GÖMÜLÜ SİSTEMLER İÇİN TCP/IP TASARIMI

Murat ARTUN

Yüksek Lisans Tezi

Fen Bilimleri Enstitüsü
Elektrik-Elektronik Mühendisliği Anabilim Dalı

Aralık - 2004

JÜRİ VE ENSTİTÜ ONAYI

Murat Artun'un "Gömülü Sistemler için TCP/IP Tasarımı" başlıklı Elektrik-Elektronik Mühendisliği Anabilim Dalı'ndaki, Yüksek Lisans tezi 14.12.2004... tarihinde, aşağıdaki jüri tarafından Anadolu Üniversitesi Lisansüstü Eğitim-Öğretim ve Sınav Yönetmeliği'nin ilgili maddeleri uyarınca değerlendirilerek kabul edilmiştir.

	Adı – Soyadı	İmza
Üye (Tez Danışmanı)	: Yard. Doç. Dr. Hakan Güray ŞENEL	
Üye	: Yard. Doç. Dr. Emin GERMEN	
Üye	: Yard. Doç. Dr. Cüneyt AKINLAR	

Anadolu Üniversitesi Fen Bilimleri Enstitüsü Yönetim Kurulu'nun 12.01.2005 tarih ve 3/3.... sayılı kararıyla onaylanmıştır.

Enstitü Müdürü

Prof. Dr. Altuğ İFTAR
Fen Bilimleri Enstitüsü
Müdürü

ÖZET

Yüksek Lisans Tezi

GÖMÜLÜ SİSTEMLER İÇİN TCP/IP TASARIMI

MURAT ARTUN

Anadolu Üniversitesi
Fen Bilimleri Enstitüsü
Elektrik-Elektronik Mühendisliği Anabilim Dalı

Danışman: Yard. Doç. Dr. Hakan G. ŞENEL

2004, 133 sayfa

Bu çalışmada, çok fazla işlem yükü ve bellek kullanımını gerektiren PPP/TCP/IP protokollerinin, düşük işlem gücüne sahip ve kısıtlı bellek kapasitesi olan mikroişlemci içeren gömülü sistemler üzerinde çalışacak şekilde yeniden tasarlanması gerçekleştirilmiştir. Protokol yığıtını çalıştıracak gömülü sistemin taşınabilir olabileceği de göz önünde bulundurulduğundan, Bağ Katmanında PPP ve bağlantı aygıtı olarak da telefon modem kullanılmıştır. Protokol Yığıtı yazılımı, hem çok farklı mikrodenetleyici sistemlerine kolaylıkla uyarlanabilmesi hem de gerektiğinde kolaylıkla genişletilebilmesi için C programlama dilinde, modüler programlama yaklaşımıyla gerçekleştirilmiştir.

Basit bir gerçekleme için, genel bir protokol yazılımında var olan, ama gerektiğinde kullanılmayan protokoller kodlanmamıştır. Protokollerin gerçeklenmesi sırasında, protokollerde bulunan çeşitli öğelerden ve durumlardan hangilerinin göz ardı edileceği belirlenmiş ve buna göre algoritmalarda değişiklikler yapılmıştır. Elde edilen protokol yazılımı 12 KB ROM ve 4KB RAM içine sığacak şekle getirilmiştir.

Anahtar Kelimeler: TCP/IP Protokol Yığıtı, Gömülü Sistemler, PPP Protokolü, Ağ Programlama, C Programlama Dili

ABSTRACT

Master of Science Thesis

TCP/IP DESIGN FOR EMBEDDED SYSTEMS

MURAT ARTUN

Anadolu University

Graduate School of Sciences

Electrical and Electronics Engineering Program

Supervisor: Assist. Prof. Dr. Hakan G. ŞENEL

2004, 133 pages

In this thesis, PPP/TCP/IP protocol stack, which significantly demands much computation power and a large memory, is redesigned for embedded systems with limited memory and computation power. Since it is also considered that the Embedded System running this Protocol Stack will be portable, PPP Protocol is chosen to be the Link Layer Protocol and a modem is used to establish the physical link. The Protocol Stack software is developed using the C Programming Language with modular programming approach in order both to be easily adaptable to different microcontroller systems and to be easily expandable as needed.

For the sake of simplicity, some of the protocols that are usually implemented in a general stack implementation have not been coded. Initially, it's been studied which properties and special cases can be discarded. According to discarded components, the algorithms are redesigned. The resultant protocol stack fits into 12 KB ROM and 4 KB RAM.

Keywords: TCP/IP Protocol Stack, Embedded Systems, PPP Protocol, Network Programming, C Programming Language

İÇİNDEKİLER

Sayfa

ŞEKİLLER DİZİNİ

SİMGELER ve KISALTMALAR DİZİNİ

ACCM : Asynchronous Control Character Map
ACFC : Address Control Field Compression
AHDLC : Asynchronous High Level Data Link Control
ANSI : American National Standarts Institute
ARP : Address Resolution Protocol
ASCII : American Standard Code for Information Interchange
ATCP : AppleTalk Control Protocol
ATM : Asynchronous Transfer Mode

BDM : Background Debug Mode

CHAP : Challenge Handshake Authentication Protocol
CRC : Cyclic Redundancy Check
CTS : Clear-to-Send

DARPA : Defense Advanced Research Projects Agency
DCD : Data Carrier Detect
DDP : Datagram Delivery Protocol
DNS : Domain Name System
DSP : Digital Signal Processing
DTE : Data Terminal Ready

EAP : Extensible Authentication Protocol,
EIA : Electronic Industries Alliance
EX-OR : Exclusive OR

FCS : Frame Check Sequence
FIFO : First-In-First-Out
FTP : File Transfer Protocol
FUNI : Frame User Network Interface

HDLC : High Level Data Link Control
HSSI : High-Speed Serial Interface
HTTP : HyperText Transfer Protocol

ICMP : Internet Control Message Protocol
IHL : Internet Header Length
InterNIC : Internet Network Information Center
IPX : Internet Packet Exchange
IP : Internet Protocol
IPCP : Internet Protocol Control Protocol
ISDN : Integrated Services Digital Network
ISN : Initial Sequence Number
ISO : International Standardization Organization
ISP : Internet Service Provider
ISR : Interrupt Service Routine

LAN : Local Area Network
LCP : Link Control Protocol

MRU : Maximum Receive Unit
MSS : Maximum Segment Size

NCP : Network Control Protocol
NRZ : Non-Return-to-Zero

OSI : Open-Systems Interconnected

PAP : Password Authentication Protocol
PFC : Protocol Field Compression
PPP : Point-to-Point Protokol

RFC : Request for Comments

RTP : Real-time Protocol

RTS : Request-to-Send

SCI : Serial Communication Interface

SDLC : Synchronous Data Link Control

SLIP : Serial Line Internet Protocol

SMTP : Simple Mail Transfer Protocol

SPAP : Shiva Password Authentication Protocol

SPX : Sequenced packet exchange

TCP : Transmission Control Protocol

TOS : Type Of Service

TTL : Time To Live

UDP : User Datagram Protocol

WAP : Wireless Application Protocol

1. GİRİŞ

Bilgisayar ağları, günümüz bilgi teknolojilerinde temel bir yere sahiptir. Bilginin bir kurum veya organizasyon içinde paylaşılması ihtiyacı, bilgisayarların birbirlerine bağlanması yolu ile bir ağ oluşturulmasını gündeme getirmiştir. İzleyen dönemde oluşturulan yerel ağların, dünya çapında bir bilgi alışverişine olanak sağlayabilecek şekilde birbirlerine bağlanması sonucunda İnternet doğmuştur.

Bu gelişmeyle birlikte, daha önceleri her biri kendi içinde özel bir ağ yapılanmasına ve farklı bilgisayar sistemlerine sahip olan bilgisayar ağlarının birbirileriyle karşılıklı olarak haberleşmesi probleminin çözümü, tüm yapı ve sistemler tarafından uygulanacak ortak kural ve standartların kabulünü kaçınılmaz kılmıştır. Bu probleme en kapsamlı, pratik ve esnek çözümü, bilgisayar ağlarının tüm dünya çapında birbirine bağlanması yoluyla İnternet'i oluşturma fikrini de ilk ortaya atan kurum olan ABD Savunma İleri Araştırma Projeleri Ajansı (Defense Advanced Projects Agency, DARPA) getirmiştir. DARPA tarafından önerilen bu çözümde, İnternet üzerindeki her bir aygıtın IP (Internet Protokolü) numarası denen tek-bir numarayla ifade edilmesi ve tüm global ağ iletişiminde uyulacak ağ protokollerinin katmanlı bir yapı oluşturması esas alınmaktadır. İletim Denetim Protokolü/İnternet Protokolü (Transmission Control Protocol/Internet Protocol, TCP/IP) protokol çifti ile adlandırılan bu yaklaşım ağ teknolojilerinin gelişimiyle birlikte esneklik ve pratikliğini ispatlamış ve İnternet'in temel iletişim protokolü haline gelmiştir. Ağ protokol yığıtı olarak adlandırılan, katmanlı yapının işlemesi konusunda bir standart oluşturması amacıyla Uluslararası Standardizasyon Organizasyonu (International Standardization Organization, ISO), yedi katmanlı bir yapıya sahip olan OSI (Open-Systems Interconnected) referans modelini ortaya koymuştur. Bu referans modelinden daha önce ortaya konan TCP/IP protokol yığıtı ise dört katmandan oluşmaktadır. Her iki modeldeki kilit nokta, belli bir katmanda çalışan bir protokolün ağ üzerindeki başka bir aygıtta yine aynı katmanda çalışan aynı protokolle iletişim kurmasıdır. Her ne kadar fonksiyonel bir karşılaştırmayı ifade etmekten çok uzak olsa da TCP/IP protokol yığıtı ile

ISO/OSI referans modelinin katmanlı yapısının eşleştirilmesi ve her katmanda çalışan ağ protokolleri Şekil 1.1'de gösterilmiştir [1, 2].

<table>
<tr><th>OSI Referans Modeli</th><th>TCP/IP Protokol Yığıtı</th><th colspan="5">TCP/IP Protokolleri</th></tr>
<tr><td>Uygulama</td><td rowspan="3">Uygulamalar</td><td>DNS</td><td>SMTP</td><td>HTTP</td><td>FTP</td><td>WAP</td></tr>
<tr><td>Sunum</td><td colspan="5">Soket Arabirimi</td></tr>
<tr><td>Oturum</td><td colspan="5"></td></tr>
<tr><td>İletim</td><td>İletim</td><td colspan="2">TCP</td><td colspan="2">UDP</td></tr>
<tr><td>Ağ</td><td>İnternet</td><td colspan="3">IP</td><td>ICMP</td></tr>
<tr><td>Veribağı</td><td rowspan="2">Ağ Arabirimi ve Donanım</td><td>ARP</td><td>PPP</td><td>SLIP</td><td colspan="2">HDLC/SDLC</td></tr>
<tr><td>Fiziksel</td><td colspan="2">Ethernet 802.3</td><td>Kablosuz</td><td colspan="2">Seri Haberleşme</td></tr>
</table>

Şekil 1.1. TCP/IP Protokol Yığıtı ve ISO/OSI Referans Modeli

Günümüzde birçok insan alışverişten günlük haberlere, elektronik postadan anlık mesajlaşmaya kadar pek çok konuda İnternet'i günlük hayatlarında kullanmaktadır. İnternet kullanımı ve bilgi alışverişi, genel olarak gelişmiş donanımlara sahip kişisel masaüstü, cep bilgisayarları ve mobil iletişim cihazları aracılığıyla yapılmaktadır.

Uzaktan bilgiye ulaşmanın insanlara kazandırmış olduğu kolaylık ve olanaklara paralel olarak günümüzdeki bazı uygulamalar İnternet'in yeni bir döneme girdiğini göstermektedir. Bu farklı uygulama ve yaklaşımların en yaygını daha önceleri İnternet'e bağlanamayan ve günlük hayatın vazgeçilmezleri arasında yer alan çeşitli elektronik aygıtların ve çeşitli endüstriyel uygulamaların da, artık, İnternet bağlantılı olarak düşünülmesidir [3].

Bu elektronik aygıtlar arasında kesintisiz güç kaynakları, bilgisayar yazıcıları gibi artık İnternet bağlantılı pratik uygulamalarına günlük hayatta sıklıkla rastlananlar sayılabileceği gibi; İnternet bağlantılı modellerinin yaygınlaşmaya başladığı televizyonlar, buzdolapları, çamaşır makineleri, fırınlar gibi günlük ev aletleri de sayılabilir.

Bu tür aygıtların İnternet bağlantılı olarak düşünülmesindeki amaç, yaygınlaşan küresel iletişim kavramının bu tür uygulamalarda da yerini alıp

günlük hayatı kolaylaştırmasıdır. Günümüzde pratik kullanımı gittikçe yaygınlaşan ve insanın iş yükünü önemli ölçüde azaltan mobil robotlar da İnternet bağlantılı olarak düşünülmekte, böylece bu tür aygıtlarla gerektiğinde kilometrelerce uzaktan etkin ve daha az bir maliyetle idaresi sağlanabilmektedir [4].

Bununla birlikte, 21. yüzyılın en önemli teknolojileri arasında sayılan sensör ağlarının İnternet bağlantılı olarak düşünülmesi yeni bir uygulamadır. Yeraltında, havada, sualtında, insan vücudu içerisinde, çeşitli kara ve hava araçlarında, binaların içlerinde kullanılacak olan bu sensörler çeşitli tehlikelerin ve tehditlerin algılanması ve takip edilmesini mümkün kılacaktır. [5].

Uzaktan erişimin İnternet üzerinden gerçekleştirilmesindeki temel neden, şüphesiz, düşük maliyet avantajından yararlanılmasının hedeflenmesidir. Çünkü İnternet'in altyapısı, ilk ortaya atıldığı 70li yılların başlarından beri artan bir ivmeyle gelişmektedir. Gün geçtikçe bağlantı maliyetleri azalmakta ve erişim hızları artmaktadır. Bu gelişmelerin, hem mevcut ağ yapılarının iyileştirilmesi (eski teknolojilerin daha yüksek kapasiteli yeni teknolojilerle değiştirilmesi), hem de yeni alternatiflerin (kablosuz iletişim vb.) gündeme girmesi gibi farklı yönleri vardır. Bu şartlar çerçevesinde düşünüldüğünde yukarıda anılan farklı aygıtlara uzaktan erişim için düşünülecek en basit çözüm İnternet olacaktır.

Kişisel bilgisayarlar dışında, İnternet bağlantılı olması istenen aygıtların en önemli ortak özelliği kısıtlı bellek kapasiteli ve düşük işlem gücüne sahip, fakat ucuz olan, 8-bitlik veya 16-bitlik mikrodenetleyici sistemleri olarak tasarlanmış olmalarıdır. Kesintisiz güç kaynakları, yazıcılar, evlerde bulunan cihazlar ve sensör ağlarındaki sensör düğümleri için bu tür bir sistem tasarımı fazlasıyla yeterli olabilmekte ve dolayısıyla düşük maliyet unsurunun gerekliliği göz önüne alındığında, daha yüksek kapasiteli ama pahalı mikroişlemci sistemlerinin kullanılmasının göz ardı edilmesi sonucunu getirmektedir.

İnternet bağlantısı için bir sistemde ağ bağlantısını sağlayacak gerekli donanımın bulunması ve bir ağ protokol yığıtı gerçeklemesinin çalışıyor olması gereklidir. Bu tür uygulamalar için düşünülen küçük ve basit mikrodenetleyicilerin bu iki özelliği barındırması onların birincil üretim amaçları olan kontrolün önüne geçmemelidir. Fakat tüm bir İnternet bağlantı yığıtının 8-

bitlik veya 16-bitlik bir mikrodenetleyici sisteminde gerçeklenmesi önemli miktarda bellek ve işlem kaynağı gerektirmektedir. Bunların, birincil görevi denetim olan sisteme entegrasyonu, maliyeti önemli ölçüde artırmakta ve sistemlerin tasarım amacını gölgelemektedir.

Tüm bu nedenler göz önüne alındığında, yüksek kapasiteli bilgisayar sistemleri dışında, düşük işlem kapasiteli aygıtların İnternet bağlanabilmeleri için farklı yaklaşımların kaçınılmaz olduğu ortaya çıkmaktadır.

Bu amaç için önerilen farklı yaklaşımlar arasında, TCP/IP'nin sınırlı işlevsellikle gerçeklenmesi başı çekmektedir. Yongalar üzerinde gerçeklenmiş yığıtlar ve bağlantı amacıyla tasarlanmış mikroişlemci temelli küçük sistemler diğer bir olasılıktır. Bu yöntemlerin hepsi için çeşitli avantaj ve dezavantajların varlığından bahsedilebilir [6].

Bu çalışmada, yukarıda belirtilen özelliklerde düşük bellek kapasiteli ve kısıtlı işlem gücüne sahip bir mikrodenetleyici için TCP/IP protokol yığıtının gerçeklenmesi üzerinde durulmuştur.

Aynı zamanda bu çalışmada, Ağ Arabirimi ve Donanım katmanı için seri haberleşme üzerinden modem ile bir İnternet Servis Sağlayıcının (Internet Service Provider, ISP) numarasını çevirerek Noktadan Noktaya Protokol (Point to Point Protocol, PPP) bağlantısı kuracak bir yapı düşünülmüştür. Bu düşüncenin sebebi, İnternet bağlantılı Ethernet ağının bulunmadığı, ancak telefon ağının kolaylıkla bulunabileceği durumlarda gömülü sistemin İnternet bağlantısı için geniş alternatifler sağlamaktır. Mobil bir sistem tasarımında GPRS modem kullanılırsa, PPP uygulaması yeterli olacaktır.

Bu çalışma metninin yazımında, daha çok İngilizce'lerinin yaygın olarak bilindiği ağ donanım ve protokollerinin isimlerini Türkçeleştirirken Türk Dil Kurumu'nun yayınlamış olduğu Bilgisayar Terimleri Karşılıklar Kılavuzu esas alınmıştır [7].

2. NOKTADAN NOKTAYA PROTOKOL

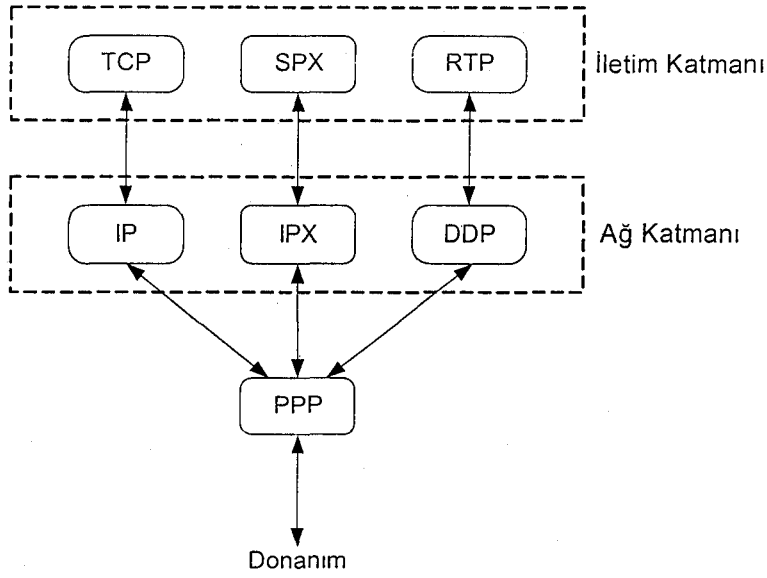
Seri port üzerinden noktadan noktaya bağlantı, veri haberleşmesinde kullanılan en eski yöntemlerden biridir. Genellikle her mikrobilgisayar ünitesi tarafından desteklenmektedir. Geçmişte, bilgisayar sistemlerinin bazıları, bu basit noktadan noktaya bağlar ile İnternet'e bağlanmışlardır.

Seri bağlantı üzerinden, Internet haberleşmesi konusundaki ilk başarılı standart, Rick Adam tarafından geliştirilen Seri Hat İnternet Protokolü'dür (Serial Line Internet Protocol, SLIP). Fakat, bu protokol sadece İnternet Protokolü (Internet Protokol, IP) datagramlarının asenkron seri hatlar üzerinden iletimini desteklemekte; IP dışındaki ağ protokollerinin desteklenmesi ve senkron hatların kullanımı gibi durumları göz ardı etmektedir. Dolayısıyla, tüm noktadan noktaya hat çeşitleri üzerinden İnternet bağlantısını sağlayacak bir protokol standardı gereksinimi Noktadan Noktaya Protokol'ün (Point-to-Point Protocol, PPP) geliştirilmesini gerekli kılmıştır [8].

