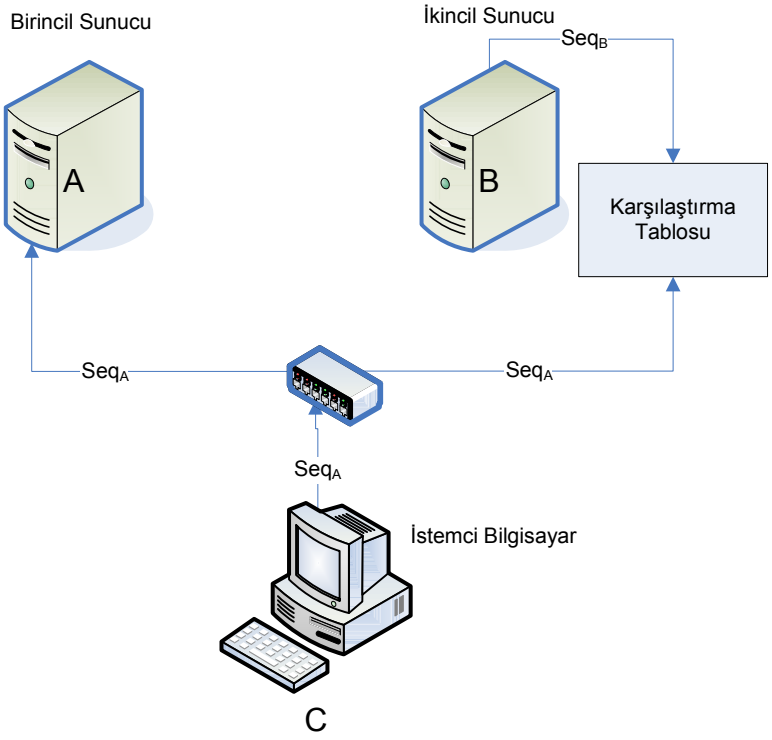


Şekil 7.4 Sistemin TCP bağlantısına ait 3 zamanlı el sıkışma akışı.

Tasarlanan mekanizmada aynı anda sadece bir sunucu bilgisayar istemci bilgisayara gönderim yapabilmektedir. Normal çalışma kipinde ikincil sunucu gelen ağ paketlerini almakta ve bu paketlerden karşılaştırma tablosundaki TCP oturum bilgisini güncellemektedir. Daha sonra paket işlenmek üzere kullanıcı uzayına gönderilmektedir. Cevap olarak oluşturulan paket mekanizma tarafından tekrar incelenmekte ve oturum bilgisi güncellenmektedir. Ancak, paket en son olarak bilgisayar ağına verilmemekte, yok edilmektedir. Böylece her iki sunucu da aynı TCP oturum bilgisini birbirleriyle haberleşmek zorunda kalmadan eş zamanlı olarak elde etmektedirler.

Bir hata durumu oluştuğunda ve birincil bilgisayar işlevsiz duruma geçtiği zaman ikincil bilgisayarın TCP oturumunu devam ettirmesi gerekmektedir. Bu durumda gönderme yönünde oluşturulan paketlerin en son aşamada silinmeyerek bilgisayar ağına verilmesi gerekmektedir. Bu aşamada dikkat edilmesi gereken nokta daha önce kurulan oturumlara ait sıra numarası

birincil sunucuya ait olan sıra numarasıdır. Ancak, ikincil sunucu da açılan oturumlar için kendi sıra numarası üretmektedir. Bu durumda karşılaştırma tablosunda her iki sunucunun da sıra numarası tutularak o anda aktif olan TCP oturumlarına ait paketlerin sıra numarasında bir dönüşüm işlemi yapılmaktadır. Gelen paketlerden birincil sunucunun sıra numarası elde edilerek karşılaştırma tablosuna yazılmaktadır. İkincil sunucu tarafından üretilen yanıttan da ikincil bilgisayarın sıra numarası elde edilmektedir. Şekil 7.5'de gösterildiği gibi gelen paketlerdeki birincil sunucuya ait sıra numarası, ikincil sunucuya ait sıra numarası ile değiştirilmekte ($Seq_A \rightarrow Seq_B$) , giden paket için ise tam tersi işlem yapılmaktadır ($Seq_A \rightarrow Seq_A$). Böylece istemci sadece Seq_A 'yı görmektedir. İkincil bilgisayar ise kendi sıra numarası ile (Seq_A) oturuma devam etmektedir. Bu dönüşüm sadece hata anındaki aktif TCP oturumları için gerekmektedir. Daha sonra açılacak yeni oturumlar için Seq_B sıra numarası kullanılmaktadır.



Şekil 7.5 Sıra numaralarının yönetilmesi

Karşılaştırma tablosu, TCP oturumlarını ayırt etmek için gerekli oturum bilgisini ve dönüşüm için gerekli sıra numaralarını Şekil 7.6'te gösterilen TCP başlık bilgisinden almaktadır.

```
typedef struct _TCPHdr
{
    u_int16_t th_sport;     /* source port */
    u_int16_t th_dport;     /* destination port */
    u_int32_t th_seq;       /* sequence number */
    u_int32_t th_ack;       /* acknowledgement number */
    u_int8_t th_offx2;      /* offset and reserved */
    u_int8_t th_flags;
    u_int16_t th_win;       /* window */
    u_int16_t th_sum;       /* checksum */
    u_int16_t th_urp;       /* urgent pointer */

}       TCPHdr;
```

Şekil 7.6 TCP başlık yapısı.

Karşılaştırma tablosu, her iki yöndeki (gelen ve giden) ağ trafiği için incelenmekte ve güncellenmektedir. Şekil 7.7'te karşılaştırma tablosunun veri yapısı gösterilmektedir.

```
typedef struct _sessionkey {
SessionKey key;
u_int32_t seqa;
u_int32_t seqb;
u_int32_t seqc;
} SessionKey;
```

Şekil 7.7 Karşılaştırma tablosunun yapısı.

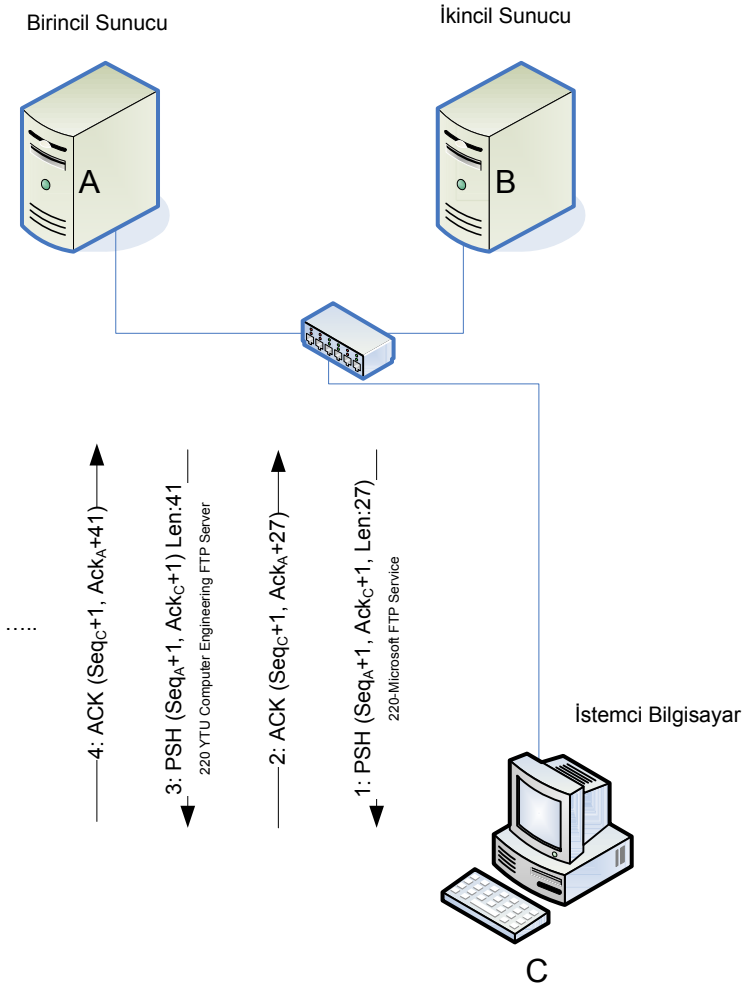
Karşılaştırma tablosunda gelen ve giden her paket için ait oldukları oturumların aranması gerektiği için oldukça işlem zamanı gerektiren bir işlemdir. Bu nedenle arama için kıyımlı arama (hashing) yöntemi kullanılmıştır. Üretilen eşsiz bir anahtar değerine göre tablo içinde arama yapılmaktadır. Anahtar yapısı Şekil 7.8'te gösterilmektedir.

```
typedef struct _sessionkey {
    u_int32_t srcIP;
    u_int32_t dstIP;
    u_int16_t srcPort;
    u_int16_t dstPort;
} SessionKey;
```

Şekil 7.8 Anahtar yapısı

Bir TCP oturumunun başlangıcında istemci ve sunucu kendilerine ait rasgele bir sıra numarası belirlerler. Bu sıra numarası TCP oturumu boyunca veri iletiminin kontrolü için kullanılmaktadır. Sıra numaraları iletişim boyunca artarak değer değiştirirler. Gönderilen bir paketi sorunsuz bir şekilde alan istemci veya sunucu gelen paketin sıra numarasını arttırarak

göndereceği bir sonraki paketin alındı kısmına yazar. Böylece bir önceki paketi başarılı bir şekilde aldığını ve bir sonraki paketi almaya hazır olduğunu diğer bilgisayara söyler. Arttırma miktarı 3-yönlü el sıkışma esnasında 1'dir. Şekil 7.9'da görüldüğü üzere oturum kurulduktan sonra ise sıra numarası gelen paketin veri miktarı kadar arttırılır. Böylece gönderilen veri miktarı ile alınan veri miktarı karşılaştırılabilir. Sıra numaraları, anlatılan bu akışa uygun olarak karşılaştırma tablosunda güncellenmektedir.



Şekil 7.9 TCP oturumu akışı.

7.2.4 Sıcak Yedeğe Geçiş

Kalp-Atış süreölçeri, *hb_err_seq* değeri ile kontrol edilen ardışık hata sayısı *hb_err_limit* değerine ulaştığında aktif sunucunun işlevsiz duruma geçtiğini öngörerek yedeğe geçme işlemlerini başlatır. Yedeğe geçme işlemini yedekte bekleyen düğüm yönlendirir. Kalp-atışı paketlerine cevap vermeyen sunucuya yine kalp-atışı hattı üzerinden yedeğe geçme

(*switchover)* işleminin başladığını bildiren bir paket gönderilir. Böylelikle durumu belirsiz olan düğümün, paket alıp cevap gönderemiyor olması durumunda aynı anda iki düğümün aktif çalışma kipine geçmesi engellenir. Yedeğe geçme işleminin başladığını bildiren yedek sunucu, *SetActive* fonksiyonunu çalıştırarak servis arayüzünün aktif olarak konuşması için gerekli parametreleri değiştirir.