Bütün noktadan noktaya bağlarda çalışacak şekilde düşünülen PPP, ağ protokol yığıtındaki tüm diğer protokoller gibi alt ve üst katmanda çalışan protokoller arasında bir ara katman görevini yerine getirecek şekilde tasarlanmıştır. Katmanlı protokol yapısı yaklaşımının da bir gereği olarak PPP'nin çalışma prensibi, gerek üst gerekse alt katmanların içeriğinden ve çalışma prensibinden bağımsızdır. Bununla beraber, alt katmandan gelen verilerdeki protokol bilgisini yorumlayıp üst katmandaki doğru protokole teslim etmek ve üst katmandan gelen verileri, alt katmana iletmekle yükümlüdür.

PPP'nin altında fiziksel katman olarak tam çift yönlü bağların bulunması gerekmekle birlikte, PPP yarım çift yönlü bağlara da uyarlanabilmektedir. PPP'nin üstünde ise ağ katmanı olarak IP, İnternet Paket Değişimi (Internet Packet Exchange, IPX) veya Datagram Teslim Protokolü (Datagram Delivery Protocol, DDP) gibi protokollerin herhangi biri bulunabilir. PPP'nin, öncülü SLIP'in özelliklerine ek olarak, IP dışındaki ağ protokollerini destekleyecek şekilde tasarlanmasının nedeni, tasarlandığı dönemde hem IP'nin günümüzdeki kadar yaygın olmaması, hem de tasarımcılarının ilerleyen zamanlarda TCP ve IP'den farklı ağ protokollerinin yaygınlaşacağı konusundaki varsayımıdır.

PPP'nin ağ protokol yığıtındaki yeri ve diğer katmanlarla etkileşimi Şekil 2.1'de gösterilmiştir.



Şekil 2.1. PPP'nin ağ protokol yığıtındaki yeri [2]

PPP, noktadan noktaya bağlar üzerinden çok-protokollü datagramların taşınması için standart bir yöntem sağlayan protokollerin birleşimi olarak tanımlanabilir. PPP üç ana bileşenden oluşmaktadır:

1. Çok-protokollü datagramları kapsülleme metodu.
2. Veri alışverişi için bağlantıyı kurmak, biçimlendirmek ve sınamak için bir Bağ Denetim Protokolü (Link Control Protocol, LCP).
3. Farklı ağ protokollerini kurmak ve biçimlendirmek için bir Ağ Denetim Protokolleri (Network Control Protocols, NCPs) ailesi [9].

Kapsülleme

Kapsülleme, üst katmanlardan gelen paketlerin, bir zarf içine konularak, alt katman üzerinden gönderilmeye hazır hale getirilmesidir.

Noktadan noktaya IP bağlarının başlangıçta tercih edilmemesinin nedeni, bu bağlar için bir kapsülleme standardının bulunmamasıdır. Bu nedenle PPP'de, hem bit-tabanlı senkron bağlar, hem de 8 veri, bir eşitlik biti ile çalışan asenkron bağlar için standart bir kapsülleme metodu bulunmaktadır. PPP, Yüksek Dereceli Veri

Bağı Denetimi (High Level Data Link Control, HDLC) metodunu kapsülleme için kullanır.

PPP'nin tüm noktadan noktaya bağları desteklemesi tasarımı sırasında öngörülen koşullardan biridir. Bu nedenle, yazılım denetimiyle (XON/XOFF Control) çalışan bağlarda, denetim verisi olarak algılanabilecek tipteki verileri uygun bir şekilde aktarması için bir kaçış mekanizması (escape mechanism) belirlenmiştir. Aynı zamanda kaçış dizisi ile gizlenmiş veriyi ortaya çıkarmak için aynı mekanizmanın ters çalıştırılması gerekmektedir.

PPP kapsülleme metodu, aynı bağ üzerinden farklı ağ protokollerinin çoklanmasını da destekler. Çünkü PPP'nin tasarımındaki temel düşünce, bu protokolün çok çeşitli host, köprü ve yönlendiricilerin kolaylıkla birbirine bağlanması için ortak bir çözüm sağlamasıdır

Ayrıca, PPP kapsülleme sırasında, iletim hatalarının uçlarda belirlenmesi amacıyla, HDLC Çerçeve Sağlama Dizisi'ni (Frame Check Sequence, FCS) kullanmaktadır. Bu yöntemle, alınan paketlerin iletim sırasında bozulmuş olup olmadığı, alıcı tarafından belirlenebilmektedir.

Bağ Denetim Protokolü (Link Control Protocol, LCP)

PPP'nin bir Bağ Denetim Protokolü kullanması, çeşitli fiziksel ortamlar için geliştirilmiş olmasındandır. Bağ denetim protokolünün temel amacı, kurulan bağlantı hakkında her iki ucun, birbirini bilgilendirmesidir. LCP, aşağıda sayılanların yerine getirilmesi amacıyla kullanılır:

- Kapsülleme yapısı için, maksimum paket büyüklüğünün ne kadar olacağı, kaçış karakterlerinin hangi değerler için kullanılacağı gibi seçeneklerin belirlenmesi,
- Paket büyüklüklerinde değişen sınırların ele alınması,
- Geri döngü ve diğer genel hataların algılanması,
- Bağın kesilmesi.

Ağ Denetim Protokolleri (Network Control Protokols, NCPs)

Ağ Denetim Protokolleri (Network Control Protokols, NCPs), noktadan noktaya bağlarda ağ protokollerini biçimlendirmek için kullanılmaktadır. Örneğin,

IP tabanlı ağlarda, IP adresinin atanması LAN ortamında bile ciddi bir sorundur. Noktadan noktaya bağlantılarda (örneğin, dial-up modem sunucuları) bu problem daha da zorlaşmaktadır. Bu tip zorlukların aşılması için her ağ protokolünün belirlenmesi ve biçimlendirilmesi amacıyla o protokole özel bir NCP geliştirilmiştir. Bunlar arasında günümüzde en çok kullanılan, İnternet Protokolü Denetim Protokolü'dür (Internet Protocol Control Protocol, IPCP).

2.1. PPP için Fiziksel Katman Gerekleri

Noktadan noktaya bağlar içinde, seri bağları destekleyen SLIP'in aksine PPP çok farklı tipteki bağ üzerinde çalışabilecek şekilde tasarlanmıştır. PPP'nin üzerinde çalışacağı bu bağlar, tam çift yönlü (full-duplex) olmak zorundadır. Bununla birlikte PPP yarım çift yönlü (half-duplex) bağlar üzerinde de çalışabilecek şekilde uyarlanabilir. PPP, asenkron, bit-sekron veya oktet-senkron bağlarda çalışabilir. Başka bir deyişle, göndermiş olduğu Veri Bağı Katmanı çerçevelerinden etkilenmeyecek yapıdaki her bağın, PPP üzerinde çalışması mümkündür.

Asenkron bağ donanımı, bir anda sadece bir karakteri gönderebilir. Senkron bağ donanımı ise bir anda farklı uzunluklarda bayt bloklarının gönderip alabilir. Bu çeşitli bağ donanımları için şu örnekler verilebilir:

- **EIA RS-232:** Bu bağ ile kişisel bilgisayar ve modem arasında olduğu gibi asenkron seri veri alışverişi yapılabilir. Karakterler başlangıç ve bitiş bitleri ile çerçevelenmiş olarak gönderilir ve alınır. Karakterlerdeki bitler bağ üzerinde sıfıra geri dönüşsüz (non-return-to-zero, NRZ) olarak kodlanır. Bitlerin ayrımı, veri alışverişi öncesinde belirlenmiş olan baud oranıyla yapılır. Baud oranı, bir hat üzerinde birim zamandaki sinyal değişimini ifade eder.
- **EIA RS-422:** İletim mantığı RS-232'ye benzer, ama, daha düşük gerilim kullanan ve daha yüksek hız sağlayan bir standarttır.
- **EIA RS-485:** RS-422'nin çok düğümlü türüdür. PPP'nin bu sistemde çalışması için bazı uyarlamalar gereklidir.
- **V.35:** Kısa mesafeli senkron bağlarda kullanılan yaygın bir arayüzdür.

- **HSSI:** Yaygın olmayan bir seri arayüzdür.
- **T1/E1:** Her ikisi de senkron olan standart telekomünikasyon arayüzleridir. T1, dört kablolu bağlar ve RJ-48 konnektörlerle çalışır ve 1.544 Mbps hızlarda veri aktarımını destekler. E1 ise, koaksiyel kablolar üzerinden 2.048 Mbps hızına kadar veri aktarımını destekler.
- **OC-3:** Optik bir telekomünikasyon arayüzüdür. PPP için 149.76 Mbps'lik hızlarda veri aktarımı mümkündür.

Bu fiziksel ortamlar dışında PPP, farklı protokollerin fiziksel ortamlarında da kullanılabilir:

- X.25 ile PPP, RFC 1598'de açıklanmıştır [10].
- ISDN üzerinden PPP, RFC 1618'de açıklanmıştır [11].
- Frame Relay ile PPP, RFC 1973'te açıklanmıştır [12].
- FUNI (Frame User Network Interface) üzerinden PPP, RFC 2363'te açıklanmıştır [13].
- AAL-5 (ATM Adaptation Layer-5) üzerinden PPP, RFC 2364'te açıklanmıştır [14].
- Eternet üzerinden PPP, RFC 2516'da açıklanmıştır [15].

PPP protokolünün çalışması için, RS232'de iletim denetimi sinyalleri olan RTS (Request-to-Send), CTS (Clear-to-Send), Data Carrier Detect (DCD), Data Terminal Ready (DTE) gibi denetim sinyallerinin kullanılması zorunlu değildir. Fakat bu sinyallerin kullanımı PPP'nin performansını artırır.

RTS, CTS, DCD, DTE benzeri donanım akış denetim sinyallerinin kullanılmasının zorunlu olmaması, PPP'nin fiziksel katmandan ayrılabilmesine olanak sağlar. Bu, hücresel radyo ağları gibi çeşitli hızlı anahtarlamalı ağ sistemlerinde de PPP'nin Veri Bağı Katmanı protokolü olarak kullanılabilmesini mümkün kılar.

2.2. PPP'de Çerçeveleme ve HDLC

2.2.1. HDLC

Şekil 2.2'de de gösterilmiş olan tipik bir HDLC çerçevesi üçü değişken olmak üzere dört ana alandan oluşur. Bu alanlardan değişken olanlar, adres, denetim ve bilgi alanları; değişken olmayan ise sağlama alanıdır. Sağlama alanı dışındaki tüm alanlar soldan sağa olmak üzere büyük oktet ilk olarak iletilirken; sağlama alanının küçük okteti ilk olarak iletilir. Sağlama alanı, farklı donanım arayüzleri için 16-bitlik veya 32-bitlik olmak üzere değişebilir. Sağlama bilgisi, önceki üç alan ve başlangıç değeri olarak sıfır kabul edilen iki veya dört baytlık doğrulama bilgisi üzerinden standart Çevrimsel Fazlalık Sağlaması (Cyclic Redundancy Check, CRC) yoluyla hesaplanır.

Bayrak 01111110	Adres	Denetim	Bilgi ...	FCS 16-32 bit	Bayrak 01111110

Şekil 2.2. Bir HDLC çerçevesi

Temel bir HDLC çerçevesinde adres ve denetim alanları tek baytlıktır. Adres alanının son biti iki baytlık adres alanı bilgisi kullanıldığında adres alanının sonunu ifade eder. Bu bitin 0 olması adres alanının ikinci bayta taştığını, 1 olması ise adres alanının sonuna gelindiğini gösterir. Dolayısıyla adres alanı okunurken son biti 1 olan ilk bayta kadarki bilginin adres alanını ifade ettiği unutulmamalıdır. 127_{10}, "tüm istasyonlar" veya "broadcast" anlamında rezerve edilmiş bir adres bilgisidir (Onaltılı sistemde son bit için 1 bilgisiyle 0xFF olarak gösterilir). Denetim alanındaki veri iletinin tipini belirtir. Bilgi alanının uzunluğu kullanılan uygulamaya bağlı olarak değişir ve bu alan kullanıcı verisini içerir [2].

2.2.2. HDLC ve PPP

PPP, RFC 1662'de belirtildiği üzere [16], ISO 3309-1979 HDLC tipi çerçeve yapısını kullanır ve asenkron bağlarda iletim sırasında eklenen başlangıç

ve bitiş bitleri bu yapının içinde değildir (Şekil 2.3). Bu çerçeve yapısında adres alanı "tüm istasyonlar" (all stations) anlamındaki rezerve 0xFF baytını ve denetim alanı da "sayılmamış bilgi" (unnumbered information) anlamındaki 0x03 baytını içerir. Her alıcı, bilgi alanı en az 1502 byte içeren bir paketi alabilecek özellikte olmalıdır. Bilgi alanı 8-bitin tamsayı katları uzunluğunda olmak zorundadır ve protokol, bilgi ve padding alanlarının bileşiminden oluşmaktadır. Üç bileşenden biri olan padding, eğer üst protokol tarafından teslim edilen veri maksimum iletim biriminden küçükse bu çerçevenin büyüklüğünü artırmak için çerçeveleme sırasında eklenir ve opsiyoneldir. Dolayısıyla HDLC tipi çerçevelenmiş bir veri kesinlikle protokol ve bilgi alanlarını içermek zorundadır. Protokol alanı, PPP paketi içinde kapsüllenmiş olan bilginin hangi protokole ait olduğunu belirtir. Sağlama alanda CRC değeri bulunur. Bu değer, tüm HDLC tipi çerçevelenmiş paket üzerinden 16-bitlik FCS ile hesaplanır [2, 16].

Bayrak 0x7E	Adres 0xFF	Denetim 0x03	Protokol 8/16 bit	Bilgi ...	Padding ...	FCS 16-32 bit	Bayrak 0x7E

Şekil 2.3. PPP HDLC tipi paket yapısı [16]

2.2.3. PPP'de Çerçeveleme

PPP, asenkron HDLC (AHDLC), bit-temelli HDLC, oktet-tabanlı HDLC olmak üzere üç çeşit çerçeveleme metodu kullanır.

Noktadan noktaya seri bağlar arasında veri alışverişinde veriler, AHDLC ile çerçevelenirler. Bu çerçeveleme yönteminde, onaltılı sayı sisteminde karşılıkları 0x7E ve 0x7D olan iki özel değer kullanılır. 0x7E, çerçeve sınırlayıcısı olarak AHDLC ile çerçevelenmiş bir PPP paketinin sonunu ve ikinci bir çerçevenin başlangıcını ifade eder. Buna göre ardışık şekilde gönderilen çerçevelenmiş veriler için çerçevenin sonunu ifade eden 0x7E oktetinden sonra yeni bir 0x7E okteti bulunmaz ve aradaki 0x7E okteti hem önceki çerçevenin sonunu hem de yeni çerçevenin başlangıcını ifade eder.

0x7D okteti ise, kullanıcı verisi içinde bulunan hem 0x7E verisini, hem de ASCII kontrol setindeki (ASCII karakter setinin ilk 32 karakteri) diğer kontrol

karakterlerini alttaki donanım arayüzünden gizlemek için bir kaçış dizisi oluşturmak üzere kullanılır. Kullanıcı bilgisindeki gizlenecek karakterlerin 0x20 değeriyle EX-OR (Exclusive OR, Dışlayan Veya) işlemine tabi tutularak gerçek değer yerine önce 0x7D ardından da EX-OR işleminin sonucu donanım arayüzüne gönderilir. Buna göre, örneğin, kullanıcı bilgisinde bulunan bir 0x7E değeri, karşıdaki alıcı tarafından çerçeve sınırlayıcısı olarak algılanmaması için önce 0x7D ve ardından da 0x5E (0x7E ile 0x20 oktetlerinin EX-OR işlemi sonucu) olarak gönderilir.

Bu noktada belirtilmesi gereken önemli bir konu, 16-bitlik CRC değerinin kaçış dizileri oluşturulmadan önce tüm paket üzerinden hesaplandığıdır. AHDLC ile çerçevelenerek oluşturulmuş, 0x7E ve tüm ASCII kontrol karakterleri yerine karşılık gelen kaçış dizileri yerleştirilmiş bir PPP paketi Şekil 2.4'te gösterildiği gibidir. Şekilde görüldüğü gibi kaçış dizilerinin yerleştirilmesi paket boyunu en az bir bayt; en fazla yaklaşık iki katı kadar büyütebilmektedir [2, 16].

```
0x7E 0xFF 0x7D 0x23 0xC0 0x21 0x7D 0x21 0x7D
0x21 0x7D 0x20 0x7D 0x3C 0x7D 0x21 0x7D 0x24
0x7D 0x25 0xDC 0x7D 0x22 0x7D 0x26 0x7D 0x20
0x7D 0x2A 0x7D 0x20 0x7D 0x20 0x7D 0x23 0x7D
0x24 0xC0 0x23 0x7D 0x25 0x7D 0x26 0x82 0xC0
0x59 0xE7 0x7D 0x27 0x7D 0x22 0x7D 0x28 0x7D
0x22 0xEA 0xDE 0x7E
```

Şekil 2.4. AHDLC ile çerçevelenmiş bir PPP paketi

Bit-temelli HDLC, PPP protokolünün T1/E1 veya ISDN gibi telekomünikasyon ortamları üzerinden kullanımı için geliştirilmiştir. Çerçeveleme ve CRC hesabı AHDLC'nin tersine donanım tarafından gerçekleştirilir. Kaçış dizileri kullanılmaz ve akış denetimi yoktur.

Oktet-temelli HDLC ise çok daha az kullanılır. Oktet-temelli çerçeveleme, kaçış dizileri ve çerçeveleme kodları bakımından AHDLC'ye benzemekle birlikte en belirgin farkı ASCII kontrol setinin kaçış işlemine tabi tutulmamasıdır. Bu, çok

hızlı veri aktarımlarının yapıldığı özel ortamlar üzerinden PPP uygulamalarında kullanılır [2].

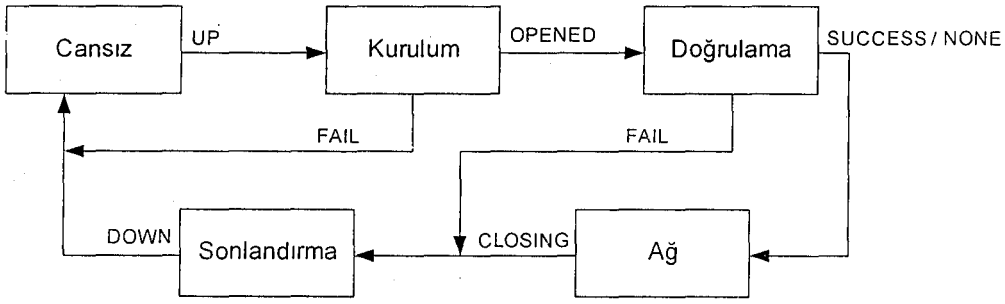
2.3. PPP'de Bağ İşlemi

Noktadan noktaya bir bağ üzerinden bilgi alışverişi için PPP bağının karşılıklı olarak kurulması gerekmektedir. Öncelikle PPP bağının her iki ucu, bağı biçimlendirmek ve sınamak için, birbirlerine karşılıklı olarak LCP paketleri gönderirler. Ardından PPP bağının kurulması için ilk paketi gönderen ucun kendisini, kullanıcı adı ve şifresiyle PPP sunucusuna doğrulatması gerekir. Bağ kurulum evresinin veya eğer gerekiyorsa doğrulama evresinin başarılı bir şekilde son bulması ağ evresini başlatır. Bu evrenin başında bağın iki ucu tarafından belli bir NCP belirlenir. Bağın iki ucu bu belirlemenin ardından NCP paketlerinin alış/verişi kurallarına uyarak biçimlendirmeye devam eder.

Bazı NCP protokolleri arasında ATCP (AppleTalk Control Protocol, AppleTalk Denetim Protkolü), IPCP (Internet Protocol Control Protocol), Novell IPX Control Protocol sayılabilir. Bunların ileti yapısı çok az farklılıklarla LCP paketleriyle aynıdır. Bu evrede ağ adreslerinin belirlenmesi için iletiler gönderilir. IPCP protokolünde, IP numarası sunucu tarafından istemciye atanır.

Kurulan PPP bağı, kapatma amaçlı LCP ve NCP paketleri yollanmadıkça veya bir dış olay (bir hareketsizlik zamanlayıcısının süresinin dolması veya ağ yöneticisini bağı kapatması) gerçekleşmedikçe açık kalır.