İşlevsiz durumdaki düğüm bir süre sonra tekrar aktif hale geçtiği takdirde aynı ağ üzerinde birden fazla aktif düğüm oluşacağından sistemin çalışmasını bozacaktır. Bu durumun engellenebilmesi için tekrar aktif olmaya çalışan düğüm, öncelikle ortamda başka aktif sunucu olup olmadığını kontrol etmektedir. Eğer ortamdaki diğer düğüm aktif durumda ise, kendini yedekli çalışma kipine uygun şekilde yapılandırmakta ve aktif sunucuyla kalp-atışı işaretleşmesi kurmaktadır.

7.2.5 Yetkili Kullanıcı Etkileşimi

LKM'ler çekirdek seviyesinde işlem gördüklerinden, kullanıcı seviyesiyle direkt olarak iletişim kuramazlar. Tasarlanan sisteme çalışma anında parametre aktarımı ve sistemin çalışma kiplerinin değiştirilebilmesi amacıyla kullanıcı seviyesinde çalışan bir yönetim arayüzü tasarlanmıştır. Bu arayüz kullanıcı seviyesiyle çekirdek seviyesi arasında haberleşmeyi sağlayarak tasarlanan sistemi yapılandırmaktadır. Kullanıcı seviyesi ile çekirdek seviyesi arasındaki haberleşme */dev* dosya sistemi altında bir *chardev* oluşturularak sağlanmaktadır.

Bu haberleşme arayüzü ile sistemin *hb_interval, hb_err_limit* gibi parametreleri çalışma esnasında değiştirilebilmektedir. Bunun yanında bakıma alınması planlanan aktif düğümün görevinin yedek düğüme el yordamıyla aktarılması sağlanmaktadır.

7.2.6 Sistemin Durdurulması

Sistemin durdurulması istendiğinde, *rmmod* komutuyla LKM'nin Linux çekirdeğinden çıkarılması yeterlidir. Ancak LKM sistemden ayrılmadan önce ATAFS sisteminin işlevselliğini korumak için bir takım önlemler alır. Rmmod komutu yedekte bekleyen sunucu üzerinde çalıştırıldığında yedek sunucu çıkış mesajını aktif sunucuya Kalp-Atışı hattı

üzerinden gönderir. Mesajı göndermesinin ardından hafıza alanında tutmakta olduğu tüm yapıları boşaltarak Linux çekirdeğinden ayrılır. Rmmod komutu aktif sunucu üzerinde çalıştırıldığında, aktif sunucu önce görevini devredebileceği yedek düğüm olup olmadığını kontrol eder. Yedekte bekleyen bir düğüm mevcut ise çıkış yaptığı mesajını kalp-atışı hattı üzerinden yedek düğüme gönderir ve LKM'yi çekirdekten ayırır. Aktif düğümün çıkış mesajını alan yedek düğüm, *SetActive* fonksiyonunu çağırarak ATAFS sisteminin yeni aktif düğümü olarak hizmet vermeye devam eder. Böylece ATAFS sisteminin sunduğu hizmetlerin devamlılığı sağlanmış olur. rmmod komutunun çalıştırıldığı aktif sunucu, yedekte bekleyen bir düğüm bulamaz ise hafızada tutulan yapıları boşaltarak Linux çekirdeğinden ayrılır ve ATAFS sisteminin sunduğu hizmetler tamamen durdurulmuş olur.

8. KULLANILAN FONKSİYONLAR

ATAFS sistemi içerisinde kullanılan fonksiyonlar temel işlev fonksiyonları, yardımcı fonksiyonlar ve sistem fonksiyonları olmak üzere üç grupta toplanmışlardır. Temel işlev fonksiyonları, LKM'nin giriş ve çıkış fonksiyonları, LKM'e yardımcı diğer giriş çıkış fonksiyonları ve ATAFS paket işleyicisi fonksiyonu gibi ATAFS'nin özünü oluşturan fonksiyonlarından oluşurlar ve tamamı *atafs.c* kaynak dosyası içinde yer alırlar. Yardımcı fonksiyonlar, ana fonksiyonlara destek vermek amacıyla üretilmiş diğer fonksiyonları kapsamakta olup *func.c* kaynak dosyası içinde yer alırlar. Sistem fonksiyonları ise ATAFS içerisinden Linux çekirdek seviyesinde çağrılan çekirdek fonksiyonlarını kapsamaktadır.

8.1 Temel İşlev Fonksiyonları

Burada ATAFS LKM'nin en temel bileşenleri listelenmiştir.

8.1.1 Temel LKM Fonksiyonları

Bu bölümde bir LKM için zorunlu olan giriş ve çıkış fonksiyonları tanıtılmıştır.

8.1.1.1 atafs_init Fonksiyonu

ATAFS LKM'nin ilk giriş fonksiyonudur. ReadConfig, AtafsInit, dev_add_pack, StartHBTimer gibi temel fonksiyonlar buradan çağrılırlar.

8.1.1.2 atafs_exit Fonksiyonu

ATAFS LKM'nin çıkış fonksiyonudur. Temel olarak atafs_init fonksiyonunun yaptıklarını geriye döndürme görevini üstlenmiştir.

8.1.2 LKM İçindeki Diğer Fonksiyonlar

LKM içerisinde LKM için zaruri giriş ve çıkış fonksiyonları dışında, ATAFS sistemi için gerekli işlemlerin modüler biçimde tamamlandığı diğer yardımcı giriş ve çıkış fonksiyonları tanıtılmaktadır.

8.1.2.1 AtafsInit Fonksiyonu

İlklendirme işlemleri bu fonksiyonda tamamlanır. Servis ve kalp-atışı arayüzlerine dair kontroller, varolan IP yapılandırmasının kaydedilmesi, yeni IP yapılandırmasının arayüze uygulanması bu fonksiyonda gerçekleştirilir.

8.1.2.2 AtafsQuit Fonksiyonu

İlklendirme fonksiyonu olan AtafsInit'in çıkış işlemlerini yapan fonksiyondur. LKM çekirdekten ayrılmadan önce çağrılır. Servis arayüzünün eski IP yapılandırmasını geri yükler.

8.1.3 ATAFS Paket İşleyicisi Fonksiyonu

ATAFS paket işleyicisi fonksiyonu olan AtafsPacketHandler, ATAFS sisteminin en temel fonksiyonudur. Sisteme giren ve çıkan tüm paketler bu fonksiyon üzerinden geçer, bu fonksiyonda filtrelenir veya yönlendirilir. Düğümün ağ hattına çıkan musluğu gibidir. Düğümün çalışma kipine göre paketlenir düşürülmesi veya yönlendirilmesinden sorumludur.

8.2 Yardımcı Fonksiyonlar

ATAFS LKM için yazılmış diğer fonksiyonlar burada anlatılmaktadır. func.c dosyası içerisinde yer alırlar. Bu bölümde yer alan fonksiyonlar temel olarak dört bölümde incelenebilirler:

1. Yapılandırma dosyası ile ilgili fonksiyonlar

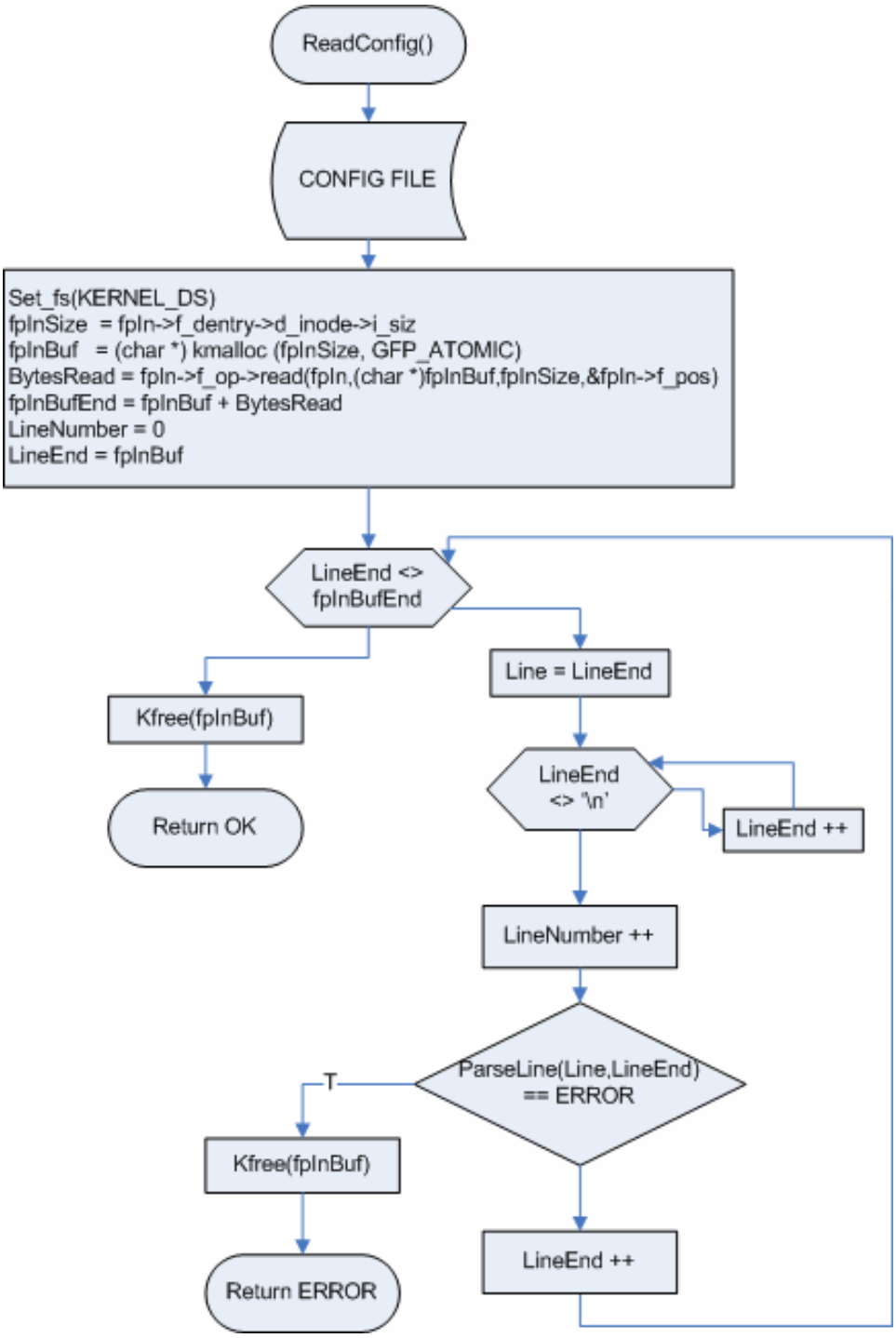
2. Ağ bağdaştırıcıları ile ilgili fonksiyonlar
3. Süreölçer ile ilgili fonksiyonlar
4. Ağ haberleşme fonksiyonları

8.2.1 Yapılandırma dosyası ile ilgili fonksiyonlar

Yapılandırma dosyasının okunması, hataların kullanıcıya iletilmesi ve yapılandırma dosyasının içeriğine göre sistem parametrelerinin atanmasına dair fonksiyonlar bu kısımda yer alırlar.

8.2.1.1 ReadConfig Fonksiyonu

Yapılandırma dosyasının okunmasından sorumlu alt program integer tipindeki *ReadConfig()* fonksiyonudur. Bu fonksiyon yapılandırma dosyasını açar, dosyanın sağlamlığını kontrol eder ve daha sonra dosyadaki her satırı okuyarak, içindeki bilgileri değerlendirmek üzere integer tipindeki *ParseLine(char *Line, char *LineEnd)* fonksiyonuna gönderir.



Şekil 8.1 Yapılandırma dosyasının okunması

8.2.1.2 ParseLine Fonksiyonu

Parseline fonksiyonu, eline geçen satırın yorum satırı, boş bir satır veya yazım yanlışı bulunan bir satır olup olmadığını kontrol eder. Hata bulmadığı takdirde satırdaki parametre ve

karşılık gelen değer bilgisini ayıklayarak integer tipindeki *SetParameter(char *parameter, char *value)* fonksiyonuna gönderir.

8.2.1.3 SetParameter Fonksiyonu

SetParameter fonksiyonunun ilk görevi eline gelen parametre bilgisini sistemde kayıtlı veri yapıları ve değişkenlerle karşılaştırarak hatalı veya doğru bir parametre tipi olduğuna karar vermektir. Parametre sistemde kayıtlı değişkenlerden biriyle uyuşuyorsa fonksiyona gönderilmiş değer bilgisi bu parametreye atanır. Parametre sistemde yer almıyorsa bu hata sistem günlüğüne yazdırılır.

8.2.1.4 AddKnownIP Fonksiyonu

LKM geliştirme sürecinde kullanılan bu fonksiyon, sistemin haberleşeceği filtre sistemine bilinen yeni bir IP ekleme görevini üstlenmiştir.

8.2.1.5 InAtoi Fonksiyonu

ASCII karakter olarak verilen bir sayı dizisini integer tipine döndürür.

8.2.1.6 IsDigit Fonksiyonu

Verilen tek bir karakterin varsa rakam karşılığını, yoksa hata kodunu döndürür.

8.2.2 Ağ Bağdaştırıcıları İle İlgili Fonksiyonlar

Ağ bağdaştırıcılarının çekirdek işaretçisinin bulunması, durumunun kontrol edilmesi, IP ve MAC adres bilgilerinin kaydedilmesi ve gerektiğinde değiştirilmesi işlemlerine dair fonksiyonlardır.

8.2.2.1 in_aton Fonksiyonu

ASCII karakterleriyle yazılmış olan IP bilgisini u32 tipine çevirir.

8.2.2.2 *in_ntoa Fonksiyonu

u32 tipinde verilen IP bilgisini ASCII karakteriyle yazılmış hale getirir.

8.2.2.3 atafs_drop_packet Fonksiyonu

Kendisine gönderilen soket ara belleğinin ttl değerini 1'e eşitlemek ve paket tipini OTHERHOST olarak belirlemek suretiyle paketlerin düşürülmesini sağlar.

8.2.2.4 AtafsGetIF Fonksiyonu

İsmi verilen ağ kartının çekirdek işaretleyicisini bulur ve döndürür.

8.2.2.5 atafs_dev_ioctl Fonksiyonu

Linux çekirdeğinin dev olarak tanımladığı birimler için sunduğu giriş çıkış kontrolü (ioctl) fonksiyonlarını çağırır. Kendisine gönderilen parametreyi çekirdeğe iletir.

8.2.2.6 sockaddr Fonksiyonu

Verilen bir soketin adres ve port bilgisini düzenler.

8.2.2.7 SaveIpAddress Fonksiyonu

Kendisine parametre olarak aktarılan ağ cihazına dair IP bilgilerini, IpPack veri yapısıyla tanımlanan alana kaydedilmesini sağlar.

8.2.2.8 SetIpPack Fonksiyonu

Kendisine parametre olarak verilen ağ cihazına, yine parametre olarak verilen IpPack içerisinde yer alan IP ve fiziksel adres bilgilerini uygular.

8.2.2.9 atafs_route_ioctl Fonksiyonu

Linux çekirdeğinin route olarak tanımladığı birimler için sunduğu giriş çıkış kontrolü (ioctl) fonksiyonlarını çağırır. Kendisine gönderilen parametreyi çekirdeğe iletir.

8.2.2.10 SetRoute Fonksiyonu

Parametre olarak verilen IpPack içinde yer alan ağ geçidi adresini sitemin varsayılan ağ geçidi olarak atar.

8.2.2.11 SetupNewIP Fonksiyonu

Yeni bir IpPack pakedi oluşturmayı sağlar.

8.2.2.12 *mac_ltos Fonksiyonu

Parametre olarak aktarılan uzun formattaki fiziksel adres bilgisini 6 byte'lık kısa formata çevirir.

8.2.2.13 xtoi Fonksiyonu

Verilen bir hexadecimal karakteri integer tipine dönüştürür.

8.2.2.14 *mac_stol Fonksiyonu

Parametre olarak aktarılan kısa formattaki fiziksel adres bilgisini uzun formata (FF:FF:FF:FF:FF) çevirir.

8.2.2.15 ShowIFAddresses Fonksiyonu

Parametre ile verilen ağ cihazına ait adres bilgilerini sistem günlüğüne yazdırır.

8.2.2.16 SetMacAddress Fonksiyonu

Verilen IpPack içerisindeki fiziksel adres bilgisini yine parametre olarak verilen ağ arayüzüne uygular.

8.2.3 Süreölçer İle İlgili Fonksiyonlar

Süreölçerlerin başlatılması, eklenmesi ve çekirdekten silinmesi ile ilgili fonksiyonlardır. Aktif ve yedek düğümde çalışan süreölçere dair fonksiyonlar da bu bölümde yer alırlar.

8.2.3.1 SetTimer Fonksiyonu

Kendisine parametre olarak verilen süreölçeri, yine parametre olarak verilen zaman aralığı sonrasında çalıştırılmak üzere kaydeder.

8.2.3.2 StopTimer Fonksiyonu

Kendisine parametre olarak verilen süreölçeri çekirdekten siler.

8.2.3.3 SendHB_SB Fonksiyonu

Bekleme kipindeki sunucuda aktif halde çalışan HBTimer_SB süreölçerini kullanır. Aktif sunucuya kalp-atışı senkronizasyon sinyali gönderir.

8.2.3.4 SendHB_PR Fonksiyonu

Aktif çalışma kipindeki sunucuda çekirdeğe kaydedilmiş olan HBTimer_PR süreölçerini kullanır. Bekleme kipindeki sunucudan gelen senkronizasyon sinyallerine cevap gönderir.

8.2.4 Ağ Haberleşme Fonksiyonları

Çekirdek seviyesinde kalp-atışı haberleşmesi ve ATAFS sisteminin ihtiyaç duyacağı diğer paket gönderme ve alma rutinleri bu kısımda yer alır.

8.2.4.1 Forward2HB Fonksiyonu

Kendisine verilen bir soket ara belleğini kopyalayarak istenilen ağ arayüzü üzerinden istenilen hedefe göndermeyi sağlar.

8.2.4.2 AtafsBuildRawPacket Fonksiyonu

Verilen parametrelere bağlı olarak, herhangi bir MAC adresinden herhangi bir MAC adresine gönderilmek üzere bir paket üretir ve çağıran üst fonksiyona döndürür. Paketin içinde yer alan IP başlığı bilgisi fonksiyona gönderilen parametrelere göre MAC adreslerinden bağımsız olarak işlenir. Paket, sistemde tanımlı olan ve parametrelerle aktarılan herhangi bir ağ bağdaştırıcısı üzerinden gönderilmek üzere düzenlenir.

8.2.4.3 AtafsSendRawPacket Fonksiyonu

AtafsBuildRawPacket fonksiyonu ile oluşturulan ağ paketi bu fonksiyon ile ağ bağdaştırıcısının gönderme kuyruğuna konulur.

8.2.4.4 AtafsBuildAndSendRawPacket Fonksiyonu

AtafsBuildRawPacket ve AtafsBuildAndSendRawPacket fonksiyonlarını ardı ardına çağıran tek bir fonksiyon olarak düzenlenmiştir.

8.2.4.5 AtafsSendHB Fonksiyonu

Main.h dosyası içinde tanımlı AtafsSendHB_SYN(),AtafsSendHB_ACK() gibi makroların çağırdığı ortak fonksiyondur. Bu tip makroların içerdiği parametrelere göre AtafsSendHB fonksiyonu, AtafsBuildAndSendRawPacket fonksiyonunu çağırır ve kalp-atışı sinyalizasyonuna hizmet eder.

9. KULLANILAN VERİ YAPILARI

Sistemin geneline ait olan veri yapıları *main.h* dosyası içerisinde tanımlanmışlardır. Sistemin genelinde kullanılan değişkenler bu dosyadaki yapılar referans alınarak *param.h* içerisinde tanımlanmıştır. Bu değişkenlerde tutulan bilgilerin bir kısmı çalışma esnasında her an değişebileceği gibi, LKM'nin yüklenmesi sırasında başlangıç değeri belirlenen ve tüm sistemin kullanımına sunulmuş değişkenler de bulunmaktadır.

Çalışma esnasında erişilen ve içeriği değiştirilen değişkenlerden bazıları şunlardır:

1. Gelen paket sayısı
2. Giden paket sayısı
3. Gönderilen Kalp-Atışı sayısı
4. Alınan Kalp-Atışı sayısı
5. Kalp-Atışı Hatası sayısı
6. Ardışık Kalp-Atışı Hatası sayısı
7. TCP Oturum karşılaştırma tablosu
8. Yedeğe geçmelilik durum bilgisi

9.1 Yerel Veri Yapıları

Bu bölümde sistemde tanımlanan veri yapıları ve amaçları tanımlanmıştır.