Noktadan noktaya bağın kurulumu, biçimlendirilmesi, devam ettirilmesi ve sonlandırılması işlemlerinin tamamında PPP'nin geçmiş olduğu evreler, Şekil 2.5'te gösterilmiştir. İlk aşamada CANSIZ evresinde başlayan PPP, karşılıklı kurulum paketlerinin değişildiği KURULUM evresine gelir. Maksimum paket boyutu, kaçış karakterleri, şifre doğrulama protokolü gibi seçeneklerin bağlantı kuracak uçlar tarafından birbirlerine gönderdikleri paketlerle belirlendiği KURULUM evresi bitince, kullanıcı adı ve şifrenin değişildiği DOĞRULAMA evresi başlar. Her hangi bir durumda, hata olması sistemin SONLANDIRMA evresine geçmesine neden olur. Sonuçta, DOĞRULAMA evresinin bitmesinden sonra gelinen AĞ evresinde, IP paketlerinin gönderilmesine başlanır.



Şekil 2.5. PPP bağ evreleri [17]

Bu tezin konusu olan, kısıtlı PPP/TCP/IP gerçeklemesinde, bu evrelerin hepsinin, standartlarında bulunduğu şekilleriyle ele alınmasının gerekli olmadığı görülmüştür. Fakat, şekilde de görülen üç temel evre mutlaka gerçekleştirilmelidir. Bunlar, Kurulum (LCP sorgulamaları ile), Doğrulama (Doğrulama Protokolleri ile) ve Ağ (NCP sorgulamaları ile) evreleridir. Bu üç evre dışındaki diğer evreler Cansız evre ve Sonlandırma evreleridir.

PPP'nin gerçeklemeleri aracılığıyla iletilen veriler her katmanda farklı ifadelerle tanımlanmaktadır. Datagram, Ağ katmanındaki iletim birimidir. Ağ katmanındaki bir datagram Veri Bağı (Data Link) katmanında birden fazla paket ile kapsüllenebilir. Çerçeve, Veri Bağı katmanındaki iletim birimidir. Bir çerçeve tipik olarak üst bilgi, alt bilgi ve belli miktarda veri içerir. Paket, Ağ katmanı ile Veri Bağı katmanı arasında iletilen temel kapsülleme birimidir.

Bir paketin PPP gerçeklemesi tarafından algılanamayarak işleme konulmaksızın gözardı edilmesi olayı "sessizce atma" (silently discard) kavramıyla tanımlanmıştır [17].

2.3.1. Cansız Evre

Bağ mutlaka bu evre ile başlar ve biter. Harici bir olay (modemin fiziksel bağlantıyı kurması veya programcı tarafından verilen bağlan komutu) sonucu fiziksel katmanın kullanım için hazır olduğunun bilinmesiyle, PPP Kurulum evresine geçer.

2.3.2. Kurulum Evresi

Bu evrede bağın kurulması için, LCP protokolü kullanılarak karşılıklı olarak biçimlendirme paketleri değişilir. Bu değişimin başarılı olarak sonuçlanmış sayılması için, bir çift iletiden oluşan Kur – Onayla (Configure-Ack) dizisinin uçlar arasında değişilmiş olması gerekmektedir. Bu evrede sorgulanmayan herhangi bir biçimlendirme seçeneği için, standartta varsayılan değer kabul edilmiş sayılır.

Bu evrede biçimlendirilen seçenekler ve parametreler, üst katman olan Ağ katmanı protokollerinden bağımsızdır. Bu seçeneklerin ayrıntıları Bölüm 2.6'da ele alınacaktır.

İleti değiş tokuşu sırasında LCP protokolünü içermeyen herhangi bir paket sessizce atılır. Ağ veya Doğrulama evresinde alınacak herhangi bir Kur – İstek (Configure-Request) paketi Kurulum evresine dönüşe neden olmaktadır.

2.3.3. Doğrulama Evresi

Bazı bağlarda (örneğin bir İnternet Servis Sağlayıcısı'nın arandığı çevirmeli bağlarda) bağın bir ucunun diğer uca kendini doğrulaması (authenticate) gerekmektedir. Aslında doğrulama her bağda gerçekleştirilir. Doğrulama amacıyla hangi protokolün kullanılacağı doğrulamayı isteyen uç tarafından belirlenir ve diğer uç bu protokole uymak zorundadır. Bu doğrulama protokolü Kurulum evresinde belirlenir. En basit doğrulama protokolü, kullanıcı ismi ve şifrelerin ASCII karakter dizileri olarak, şifrelenmeden gönderildiği, Şifre Doğrulama Protokolü'dür (Password Authentication Protocol, PAP). PPP gerçeklemelerindeki doğrulama işlemi Bölüm 2.7'de ayrıntılı olarak ele alınmıştır.

Doğrulama evresinden Ağ evresine geçiş için doğrulamanın tam olarak bitmiş olması gerekir. Eğer doğrulama sağlanamamışsa, doğrulamayı isteyen uç Sonlandırma evresine geçer ve bağı koparır.

Bu evrede doğrulama protokolü ve bağ kalite izleme paketleri dışında alınan tüm paketler sessizce atılır.

2.3.4. Ağ Evresi

Bu evrede PPP üzerinde ağ katmanında kullanılacak ağ protokollerinden her biri ayrı ayrı biçimlendirilmelidir. Bu evrenin başında kullanılan bir NCP tam olarak bittiğinde, PPP gerçeklemesi artık o NCP tarafından biçimlendirilmiş olan ağ protokolünün paketlerini taşıyacaktır.

Bu evrede artık bağ trafiği, LCP, NCP veya herhangi bir ağ protokolü paketlerinden herhangi birini içerebilir.

LCP, "bağlantı açık" durumuna geçtikten sonra bir uçtaki PPP gerçeklemesi tarafından desteklenmeyen protokolü içeren bir paket için Protokol – Reddet (Protocol-Reject) iletisi gönderilir. Sessizce atma işlemi sadece protokolü desteklenen ama hatalı olduğu belirlenen paketlere uygulanır.

2.3.5. Sonlandırma Evresi

Bir uçtaki PPP gerçeklemesi kurulan bağı herhangi bir zamanda sonlandırabilir. Sonlandırma işlemi, taşıyıcı sinyalinin kaybolması, doğrulamanın sağlanamaması, bağ kalitesinin sağlanamaması, boş-zaman zamanlayıcısının süresinin dolması, veya programcının bağı kapatması sebebiyle olabilir.

Bağı sonlandırmak için LCP protokolü kullanılarak karşılıklı olarak Kapat (Terminate) paketleri değişilir. Bağ kapanırken, PPP, ağ protokolünü, bağın kapandığı konusunda haberdar eder. Aynı zamanda, fiziksel katmana da bağın sonlandırılması için bir sinyal gönderilir.

Kapat – İstek (Terminate-Request) paketi gönderen bir uç karşı taraftan Kapat – Kabul (Terminate-Ack) paketi almadıkça bağı sonlandırmaz. Bağ sonlandırıldıktan sonra PPP gerçeklemesi Cansız (Dead) evresine geri döner.

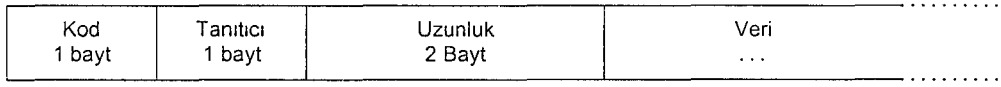
Bu evrede LCP protokolü dışında protokol içeren herhangi bir paket sessizce atılmalıdır.

2.4. LCP Paket Biçimleri

LCP protokolü içinde tanımlanmış paketler dört grupta incelenebilir:

1. *Bağ biçimlendirme paketleri*, PPP bağının kurulması ve biçimlendirilmesi için gönderilir ve alınır (Configure-Request, Configure-Ack, Configure-Nak ve Configure-Reject).
2. *Bağ sonlandırma paketleri*, bir bağın sonlandırılması için gönderilir ve alınır (Terminate-Request ve Terminate-Ack).
3. Bağ bakım paketleri bir bağın yönetimi ve hatalarının ayıklanması için gönderilir ve alınır (Code-Reject, Protocol-Reject, Echo-Request, Echo-Reply ve Discard-Request).
4. Diğer paketler (Identification, Time-Remaining, Reset-Request, Reset-Ack, Vendor Extension).

Bilgi alanında LCP paketi bulunan bir PPP paketinin protokol alanı onaltılı sayı düzeninde, 0xC021 değeri içerir [18]. Bir PPP paketinin bilgi alanında sadece bir tane LCP paketi bulunabilir. Bir LCP paket biçimi Şekil 2.6'da gösterilmiştir.



Şekil 2.6. LCP paket biçimi [17]

Kod Alanı

Kod alanı bir bayt uzunluğunda olup LCP paketinin tipini belirtir. Bilinmeyen kod içeren bir LCP paketi karşılığında Kur – Reddet (Configure-Reject) paketi iletilir. Yaygın kullanılan LCP kod değerleri şunlardır [2, 18]:

Kod	Paket Tipi
1	Kur – İstek (Configure-Request)
2	Kur – Onayla (Configure-Ack)
3	Kur – Onaylama (Configure-Nak)
4	Kur – Reddet (Configure-Reject)
5	Kapat – İstek (Terminate-Request)
6	Kapat – Onayla (Terminate-Ack)
7	Kod – Reddet (Code-Reject)

8	Protokol – Reddet (Protocol-Reject)
9	Yankı – İstek (Echo-Request)
10	Yankı – Yanıtla (Echo-Reply)
11	Gözardı – İstek (Discard-Request)
12	Kimlik (Identification)
13	Kalan – Zaman (Time-Remaining)
14	Reset – İstek (Reset-Request)
15	Reset – Onayla (Reset-Ack)
0	Üretici Eklentisi (Vendor Extension)

Tanıtıcı :

Bir baytlık olan Tanıtıcı alanından istekler ve cevapların eşleştirilmesi amacıyla yararlanılır. Geçersiz tanıtıcı değeri içeren bir paket sessizce atılır.

Uzunluk :

İki baytlık olan Uzunluk alanındaki değer LCP paketinin uzunluğunu belirtir. Bu değer Kod, Tanıtıcı, Uzunluk ve Veri alanlarının toplam uzunluğunu ifade eder. Uzunluk alanındaki değerin belirttiği uzunluğun dışındaki değerlerin padding alanına ait olduğu varsayılır ve ihmal edilir.

Veri :

Veri alanı sıfır veya daha fazla bayttan oluşur. Bu alandaki verinin tipi Kod bilgisi tarafından tanımlanır.

2.4.1. Bağ Biçimlendirme Paketleri

Tüm bağ biçimlendirme paketlerinin ortak biçimi Şekil 2.7'de gösterildiği gibidir. Bu gruptaki paket kodlarını içeren paketlerin biçimi genel bir LCP paketiyle aynıdır. Sadece genel bir LCP paketi için Veri alanı olarak tanımlanan alan bu paket kodlarını içeren paketlerde Seçenekler alanı olarak tanımlanmıştır. Bu Seçenekler alanı Bölüm 2.5'te ele alınan çeşitli biçimlendirme seçeneklerini içerir.

Kod 1 bayt	Tanıtıcı 1 bayt	Uzunluk 2 Bayt	Seçenekler ...

Şekil 2.7. Bağ biçimlendirme paketlerinin ortak biçimi [17]

Bağ biçimlendirme paketleri için LCP kod değeri, Kur – İstek (Configure-Request) için 1, Kur – Onayla (Configure-Ack) için 2, Kur – Onaylama (Configure-Nak) için 3 ve Kur – Reddet (Configure-Reject) için 4 olacaktır.

Tanıtıcı (Identification) değeri sorgulama paketini gönderen uç tarafından rasgele belirlenir ve ilk paketi gönderen uç seçenekleri değiştirdiği her yeni sorgulama paketini gönderdiğinde veya bir önceki istek karşılığında geçerli bir cevap aldığında bu değeri bir artırır. Bir paket, zaman aşımı nedeniyle yeniden gönderiliyorsa bu değer aynı kalacaktır.

Uzunluk alanındaki değer, Kod, Tanıtıcı, Uzunluk ve tüm seçenekleri içeren Seçenekler alanlarının toplam uzunluğunu ifade eder.

Bağlantıyı başlatmak isteyen bir PPP gerçeklemesi Kur – İstek (Configure-Request) paketi göndermek zorundadır. Eğer kurulacak bağlantı için varsayılan seçenek değerleri dışında başka seçenek değerleri isteniyorsa bu Seçenekler alanında sıralanmalıdır. Normalde bağlantının varsayılan seçenek değerleri sorgulanmaz.

Eğer alınan bir Kur – İstek (Configure-Request) paketinde bulunan bütün seçenekler alıcı uçtaki PPP gerçeklemesi tarafından algılanabiliyor ve belirtilen değerler kabul ediliyorsa, karşı uca bir Kur – Onayla (Configure-Ack) paketi gönderilir. Gönderilen Kur – Onayla (Configure-Ack) paketindeki seçenekler alınan, kabul edilebilir seçenekleri içeren Kur – İstek (Configure-Request) paketin içeriğiyle aynıları olmalıdır. Dolayısıyla bir Kur – Onayla (Configure-Ack) paketi alındığında bu paketin en son gönderilen Kur – İstek (Configure-Request) paketiyle aynı seçenekleri içermesi beklenir ve bu içerikte olmayan paketler sessizce atılır.

Eğer alınan bir Kur – İstek (Configure-Request) paketindeki seçenekler algılanabiliyor fakat değerleri kabul edilemiyorsa, alıcı uç, bu pakete karşılık bir Kur – Onaylama (Configure-Nak) paketi gönderir. Gönderilen Kur – Onaylama (Configure-Nak) paketinde, sadece alınan Kur – İstek (Configure-Request)

paketindeki değerleri kabul edilmeyen seçenekler, alıcı ucun istediği değerleriyle bulunur. Alınan bir Kur – Onaylama (Configure-Nak) paketinde tanıtıcı değerinin son gönderilen Kur – İstek (Configure-Request) paketininkiyle aynı olması beklenir ve bu şartı sağlamayan paket sessizce bırakılır. Geçerli bir Kur – Onaylama (Configure-Nak) paketi alındığında bu pakette belirtilen seçenek değerleriyle yeni bir Kur – İstek (Configure-Request) paketi oluşturulup gönderilir.

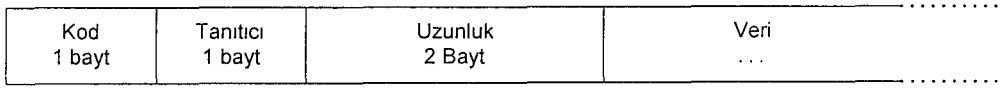
Eğer alınan bir Kur – İstek (Configure-Request) paketindeki seçenekler PPP gerçeklemesi tarafından algılanamıyorsa veya sorgulama için kabul edilebilir değilse, alıcı uç bu pakete karşılık bir Kur – Reddet (Configure-Reject) paketi gönderir. Gönderilecek bu Kur – Reddet (Configure-Reject) paketinin Seçenekler alanı, alınan paketteki kabul edilemez seçeneklerle doldurulur. Alınan bir Kur – Reddet (Configure-Reject) paketindeki tanıtıcı değerinin son gönderilen Kur – Reddet (Configure-Reject) paketininkiyle aynı olması beklenir ve bu şartı sağlamayan paket sessizce atılır. Geçerli bir Kur – Reddet (Configure-Reject) paketi alındığında bu pakette belirtilen seçenekleri içermeyen yeni bir Kur – İstek (Configure-Request) paketi oluşturulup gönderilir.

2.4.2. Bağ Sonlandırma Paketleri

LCP kod değeri olarak, Kapat – İstek (Terminate-Request) için 5 ve Kapat – Onayla (Terminate-Ack) için 6 değerlerini içeren bağ sonlandırma paketlerinin ortak biçimi, Şekil 2.8'de gösterilmiştir.

Bu paketlerin Tanıtıcı alanlarındaki değerler, Veri alanındaki değer değiştiğinde veya bir önceki isteğe geçerli bir yanıt alındığında değiştirilir.

Veri alanında ise hiçbir değer bulunmayabilir; veya yazdırılabilir ASCII karakterlerinden oluşan bir ileti veya gönderici tarafından yorumlanamaz veriler bulunabilir.



Şekil 2.8. Bağ sonlandırma paketlerinin ortak biçimi [17]

Bu iki paket, bir PPP bağının kapatılması için karşılıklı olarak yollanır. Bir uçtaki PPP gerçeklemesi, bağı kapatmak istediğinde bir Kapat – Onayla (Terminate-Ack) paketi gönderir. Gönderilen Kapat – Onayla (Terminate-Ack) paketi, geçerli bir yanıt alınana kadar yeniden gönderilir. Bağ, yeterli sayıda yanıtsız Kapat – İstek (Terminate-Request) gönderildikten sonra Kapat – Onayla (Terminate-Ack) paketi alınmamış olmasına rağmen kapatılabilir. Karşı uçtan Kapat – İstek (Terminate-Request) Paketi alan bir PPP gerçeklemesi yanıt olarak mutlaka aynı Tanıtıcı değerini içeren bir Kapat – Onayla (Terminate-Ack) paketi göndermelidir.

2.4.3. Bağ Bakım Paketleri

Bağ bakım paketlerinin LCP Kod alanı değerleri, Kod – Reddet (Code-Reject) için 7, Protokol – Reddet (Protocol-Reject) için 8, Yankı – İstek (Echo-Request) için 9, Yankı – Yanıtla (Echo-Reply) için 10 (onaltılı 0x0A) ve Gözardı – İstek (Discard-Request) için 11'dir (onaltılı 0x0B).

Bu paketlerden Kod- Reddet (Code-Reject), gönderilen paketteki Kod değerinin alıcı uçtaki PPP gerçeklemesi tarafından bilinmediğini ya da desteklenmediğini ifade etmek üzere gönderilir. Karşı uçtaki PPP gerçeklemesinin gönderdiği paketteki Kod değerinin alıcı tarafından bilinmemesi, bağın uçlarındaki PPP gerçeklemelerinin farklı sürümlere ait olduğunu gösterir. Kod – Reddet (Code-Reject) paketi herhangi bir protokol düzeyindeyken gönderilebilir. Kod – Reddet (Code-Reject) paket biçimi Şekil 2.9'da görüldüğü gibidir.

Kod 1 bayt	Tanıtıcı 1 bayt	Uzunluk 2 Bayt	Reddedilen Paket ...

Şekil 2.9. Kod – Reddet (Code-Reject) paket biçimi [17]

Tanıtıcı değeri, her yeni Kod – Reddet (Code-Reject) paketi için değiştirilmelidir.

Reddedilen Paket alanında reddedilen LCP paketinin bir kopyası bulunur. Bu kopya, Bilgi alanı ile başlar ve Veri Bağı Katmanı üst bilgilerinin hiç birini ve FCS verisini içermez. Reddedilen paket karşı ucun MRU'su (Maximum Receive Unit, Maksimum Alım Birimi) ile uyumlu olması için gerektiğinde kesilebilir.

Kod – Reddet (Code-Reject) paketini alan bir PPP gerçeklemesi, hangi kod için bu paketi almışsa o kod değerini göndermeye son vermelidir.

Protokol – Reddet (Protocol-Reject) paketi ise, alınan paketteki protokol değerinin bilinmediği durumlarda gönderilir. Alınan paketteki protokol değerinin bilinmemesi, bu protokolün alıcı uçtaki PPP gerçeklemesi tarafından gerçeklenmediğini ifade eder. Bu durum genellikle karşı ucun yeni bir protokol biçimlendirme girişiminde bulunması halinde ortaya çıkar. Protokol – Reddet (Protocol-Reject) paketinin gönderilebilmesi için LCP'nin açık durumda olması gerekir. LCP henüz açık duruma gelmeden gönderilen Protokol – Reddet (Protocol-Reject) paketleri sessizce atılır. Protokol – Reddet (Protocol-Reject) paketinin biçimi Şekil 2.10'da gösterilmiştir.

Kod 1 bayt	Tanıtıcı 1 bayt	Uzunluk 2 Bayt	Reddedilen Protokol 2 Bayt	Reddedilen Paket ...

Şekil 2.10. Protokol – Reddet (Protocol-Reject) paketinin biçimi [17]

Tanıtıcı değeri her yeni Protokol – Reddet (Protocol-Reject) paketi için değiştirilmelidir.

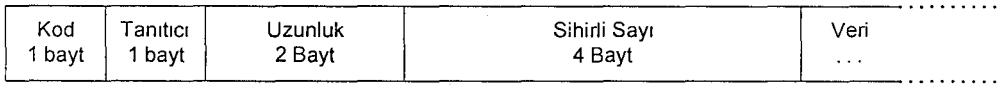
Reddedilen Protokol değeri reddedilen PPP paketindeki protokol değerinin aynıdır.