9.1.1 IpPack Veri Yapısı

Bir ağ arayüzüne dair IP, yayım adresleri, ağ maskesi, ağ geçidi IP'si ve fiziksel adres bilgilerinin saklanabildiği bir veri yapısıdır. ATAFS sistemi ilk çalıştırıldığında servis arayüzünün bilgilerinin saklanması ve gerektiğinde tekrar geri yüklenebilmesi için kullanılmıştır.

```
struct IpPack{
    u32    local;
```

```
    u32     address;
    u32     broadcast;
    u32     mask;
    u32     anycast;
    u32     gateway;
    unsigned char mac[MAX_ADDR_LEN];
};
```

9.2 Linux çekirdeğine ait veri yapıları

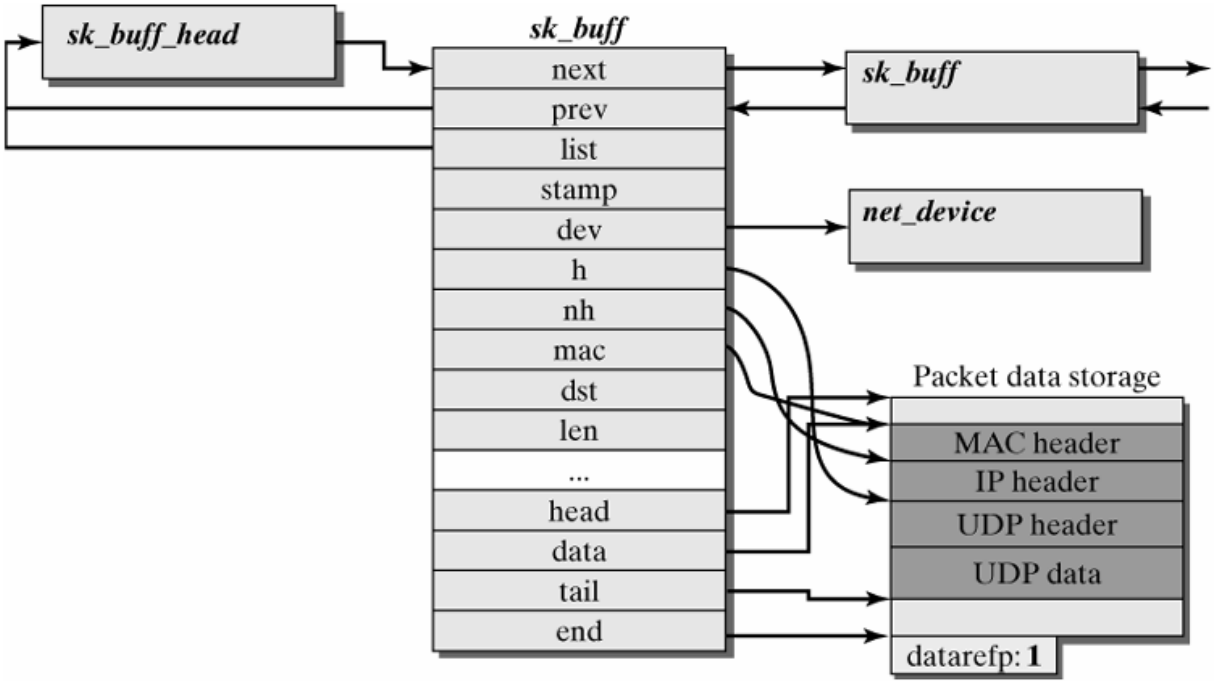
Bu bölümde Linux çekirdeğinin sağladığı ve ATAFS içerisinde kullanılan başlıca veri yapıları anlatılmaktadır.

9.3 Soket Arabelleği Veri Yapısı

Linux işletim sisteminin bilgisayar ağı gerçeklemesi belirli bir protokolden bağımsız olacak şekilde tasarlanmıştır. Bu tasarım hem ağ hem de taşıma katmanı protokollerini (TCP/IP, IPX/SPX, vb.) ve ağ bağdaştırıcı protokollerini (Ethernet, token ring, vb.) desteklemektedir. Diğer protokoller de önemli değişikliklere ihtiyaç duyulmadan herhangi bir ağ katmanına eklenebilmektedir. Soket ara bellekleri, Linux çekirdeği içerisinde paketleri temsil etmek ve yönetmek için kullanılan veri yapılarıdır.

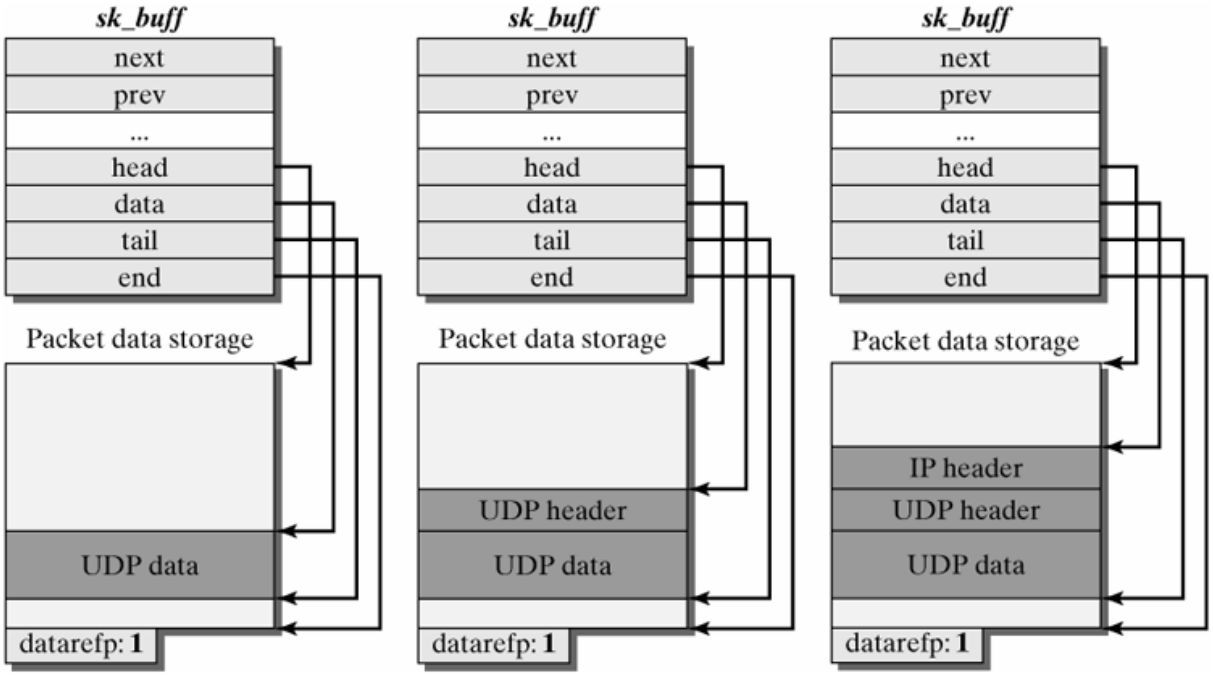
Bir soket arabelleği
Şekil **9.1**'de gösterildiği üzere iki parçadan oluşmaktadır:

- **Paket verisi:** Bu alan bilgisayar ağı üzerinde iletilen veriyi saklamaktadır. Bu veri alanı protokole ait veri birimine uygun gelmektedir.
- **Yönetim verisi (struct sk_buff):** Paket, Linux çekirdeği tarafından işlenirken, gerçek paket verisi içerisinde tutulmayan bazı fazladan veriye ihtiyaç duyar. Bunlar daha çok uygulamaya özgü verilerdir (işaretçiler, zamanlayıcılar, vb.). Bu veriler, fonksiyonlara iletilen parametrelere ek olarak, protokol kopyaları arasında değiştirilen kontrol bilgisi arayüzünü oluştururlar.



Şekil 9.1 Soket Arabelleklerinin yapısı.

Soket ara belleği, bir paketin çekirdek içinde işlendiği tüm zaman boyunca adreslenmesini ve yönetilmesini sağlayan veri yapısıdır. Bir uygulama sokete veri gönderdiğinde, soket uygun bir soket ara belleği yapısı yaratarak taşınan verinin bulunduğu adres değerini soket ara belleği değişkenlerine yazar. Katmanlar arasındaki ilerleyişinde (bkz) her katmanın paket başlık bilgisi bir önceki taşınan verinin önüne eklenir. Pakete ait başlık bilgileri için yeteri kadar yer ayrılmıştır. Böylece başlık bilgisinin birden fazla kopyalanmasından sakınılmıştır. Taşınan veri sadece iki kez kopyalanır: Bir kez kullanıcı uzayından çekirdek uzayına iletilirken ve ikincisi ise paket bilgisayar ağ bağdaştırıcısına gönderilirken. Mevcut geçerli paket verisinin önündeki boş alana ön boşluk payı (headroom) denir. Mevcut paket verisinin arkasındaki boşluğa ise arka boşluk payı (tailroom) denir.



Şekil 9.2 Protokol katmanları boyunca soket ara belleğinin değişimi.

Bir paket bilgisayar ağı bağdaştırıcısı tarafından alındığı zaman, oluşan kesme süresinde *dev_alloc_skb()* metodu kulanılarak yeni bir *sk_buff* yapısı yaratılır. Daha sonra bu yapı alınan paketin saklanmasında kullanılır. Pakete, gönderilinceye kadar yaratılan soket ara belleği üzerinden erişilir.

Şekil 9.3'te belirtilen *Sk_buff* veri yapısındaki parametreler kısaca aşağıda açıklanmıştır:

- **Next, prev:** Soket ara belleklerini kuyruk yapılarına eklemek için kullanılmaktadır (*struct skb_queue_head*). Kuyruk yapısı, sağlanan özel fonksiyonlar ile işlenmelidir (*skb_queue_head(), skb_dequeue_tail(),* vb.). Kuyruk yapısı üzerinde doğrudan değişiklik yapılmamalıdır.
- **List:** Soket ara belleğinin bulunduğu kuyruk yapısını işaret eder. Bu nedenle kuyruklar her zaman mutlaka *sk_buff_head* tipinde olmalıdır. Böylece soket ara belleği işlemleri tarafından yönetilebilirler. Eğer paket herhangi bir kuyruğa bağlı değilse bu işaretçinin değeri *null* olmalıdır.
- **Sk:** Paketin yaratıldığı soketi işaretler. Yazılımsal bir yönlendirici için, sürücüsü soket ara yüzü veri yapısını bilgisayar ağı bağdaştırıcısı yaratır. Bu durumda paket geçerli bir sokete atanamaz ve işaretçi *null* değerini alır.
- **Stamp:** Paketin Linux sistemine varış zamanını belirtir.

- **Dev ve rx_dev:** Bilgisayar ağı cihazlarına referans verirler. *Dev* soket ara belleğinin işlendiği bilgisayar ağ cihazını gösterir. Yönlendirme kararı verilinceye kadar *dev* paketin alındığı cihazı, yönlendirme kararından sonra ise paketin gönderileceği cihazı gösterir. *Rx_dev* ise her zaman paketin alındığı bilgisayar ağ cihazını gösterir.

```
struct sk_buff
{
  struct sk_buff       *next,*prev;
  struct sk_buff_head  *list;
  struct sock          *sk;
  struct timeval       stamp;
  struct net_device    *dev, *rx_dev;

  union /* Transport layer header */
  {
      struct tcphdr   *th;
      struct udphdr   *uh;
      struct icmphdr  *icmph;
      struct igmphdr  *igmph;
      struct iphdr    *ipiph;
      struct spxhdr   *spxh;
      unsigned char   *raw;
  } h;

  union /* Network layer header */
  {
      struct iphdr    *iph;
      struct ipv6hdr  *ipv6h;
      struct arphdr   *arph;
      struct ipxhdr   *ipxh;
      unsigned char   *raw;
  } nh;

  union /* Link layer header */
  {
      struct ethhdr   *ethernet;
      unsigned char   *raw;
  } mac;

  struct dst_entry     *dst;
  char                 cb[48];
  unsigned int         len, csum;
  volatile char        used;
  unsigned char        is_clone, cloned, pkt_type, ip_summed;
  __u32                priority;
  atomic_t             users;
  unsigned short       protocol, security;
  unsigned int         truesize;
  unsigned char        *head, *data, *tail, *end;
  void                 (*destructor)(struct sk_buff *);

   ...
};
```

Şekil 9.3 Sk_buf veri yapısı.

- **H, nh ve mac:** İletim(*h*), ağ(*nh*) ve ortama erişim kontrolü (*mac*) katmanlarına ait paket başlık bilgilerni işaretlerler. Bu işaretçiler paketin çekirdekteki ilerleyişi boyunca belirlenirler. Örneğin, IP paketine ait *h* işaretçisine *ip_rcv()* fonksiyonunda IP protokol başlık bilgisinin adresi atanır.

- **Dst:** Yönlendirme arabelleğindeki bir alanı işaret eder. Bunun anlamı ya paketin ileriki yolculuğuna dair bir bilgi içermektedir (paketin bilgisayardan ayrılacağı bağdaştırıcının bilgisi) ya da kalıcı başlık arabelleğinde tutulan bir MAC başlığını işaret etmektedir.
- **Cloned:** Paketin kopyalanıp kopyalanmadığını belirtir.
- **Pkt_type:** paketin tipini belirtir. Aşağıdaki değerlerden birini alır:
    - PACKET_HOST: Yerel sunucuya gönderilmiş bir paketi belirtir.
    - PACKET_BROADCAST: Genel yayın iletisini belirtir.
    - PACKET_MULTICAST: Çoğa gönderim iletisini belirtir.
    - PACKET_OTHERHOST: Yerel bilgisayara gönderilmemiş olan ancak özel durumlarda (örn., promiscuous modu) alınmış paketleri belirtir.
    - PACKET_OUTGOING: Giden yöndeki paketleri belirtir.
    - PACKET_LOOPBACK: Yerel bilgisayarın yine kendine gönderdiği paketleri belirtir.
    - PACKET_FASTROUTE: Özel bilgisayar ağı bağdaştırıcıları arasında hızlı iletim kipinde iletilen paketleri belirtilir.

- **Len:** Soket ara belleği tarafından sunulan bir paketin uzunluğunu verir. Sadece çekirdek tarafından erişilebilen veri kısmını ifade eder. Yani Ethernet paketinde sadece iki MAC adresi ve *type/length* alanları dikkate alınır. Diğer alanlar bilgisayar ağ bağdaştırıcısı tarafından daha sonra ekleneceği için çekirdek tarafından tutulmazlar.
- **Data, head, tail, end:** *data* ve *tail* işaretçileri mevcut geçerli paket verisini gösterir. Paketin işlendiği katmana bağlı olarak mevcut geçerli protokol veri birimini gösterirler. *Head* ve *end* paket verisi için kullanılabilir toplam alanı belirtirler. Belirtilen alan biraz büyük tutularak paketin önüne ve arkasına eklenecek protokol verisine yeterli alan bırakılmaktadır. Böylece veri ekleme esnasında yetersiz yer nedeniyle oluşabilecek pahalı paket kopyalama işleminin önüne geçilir. *Head* ile *data* arasındaki kalan alana *headroom*;*tail* ile *end* arasında kalan alana ise *tailroom* denir.

9.3.1 Soket Ara belleklerindeki İşlemler

Linux çekirdeği soket ara belleklerini düzenlemek için birkaç fonksiyon sunmaktadır. Bu fonksiyonlar genel olarak üç grupta incelenebilir:

- Soket ara belleği yaratan, bırakan ve çoğaltan fonksiyonlar.
- *Sk_buff* veri yapısındaki parametre ve işaretçileri işleyen fonksiyonlar.
- Soket ara belleği kuyruklarını yöneten fonksiyonlar.

9.3.2 Soket Ara Belleği Yaratma ve Bırakma

9.3.2.1 Alloc_skb()

Soket ara belleği için hafızada bir yer açar. Yeni bir soket ara belleği yaratılması esnasında *kmalloc()* fonksiyonu ile yeni bir hafıza alanı kullanılmasından ziyade daha önce oluşturulmuş bitik *sk_buff* veri yapıları değerlendirilir. Çekirdek seviyesine ayrılan hafıza alanı kısıtlı olduğu için bu alanı en verimli şekilde kullanmak gerekmektedir. Tamamen harcanmış soket ara belleği yapılarını yönetmek için iki farklı yapı vardır:

- İlk olarak, her CPU *skb_head_cache* denilen ve tamamen harcanmış paketleri tutan bir yapıyı yönetirler. Bu *alloc_skb()* fonksiyonunun yararlandığı basit bir soket ara belleği kuyruk yapısıdır.
- İkincisi ise, tamamen harcanmış *sk._buff* veri yapıları için merkezi bir yığıt yapısıdır (*skbuff_head_cache*).

Eğer, mevcut işlemci için uygun hiçbir *sk_buff* yapısı kalmamışsa, *kmem_cache_alloc()* fonksiyonu kullanılarak merkezi yapıdan bir paket elde edilmeye çalışılır. Eğer bunda da başarısız olunursa en sonunda *kmalloc()* fonksiyonu kullanılarak hafızada yeni bir yer açılacaktır.

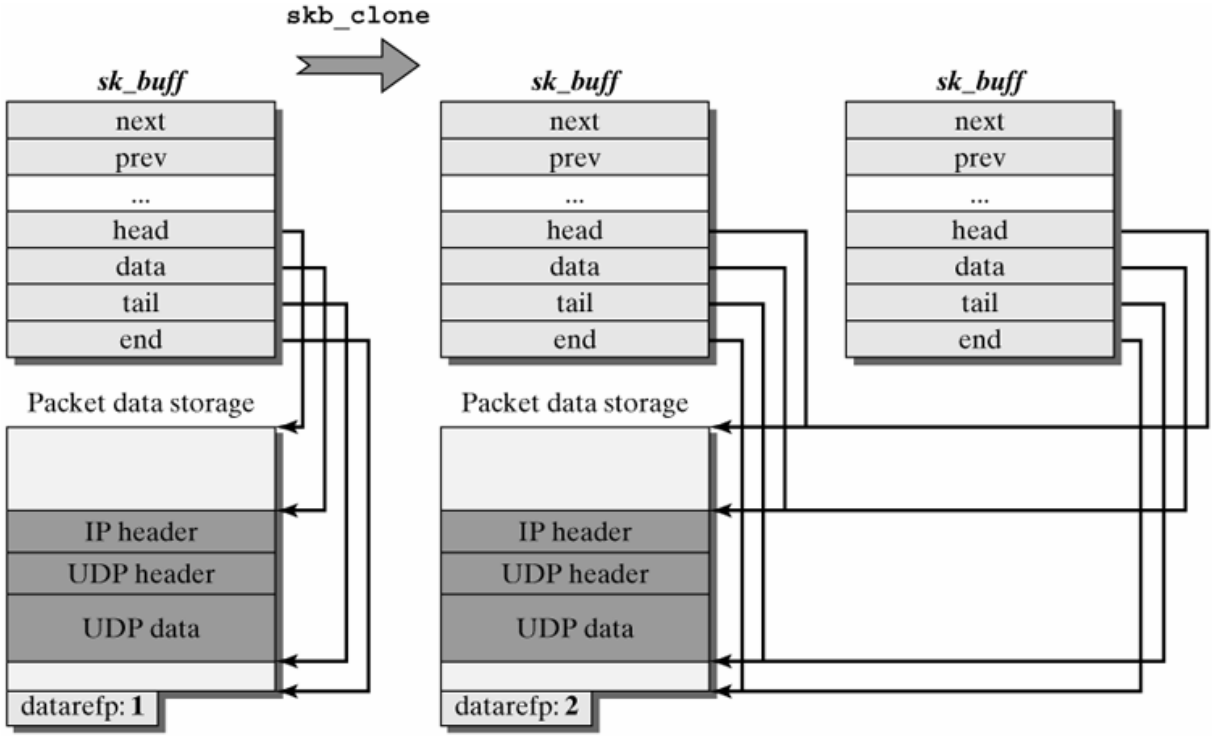
9.3.2.2 dev_alloc_skb()

alloc_skb() fonksiyonunu kullanarak bir soket arabelleği yapısı yaratır. Bu soket arabelleğinin paket veri boyutu (*length* + 16 byte)'tır. Sonradan *skb_reserve(skb, 16)* fonksiyonu kullanılarak mevcut paket 16 byte geriye taşınır. Bunun anlamı paket için 16 byte'lık ön boşluk payı ve *length* kadar arka boşluk payı ayrılır.

9.3.2.3 Skb_copy()

Soket ara belleğinin bir kopyasını çıkarır. Hem *sk_buff* veri yapısını hem de paket verisini kopyalar.
Şekil **9.4**'te görüldüğü üzere ilk fonksiyon *alloc_skb()* fonksiyonunu kullanarak yeni bir *sk_buff* yapısı elde eder. Daha sonra bazı değişkenlere değerlerini atar. İki kopyayı birleştiren tüm değerlere *null* değeri atanır.

Yeni soket arabelleğindeki taşınan veri alanı için *kmalloc()* fonksiyonu ile yeni bir alan açılır ve *memcopy()* ile kopyalanır. Daha sonra yeni veri yapısındaki işaretçiler ayarlanır. *Skb_copy()* sonucunda orijinalinden bağımsız yeni bir soket ara belleği yaratılır.