Reddedilen Bilgi alanı reddedilen paketin bir kopyasını içerir. Bu kopya, Bilgi alanıyla başlar ve Veri Bağı Katmanı üst bilgilerinin hiç birini ve FCS verisini içermez. Reddedilen paket karşı ucun MRU'su (Maximum Receive Unit, Maksimum Alım Birimi) ile uyumlu olması için gerektiğinde kesilebilir.

Protokol – Reddet (Protocol-Reject) paketinin alan bir PPP gerçeklemesi, reddedilen protokol içeren paketleri göndermeye son vermelidir.

Yankı – İstek (Echo-Request), Yankı – Yanıtla (Echo-Reply) ve Gözardı – İstek (Discard-Request) paketleri, bağın güvenirliğini izlemek veya bir PPP

gerçeklemesinde hata ayıklamak ve gerçeklemenin performansını test etmek için kullanılır. Yankı – İstek (Echo-Request) paketini alan bir PPP gerçeklemesi yanıt olarak Yankı – Yanıtla (Echo-Reply) paketi göndermelidir. Gözardı – İstek (Discard-Request) paketini alındığında ise bu paket sessizce atılmalıdır. Bu üç paket sadece LCP açık durumdayken gönderilmelidir. Bu paketlerin ortak biçimi Şekil 2.11'de gösterilmiştir.



Şekil 2.11. Yankı – İstek (Echo-Request), Yankı – Yanıtla (Echo-Reply) ve Gözardı – İstek (Discard-Request) paketlerinin ortak biçimi [17]

Tanıtıcı alanındaki değer Gözardı – İstek (Discard-Request) paketi için her paket gönderilişinde değiştirilmelidir. Yankı – İstek (Echo-Request), Yankı – Yanıtla (Echo-Reply) paketlerinde ise bu değer ya veri alanının içeriği değiştirildiğinde ya da bir önceki isteğe geçerli bir yanıt alındığında değiştirilmelidir. Fakat tekrar gönderilen paketlerde bu değer aynı kalmalıdır.

Sihirli Sayı alanındaki dört baytlık değer bağın geri döngü (loop-back) durumunda olup olmadığını anlamaya yardımcı olur. Gönderilen sihirli sayılı paket, gönderici tarafından alınıyorsa, bağlantıda geri döngü vardır. Bu değer, LCP'nin Sihirli Sayı Biçimlendirme Seçeneği ile Kurulum evresinde sorgulanarak belirlenir. Eğer sorgulanmamışsa varsayılan değeri sıfırdır.

Veri alanında ise hiçbir değer bulunmayabilir; veya yazdırılabilir ASCII karakterlerinden oluşan bir ileti veya gönderici tarafından yorumlanamaz veriler bulunabilir.

2.4.4. Diğer Paketler

Kimlik (Identification) paketinin LCP Kod değeri 12'dir (onaltılı 0x0C). Bu ileti için bir sorgulama tanımlanmamıştır. Bu kod ile gönderilen bir paketle PPP gerçeklemesi kendisini karşı uçtaki gerçeklemeye doğrulanmamış basit bir katar ile tanıtır. Bu katarda, versiyon numaraları, üretici bilgisi, gibi bilgiler bulunabilir. Bu ileti LCP açık duruma gelmeden önce de gönderilebilir.

Sorgulama tanımlanmamış bir diğer ileti çeşidi olan Kalan – Zaman (Time-Remaining) paketinin LCP kod değeri 13'tür (onaltılı 0x0D). Kalan – Zaman (Time-Remaining) paketi ile bir PPP gerçeklemesi, karşı uçtaki gerçeklemeye bağın belli bir süre sonra bir çeşit yönetimsel denetim nedeniyle sonlandırılacağı bildirir.

LCP Kod değerleri 14 ve 15 (onaltılı 0x0E ve 0x0F) olan Reset – İstek (Reset-Request) ve Reset – Onayla (Reset-Ack) iletileri veri sıkıştırma protokolleri işlemdeyken kullanılır.

Üretici Eklentisi (Vendor-Extension) paketi ise bazı özel amaçlar için iki uç arasında paket değişiminin gerektiği durumlarda kullanılır. LCP Kod değeri 0 olan bu paketler herhangi bir anda yollanabilir [2, 17].

2.5. LCP Biçimlendirme Seçenekleri

PPP'deki diğer protokollere oranla LCP'de çok fazla biçimlendirme seçeneği vardır. Bu biçimlendirme seçenekleri, bir noktadan noktaya bağın geçerli özelliklerine göre sorgulama yapılmasına olanak tanır. LCP'de PPP'nin diğer protokollerine oranla daha fazla seçenek bulunmasının nedenlerinden biri, seçeneklerin bazılarının çerçeveleme derecesinde değişikleri içermesi dolayısıyla ağ kurulumunda en önce belirlenmesinin gerekliliğidir. Diğer nedense, bazı seçeneklerin doğrulama protokollerinin belirlenmesi için sorgulanması dolayısıyla doğrulama evresinden önce belirlenmesinin gerekmesidir. Bir biçimlendirme seçeneği Kur – İstek (Configure-Request) paketi içinde yollanmamışsa o seçeneğin varsayılan değeri kabul edilmiş sayılmaktadır.

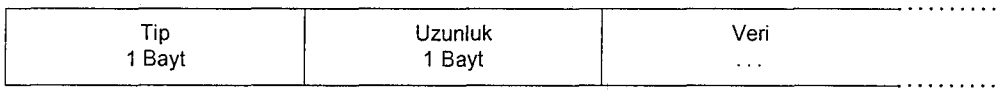
Tüm biçimlendirme seçenekleri aksi belirtilmediği sürece yarı çift yönlü olarak sorgulanırlar.

PPP gerçeklemelerinde seçeneklerin sorgulanması konusunda bir kaç önemli ayrıntı dikkate alınmalıdır:

1. Seçenekler, sorgulayan ucun ya ek bir özelliğini veya gerek duyduğu bir noktayı belirtirler. Hiç bir seçenek gerçeklenmemiş bir PPP gerçeklemesi mutlaka tüm seçenekleri gerçeklemiş bir PPP gerçeklemesi ile bağlantı kurmalıdır.

2. Tüm seçenekler için tanımlanan varsayılan değerler kabul edildiğinde bağ doğru olarak çalışır. Fakat kapasitesinden daha düşük bir performans sergiler.
3. Seçeneklerin varsayılan değerlerinin Configure-Request paketleri ile gönderilmesi gereksizdir.

Biçimlendirme seçeneklerinin ortak biçimi Şekil 2.12'de gösterilmiştir.



Şekil 2.12. Biçimlendirme seçeneklerinin ortak biçimi

Tip alanı bir baytlık uzunlukta olup aşağıdaki seçenek tiplerinden birinin değerini içerir.

Diğer bir baytlık alan olan Uzunluk alanı ise seçeneğin uzunluğunu Tip, Uzunluk ve Veri alanlarının toplamı olacak şekilde gösterir.

Veri alanı ise ya sıfır veya daha fazla bayt uzunlukta olabilir; ve o seçeneğe özel bilgiyi içerir.

Eğer alınan paketteki uzunluk alanı yanlışsa veya algılanamıyorsa, istenen biçimlendirme seçeneğinin uygun uzunluk ve veri ile doğru olarak belirtildiği bir Kur – Onaylama (Cofigure-Nak) paketi gönderilmelidir. Eğer veri alanı belirtilen paket bilgi uzunluğunu aşıyorsa bu paket sessizce atılır.

PPP Kurulum, Doğrulama ve Ağ evrelerindeki sorgulamaların çeşitli örnekleri, EK-1'de ayrıntılı olarak verilmektedir.

Bu çalışmada, kısıtlı bir PPP gerçeklemesi düşünüldüğünden, bazı seçeneklerin göz ardı edilmesi gerekmiştir. Diğerleri arasında en çok sorgulanmaları nedeniyle evrensel olarak nitelenebilecek seçenekler 01 (MRU), 02 (ACCM), 03 (Doğrulama Protokolü), 05 (Sihirli Sayı), 07 (PFC) ve 08'dir (ACFC). Bu seçenekler içinde 01 (MRU) dışındakiler, Türkiye'deki ISP'ler tarafından da standart olarak sorgulanmaktadır. Bunlar dışında genellikle sorgulanan iki seçenek ise 17 (MRRU) ve 19'dur (Son Nokta Ayıracı). Bu bölümde tüm LCP biçimlendirme seçeneklerin bir listesi verildikten sonra sadece genel olarak sorgulanan seçeneklerin açıklamaları verilecektir [18]. Tüm

biçimlendirme seçeneklerinin ayrıntılı açıklaması için Carlson'ın PPP Design, Implementation and Debugging [2] adlı kitabına başvurulabilir.

Tip	Biçimlendirme Seçeneği
1	Maximum-Receive-Unit
2	Async-Control-Character-Map
3	Authentication-Protocol
4	Quality-Protocol
5	Magic-Number
6	RESERVED
7	Protocol-Field-Compression
8	Address-and-Control-Field-Compression
9	FCS-Alternatives
10	Self-Describing-Pad
11	Numbered-Mode
12	Multi-Link-Procedure
13	Callback
14	Connect-Time
15	Compound-Frames
16	Nominal-Data-Encapsulation
17	Multilink-MRRU
18	Multilink-Short-Sequence-Number-Header-Format
19	Multilink-Endpoint-Discriminator
20	Proprietary [KEN]
21	DCE-Identifier [SCHNEIDER]

Bu çalışmada PPP sunucusu olarak belirlenen ISP'lerden biri olan Superonline'dan alınan ilk LCP Kur – İstek (Configure-Request) paketi ve bu paketteki verilerin neler oldukları Şekil 2.13'te görülmektedir.

Alan Tipi	Paket İçeriği	Anlamı
Çerçeveleme	7E	Başlama Bayrağı
	FF 03	Adres ve Denetim
Protokol	C0 21	LCP Protokolü
Kod	01	LCP Configure_request
Tanıtıcı	1C	Bu paketin tanıtıcı değeri
Uzunluk	00 2A	Paket boyu (protokol ve sağlama arası)
Seçenekler	02	Seçenek 2, ACCM
	06	Seçenek 2'nin uzunluğu
	00 0A 00 00	^S ve ^Q gizlenmeli
	03	Seçenek 3, Doğrulama Protokolü
	04	Seçenek 3'ün uzunluğu
	C0 23	PAP
	05	Seçenek 5, Sihirli Sayı
	06	Seçenek 6'nın uzunluğu
	4E B0 6E 3F	Sihirli Sayı'nın değeri
	07	Seçenek 7, PFC
	02	Seçenek 7'nin uzunluğu
	08	Seçenek 8, ACFC
	02	Seçenek 8'in uzunluğu
	11	Seçenek 17, Multilink-MRRU
	04	Seçenek 17'nin uzunluğu
	05 F4	Seçenek 17'nin değeri
	13	Seçenek 19, Multilink-ED
	0E	Seçenek 19'un uzunluğu
	01 75 73 65 72 33 31 34 34 30 31 36	Seçenek 19'un değeri
Sağlama	9A 94	İki bayrak arası için sağlama değeri
Çerçeveleme	7E	Bitiş Bayrağı

Şekil 2.13. Örnek bir Kur – İstek (Configure-Request) paketi

2.5.1. Maksimum Alım Birimi

Bu seçenek, karşı uçtaki PPP sunucusuna, en çok ne kadar büyüklükteki bir paketin alınabileceğini belirtir.. Uzunluk alanı değeri her zaman 4'tür. İki baytlık Veri alanında ise, sorgulanmak istenen Maksimum Alım Birimi (Maximum Receive Unit, MRU) değeri bulunur. HDLC çerçevesindeki adres, denetim ve sağlama alanlarının uzunluğu belirtilen değerin içinde değildir.

Bu birim için varsayılan değer 1500 bayttır. Daha küçük bir paket değeri ilk sorgulamada istenmiş olsa bile, bağ uyumunun kaybolması olasılığına karşı her PPP gerçeklemesi, kesinlikle en az 1500 bayt uzunluğundaki paketleri kabul edebilir durumda olmalıdır [2, 17].

2.5.2. Asenkron Denetim Karakter Haritası

Asenkron denetim karakterleri, ASCII karakter tablosundaki yazdırılamayan ilk 32 karakterdir. Bu biçimlendirme seçeneği ASCII denetim karakterlerinin asenkron bağlar tarafından algılanamayacak şekilde iletilmeleri için bir yöntem tanımlar. Çünkü bu karakterlerden bazıları asenkron noktadan noktaya bağlantılar üzerindeki bazı modemler tarafından karşı uca gönderilecek bilgi yerine, belli komutlar olarak algılanabilir.

Bu seçenekte Uzunluk alanının değeri her zaman 6'dır. Veri alanında ise 4 bayt uzunluğundaki ACCM (Asynchronous Control Character Map, Asenkron Denetim Karakter Haritası) değeri bulunur. Bu değerdeki her bir bit denetim karaketerlerinden birine karşılık gelmektedir.

ACCM'nin varsayılan değeri olan 0xFFFFFFFF tüm denetim karakterlerinin (32'den küçük ASCII numaralı karakterler) kaçış karakterleriyle birlikte gönderilmesi gerektiği anlamına gelir. Bir denetim karakterinin gizlenmesi işleminde paketteki bu karakter değeri, önce denetim kaçış karakteri olan 0x7D ve ardından bu karakterin 0x20 değeriyle EX-OR işlemimin sonucu ile değiştirilir. Tüm ASCII denetim karakterleri kaçış işlemine tabi tutulmuş bir PPP paketi Şekil 2.4'te görülmektedir.

Fakat tüm denetim karakterlerinin kaçış dizisiyle ifade edilmesi paket boyutunu çok fazla büyütecektir. Bununla beraber hiçbir denetim karakterinin kaçış dizisine tabi tutulmaması da çok rastlanan bir durum değildir. Genellikle kaçış işlemine tabi tutularak iletilmesi istenen denetim karakterleri, ASCII karakter tablosunun 17 ve 19. karakterlerdir (DC1, CTRL-X ve DC3, CTRL-S karakterleri). Bu karakterler standart XON/XOFF iletim denetimi ile çalışan bağlarda denetim amaçlı kullanılırlar. Bunun için ACCM değeri, 0x000A0000 olarak verilir.

LCP sorgulamaları yapılırken ACCM'nin en uygun zamanda ayarlanması gerekir. Tüm çeşitli gerçeklemelerle uyum sağlanabilmesi için alım ACCM'sinin, karşı uçtan Kur – Onayla (Configure-Ack) veya Kur – Onaylama (Configure-Nak) paketi alır almaz hemen belirtilen değere ayarlanması gerekir. Gönderim ACCM'si ise, LCP açık durumuna geçtikten ve tüm bekleyen çıktılar gönderildikten sonra belirtilen değere ayarlanmalıdır [19].

2.5.3. Kimlik Doğrulama Protokolü

Kimlik Doğrulama (authentication) protokolü seçeneğini içeren bir Kur – İstek (Configure-Request) paketinin alımı karşı uçtaki PPP gerçeklemesinin bağlantının kurulması için belirtilen protokol ile doğrulamanın yapılmasını istediği anlamını ifade eder. Bir paket bir anda birden fazla kimlik doğrulama protokolü seçeneğini içeremez. Kimlik doğrulama protokolü için varsayılan, doğrulamanın yapılmamasıdır.

Bu seçeneği içeren bir Kur – İstek (Configure-Request) paketi alan bir PPP gerçeklemesi, eğer doğrulama için istenen protokol gerçeklenmiş bir protokolse yanıt olarak Kur – Onayla (Configure-Ack); eğer gerçeklenmemiş bir protokolse gerçeklenmiş olan bir protokol istenene kadar Kur – Onaylama (Configure-Nak) paketi göndermelidir. Fakat Kur – İstek (Configure-Request) paketine karşılık Kur – Onaylama (Configure-Nak) paketini alan bir PPP gerçeklemesi bağı sonlandırabilir. Kabul edilebilir bir doğrulama protokolü için Kur – Onayla (Configure-Ack) paketi gönderen bir PPP gerçeklemesinin, bu belirlenen protokolü kullanarak kendini karşı uca doğrulaması gerekir. Kabul edilmeyen bir doğrulama protokolünü içeren bir Kur – İstek (Configure-Request) paketine yanıt olarak protokolün değiştirilmesi için Kur – Reddet (Configure-Reject) paketi gönderilmez.

Kimlik doğrulama protokolü seçeneğindeki Veri alanı 2 veya 3 baytlık olabilir. İlk 2 bayt istenen doğrulama protokolünü, eğer varsa 3. bayt istenen bu protokole ait ek bilgiyi ifade eder [2, 17].

Kullanımda olan bazı doğrulama protokolleri arasında, 0xC023 Şifre Doğrulama Protokolü (Password Authentication Protocol, PAP) [20], 0xC223 Kimlik Sorma Anlaşma Doğrulama Protokolü (Challenge Handshake

Authentication Protocol, CHAP) [21], 0xC227 Genişletilebilir Doğrulama Protokolü (Extensible Authentication Protocol, EAP) [22] sayılabilir.

2.5.4. Sihirli Sayı

Bu seçeneğin uzunluğu 6 bayt olup Veri alanında 4 bayt uzunluğunda bir "sihirli sayı" içerir.

Sihirli Sayı, PPP bağının bir ucu için ayırt edici nitelikte rassal bir sayıdır. Bu rassal sayının tek-bir (unique) sayı olması önemlidir. Bunun için sihirli sayının seçiminde donanım seri numaraları veya sistem saatleri gibi tek-bir (unique) değere sahip olma olasılığı yüksek olan kaynaklar kullanılır.

Sihirli sayı geri döngü veya geri yansıma gibi hatalı durumların ayırt edilebilmesi amacıyla kullanılır. Aynı zamanda Kalite Protokolü gibi bazı diğer seçenekler tarafından da kullanılır.

Sihirli sayı sorgulanmamışsa, kullanımı gerektiğinde varsayılan değer olan sıfır kullanılır.

Bu seçeneğin sorgulanması diğerlerinden farklıdır. Her yeni alınan Kur – İstek (Configure-Request) paketindeki sihirli sayının, son gönderilen Kur – İstek (Configure-Request) paketindeki sihirli sayı ile karşılaştırılması gerekir. Bu iki sayı farklıysa bağda geri döngü (loop-back) yoktur; ve alınan Kur – İstek (Configure-Request) paketindeki sihirli sayı için Kur – Onayla (Configure-Ack) gönderilir. Eğer iki sayı eşitse, bağda geri döngü olması muhtemeldir. Bunun belirlenmesi için farklı bir sihirli sayı içeren bir Kur – Onaylama (Configure-Nak) paketi gönderilir. Bu durumda, Kur – Onaylama (Configure-Nak) paketi alınana veya yeniden başlama zamanlayıcısı bitene kadar yeni bir Kur – İstek (Configure-Request) paketi gönderilmemelidir. Kur – Onaylama (Configure-Nak) paketi alındığında ise, paketteki sihirli sayı son gönderilen Kur – Onaylama (Configure-Nak) paketindeki sihirli sayı ile aynıysa yeni bir sihirli sayı belirlenip Kur – İstek (Configure-Request) paketi gönderilir [2, 17].

2.5.5. Protokol Alanı Sıkıştırması

PPP Protokol alanındaki iki baytlık değerler, bir baytlık değerlere sıkıştırıldığında ayırt edilebilecek şekilde belirlenmiştir. Bu biçimlendirme seçeneğini gönderen bir PPP gerçeklemesi, tek bayta sıkıştırılmış Protokol alanı değerlerini alabileceğini karşı uçtaki gerçeklemeye bildirmiş olur. Bu seçenek için varsayılan uygulama Protokol alanı değerlerinin iki bayt olarak gönderilmesidir. Eğer bu seçenek sorgulanmamışsa sıkıştırılmış Protokol alanı değerleri gönderilmemelidir. LCP paketi gönderilirken Protokol alanı değeri kesinlikle sıkıştırılmaz.

İki baytlık olan bu seçenek için, Veri alanı tanımlanmamıştır [17].

2.5.6. Adres Kontrol Alanı Sıkıştırması

Veri Bağı Katmanı'nın Adres ve Denetim alanlarındaki değerler PPP için sabit değerler oldukları için (0xFF ve 0x03) kolaylıkla sıkıştırılabilirler. Bu seçeneği göndermekle, PPP gerçeklemesi, Adres ve Denetim alanlarındaki değerleri sıkıştırılmış paketleri alabileceğini karşı uca bildirmiş olur. Bu seçenek sorgulanmadan önce Adres ve Denetim alanları sıkıştırılmış bir paket alındığında bu paket sessizce atılır.

Bu seçenek de iki baytlık olup Veri alanını içermez [17].