Şekil 9.4 Skb_copy()

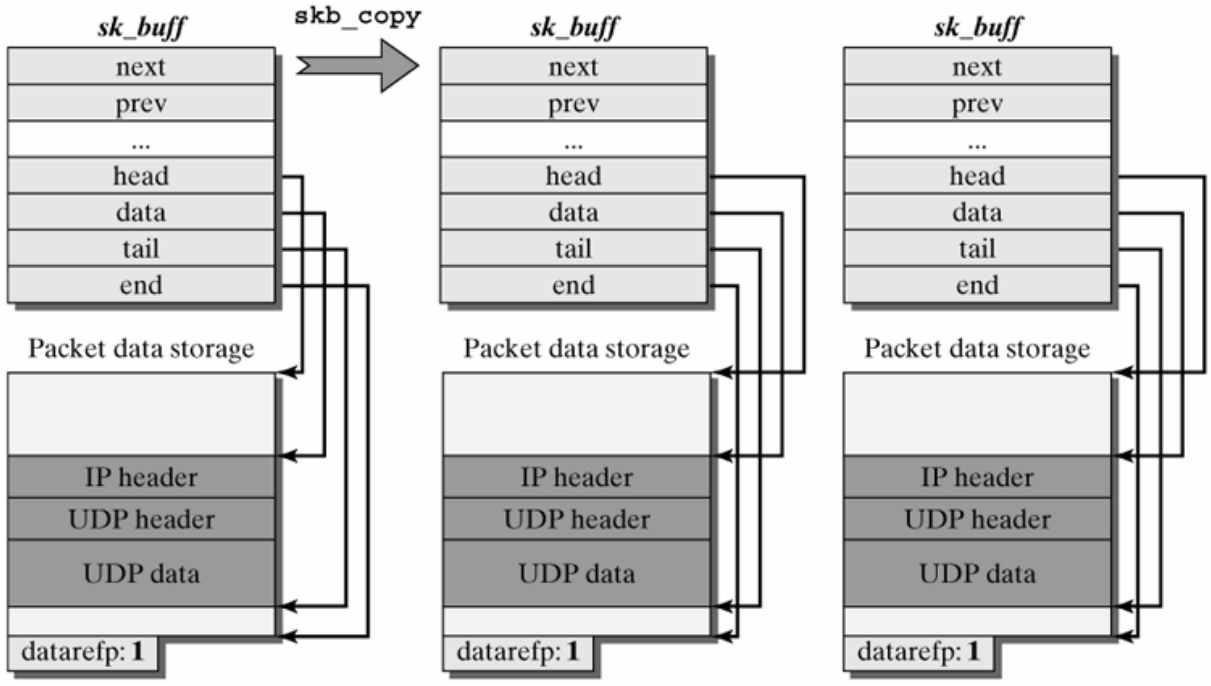
9.3.2.4 Skb_copy_expand()

skb_copy() fonksiyonu gibi yeni bir soket ara bellek yapısı yaratır. Ancak, diğerinden farklı olarak ön boşluk payı ve arka boşluk payı değerlerini parametre olarak alır. İstenen boyutta ön ve arka boşlukları bırakılmasını sağlar.

9.3.2.5 Skb_clone()

Skb_clone fonksiyonu da yeni bir soket ara belleği oluşturur. Ancak, sadece yeni bir *sk_buff* yapısı yaratırken, paket verisi için bir alan açmaz. Yeni ve orijinal yapılar aynı paket verisini işaret eder.

Şekil **9.5**'de skb_clone() fonksiyonu çağrılmadan önce ve çağırıldıktan sonraki durum görülmektedir. Paket verisinin bulunduğu hafıza alanı *datarefp* parametresinin değeri bir olana kadar serbest bırakılmaz (Paket verisine sadece bir referans kalıncaya kadar).



Şekil 9.5 Soket arabelleğinin kopyalanması.

9.3.2.6 Kfree_skb(), Dev_kfree_skb()

Kfree_skb() soket ara belleğini siler. Ayrıca, soket arabelleğinin hala bir kuyrukta olup olmadığını kontrol eder. Bunun yanında route ara belleğinden referansını siler ve eğer mevcutsa soket arabelleği için *destructor()* fonksiyonunu çağırır. *Kfree_skb()* ek güvenlik kontrolleri nedeniyle diğer fonksiyonlara göre tercih edilmelidir.

9.3.2.7 Kfree_skbmem()

Kfree_skbmem(), klonlanmamış ve kendisine referans veren örneği olmayan soket ara belleklerini temizler. Ancak, *kfree()* ile silmek yerine *skb_head_to_pool()* fonksiyonunu kullanarak yeniden kullanılmak üzere soket-ara belleği ön belleğine ekler.

9.3.2.8 Skb_header_init()

Sk_buff yapısındaki bazı alanları varsayılan değerlere ayarlar.

9.3.3 Soket Ara Belleği Kuyruklarını Yöneten Fonksiyonlar

Aşağıdaki fonksiyonlar <Linux/skbuff.h> başlık dosyasında tanımlanmışlardır.

9.3.3.1 skb_dequeue*()*

skb_dequeue() listedeki ilk arabelleği alır. Eğer liste boşsa NULL işaretçisi döndürür. Bu fonksiyon arabellekleri kuyruktan çekmek için kullanılır. Arabellekler, *skb_queue_head()* ve *skb_queue_tail()* yordamları ile eklenir.

9.3.3.2 skb_queue_head ()

skb_queue_head() arabelleği listenin başına yerleştirir. Diğer bütün liste işlemleri gibi atomik bir yapıya sahiptir.

9.3.3.3 skb_queue_tail ()

skb_queue_tail() arabelleği listenin sonuna yerleştirir ve en çok kullanılan fonksiyondur. Neredeyse bütün kuyruklar, bu fonksiyon ile veri sıralayan bir yordam seti ve *skb_dequeue()* ile aynı kuyruktan parçaları kaldıran başka bir yordam seti ile idare edilir.

9.3.3.4 skb_unlink ()

skb_unlink() bir arabelleği bulunduğu listeden kaldırır. Arabellek boşaltılmamıştır, sadece listeden çıkarılmıştır. Bazı işlemleri kolaylaştırması için arabelleğin hangi listede bulunduğunu bilmeye gerek yoktur ve her zaman herhangi bir listeye dahil olmayan bir arabellek için *skb_unlink()* çağrılabilir. Bu fonksiyon ağ kodunun, ağ protokolünün kimin kullandığını bilmediği zaman bile, bir arabelleği kullanım dışı bırakmasını sağlar. Cihaz sürücülerinin o anda kullandıkları bir arabelleğin kaldırıldığını anlamalarını önlemek amacıyla ayrı bir kilitleme mekanizması sağlanmıştır.

TCP gibi bazı karmaşık protokoller çerçeveleri düzenli tutarlar ve veri alınırken girişi tekrar düzenlerler. *skb_insert()* ve *skb_append()* isimli iki fonksiyon, kullanıcıların *sk_buff*'ları belirli bir arabelleğin önüne ve arkasına koymasını sağlarlar.

10. UYGULAMA SENARYOLARI

Uygulamanın geliştirildiği ve uygulandığı ortam Çizelge 10.1'de verilmiştir.

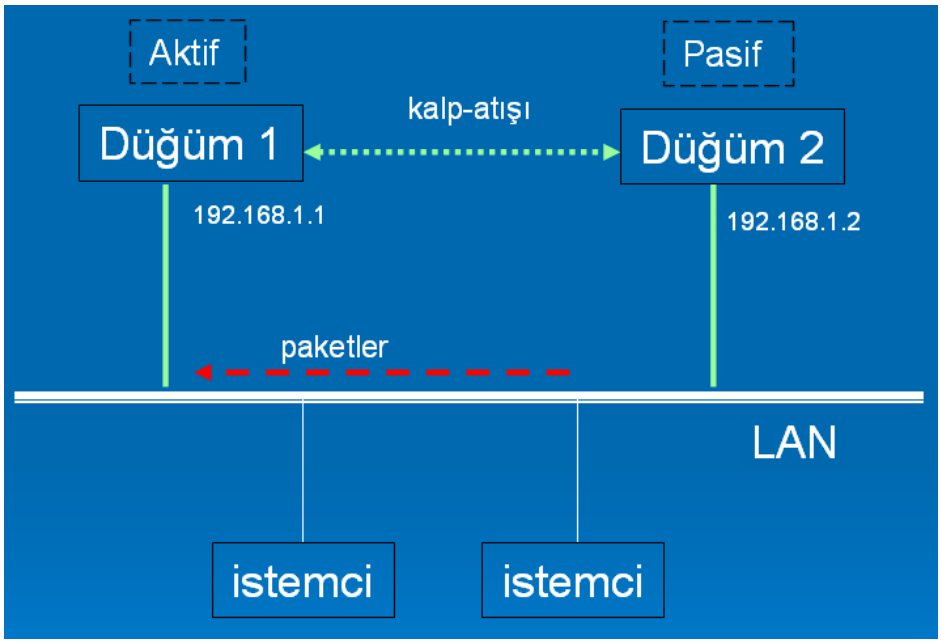
Çizelge 10.1 Uygulama ortamı

Marka	IBM Thinkcentre
İşlemci	Pentium 4 3.00GHz
Bellek	256 MB DDR
İşletim Sistemi	Suse Linux v10.1
Çekirdek Sürümü	2.6.16.13-4-smp
Derleyici Sürümü	Gcc v4.1.0 ve GNU Make v3.80
Ağ Bağdaştırıcısı (Servis Arayüzü)	Broadcom Corporation NetXtreme BCM5751 Gigabit Ethernet
Ağ Bağdaştırıcısı (Kalp-Atışı Arayüzü)	3Com Corporation 3c905 100BaseTX [Boomerang]

Uygulama sırasında oluşabilecek arıza senaryoları şunlardır:

1. Aktif düğümün servis arabirimi devre dışı kalır, servis yedek düğüme aktarılır.
2. Aktif düğüm bütünüyle devre dışı kalır. Yedek düğüm arızayı kalp-atışları ile algılar ve aktif duruma geçer.
3. Aktif düğüm ile yedek düğüm arasındaki kalp-atış bağlantısı devre dışı kalır. Yedek düğüm, aktif düğümün servis arabirimini kontrol ederek karar verir.
4. Aktif düğüm, geçici bir sıkışma nedeniyle kalp-atışı ve servis arabirimine gelen isteklere cevap veremez. Yedek düğüm, aktif düğüme ulaşamadığı süre sayacı üst sınıra ulaştığında aktif düğümün arızalandığına karar vererek aktif çalışma kipine geçer. Aktif düğüm, geçici sıkışmayı atlatıp yeniden isteklere cevap verebilecek duruma geldiğinde yedek düğümün kalp-atış bağlantısı üzerinden gönderdiği işaretlere göre bekleme kipine geçmesi gerektiği bilgisini alır.

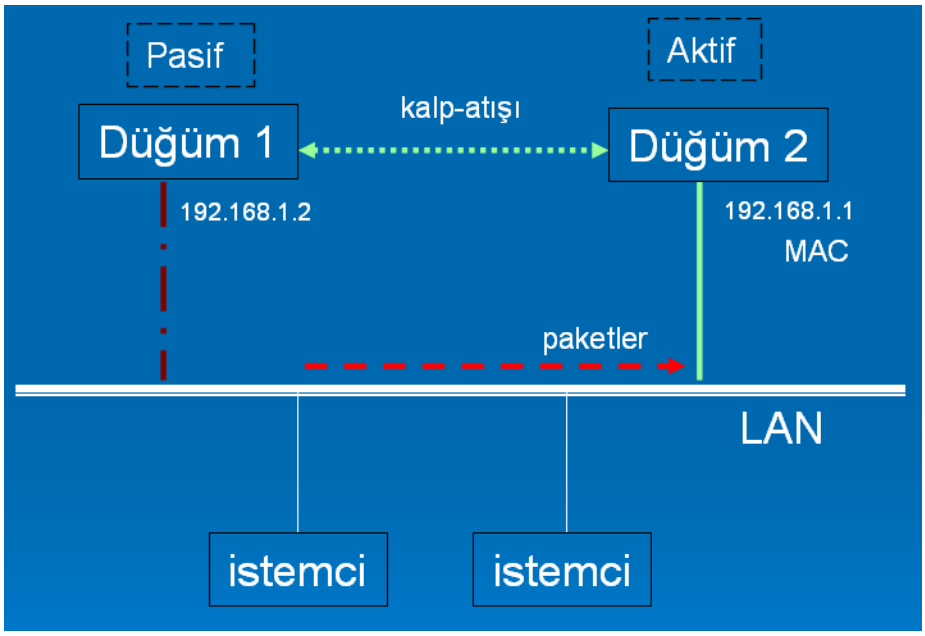
Sistemin olağan çalışması durumundaki görünümü Şekil 10.1'de verilmiştir.