2.6. Sayıcılar ve Zamanlayıcılar

Yeniden Başlama Zamanlayıcısı (Restart Timer)

Bir PPP gerçeklemesinde kullanılan tek zamanlayıcı budur. Yeniden Başlama Zamanlayıcısı (Restart Timer), Kur – İstek (Configure-Request) ve Kapat – İstek (Terminate-Request) paketlerinin zamanlanması için kullanılır. Bu paketlerden birisi gönderildiğinde belirlenen süre içinde cevap gelmesi beklenir. Belirlenen süre, yeniden başlama zamanlayıcısı ile ölçülür. Yeniden başlama zamanlayıcısı dolmasına rağmen bir cevap gelmemişse, zaman aşımı (timeout) oluşur ve gönderilmiş olan paket tekrar gönderilir. Yeniden başlama zamanlayıcısı için varsayılan zaman aşımı değeri üç saniyedir.

Max – Kapat (Max-Terminate)

Max – Kapat (Max-Terminate), karşı uçtan Kapat – Onayla (Terminate-Ack) alınmadan gönderilen Kapat – İstek (Terminate-Request) paketlerinin sayısını tutan bir sayıcıdır. Max – Kapat (Max-Terminate) sayıcısı belli bir değeri aştığında karşı ucun tepki veremediği sonucuna varılır ve bağ kapatılır. Bu sayıcı için varsayılan değer ikidir (2) [17].

Max – Kur (Max-Configure)

Max – Kur (Max-Configure) sayıcısı ise Kur – Onayla (Configure-Ack), Kur – Onaylama (Configure-Nak) veya Kur – Reddet (Configure-Reject) alınmadan gönderilmiş olan Kur – İstek (Configure-Request) paketlerinin sayısını tutar. Belli bir değeri aştığında bağ paketleri gönderen uç tarafından kapatılır. Bu sayıcının varsayılan değeri ondur (10) [17].

Max – Başarısızlık (Max-Failure)

Bu sayıcı ise geçerli bir Kur – Onayla (Configure-Ack) alınmadan gönderilmiş bulunan Kur – Onaylama (Configure-Nak) paketlerinin sayısını tutar. Varsayılan değeri beştir (5) [17].

2.7. Kimlik Doğrulama

Doğrulama protokollerinde PPP bağının Kurulum evresinde olduğu gibi sorgulamalar söz konusu değildir. Bu protokoller aracılığıyla, istek paketinde doğrulama için gereken bilgiler gönderilir ve karşılığında doğrulama veya başarısızlık iletileri içeren paketler alınır.

PPP bağının kurulması için kimlik doğrulaması zorunludur. Çünkü ağ üzerinden bilgi alışverişi yapmak isteyen host ve yönlendiriciler genellikle, PPP ağ sunucusuna anahtarlamalı devreler veya çevirmeli hatlar üzerinden bağlanırlar. Dolayısıyla, PPP sunucusu bağlanmak isteyen host veya yönlendiricilerin kendilerini tanıtmalarını bekleyecektir. Fakat bazen doğrulama evresi ihmal edilerek de PPP bağı kurulabilir.

Doğrulama için kullanılacak doğrulama protokolü, Kurulum evresinde LCP sorgulamaları ile belirlenir (3 numaralı LCP biçimlendirme seçeneği).

Bir PPP ağ sunucusu bağ kurmak isteyen bir PPP gerçeklemesinin doğrulama yoluyla kendisini tanıtmasını bağ için şart koşmuş ise LCP sorgulamalarında en güçlü doğrulama seçeneğini ilk olarak sorgulamalıdır. Eğer sorgulanan doğrulama protokolü bağı kurmak isteyen PPP gerçeklemesinde yoksa, farklı bir doğrulama protokolünün kullanılması için bağı kurmak isteyen PPP gerçeklemesi istediği doğrulama protokolünü içeren bir Kur – Onaylama (Configure-Nak) paketi göndermelidir. Bu aşamada Kur – Reddet (Configure-Reject) paketini gönderilmesi doğrulamanın iptal edilmesinin istendiği anlamına geleceği için doğrulamayı isteyen PPP sunucusu tarafında bağ sonlandırılır. Bölüm 2.5.3'ün sonunda sıralanmış olan çeşitli doğrulama protokollerden en az işlem yükü gerektiren ve Türkiye'deki ISP'lerin bir çoğu tarafından kullanılan Şifre Doğrulama Protokolü'dür (Password Authentication Protocol, PAP). Fakat bu protokol, kullanıcı adı ve şifre bilgileri hat üzerinden doğrudan gönderildiği için güvenli değildir. Bu çalışmada doğrulama evresinde PAP kullanıldığı için bu protokolün çalışma prensibi ve paket yapısına kısaca değinilecektir.

2.7.1. Şifre Doğrulama Protokolü

Kurulum evresinin tamamlanmasından sonra bağlantı isteğinde bulunan uç, PPP sunucuya kullanıcı adı ve şifre ikilisini sunucu, bir kabul veya ret iletisi gönderinceye kadar tekrarlı olarak gönderir. Doğrulama işlemi, gönderilen kullanıcı adı ve şifre bilgilerinin bir veritabanında aranıp karşılaştırma yapılması gibi uzun süre gerektiren bir işlemin sonucunda yapılmaktadır. Bundan dolayı bu bilgileri gönderen uç, tekrar gönderim yapmadan önce yeterli bir süre beklemiş olmalıdır.

PAP paketi, bir PPP çerçevesinin bilgi alanını oluşturacak şekilde çerçevelenir ve bu kapsülün protokol alanındaki değer için 0xC023 yazılır. PAP paketinin yapısı Şekil 2.14'te verilmiştir.

Kod 1 bayt	Tanıtıcı 1 bayt	Uzunluk 2 Bayt	Veri ...

Şekil 2.14. PAP paketinin yapısı [20]

Doğrulama evresi sırasında yapılan paket değişimleri EK-1'de ayrıntılı olarak gösterilmiştir.

Bir bayt uzunluğundaki Kod, PAP paketinin tipini belirler. Bir PAP paketi için üç farklı kod değeri vardır:

Kod	Paket Tipi
1	Kimlik Doğrulama – İstek (Authenticate-Request)
2	Kimlik Doğrulama – Onayla (Authenticate-Ack)
3	Kimlik Doğrulama – Onaylama (Authenticate-Nak)

Tanıtıcı değeri de bir baytlık olup istek ve cevapların eşleştirilmesine yarar.

Uzunluk alanındaki değer, Kod, Tanıtıcı, Uzunluk ve Veri alanlarının toplam uzunluğunu ifade eder.

Veri alanının uzunluğu değişebilir ve buradaki verinin tipini Kod alanındaki değer belirler.

Kimlik Doğrulama – İstek (Authenticate-Request)

PAP protokolü bu paketin gönderilmesiyle başlatılır. Bu paketin gönderimi geçerli bir Kimlik Doğrulama – Onayla (Authenticate-Ack) veya Kimlik Doğrulama – Onaylama (Authenticate-Nak) paketi alınana kadar tekrarlanır. Kimlik Doğrulama – İstek (Authenticate-Request) paketi, normal bir PAP paketinin veri alanı sırasıyla Peer-ID-Length, Peer-ID, Passwd-Length, Password alanları ile doldurularak hazırlanır. Peer-ID-Length alanı bir bayt uzunluğunda olup Peer-ID alanındaki bilginin uzunluğunu belirtir. Peer-ID alanı Kullanıcı Adı bilgisini içerir. Passwd-Length alanı bir bayt uzunluğunda olup Password alanındaki bilginin uzunluğunu belirtir. Password alanı Şifre bilgisini içerir.

Doğrulama – Onayla (Authenticate-Ack) ve Doğrulama – Onaylama (Authenticate-Nak)

Bu PAP paketlerinin ilki gönderilen Kullanıcı Adı ve Şifre bilgileri doğruysa, ikincisi ise yanlışsa gönderilir. Bu paketler için PAP paketinin veri

alanında Msg-Length ve Message alanları vardır. Message alanı sıfır veya daha fazla uzunlukta olabilir. Bu alan, insan tarafından okunabilir bir ileti içerir.

2.8. Ağ Denetimi

Ağ denetimi, Ağ evresinin başlangıcında PPP gerçeklemesinin üstünde (Ağ Katmanında) çalışacak olan protokolün seçimi ve bu protokol için seçeneklerin belirlenmesi amacıyla yapılır. Her ağ protokolü için genellikle iki PPP protokol değeri vardır. Bunlardan ilki 0x8000 ve 0xBFFF arasında değişen değerler olup o ağ protokolüne ait Ağ Denetim Protokolü'nün (NCP) değerleridir. İkincisi ise 0x0000 ve 0x03F arasındaki Ağ Protokolü değerleridir. En çok kullanılan Ağ Protokolü, 0x0021 PPP protokol değeriyle İnternet Protokolü (Internet Protocol, IP); ve bu protokolün Ağ Denetim Protokolü, 0x8021 PPP değeriyle İnternet Protokolü Denetim Protokolü'dür (Internet Protocol Control Protocol, IPCP).

Her NCP'nin paket biçimi, Kod değerleri ve sorgulama işlemi LCP'nin aynıdır. Bununla birlikte, sadece LCP'deki ilk yedi kod çeşidi NCP sorgulamalarında kullanılır.

Bu çalışmada kullanılan NCP'nin IPCP olması nedeniyle bu bölümde ayrıntılı olarak bu protokolün özellikleri ele alınacaktır.

2.8.1. İnternet Protokolü Denetim Protokolü

Bu protokol bir PPP bağında Ağ Protokolü olarak kullanılacak İnternet Protokolü'nün biçimlendirilmesi ve etkinleştirilmesi için kullanılır. Fakat IPCP paketleri PPP gerçeklemesi Ağ evresinde geçmeden önce kesinlikle yollanmaz. Daha önceki evrelerde alınan IPCP paketleri sessizce atılır.

Her PPP paketinin içerisinde sadece bir tane IPCP paketi kapsüllenebilir. IPCP protokolünde diğer NCP'lerde olduğu gibi sadece ilk yedi kod çeşidi kullanıldığından diğer kod değerlerini içeren bir paket alındığında bu pakete yanıt olarak Kod – Reddet (Code-Reject) paketi gönderilir.

IPCP protokolü, LCP'nin aynı biçimlendirme seçeneği paket yapısını kullanır. Sorgulamaların yapıldığı üç çeşit seçenek vardır [18]:

1	IP-Adresleri
2	IP-Sıkıştırma-Protokolü
3	IP-Adresi

IP-Adresleri

Bu seçenek ile yapılan sorgulamalar sonucunda bir noktada birleşmek mümkün olamadığı için bu seçeneğin kullanımı iptal edilmiştir. Bu seçenek yerine üçüncü seçenek olan IP-Adresi sorgulanmaktadır.

IP-Sıkıştırma-Protokolü

Bu biçimlendirme seçeneği ile belli bir sıkıştırma protokolünün kullanımı sorgulanır. Bu seçeneğin yapısı LCP biçimlendirme seçeneklerinin yapısını aynıdır. Yalnızca Veri alanının, iki baytlık IP-Sıkıştırma-Protokolü ve Veri olmak üzere iki bölümü vardır. IP-Sıkıştırma-Protokolü bölümünde kullanımı istenen sıkıştırma protokolünün değeri, Veri bölümünde ise istenen protokol tarafından belirlenen veriler vardır. Belli bir sıkıştırma protokolü için IPCP sorgulaması yapılmadığında IP paketleri hiçbir sıkıştırma yapılmaksızın gönderilir.

IP-Adresi

Bu seçenek ile hem bağlantı kurmak isteyen PPP gerçeklemesine diğer uçtaki PPP sunucu tarafından bir IP numarası atanır; hem de PPP sunucu kendi IP numarasını PPP gerçeklemesine bildirir. Bu seçeneğin yapısı Veri alanında IP adresi olmak üzere diğer LCP biçimlendirme seçeneklerinin aynıdır. PPP gerçeklemesine IP adresinin atanması dört yollu el sıkışma ile yapılır.

PPP gerçeklemesine IP adresi atanması işleminde ya bu gerçeklemenin istemiş olduğu IP adresi bir IPCP Kur – İstek (Configure-Request) paketi ile PPP sunucuya bildirilir; veya PPP sunucunun bir IP adresi bildirmesi için istekte bulunulur. İstekte bulunma, IP adresi atanacak gerçeklemenin tüm IP değerlerinin sıfır olduğu bir IPCP Kur – İstek (Configure-Request) paketi göndermesi ile yapılır. PPP sunucu, bu pakete karşılık, IP değerlerini atayacağı IP adresi ile değiştirdiği bir Kur – Onaylama (Configure-Nak) paketi gönderir. PPP gerçeklemesi ise bu gönderilen IP adresi ile yeni bir Kur – İstek (Configure-

Request) paketi gönderir ve karşılığında alınan Kur – Onayla (Configure-Ack) paketi ile IP adresi atanmış olur.

3. İLETİM DENETİM PROTOKOLÜ / İNTERNET PROTOKOLÜ

Günümüz İnternet haberleşmesinde OSI referans modelinin Ağ katmanının değişmez protokolü haline gelmiş olan İnternet Protokolü, tüm ağ yapısı üzerinde datagramların farklı fiziksel katmanlardan geçip doğru yolu bularak yönlendirilmesi amacını grçekleştirecek şekilde tasarlanmıştır. İletim Denetim Protokolü ise bir üst katman olan İletim katmanında çalışan ve verilerin bir kullanıcıdan diğerine bağlantılı (connection-oriented) ve güvenilir (reliable) olarak iletilmesini sağlayan bir protokoldür. Genellikle birlikte anılan bu iki protokol (TCP/IP), Internet haberleşmesinin temelini oluşturmaktadır. TCP protokolünün bir alt katmanında, mutlaka IP protokolü bulunması gerekmekle birlikte, IP protokolü gelen veriyi içindeki protokol bilgisine göre, ya aynı katmandaki ICMP protokolüne veya bir üst katman protokollerinden TCP'ye veya TCP'nin bağlantısız ve güvenilir olmayan bir türü olarak kabul edilebilen UDP'ye de verebilmektedir.

3.1. İnternet Protokolü

İnternet Protokolü, belli uzunluktaki IP adresleriyle adreslenmiş kullanıcılar arasında datagramların gönderilmesini ve alınması işini gerçekleştirir. Bu protokolün bir diğer özelliği, uzun bir veri bloğunu, iletimin yapılacağı fiziksel ortamın gereklerine göre parçalayabilip daha sonra yeniden birleştirebilmesidir. Buna göre IP protokolünün, adresleme (addressing) ve parçalama (fragmentation) şeklinde ifade edilebilecek iki temel işlevi olduğu söylenebilir.

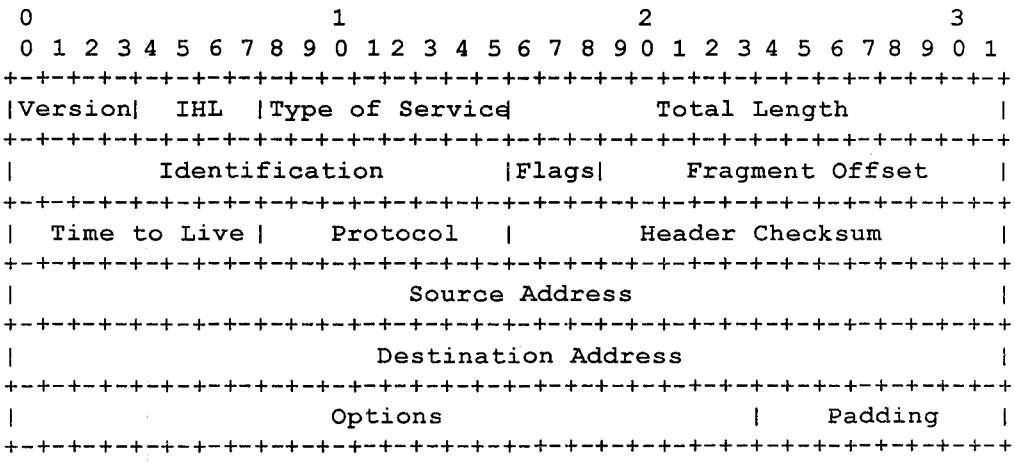
İnternet Protokolü'nde veri iletiminde güvenilir olmayan ve bağlantısız bir yapı uygulanmıştır. Güvenilirlik ve bağlantılılık yerine bu protokolün tasarımında, datagramların esnek bir ağ yapısı üzerinde farklı kullanıcılara, uzaklık veya donanım farkı olmaksızın ve mümkün olan en kısa zamanda iletim garantisi üzerinde yoğunlaşılmıştır. Buna göre IP protokolü, bir datagramın iletim ortamında karşılaşabileceği bit hataları veya tıkanıklıklar sebebiyle atılması gibi

olumsuz durumlarla ilgilenmez. Bunlar bir üst katmanda çalışan TCP'nin ilgileneceği sorunlardır.

3.1.1. IP Datagram Başlığı

Bir IP datagramı, üst katmandan gelen verinin önüne bir IP Datagram başlığı eklenerek paketlenmesi ile oluşturulur. Şekil 3.1'de yapısı gösterilen bir IP Datagram başlığı, normalde seçenekler bölümü olmaksızın 20 baytlık bir uzunluğa sahiptir.

Hem bu verilen IP Datagram başlığında hem de bir sonraki alt bölümde ele alınacak olan TCP parçası başlığında, birden fazla bayttan oluşan alanlardaki değerler, ağ aygıtları üzerinde en büyük bayt ilk olmak üzere iletilirler. Big-endian bayt sıralaması olarak adlandırılan bu sıralama, ağ bayt sıralamasıdır. Bir bayttan büyük verileri little-endian sıralamaya göre kaydeden mimarili işlemcilere sahip makineler, iletim öncesinde bu verileri big-endian bayt sıralamasına dönüştürmelidirler.



Şekil 3.1. IP Datagram Başlığı [23]

Şu anda kullanılmakta olan IP sürümü 4'tür. İnternet üzerinde daha fazla kullanıcı ve farklı iletim olanakları sunan sürüm 6, deneme aşamasındadır.

İnternet başlık uzunluğu (Internet Header Length, IHL), IP Datagram başlığının 32-bitlik kelimeler cinsinden ifadesidir. Bu 4 bitlik IHL alanı ile bir IP Datagram başlığı, en fazla 60 baytla sınırlanmıştır.

Servis tipi (Type of Service, TOS) alanındaki bit değerleri ile o datagramın belli bir ağ yapısı üzerinde iletiminde istenen servis kalitesi ifade edilmektedir.

Toplam uzunluk (Total Length), başlık ve veri alanlarından oluşan bir IP datagramının bayt cinsinden toplam uzunluğunu ifade etmektedir. Bu 16-bitlik alan ile ifade edilebilecek en büyük uzunluk 65.535 bayttır.

Tanımlama (Identification) alanındaki değer parçalanmış bir IP datagramının parçalarının birleştirilmesi amacıyla kullanılır. Parçalanmamış bir datagram için bu alandaki değer sıfırdır.

Bayraklar (Flags) alanındaki üç bit bir IP datagramının parçalanmasında kontrol amacıyla kullanılır. Parça offseti (Fragment Offset) değeri ise taşınan parçanın tüm IP datagramı içinde nereye karşılık geldiğini ifade etmektedir. Bu iki alanın ikisi de parçalanmamış bir datagram için sıfır değerini alır.

Yaşam süresi (Time to Live, TTL) alanındaki değer o datagramın İnternet sisteminde kalabileceği saniye cinsinden maksimum süreyi belirtmektedir. Günümüzde bir IP datagramının hedefe ilerlerken geçeceği yönlendiriciler her ne kadar bir datagramı 1 saniyeden kısa bir sürede işleyip gideceği ağ parçasına yönlendirse de bu alandaki değer datagramın her işlenişinde bir azaltılmaktadır. Bu yaklaşım sonucu yaşam süresi değeri aslında daha çok IP datagramının kaynaktan hedefe yol alırken geçmiş olduğu yönlendiricilerin sayısını ifade etmektedir. Bu alandaki değer IP Datagram başlığının her işleme tabi tutulması sırasında bir azaltılır; ve sıfıra ulaştığında o datagram silinerek bir datagramın tüm ağ üzerinde denetimsiz bir şekilde dolaşarak ağ kaynaklarını gereksiz yere meşgul etmesi önlenmiş olur.

Protokol (Protocol) alanındaki değer, bu datagramın veri alanındaki bilginin hangi üst protokole devredileceğini gösterir. Bu alandaki değerler tipik olarak İnternet Denetim Mesajı Protokolü (Internet Control Message Protocol, ICMP) için 0x01, Kullanıcı Datagram Protokolü (User Datagram Protocol, UDP) için 0x11, ve TCP için 0x06 olmaktadır.