Şekil 10.1 ATAFS Olağan Çalışma Durumu

10.1 Senaryo 1

Bu senaryoda aktif düğümün servis arabirimi hizmet dışı kalır. Aktif düğüm istemci tarafından gelen isteklere yanıt veremez. Bu durumda aktif düğüm, servis arabiriminin devre dışı kaldığını tespit ettiği takdirde yedek düğüme *switchover* işareti gönderir ve yedek düğüm aktif çalışma kipine geçer.

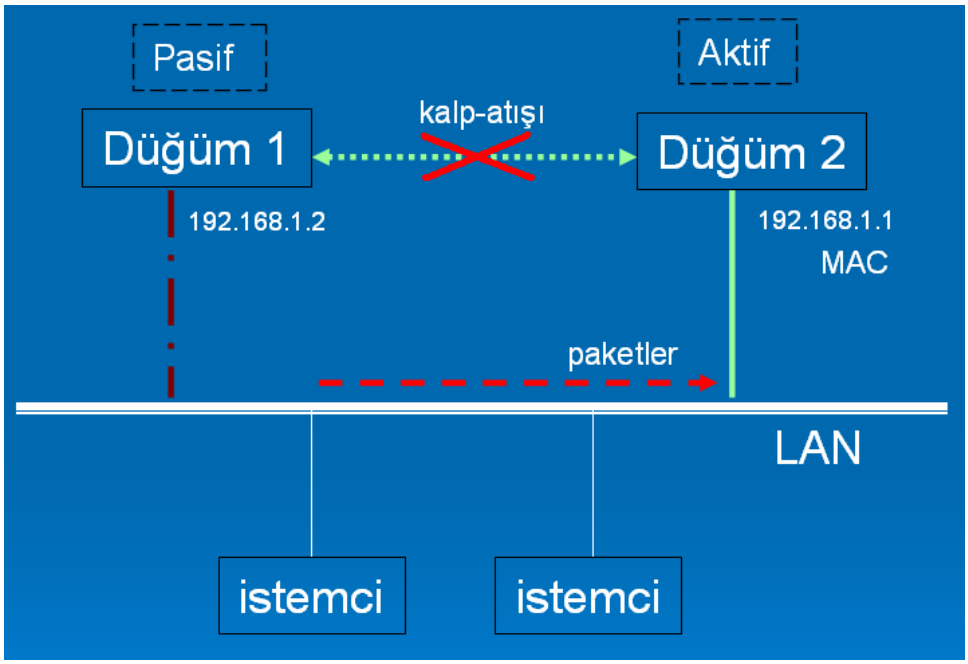


Şekil 10.2 Senaryo 1: Aktif düğümün servis arabiriminin arızalanması

Aktif düğümün arızayı fark edememesi durumunda, istemci tarafından gelen isteklere aktif düğüm tarafından cevap verilmediğini gören yedek düğüm *switchover* işlemlerini başlatır ve aktif çalışma görevini devralır.

10.2 Senaryo 2

Aktif düğümün bütünüyle devre dışı kaldığı bu senaryoda, yedek düğüm kalp-atış hattından yanıt alamadığı sürenin üst sınıra gelmesini bekler ve daha sonra öncelikle aktif düğümün servis arabirimini kontrol ederek aktif düğümün tamamen devre dışı kalıp kalmadığına karar verir. Kalp-atış ardışık hata sayısının *hb_err_limit* değerine ulaştığını gören yedek düğüm *switchover* işlemlerini başlatır ve aktif çalışma görevini devralır.

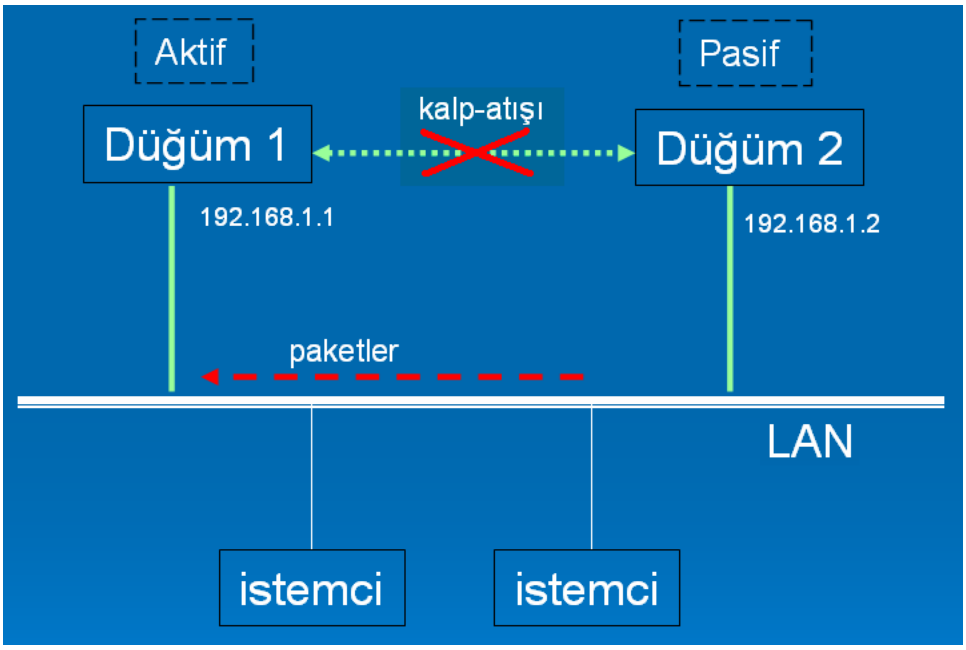


Şekil 10.3 Senaryo 2: Aktif düğümün tamamen arızalanması

10.3 Senaryo 3

Kalp-atış hattında kablo kopması, yerinden çıkması gibi her iki düğümden bağımsız bir arıza oluşması veya iki düğümden bir veya ikisinin kalp-atış arabiriminin arızalanması durumunda, düğümler arasındaki kalp-atış işaretleşmesi sağlanamaz. Bu durumda kalp-atış ardışık hata sayısının *hb_err_limit* değerine ulaştığını gören yedek düğüm, aktif düğümün servis

arabirimini kontrol ederek aktif düğümün tamamen devre dışı kalmadığını öğrenir. Bu durumda her iki düğüm, sistem günlüğüne kritik hata mesajı kaydederek, sistem yöneticisinin kalp-atış hattında arıza oluştuğunun farkına varması sağlanır. Sistemin yedeğe geçmelilik (failover) özelliği kalp-atış bağlantısı yeniden tesis edilene kadar duraksamaya uğrar.



Şekil 10.4 Senaryo 3: Kalp-atış hattının arızalanması

10.4 Senaryo 4

Bu senaryoda aktif düğüm geçici bir sıkışma nedeniyle kalp-atışı ve servis arabirimine gelen isteklere cevap veremez. İkincil düğüm, birincil düğüme ulaşamadığı süre sayacı üst sınıra ulaştığında senaryo 2'de olduğu gibi birincil düğümün arızalandığına karar vererek aktif çalışma kipine geçer. Aktif çalışma kipine geçen ikincil düğüm, birincil düğümün her an yeniden aktif çalışma kipine geçmesi ihtimalini kontrol altında tutabilmek için, kalp-atış hattına düzenli olarak kendisinin aktif çalışma kipinde olduğunu ifade eden özel bir işaret yollar. Aktif düğüm, geçici sıkışmayı atlatıp yeniden isteklere cevap verebilecek duruma geldiğinde yedek düğümün kalp-atış bağlantısı üzerinden gönderdiği işaretlere göre bekleme kipine geçmesi gerektiği bilgisini alır.

SONUÇ

Günümüzde yüksek yararlanırlıklı sistemler oluşturmak bir ihtiyaçtan öte bir mecburiyettir. Bilgisayar sistemlerinin kullanıldığı pek çok alanda yüksek yararlanırlık ve kusura dayanıklılık kritik değer arz ederken, bu sistemlerin aksamalardan toparlanma süreleri ve maliyetleri de büyük önem taşımaktadır. Yüksek yararlanırlık üzerine yapılan tasarımlar her türlü ihtiyaca cevap verebilecek genel amaçlı çözümler olduğundan, küçük çaplı sistemlerde kullanılabilecek ve düşük maliyetle yüksek performans gösterebilecek bir mimari tasarlanmaya çalışılmıştır. Bu mimari Linux işletim sistemi üzerinde C programlama dili ve Yüklenebilir Çekirdek Modülü kullanılacak şekilde tasarlanmıştır.

Yapılan tez çalışması sonucunda, tcp oturum bilgisinin uygulamaya saydam olarak bir sunucudan diğer sunucuya aktaran, aktif tcp oturumlarını kopmadan devam ettiren, sıcak yedeğe geçmeli bir sistem tasarlanmış ve hayata geçirilmesi için gerekli kodlama adımları denenmiştir.

Yapılan çalışmalar sırasında Linux çekirdek ağ mimarisinin getirdiği kolaylık ve zorluklar öğrenilmiş, paketin üst katmanlara kadar taşınmasının getireceği işlem yükü bilgisayar ağ trafiğinin en alt seviyede incelenmesiyle bertaraf edilmiştir. Çekirdek seviyesinde düşük işlem yükü ile gerçeklenen bu sistemin, her iki düğümün aynı anda devre dışı kalması gibi uç durumlar haricinde yüksek yararlanırlık oranını da beraberinde getirebileceği görülmüştür. Bu çalışma, iyi tasarlanmış, yalın bir uygulamayla basit bir donanıma yüksek yararlanırlık getirebilinmesine ve bu konuda ileride yapılacak çalışmalara ışık tutmuştur.

KAYNAKLAR

Avizienis,A., Kopetz,H., Laprie,J.C. Dependable Computing and Fault-Tolerant Systems. 1987. Springer-Verlag, New York.

Bradford,E.G. RunTime: High-performance programming Techniques on Linux and Windows 2000-Setting up timing routines. 2001. IBM.

David A.Rennels, 2004. Fault-Tolerant Computing. In: Edwin D.Reilly, (Ed.), Concise Encyclopedia of Computer Science. John Wiley & Sons, pp. 319-321.

Ge,L., Scott,L., VanderWiele,M., 2003. Putting Linux reliability to the test. The Linux Technology Center, IBM 17.

Gray,J., 1987. Why Do Computers Stop and What Can We Do About It. 6th International Conference on Reliability and Distributed Databases 3-12.