Başlık denetim toplamı (Header Checksum) sadece başlık alanındaki veriler üzerinden hesaplanmaktadır. Bu değer ilgili IP datagramının her işleme tabi tutulması sırasında tekrar hesaplanmalıdır. Hesaplama sürecinin başında denetim toplamının ilk değeri sıfır alınmakta; hesaplama yapılırken başlıktaki değerler 16-

bitlik kelimeler olarak ele alınıp 16-bitlik birin tümleyeni toplamı yoluyla hesaplanan denetim toplamı değeri, bu alana yazılmaktadır.

Kaynak adresi (Source Address), IP datagramının kaynağını, varış adresi (Destination Address) ise hedefini belirten küresel IP adresi değerleridir.

Her IP Datagram başlığında bulunmayabilen seçenekler kısmı ise IP datagramının ağ yapısı üzerinde iletiminde dikkate alınacak bir kısım ayrıntıları içermektedir.

3.1.2. IP Adresleri

İnternet Protokolü'nün kullanılan sürümüne göre İnternet üzerindeki her kullanıcı dört sekizliden oluşan ve tüm global Internet sistemi üzerinde tek-bir (unique) olan bir IP adresi taşımalıdır. Bu yaklaşım yoluyla yaklaşık dört milyar kullanıcı adreslenebilmektedir. Fakat kurulan tüm ağ yapısı üzerinde verilerin iletiminde çeşitli kolaylıklar amaçlanarak bu dört sekizliden oluşan IP adresleri ağ adresi ve yerel adres olmak üzere iki farklı bölüm şeklinde düşünülmüştür. Bunun sonucu olarak IP adres yapısı da A, B, ve C sınıfı olmak üzere sınıflandırılmıştır. A sınıfı IP adreslerinde, ilk baytın en yüksek biti sıfır olup, geri kalan yedi biti ağ adresini oluşturur. Geri kalan son üç bayt ise yerel adresi ifade eder. B sınıfı bir adreste, ilk baytın ilk iki biti bir-sıfır değerlerini alır ve bu baytın kalan altı biti ve ikinci bayt, ağ adresini oluşturur. Geri kalan son iki bayt ise yerel adres değerini taşır. C sınıfı bir adreste ise, ilk baytın ilk üç biti bir-bir-sıfır değerini alır ve bu baytın kalan 5 biti, ikinci bayt ve üçüncü bayt ağ adresi, son bayt da yerel adrestir. Buna göre A sınıfı bir adreste ilk baytın değeri 0 ile 126 arası, B sınıfı bir adreste 128 ile 191 arası, C sınıfı bir adreste ise 192 ile 223 arası olmaktadır. Bu sınıflandırmada belirtilmeyen ilk bayt değeri 127 olan adresler "geri döngü" adresleri olarak adlandırılırlar ve ağ üzerindeki bir bilgisayarın kendisini ifade ederler.

Sınıflandırma ve ağ yapısının farklı alanlar şeklinde düşünülmesi sonucu daha az sayıda kullanıcı adreslenebilmekte, fakat bununla birlikte İnternet daha kolay ve etkin bir ağ yönetimi için farklı alt ağlara bölünebilmektedir.

İnternet üzerindeki her kullanıcının tek-bir IP adresi taşıması zorunluluğu nedeniyle IP adreslerinin kullanıcılara atanmasıyla görevli tüm dünya çapında

merkezi bir tek otoritenin gerekliliği kaçınılmazdır. Bu otorite İnternet Ağ Bilgi Merkezi'dir (Internet Network Information Center, InterNIC). Fakat, bu otorite sadece ağ adreslerini atamakta; kullanıcı adreslerinin atanmasını ağ yöneticilerine bırakmaktadır.

3.2. İnternet Denetim Mesajı Protokolü

İnternet Denetim Mesajı Protokolü (Internet Control Message Protocol, ICMP), her ne kadar farklı bir protokol olarak ele alınsa da genellikle İnternet Protokolü'nün bir parçası gibi düşünülür; ve OSI referans modelinin Ağ katmanında çalışır. Bununla birlikte ICMP mesajları yine de IP tarafından paketlenerek bir alt katmana verilir.

IP protokolü, tasarımının bir sonucu olarak güvenilir değildir. Fakat bu protokol ile birlikte gerçeklenen ICMP protokolü mesajlarının ağ üzerinde değişilmesinin nedeni, IP protokolüne güvenilirlik özelliği eklemek değil, haberleşme ortamındaki çeşitli bozukluklar hakkında geri besleme sağlamaktır. Buna göre ICMP mesajları, datagramların işlenişinde ortaya çıkan çeşitli hataları rapor etmek için kullanılırlar.

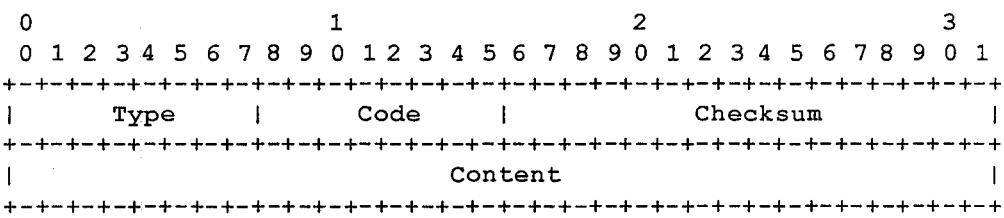
Alınan ICMP mesajları, IP katmanı veya bir üstte bulunan TCP ve UDP katmanı tarafından değerlendirilir. Bazı ICMP mesajları ise, kullanıcılara hata mesajları gönderilmesi amacıyla kullanılır.

Bir ICMP mesajının genel yapısı Şekil 3.2'de verilmiştir. Bu yapıdaki ilk dört bayt tüm mesaj çeşitlerinde aynı yapıdadır.

Tip (type) alanı, belirli bir ICMP mesajını ifade etmek üzere 15 farklı değerden birini alabilmektedir. Kod (code) alanı, ise bazı ICMP mesajlarında sadece sıfır değerini alırken, bazı mesajlarda bir kısım değişiklikleri ifade etmek üzere farklı değerler alabilmektedir.

Denetim toplamı (checksum) alanındaki değer, sadece başlık değil; hem başlık hem de veri olmak üzere tüm ICMP mesajı üzerinden IP Datagram başlığındaki denetim toplamının aynı algoritmayla hesaplanmış olan değeri içerir. Denetim toplamı, tüm ICMP mesajları tarafından içerilmelidir. Denetim toplamı alanında sıfır değeri bulunan bir ICMP mesajı sessizce atılır.

Bir ICMP mesajının içeriği (content) ise mesajın tipine göre farklılık gösterir.



Şekil 3.2. Bir ICMP mesajının genel yapısı [24]

3.2.1. ICMP Mesaj Tipleri

ICMP mesaj tipleri sorgulama ve hata mesajları olmak üzere iki farklı grupta ele alınabilir. Şekil 3.3'te listesi verilen bu ICMP mesajlarından, 0, 8, 9, 10, 13, 14, 15, 16, 17, 18 tipli mesajlar sorgulama, diğerleri ise hata mesajlarıdır. ICMP mesajlarının bu iki grupta incelenmesinin sebebi, ICMP hata mesajlarının bazı durumlarda özel olarak ele alınmasıdır.

```
0       Echo Reply
3       Destination Unreachable
4       Source Quench
5       Redirect
8       Echo
9       Router advertisement
10      Router Solicitation
11      Time Exceeded
12      Parameter Problem
13      Timestamp Request
14      Timestamp Reply
15      Information Request
16      Information Reply
17      Address Mask Request
18      Address Mask Reply
```

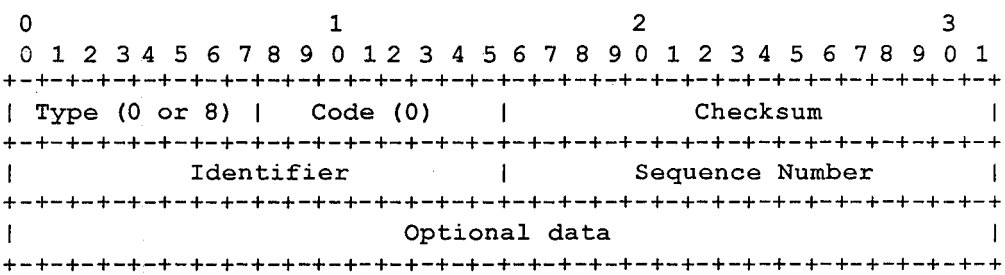
Şekil 3.3. ICMP mesaj tipleri

3.2.2. Yankı İsteği ve Yanıtı

Karşılıklı olarak değişilen bu iki ICMP mesaj tipi ağ üzerindeki bir kullanıcını erişilebilir olup olmadığını test etmek amacıyla kullanılmaktadır. Bir kullanıcının erişilebilir olduğunu test etmek üzere ICMP sık kullanılan "ping"

programı yoluyla yapılmaktadır. ICMP yankı yanıtı mesajı ise kullanıcı tarafından alınan bir yankı isteği mesajına karşılık olarak üretilip isteği gönderen kullanıcıya gönderilmektedir.

Diğer ICMP mesaj tipleri arasında, yönlendiriciler veya ağ geçitleri yanında kullanıcıları da ilgilendiren ICMP mesaj tipleri bu ikisi olduğundan bu çalışmada sadece bu ikisi gerçeklenmiştir. ICMP yankı isteği ve yanıtı mesajlarının ortak yapısı Şekil 3.4'te verilmiştir.



Şekil 3.4. ICMP yankı isteği ve yanıtı mesajlarının ortak yapısı

Bu yapıda tip (type) değeri yankı isteği için 8, yankı yanıtı için ise 0 değerini almaktadır. Kod (code) değeri ise daima sıfır olmaktadır. Denetim toplamı yukarıda belirtildiği gibi tüm ICMP mesajları için geçerli olan algoritma yoluyla hesaplanır. Tanıtıcı (identifier) değeri alınan bir yankı yanıtı mesajının gönderilenle eşleştirilmesi için kullanılır. Seri numarası (sequence number) değeri ise her yeni yankı isteği için bir artırılır. Veri alanı ise şekilde de gösterildiği gibi seçime bağlıdır (optional data). Eğer gönderilen bir ICMP yankı isteği mesajında veri alanında herhangi bir veri varsa, bu mesaj için yankı yanıtı mesajı oluşturulurken istek mesajındaki veri aynen yanıt mesajına kopyalanır.

3.3. İletim Denetim Protokolü

Bilgisayar ağlarının çeşitli getirilerinin bir sonucu olarak gelişmesi ve yaygınlaşması sonucu farklı donanım özelliklerine sahip farklı ağ yapıları arasındaki iletişimin sağlanması amacıyla standart bir iletişim protokolünün ortaya konması kaçınılmaz olmuştur. Bu gereksinimi karşılamaya yönelik ortaya konan yaklaşımlardan İletim Denetim Protokolü (Transmission Control Protocol,

TCP), küresel bir bilgisayar iletişim ağı oluşturabilmesine olanak tanımıştır. TCP protokolü, katmanlı ağ protokol yapısına uyacak, bağlantı kurulumuna dayalı (connection-oriented) ve güvenilir (reliable) bir protokol tasarımıdır [1, 25].

TCP protokolü ile taşınan veri, bir üst katmandaki uygulamadan alındıktan sonra uygun uzunluklara bölünüp önüne bir TCP başlığı eklenerek paketlenir ve bir alt katmandaki IP protokolüne verilir. Önüne TCP başlığı eklenerek paketlenmiş bu veriye TCP parçası (segment) adı verilir.

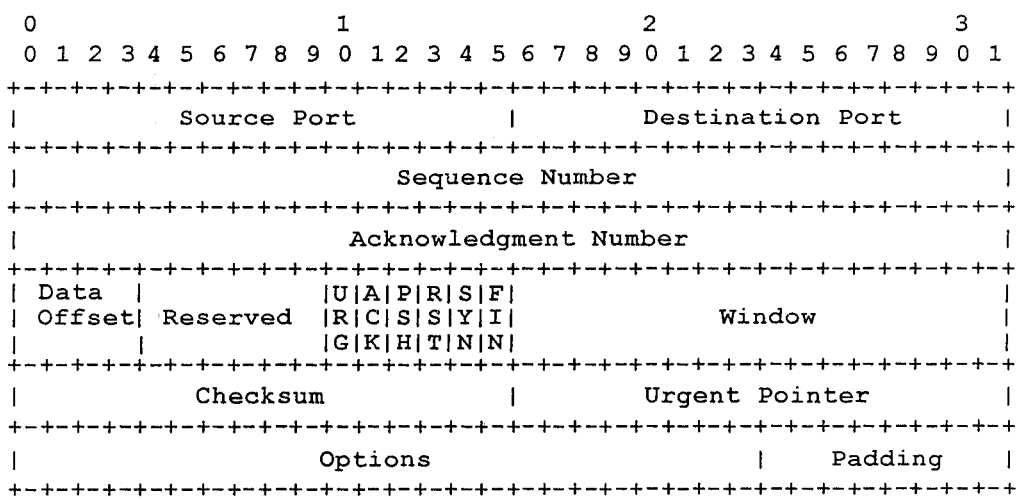
TCP protokolünü tanımlamakta kullanılan bu iki kavramdan "bağlantıya dayanması", veri değişimi için protokolün iki ucunun veri alışverişi öncesinde bir TCP bağlantısı kurmaları, diğer bir deyişle karşılıklı bağlantı kurulması konusunda anlaşmaları anlamına gelir. Bağlantının kurulması aşamasında her iki uçtaki TCP gerçeklemeleri, karşılıklı olarak bağlantı isteklerini (SYN) ve durum bilgilerini gönderirler. Ayrıca, karşı uçtan bu isteklerinin onaylandığını ve durum bilgilerinin kabul edildiğini ifade eden pozitif kabul (ACK) yanıtının gelmesini beklerler. Bu aşamanın başarılı olarak tamamlanmasıyla, iki uç arasında veri alışverişine hazır bir TCP bağlantısı kurulmuş olur. Kurulmuş olan bu bağlantı, veri alışverişi bittiğinde her iki kullanıcının da gereksiz yere meşgul olmaması için bitirilir. Bağlantının bitirilmesi için de yine karşılıklı olarak sonlandırma istekleri (FIN) yollanır ve karşılığında pozitif kabul (ACK) yanıtının gelmesi beklenir.

"Güvenilirlik" ise bir TCP gerçeklemesi tarafından gönderilen verideki her baytın bir seri numarası ile numaralandırılması, ve alıcı TCP gerçeklemesi tarafından başarılı olarak alınan her bayta karşılık bir pozitif kabul (ACK) gönderilmesiyle sağlanır. Buna bağlı olarak, eğer belirli bir zaman-aşımı süresinin sonunda karşılık olarak pozitif kabul (ACK) alınmamışsa, gönderilen verilerin karşı uçtaki TCP gerçeklemesi tarafından alınmadığı varsayılarak bu veriler tekrar gönderilir. Bununla birlikte, alıcı uçtaki TCP gerçeklemesi, alınan her parçaya bir denetim toplamı denetimi yaparak hatalı sonuç veren parçalardaki seri numaraları için pozitif kabul (ACK) göndermez. Bu durum gönderici uç tarafından ele alındığında, hatalı olarak yerine ulaşan parçalar için de yine belli bir zaman-aşımı süresi içinde pozitif kabul (ACK) gelmediği için bu parçaların da ağ üzerinde kaybolduğu varsayılır ve parçalar tekrar gönderilir.

TCP protokolünde aynı zamanda, alıcı ucun gönderici ucun göndereceği veri miktarını belirlemesi şeklinde bir akış denetimi de bulunmaktadır. Denetim mekanizması, alıcı uç tarafından sağlam olarak alınmış olan en son baytın seri numarası için pozitif kabul (ACK) gönderirken, alıcı tarafından bir de pencere bilgisi gönderilmesi yoluyla gerçekleştirilir. Bu pencere bilgisi, göndericinin alıcıdan daha önce göndermiş olduğu baytlar için herhangi bir pozitif kabul (ACK) almaksızın gönderebileceği en fazla veri miktarını ifade eder.

3.3.1. TCP Başlığı

TCP başlığı bir datagramda IP Datagram başlığından sonra gelerek TCP protokolüne ait bir kısım bilgileri içerir. Şekil 3.5'te TCP başlığının yapısı gösterilmiştir. Bu başlığın uzunluğu seçenekler olmaksızın 20 bayttır.



Şekil.3.5. TCP Başlığı [25]

Her TCP parçasındaki kaynak ve hedef port numaraları (source port, destination port) veriyi yollayan ve alan uygulamayı ifade eder. Bu iki değer, IP başlığındaki kaynak ve hedef IP numaraları değerleriyle birlikte bir TCP bağlantısını ayırt etmek için kullanılır. Bir IP adresi ve port numarası aynı zamanda "soket" adıyla da adlandırılmaktadır. Kaynak IP adresi, kaynak port numarası, hedef IP adresi ve hedef port numarası bileşenlerinden oluşan bir "soket

çifti" ise İnternet üzerindeki bir TCP bağlantısının iki uç noktasını ifade etmek için kullanılır.

Seri numarası (sequence number), gönderilen TCP parçası içindeki verinin ilk baytını ifade eder. TCP protokolünde, veri alışverişi yapan iki uygulamada bir uçtan diğerine gönderilen verilerin her bir baytı bir seri numarası ile numaralandırılır. TCP bağlantısı kurmak için gönderilen ilk parçada başlıktaki SYN bayrağı set edilir. Bu parça veri taşımaz ve seri numarası o bağlantı için SYN gönderen kullanıcı tarafından kullanılacak seri numarası değerlerinin başlangıcını ifade eden ilk seri numarasıdır (initial sequence number, ISN). Bundan sonra gönderilecek ilk TCP parçasındaki seri numarası ISN+1'e eşit olacaktır. Çünkü SYN bayrağı TCP'nin tasarımı gereği bir adet seri numarası alır.

Onay numarası (acknowledgment number) ise, gelen TCP parçasında ACK bayrağı bir değerini almışsa bir anlam ifade eder. Bu değer, ACK gönderen ucun bu parçayı gönderene başarılı bir şekilde almış olduğu en son baytın seri numarasının bir fazlasıdır. Onay numarası bu anlamda, ACK gönderen ucun beklemekte olduğu baytın seri numarasını ifade eder. SYN bayrağının tersine ACK bayrağı seri numarası almaz, dolayısıyla bağlantı boyunca değişilen her TCP parçasında ACK bayrağı bir yapılır. Ayrıca, ACK bayrağı set edilmiş bir TCP parçası FIN veya SYN bayrakları set edilmiş bir parça gibi sadece TCP bağlantısının kuruluş ve sonlandırılış aşamalarında değil, tüm veri alışverişi süresince değişilir.

TCP bağlantı kurmuş olan iki uç arasında tam çift yönlü (full-duplex) bir veri akışı sağladığı için her iki ucun da normalde sadece tek yönlü bir veri akışı olacak olması durumunda bile karşılıklı olarak birer seri numarası belirlemeleri gerekir.

Veri ofseti (data offset) aynı zamanda başlık uzunluğu olarak da adlandırılır ve başlığın 32-bitlik kelimeler cinsinden uzunluğunu ifade eder. Seçenekleri içermeyen bir TCP başlığı normal de 20 bayt uzunluğundadır; dolayısıyla bu değer 32-bitlik kelime cinsinden 5'tir.

TCP başlığında altı tane bayrak biti vardır. Bir TCP parçası içinde bu bayraklardan bir yada aynı anda birden fazlası bir yapılmış olabilir. Bu bayraklar ve anlamları aşağıdaki gibidir:

URG : Acil işaretçisi alanı geçerlidir.

ACK : Onay numarası alanı geçerlidir.

PSH : Alıcı paçasındaki veriyi bekletmeksizin uygulamaya vermelidir.

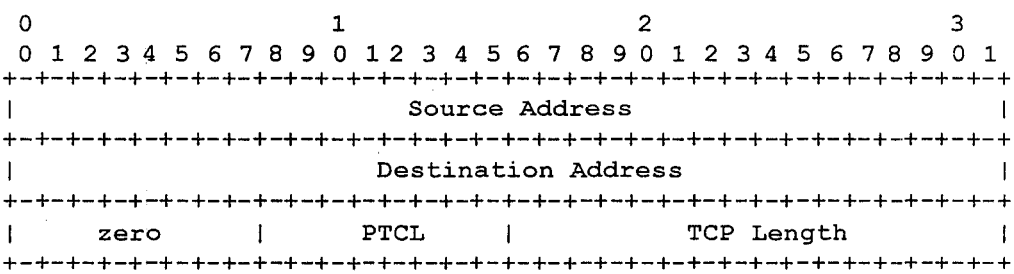
RST : Bağlantı yeniden başlatılmalıdır.