Gray,J., Siewiorek,D.P., 1991. High-Availability Computer Systems. IEEE Computer 24 (9), 39-48.

Guvensan Amac, 2006. Linux İşletim Sistemi Çekirdeği ile Bütünleşik Bir Kriptografik Sistemin Tasarımı ve Gerçeklenmesi. Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü, (11-20 pp.).

Kossak,L., 1999. Building Into The Linux Network Layer. Phrack Magazine 9.

Watanabe,E., 1985. Survey on Computer Security. Japanese Information Processing Development Corporation (JIPDEC), Mar.

INTERNET KAYNAKLARI

[1] http://en.wikipedia.org/wiki/Availability

[2] William J. Bender ve Abhinav Joshi, 2004, http://www.ppc.com/modules/knowledgecenter/highavailability.pdf

[3] Standard Glossary of Software Engineering Terminology http://standards.ieee.org/reading/ieee/std_public/description/se/610.12-1990_desc.html

[4] Oddur Benediktsson, Ağustos 2003, http://www.hi.is/pub/cs/2003-04/reliability/definitions/Fault_terms.html

[5] E-Business Foundations Without Limits, The EMC Cisco Oracle E-business "ECOstructure" initiative, May 2001 http://www.cisco.com/warp/public/779/largeent/learn/technologies/ecostructure/downloads/eco_bc.pdf

[6] www.linux.org

[7] http://en.wikipedia.org/wiki/High-availability_cluster

[8] Legard, D., 2004, "Rapid Linux Growth boosts server market" http://www.infoworld.com

[9] "Why Linux is a Better Choice than Windows", http://www.novell.com

[10] "Linux Marketplace Acceptance", http://helpdesk.du.ac.in

[11] Abramson, R., 2001 "Linux Looms Larger Than Thought", http://www.thestreet.com

[12] The Linux Kernel Module Programming Guide Peter Jay Salzman, Michael Burian, Ori Pomerantz http://www.tldp.org/LDP/lkmpg/2.6/html/index.html

[13] Linux Loadable Kernel Module HOWTO http://www.tldp.org/HOWTO/Module-HOWTO/x68.html

[14] http://www.tucows.com

[15] The Future of High-Availability, http://www.continuitycentral.com/FutureOfHA.pdf

[16] Wikipedia, High Availability Cluster, http://en.wikipedia.org/wiki/High-availability_cluster

[17] http://vista.intersystems.com/csp/docbook/DocBook.UI.Page.cls?KEY=GHA_failover

[18] IBM HACMP, http://www-03.ibm.com/systems/p/ha/

[19] Veritas Cluster Servers, http://www.veritas.com/Products/van?c=product&refId=20

[20] Linux High Availability, http://www.linux-ha.org/

[21] http://en.wikipedia.org

[22] http://www.high-availability.com

[23] http://scholar.google.com

[24] Linux Çekirdeğinde Netfilter Mimarisi
http://hmyblog.vmmatrix.net/lna/0131777203_ch19lev1sec3.html
[25] TCP/IP FTP Timeout Processes
http://www.nwrdc.fsu.edu/zOS_Support/ftp_timeout.htm
[26] RS232 Specifications
http://www.lammertbies.nl/comm/info/RS-232_specs.html
[27] The Linux Serial Programming HOWTO, Part 1 of 2 By Vernon C. Hoxie v2.0 10 Eylül 1999
http://www.lafn.org/~dave/linux/Serial-Programming-HOWTO.txt

EKLER

EK-1 Internet Protokol Numaraları

Decimal	Keyword	Protocol
0	HOPOPT	IPv6 Hop-by-Hop Option
1	ICMP	Internet Control Message
2	IGMP	Internet Group Management
3	GGP	Gateway-to-Gateway
4	IP	IP in IP (encapsulation)
5	ST	Stream
6	TCP	Transmission Control
7	CBT	CBT
8	EGP	Exterior Gateway Protocol
9	IGP	any private interior gateway (used by Cisco for their IGRP)
10	BBN-RCC-MON	BBN RCC Monitoring
11	NVP-II	Network Voice Protocol
12	PUP	PUP
13	ARGUS	ARGUS
14	EMCON	EMCON
15	XNET	Cross Net Debugger
16	CHAOS	Chaos
17	UDP	User Datagram
18	MUX	Multiplexing
19	DCN-MEAS	DCN Measurement Subsystems
20	HMP	Host Monitoring
21	PRM	Packet Radio Measurement
22	XNS-IDP	XEROX NS IDP
23	TRUNK-1	Trunk-1
24	TRUNK-2	Trunk-2
25	LEAF-1	Leaf-1
26	LEAF-2	Leaf-2
27	RDP	Reliable Data Protocol
28	IRTP	Internet Reliable Transaction
29	ISO-TP4	ISO Transport Protocol Class 4
30	NETBLT	Bulk Data Transfer Protocol
31	MFE-NSP	MFE Network Services Protocol
32	MERIT-INP	MERIT Internodal Protocol
33	SEP	Sequential Exchange Protocol
34	3PC	Third Party Connect Protocol
35	IDPR	Inter-Domain Policy Routing Protocol
36	XTP	XTP
37	DDP	Datagram Delivery Protocol
38	IDPR-CMTP	IDPR Control Message Transport Proto
39	TP++	TP++ Transport Protocol
40	IL	IL Transport Protocol
41	IPv6	Ipv6
42	SDRP	Source Demand Routing Protocol
43	IPv6-Route	Routing Header for IPv6
44	IPv6-Frag	Fragment Header for IPv6
45	IDRP	Inter-Domain Routing Protocol
46	RSVP	Reservation Protocol
47	GRE	General Routing Encapsulation
48	MHRP	Mobile Host Routing Protocol
49	BNA	BNA
50	ESP	Encap Security Payload for IPv6
51	AH	Authentication Header for IPv6
52	I-NLSP	Integrated Net Layer Security TUBA
53	SWIPE	IP with Encryption
54	NARP	NBMA Address Resolution Protocol
55	MOBILE	IP Mobility
56	TLSP	Transport Layer Security Protocol using Kryptonet key management
57	SKIP	SKIP
58	IPv6-ICMP	ICMP for IPv6
59	IPv6-NoNxt	No Next Header for IPv6
60	IPv6-Opts	Destination Options for IPv6

```
 60     IPv6-Opts   Destination Options for IPv6
 61                 any host internal protocol
 62     CFTP        CFTP
 63                 any local network
 64     SAT-EXPAK   SATNET and Backroom EXPAK
 65     KRYPTOLAN   Kryptolan
 66     RVD         MIT Remote Virtual Disk Protocol
 67     IPPC        Internet Pluribus Packet Core
 68                 any distributed file system
 69     SAT-MON     SATNET Monitoring
 70     VISA        VISA Protocol
 71     IPCV        Internet Packet Core Utility
 72     CPNX        Computer Protocol Network Executive
 73     CPHB        Computer Protocol Heart Beat
 74     WSN         Wang Span Network
 75     PVP         Packet Video Protocol
 76     BR-SAT-MON  Backroom SATNET Monitoring
 77     SUN-ND      SUN ND PROTOCOL-Temporary
 78     WB-MON      WIDEBAND Monitoring
 79     WB-EXPAK    WIDEBAND EXPAK
 80     ISO-IP      ISO Internet Protocol
 81     VMTP        VMTP
 82     SECURE-VMTP SECURE-VMTP
 83     VINES       VINES
 84     TTP         TTP
 85     NSFNET-IGP  NSFNET-IGP
 86     DGP         Dissimilar Gateway Protocol
 87     TCF         TCF
 88     EIGRP       EIGRP
 89     OSPFIGP     OSPFIGP
 90     Sprite-RPC  Sprite RPC Protocol
 91     LARP        Locus Address Resolution Protocol
 92     MTP         Multicast Transport Protocol
 93     AX.25       AX.25 Frames
 94     IPIP        IP-within-IP Encapsulation Protocol
 95     MICP        Mobile Internetworking Control Pro.
 96     SCC-SP      Semaphore Communications Sec. Pro.
 97     ETHERIP     Ethernet-within-IP Encapsulation
 98     ENCAP       Encapsulation Header
 99                 any private encryption scheme
100     GMTP        GMTP
101     IFMP        Ipsilon Flow Management Protocol
102     PNNI        PNNI over IP
103     PIM         Protocol Independent Multicast
104     ARIS        ARIS
105     SCPS        SCPS
106     QNX         QNX
107     A/N         Active Networks
108     IPComp      IP Payload Compression Protocol
109     SNP         Sitara Networks Protocol
110     Compaq-Peer Compaq Peer Protocol
111     IPX-in-IP   IPX in IP
112     VRRP        Virtual Router Redundancy Protocol
113     PGM         PGM Reliable Transport Protocol
114                 any 0-hop protocol
115     L2TP        Layer Two Tunneling Protocol
116     DDX         D-II Data Exchange (DDX)
117     IATP        Interactive Agent Transfer Protocol
118     STP         Schedule Transfer Protocol
119     SRP         SpectraLink Radio Protocol
120     UTI         UTI
```

```
121      SMP          Simple Message Protocol
122      SM           SM
123      PTP          Performance Transparency Protocol
124      ISIS over IPv4
125      FIRE
126      CRTP         Combat Radio Transport Protocol
127      CRUDP        Combat Radio User Datagram
128      SSCOPMCE
129      IPLT
130      SPS          Secure Packet Shield
131      PIPE         Private IP Encapsulation within IP
132      SCTP         Stream Control Transmission Protocol
133      FC           Fibre Channel
134-254               Unassigned
255                   Reserved
```

EK-2 ATAFS'nin derlenmesi

Aşağıda anlatılan derleme işlemleri i386 mimarisinde Suse Linux V10.1 üzerinde GNU make v3.80 kullanılarak yapılmıştır.

1. ATAFS'ye ait kaynak kod dosyalarının bulunduğu klasöre girilir.

2. make komutu çalıştırılır. Makefile dosyasındaki tanımlamaya göre atafs.ko dosyası oluşturulur.

3. Kaynak kod klasöründe yer alan atafs.conf dosyası /etc/ klasörü altına kopyalanır ve sistemin bilgilerine göre yapılandırma dosyası düzenlenir.

4. insmod atafs.ko komutuyla ATAFS LKM'si çekirdeğe yüklenir.

5. ATAFS'ye dair bilgiler sistem günlüğünden takip edilebilir.

ÖZGEÇMİŞ

Doğum tarihi	02.07.1980	
Doğum yeri	İstanbul	
Lise	1994-1997	Bayrampaşa Tuna Lisesi
Lisans	1997-2003	Yıldız Teknik Üniversitesi Elektrik-Elektronik Fak. Bilgisayar Mühendisliği Bölümü

Çalıştığı Kurumlar

2000-2003	Kuzey Maden İnşaat Taahhüt Ltd. Şirketi IT Danışmanlığı
2003-Devam ediyor	YTÜ Fen Bilimleri Enstitüsü Araştırma Görevlisi