SYN : Seri numaraları yeni bir bağlantı için ayarlanmalıdır.

FIN : Gönderici uç veri göndermeyi bitirmiştir.

Pencere büyüklüğü (window size) alanındaki değer ise bu değeri gönderen tarafın, aynı parça içindeki onay numarasına karşılık gelen seri numarasına sahip bayttan itibaren kabul edebileceği bayt sayısını ifade eder.

Denetim toplamı (checksum) alanındaki değer ise TCP başlığı ve verisi üzerinden hesaplanır. Bu alandaki denetim toplamı değeri UDP protokolünün aksine TCP için gönderici tarafından mutlaka hesaplanmış olması gereken bir değerdir. Bu değer hesaplanırken, normal TCP parçası ile birlikte aynı zamanda Şekil 3.6'da gösterildiği gibi IP başlığındaki hedef ve kaynak IP adresleri, protokol numarası ve parça uzunluğu değerlerini de içeren bir yalancı-başlık da içermelidir. Bu değer hesaplanırken aynen IP başlık denetim toplamı hesaplanırken kullanılan algoritma kullanılır.



Şekil 3.6. Yalancı TCP başlığı

Acil işaretçisi (urgent pointer) alanındaki değer ise sadece URG bayrağı bir olan parçalarda geçerlidir. Bu alandaki değer yine aynı parçadaki seri numarasına eklenerek acil verinin son baytına ulaşmaya yarayan pozitif bir ofset değeridir.

Seçenekler alanı normalde sadece bağlantı kurulması amacıyla gönderilen SYN bayrağı bir yapılmış parçalarda bulunur. Seçenekler alanında ilan edilen en genel TCP seçeneği maksimum parça büyüklüğü'dür (maximum segment size,

MSS). Bu seçenek, ilan eden uçtaki TCP gerçeklemesinin almaya hazır olduğu en büyük parçanın büyüklüğünü (MSS) ifade eder.

Bir TCP parçasının sadece TCP başlığından oluştuğu durumlar vardır. Bunların birincisi, bağlantının başlatılması amacıyla değişilen SYN bayraklarının set edilmiş olduğu ilk TCP parçalarıdır. Bu parçalarda her ucun verileri alırken uyacağı ayrıntılar seçeneklerle belirtilir. Bir diğer sadece başlıktan oluşan TCP parçası, alınan bir veriyi onaylamak amacıyla, onaylayan ucun gönderecek herhangi bir verisi olmadığında gönderdiği ACK bayrağı set edilmiş TCP parçasıdır. Sonuncusu ise, her iki tarafın artık gönderecek verisi kalmadığı durumda kurulmuş olan TCP bağlantısını bitirmek amacıyla değişmiş oldukları FIN bayraklarının set edilmiş olduğu TCP parçalarıdır. Fakat bağlantının bitirilmesi için gönderilen FIN bayrağının özellikle veri taşımayan bir TCP parçası ile özel olarak gönderilmesine gerek yoktur. Yani gönderilecek en son veriyi taşıyan TCP parçasında FIN bayrağı bir yapılarak da gönderilebilir.

3.3.2. Bir TCP Bağlantısının Kurulması ve Sonlandırılması

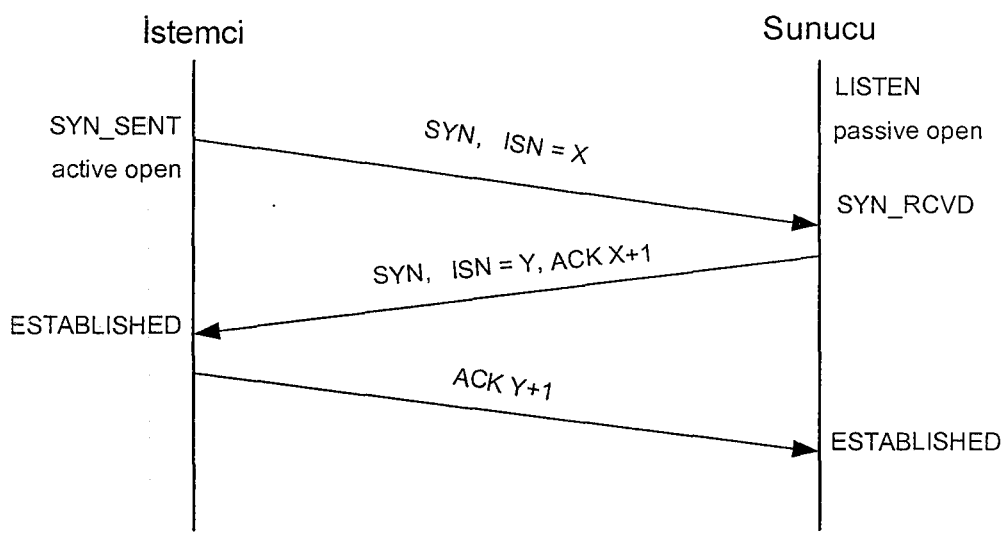
TCP protokolü yukarıda da belirtildiği gibi bağlantılı bir protokoldür. Dolayısıyla bu protokol aracılığıyla veri değişimi söz konusu olduğunda her iki uç öncelikle bir TCP bağlantısı kurarlar. Şekil 3.7'de bir TCP bağlantısının kurulmasında her iki TCP gerçeklemesinin geçmiş oldukları evreler ve karşılıklı olarak hangi parçaların değişildiği gösterilmiştir. Bu işlem, şu adımlarla ifade edilebilir:

Veri değişimine gereksinim duyan uç (istemci), karşı uçtaki sunucunun bağlanmak istediği port numarasını ve kendi ilk seri numarasını (ISN) belirten bir SYN parçası gönderir. Başlangıç aşamasında CLOSED evresinde olan bu uç, bağlantı isteğini belirten bu ilk SYN parçası gönderdikten sonra SYN_SENT evresine geçer.

Karşı uçtaki sunucu ise başlangıçta bilinen bir port numarasına yönelecek bağlantı isteklerini sürekli olarak kontrol ettiği LISTEN evresindedir. Bu uçtaki sunucu bir SYN parçası aldıktan sonra SYN_RCVD evresine geçer. Ardından hem aldığı bu SYN parçasını kabul ettiğini belirten ve gönderilen ISN'nin bir fazlasının onay numarasına yazıldığı bir ACK parçası, hem de kendi ilk seri

numarasını (ISN) içeren bir SYN segmenti gönderir. Gönderilen bu iki parça ayrı olarak değil hem SYN hem de ACK bayrakları set edilmiş ve ilgili alanlar doldurulmuş bir tek parça olarak gönderilir.

Bağlantı isteği için ilk SYN segmentini göndermiş olan istemci, karşı uçtaki sunucudan bu SYN ve ACK parçasını aldığında, ACK bayrağının bir yapılmış olması dolayısıyla göndermiş olduğu bağlantı isteği kabul edilmiş olduğundan ESTABLISHED evresine geçer. Ayrıca, SYN bayrağı da bir yapılmış olduğu için karşı ucun bağlantı isteğini de ifade eden bu parçayı seri numarasının bir fazlasının onay numarasına yazıldığı bir ACK parçası ile onaylar. Gönderilen bu son ACK parçasını alan karşı uç da ESTABLISHED evresine geçer ve karşılıklı TCP bağlantısı kurulmuş olur.



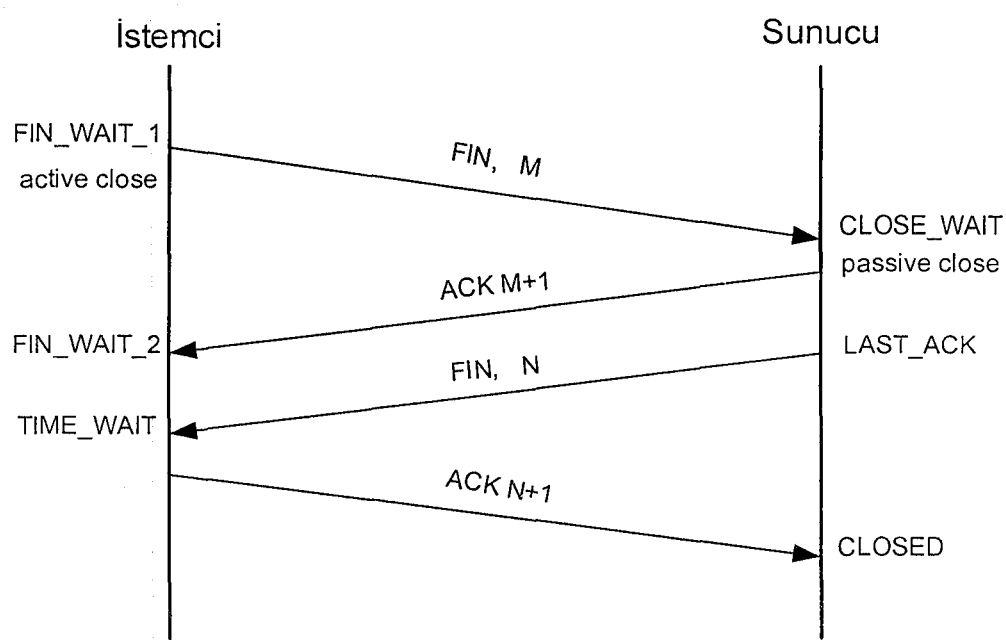
Şekil 3.7. Bir TCP bağlantısının kurulması

Bu şekilde ifade edilen bağlantı kurulumu "üç yollu el sıkışma" olarak adlandırılır. Bu işlemde bağlantı kurulumu için ilk kez SYN parçası gönderen ucun yaptığı eyleme, *aktif açım*, SYN aldıktan sonra karşılık olarak SYN ve ACK gönderen ucun yaptığı eyleme de *pasif açım* denir. Pasif açım eylemini yapan uç, herhangi bir bağlantı isteği için hazır bekleyerek belli port numaralarını sürekli kontrol etmektedir.

Bir TCP bağlantısında veri aktarımı bittikten sonra kurulmuş olan bu bağ sonlandırılır. Sonlandırma amacıyla ise FIN bayrağı bir yapılmış TCP parçaları ve

bunlara karşılık gönderilen ACK parçaları değişilir. Şekil 3.8'de değişilen bu paketler ve her iki TCP gerçeklemesinin geçmiş olduğu evreler gösterilmiştir.

TCP bağlantısının kurulması için üç parçanın değişilmesi yeterli iken, bağlantının sonlandırılması için dört parçanın değişilmesi gerekmektedir. Bir ucun karşı uçtan bir FIN parçasısını alması, artık o karşı ucun gönderecek herhangi bir verisinin kalmamış olduğunu ifade etmektedir. Fakat bir uçtaki TCP gerçeklemesi karşıdan FIN parçası almış ve onaylayan ACK parçası göndermiş olması rağmen hala veri göndermeye devam edebilir. Yarım-kapalı durumu olarak adlandırılan bu durum, nadiren kullanılır.



Şekil 3.8. Bir TCP bağlantısının sonlandırılması

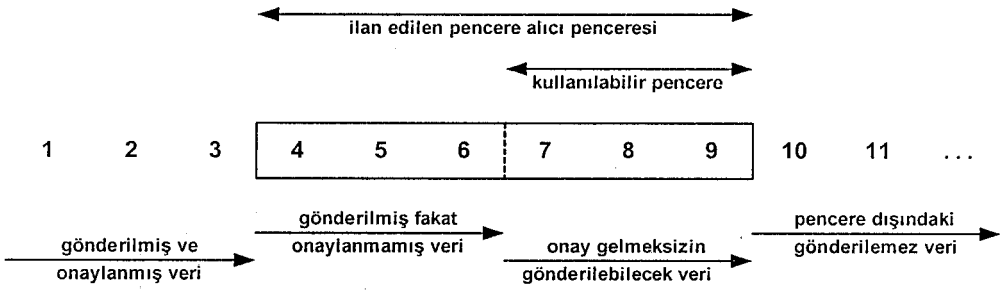
TCP bağlantısı sonlandırma işlemlerinde, farklı TCP gerçeklemeleri tarafından farklı yaklaşımlar uygulanmaktadır. Bazı durumlarda veri gönderen uç olan sunucu uç, son veri parçasını içeren parçanın FIN bayrağını da set ederek gönderir. Bazen de yine sunucu uç veri gönderimi bittikten sonra FIN bayrağı bir yapılmış yeni bir parça gönderir. Bazen ise belli bir miktarda veri alındıktan FIN bayrağı bir yapılmış parça gönderilerek sonlandırma işlemi istemci uç tarafından başlatılır.

Her üç durumda da sonlandırma işlemini ilk başlatan uç FIN parçasını gönderdikten sonra FIN_WAIT_1 evresine geçer. Karşı uçtaki TCP gerçeklemesi alınan bu FIN parçasını onayladığına dair bir ACK parçası gönderdiğinde FIN_WAIT_2 evresine geçilir. Bu işlem sonrasında, karşı uçtaki TCP gerçeklemesi de FIN parçası göndererek LAST_ACK evresine geçer. FIN_WAIT_2 evresindeki uç FIN parçası aldığında TIME_WAIT evresine geçer ve alınan bu FIN parçası onaylayan bir ACK segmenti gönderir. Bu ACK parçasını alan uç da TCP bağlantısını kapatır. TIME_WAIT evresine geçmiş olan uç da daha önceden belirlenmiş bir süre miktarınca bekledikten sonra TCP bağlantısını kapatır.

3.3.3. TCP Veri Alışverişi ve Kayan Pencereler

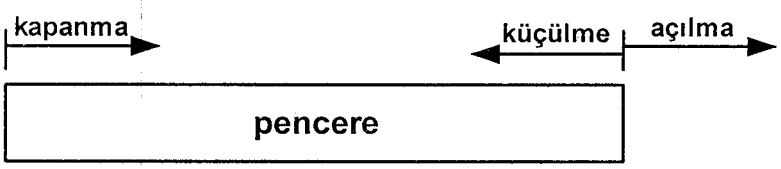
Bir TCP bağlantısı süresince değişilen her parçada, her iki uç da karşı tarafa, onay göndermeksizin alabileceği veri miktarının bayt cinsinden ifadesini TCP başlığındaki pencere alanına yazarak gönderir. Veri gönderen bir uç, karşı uçtan almış olduğu parçalardaki pencere alanı bilgisi sıfırlanmadığı sürece veri göndermeye devam eder. Alınan parçalardaki pencere bilgisi sıfırlandığında ise veri gönderimi, pencere alanı bilgisi sıfırdan farklı olan yeni bir parça alınana kadar durdurulur.

Bu veri akışını düzenlemek için TCP içerisinde "kayan pencereler" protokolü gerçeklenir. Şekil 3.9'da gösterilen bu protokolde alıcı tarafından ilan edilen pencere büyüklüğüne "alıcı penceresi" denir. Alıcı penceresinin sağ kenarı, alıcı tarafından başarılı olarak alınıp karşılığında onay gönderilmiş en son bayttan sonraki ilk bayttan başlar. Bu pencerenin sol kenarı ise bağlantı kurulması sırasında ilan edilmiş olan alıcı penceresi alanının içine alacağı son baytta son bulur. Bu alıcı penceresi ise iki farklı alandan oluşur. Bu alanlardan ilki, gönderici tarafından gönderilmiş fakat karşılık olarak onay alınmamış baytlardan oluşur. Gönderilmeye hazır ilk bayttan pencerenin sol kenarına kadar olan bölüm ise onay alınmaksızın gönderilebilecek veri miktarını ifade eder. Bu ikinci bölüm "kullanılabilir pencere" olarak adlandırılır.



Şekil 3.9. TCP kayan pencereler protokolü

Bu pencere, zaman içerisinde veri alışverişi devam ederken gönderilen veriler alıcı tarafından onaylandıkça sağa doğru kayacaktır. Pencerenin iki kenarı, gönderilen veri ve karşılığında alınan onaya bağlı olarak ve tamamen birbirlerinden bağımsız hareket edeceğinden pencerenin boyutu, veri alışverişi süresince azalacak veya artacaktır.



Şekil 3.10. Pencere kenarlarının hareketi

Şekil 3.10'da gösterildiği gibi pencere kenarlarının, kapanma, açılma ve küçülme olarak adlandırılan üç hareketi vardır. Pencerenin sol kenarı sadece sağa doğru hareket eder ve bu hareketle pencere "kapanır". Bu hareket, gönderici tarafından verilerin gönderilmesi ve karşılığında onay alınması sonucu olur. Sağ kenar ise her iki tarafa hareket edebilir. Bu kenarın sağa doğru hareketiyle pencere "açılır"; sola doğru hareketiyle ise "küçülür". Bu kenarın sağa doğru hareketi, alıcı ucun ilan ettiği pencere boyutu değerinin büyümesiyle olur; sola doğru hareketi ise alıcı uçta normalde göndericinin ilan etmiş olduğu maksimum parça büyüklüğünden daha küçük miktarda alan bulunması durumunda alıcının ağ üzerinde küçük parça iletimini önlemek amacıyla sıfır pencere büyüklüğü ilan etmesi nedeniyle olur. Fakat bu istenmeyen bir durumdur. Dolayısıyla her ne kadar küçük TCP parçalarına yol açacak olsa da pencerenin küçülmesine yol açacak pencere boyutu ilan edilmez.

3.3.4.Güvenilir Veri İletimi

TCP protokolünün ikinci önemli özelliği, veri iletiminde güvenilir bir yapı sunmasıdır. Ağ katmanındaki IP protokolü, datagramların sıralı olarak (in-order) ve bit hataları olmaksızın teslim edilmesini garanti etmemektedir. IP protokolünün sağlamış olduğu servis ile datagramlar, ağdaki yönlendiricilerde oluşan tıkanıklıklar nedeniyle alıcıya, sıralı olmaksızın (out-of-order) veya bit hataları ile ulaşabilir. İletim katmanı parçaları da protokol yığıtı yapısı gereği IP datagramları ile taşındığından bu sorunlar, İletim katmanı parçalarını da etkilemektedir.

Bir uçtaki kullanıcıdan diğerine acil ve önemli bir verinin iletiminde gerekli olan güvenirlik, İletim katmanı protokolü olan TCP tarafından sağlanmaktadır. TCP protokolünün güvenilir veri iletimi servisi, Uygulama katmanındaki bir programın İletim katmanından almış olduğu verinin boşluksuz, tekrarlamasız, bozulmamış ve sırasında olarak karşı uçtaki göndericinin göndermiş olduğu veri dizisinin aynı olmasını garanti eder.

TCP protokolünün tasarımında güvenilirlik, gönderilen veri için alınacak pozitif ACK parçaları yoluyla sağlanır. Alınacak pozitif ACK parçalarının değerlendirilmesinde ise iki nokta göz önünde bulundurulur. Bunlardan birincisi, pozitif ACK parçalarının belli bir zaman aşımı (timeout) süresi içinde gelip gelmediğinin; ikincisi ise, alınan pozitif ACK parçasının en son gönderilen veri için olup olmadığının denetimidir. Beklenen pozitif ACK parçasının zamanında gelmemesi veya son gönderilen veri için beklenen pozitif ACK yerine en son alınan pozitif ACK parçasının yeniden alınması durumunda bu veriyi taşıyan parça tekrar gönderilir.

4. GÖMÜLÜ SİTEMLER İÇİN TCP/IP TASARIMI

Bu tez çalışmasında, seri bağlar üzerinden bir modem aracılığıyla İnternet bağlantısını gerçekleştirecek 8-bitlik veya 16-bitlik bir mikrodenetleyici sistemi için, düşük bellek gereksinimi olan PPP/TCP/IP protokol yığıtının tasarımı ve kodlanması gerçekleştirilmiştir.

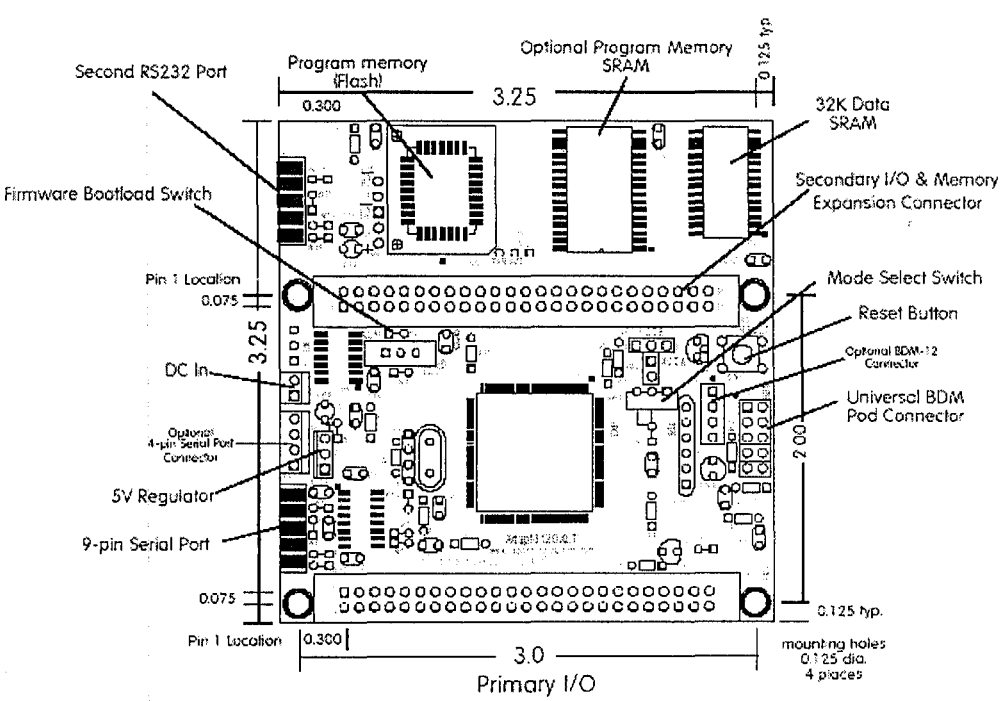
Bu çalışmada, bir mikrodenetleyici sisteminin İnternet bağlantısına sahip olması için gerekli olan iki temel bileşenden, mevcut ağ protokol yığıtlarının en yaygını olan TCP/IP protokol yığıtının bir mikrodenetleyici sistemi üzerinde çalıştırılması problemi ele alınarak bir çözüm ortaya konulmuştur. Fiziksel katman konusunda esneklik elde edilebilmesi için mevcut telefon hatlarının bir modem aracılığıyla kullanılabilmesi amaçlanmıştır. Bu amaca yönelik de seri hatlar üzerinden İnternet bağlantısının gerçekleştirilmesi için geliştirilmiş olan PPP protokolü, Veri Bağı katmanı protokolü olarak ele alınmıştır. Bir mikrodenetleyici sisteminde TCP/IP için geçerli olan düşük işlem gücü ve bellek kapasitesi kısıtları PPP için de geçerli olduğundan bu protokolün de yine bu kısıtlı donanım özelliklerine sahip sistemlerde çalışabilecek bir tasarımı gerçekleştirilmiştir. Aynı zamanda, Fiziksel katman için düşünülen modem arayüzü ile mikrodenetleyici sisteminin veri alışverişinin, sistem üzerindeki temel bir bileşen olan seri iletişim arayüzü (Serial Communication Interface, SCI) yoluyla yapılabilmesini gerçekleştirecek sürücü fonksiyonları da tasarıma eklenmiştir. Bu tasarımın TCP/IP protokolü aracılığıyla veri alışverişini sağlıklı bir şekilde yapıp yapmadığının test edilmesi amacıyla da Uygulama katmanı için çok basit bir HTTP istemci tasarımı gerçekleştirilmiştir.

Gerçekleştirilen tüm tasarımın kodlanması, hem çok farklı mikrodenetleyici sistemlerine kolaylıkla uyarlanabilmesi hem de gerektiğinde kolaylıkla genişletilebilmesi için C programlama dilinde, modüler programlama yaklaşımıyla gerçekleştirilmiştir. Yazılan C kodları, ImageCraft Creations Inc. firması tarafından Motorola MC68HC12 mikrodenetleyicisi için üretilmiş olan "ICC12" derleyicisi ile derlenerek, kullanılacak mikrodenetleyici sistemi üzerinde çalıştırılmaya uygun makina koduna çevrilmiştir [26].

ICC12, ANSI standart C dili kullanılarak geliştirilen kaynak kodlarının projeler şeklinde düzenlenerek derlenmesine olanak tanıyan bir tümleşik geliştirme ortamıdır (Integrated Development Environment, IDE). ICC12 geliştime ortamı ile kodların düzenlenmesi ve oluşturulması (build) tamamen bu IDE ile yapılabilmektedir. Kodların derlenmesi sırasında oluşan derleme hataları da IDE içindeki durum penceresinde gösterilmekte; ve fare ile bu hataların üzerine tıklanarak editör penceresinde doğrudan hatalı satırın üzerine gidilebilmektedir. Tümleşik proje yöneticisinin oluşturduğu "makefile" ile projedeki C modüllerinden elde edilen "object" dosyalarının belirlenen kod ve veri adreslerine göre birleştirilerek çalıştırılabilir makine kodunun üretilmesi sağlanmaktadır.

4.1. Mikrodenetleyici Sistemi

Bu çalışmada geliştirilen TCP/IP protokol yığıtının test edilmesi amacıyla, Technological Arts firması tarafından geliştirilmiş olan ADAPT812DXLT modeli bir mikrodenetleyici kartı kullanılmıştır. Kullanılan bu mikrodenetleyici kartının devre şeması Şekil 4.1'de verilmiştir [27].



Şekil 4.1. ADAPT812DXLT mikrodenetleyici sistemi

ADAPT812DXLT, üzerinde Motorola MC68HC12A4 mikrodenetleyicisi bulunan bir karttır. Sistemin temelini oluşturan Motorola MC68HC12A4 mikrodenetleyicisi, bu tasarımda Genişletilmiş Dar Kip'te (Expanded Narrow Mode) çalışmaktadır. Genişletilmiş Dar Kip, bu mikrodenetleyicinin üç normal (Tek-Çip Kipi, Genişletilmiş Geniş Kip, Genişletilmiş Dar Kip) çalışma kipinden biridir. Bu kipte mikrodenetleyici, 8-bit veriyolu, 16-bit adresyolu ile dışarıdan eklenecek program veya veri belleklerine ulaşacak şekilde çalıştırılır [27, 28].

Motorola MC68HC12A4 mikrodenetleyicisi, 16 Mhz kristal ile 8 Mhz'lik veriyolu hızı sağlayan, genişletilmiş komut seti ile Sayısal İşaret İşleme (Digital Signal Processing, DSP) ve Bulanık Mantık (Fuzzy Logic) uygulamalarını destekleyecek kapasitede tasarlanmış yeni nesil bir mikrodenetleyicidir. Bu yenilenmiş konut seti sayesinde öncülü sayılabilecek MC68HC11 mikrodenetleyicisine oranla %30'luk bir kod büyüklüğü avantajı sağlanmaktadır. Aynı zamanda MC68HC11'in komut seti de desteklenerek bu mikrodenetleyici sistemleri için geliştirilen programların da kolaylıkla güncellenebilmesine olanak tanınmıştır. Tek-çip kipinde, çip üzerinde gerçeklenmiş 4 kilobaytlık iç EEPROM ve 1 kilobaytlık iç SRAM uygulamalar için kullanılabilmektedir. Genişletilmiş kiplerde ise, "bellek sayfalama" yöntemi ile 4 megabayta kadar dış program belleği ve 1 megabayta kadar dış veri belleği desteklenebilmektedir. Çip üzerindeki iki seri haberleşme arayüzü (SCI) portu ise 16 MHz'lik kristal ile 38.4 Kbps'lik baud hızı desteklenebilmektedir. Ayrıca, 8-bitlik ve 16-bitlik zamanlayıcı birimleri, ve 16-bitlik darbe toplayıcısı (pulse accumulator) da yine mikrodenetleyici üzerinde gerçeklenmiştir. 8-bitlik yonga-üzeri sayısal-örneksel çevirici ise çeşitli denetim uygulamalarında kullanılabilmektedir. Arkaplan Hata Ayıklama Kipi (Background Debug Mode, BDM) portu aracılıyla hem uygulama kodlarının mikrodenetleyici sistemine yüklenmesi hem de hata ayıklama işlemleri ele alınabilmektedir [28].

ADAPT812DXLT sisteminde, çalıştırılacak program kodunun depolanması için 128 kilobaytlık bir dış flaş bellek vardır. 32 kilobaytlık SRAM ise uygulamada kullanılacak veriler için kullanılmaktadır. Aynı zamanda MC68HC12A4'teki çip üzerindeki EEPROM'a program yükleyici monitor programı yüklenmiştir. Mikrodenetleyici üzerinde bulunan iki seri haberleşme

arayüzü (SCI) portu da DB-9 tipi konnektörlere bağlanarak, sistemin dış haberleşmesinde alternatiflere olanak tanıyacak şekilde tasarlanmıştır [27].

4.2. Seri İletişim Arayüzü

Modem ile veri alışverişinde, birinci seri port (SCI0) kullanılmıştır. SCI portu üzerinde veri gönderimi, dış etkenlere bağlı olmayan bir işlem olduğu için, SCI veri kütüğüne yapılan normal yazım işlemleriyle sağlanmıştır. Gelen veriler için ise bir SCI Kesme Servisi Yordamı (Interrupt Service Routine, ISR) yazılarak, alım sırasında verilerin kaçırılmadan sistem belleğinde bulunan bir tampona kaydedilmesi sağlanmıştır.

Verilerin modeme gönderiminde derleyicide gerçeklenmiş olan bilinen "`int putchar (int c)`" ve "`int puts (const char *string)`" ANSI (American National Standarts Institute) C fonksiyonları kullanılmıştır.

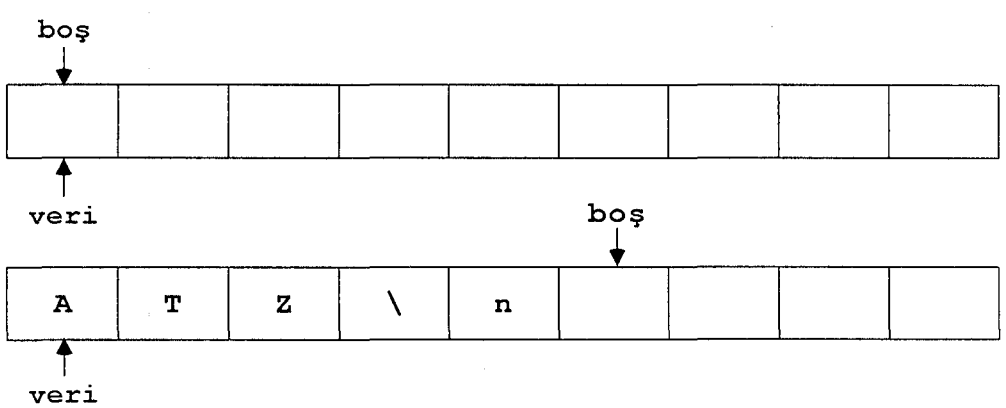
Veri alımında ise yazılan ISR, mikrodenetleyici kartının gerçeklenmesinden kaynaklanan bir gereklilik olarak, kodun derlenmesi sırasında bağlayıcının (linker) bu yordam kodunu dış flaş belleğe yönlendireceği şekilde tasarlanmıştır. Fakat yapılan gerçeklemenin çalışması sırasında normal bir modem işleminin iki kipine göre (komut kipi ve veri kipi) alım işleminde iki farklı yaklaşımın gerçekleştirilmesine gereksinim duyulmuştur. Bu gereksinim dolayısıyla, tek bir ISR kodunun iki farklı fonksiyonu yerine getirebilmesi için alternatif bir çözüm yolu olarak fonksiyon işaretçilerinin kullanımıyla ISR içinden alınan veri ile bir fonsiyon işaretçisi çağırılmıştır. Bu şekilde tasarlanan ISR kodu Şekil 4.2'de verilmiştir. Ana programda ise ISR içinden çağırılan fonksiyon işaretçisinin, modemin karşı uçtaki ISP ile bağlantı kurulması öncesinde alınan veriyi normal olarak bir FIFO (First-In-First-Out, İlk Giren İlk Çıkar) tampon üzerine alan bir fonksiyonu, ISP ile bağlantı kurulması sonrasında ise, verileri Bölüm 2.2.3'te bahsedilen çerçeve yapısına göre alan bir fonksiyonu göstermesi sağlanmıştır. Alınan verilerin yazılacağı bu FIFO tampon, mikrodenetleyici sisteminin veri belleğinde oluşturulmuş olan 1500 baytlık bir dizidir. Bu dizinin uzunluğunun derleme anında 1500 bayt olarak belirlenmesinin nedeni, en küçük maksimum alım biriminin, PPP protokolünün bir gerekliliği olarak daha küçük bir değer alamamasıdır. Maksimum alım biriminin sorgulanması, Bölüm 4.3'te açıklanacağı

üzere sadeleştirme amacıyla sorgulamalardan çıkartılarak aynı zamanda varsayılan değeri de olan en küçük 1500 bayt değerinin kabul edilmesi sağlanmıştır.

```
#pragma interrupt_handler SCI0_Handler (void) {
        // clear the RDRF flag by reading first
        SC0SR1;
        // then
        SC0DR;
        // forward the received character
        FwdISR (SC0DRL);
}
```

Şekil 4.2. SCI Kesme Servis Yordamı

Komut kipinde iken modeme, öncelikle modem konfigurasyonun resetlenmesi ve elsıkışma sinyallerinin iptali için uygun bir modem ilkleme katarı (modem initialization string) gönderilmiştir. Ardında da aranacak ISP'nin numarasının çevrilmesi için bir komut gönderilerek, modemin PPP bağı için devre anahtarlamalı bağı kurması sağlanmıştır. Bu işlemler sırasında modem komut kipinde olduğu için veri alışverişinde, SCI servis yordamından Şekil 4.3'deki FIFO tampon yapısını kullanan bir alım fonksiyonuna yönlendirilmesi gerçekleştirilmiştir.



Şekil.4.3 FIFO tampon yapısı

4.3. PPP Gerçeklemesi

İkinci bölümde de belirtildiği üzere PPP protokolü üç ana bileşenden oluşmaktadır:

1. Çok-protokollü datagramları kapsülleme metodu.
2. Veri alışverişi için bağlantıyı kurmak, biçimlendirmek ve test etmek için bir Bağ Denetim Protokolü (Link Control Protocol, LCP).
3. Farklı ağ protokollerini kurmak ve biçimlendirmek için bir Ağ Denetim Protokolleri (Network Control Protocols, NCPs) ailesi [9].

Normal bir PPP gerçeklemesi, LCP ve NCP sorgulamalarında karşı uçtaki PPP gerçeklemesinden gelebilecek tüm seçenekleri ele alabilecek şekilde tasarlanmaktadır. Bunun sonucu olarak ise tüm aşamaların bellekte tutulduğu, tüm paket tiplerinin ele alındığı bu tip bir gerçekleme için yazılan kod ve kullanılan veri tipleri, oldukça fazla bellek kapasitesi ve işlem gücü gerektirmektedir.

Aslında tipik bir PPP bağı kurulumu işlemindeki mesaj alış verişinde, istisnai bir durum olmadıkça hep aynı yol takip edilir. Bu durum PPP kodlamasının basitleştirilebilmesinde kullanılan özelliklerinden biridir. Basitleştirilme için kullanılan diğer bir özellikse, bir PPP bağı için tüm seçeneklerin sorgulanarak kabul edilmesinin, PPP sunucusunu (İnternet Servis Sağlayıcıyı) arayan kullanıcı açısından zorunlu olmamasıdır. Bu durum, normal bir PPP gerçeklemesi için, tüm özellikleri kullanmayan, diğer bir deyişle kısıtlı, bir bağ kurulması anlamına gelse de, düşük işlem kapasiteli bir mikrodenetleyici sistemi için paketlerin değişiminde daha az işlem yükünü sonuç vereceğinden bir avantajdır.

Tipik bir PPP bağı Bağ Denetim Protokolü (Link Control Protocol, LCP) paketlerinin değişilmesiyle başlar ve son bulur. Bir ucun göndermiş olduğu paketteki LCP seçenekleri o ucun bağ üzerinde paket yollarken uyacağı PPP kurallarını ifade etmektedir. PPP protokolünün işletilmesi sürecinin başında, bir Bağ Denetim Protokolü'nün bulunması, PPP protokolünün çok çeşitli fiziksel ortamlar için geliştirilmiş genel bir protokol olmasındandır. Bu seçeneklerin

sorgulanması ve hangilerinin kabul edileceği veri alışverişi için kullanılan donanıma göre değişmektedir.

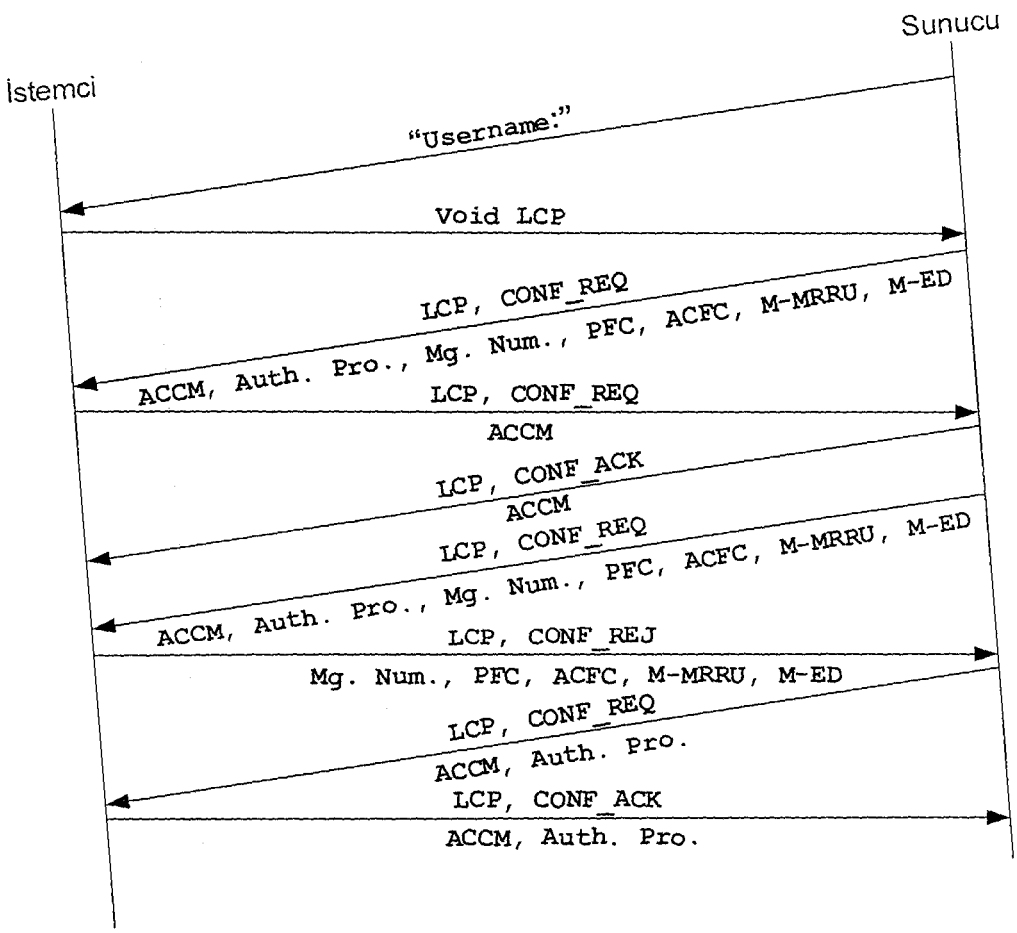
Bir LCP seçeneği herhangi bir LCP Kur – İstek (Configure-Request) paketi ile sorgulanmamışsa bu durum, o seçeneğin varsayılan değerinin geçerli olacağı anlamına gelmektedir. Aynı zamanda gönderilen herhangi bir seçeneğe karşı uç tarafından Kur – Reddet (Configure-Reject) ile yanıt verilmesi, o seçeneğin gerçeklenmediği ve yine varsayılan değerinin ele alınması gerektiğini ifade eder. Varsayılan değerler genellikle PPP'nin her ortamda çalışacak şekilde davranmasını sağlar.

Bu çalışmada PPP protokolünün sadeleştirilmesi sırasında bu iki özellikten yararlanılmıştır. Sorgulamaların başlaması için öncelikle karşı uçtan bir paketin gelmesi beklenmiştir. Karşı uç tarafından gönderilen ilk paket sessizce atılarak gerçeklemenin kendi seçeneklerini öncelikle karşı uca kabul ettirmesi sağlanmıştır. PPP protokolünde, hangi tarafından önce konuşmaya başlaması konusu, standartta belirtilmemektedir. Bu şekilde, istemcinin kendi parametrelerini karşı tarafa ilk olarak iletmesi zorlanmaktadır. LCP sorgulamalarında, karşı uçta PPP sunucu durumunda tam bir PPP gerçeklemesinin bulunduğu varsayımıyla sorgulamalar yürütülmüştür.

Bazı PPP sunucularda, LCP sorgulamaları öncesinde bir kimlik doğrulama aşaması bulunmaktadır. Kullanıcı ismini ve şifresini girdikten sonra LCP konuşmaları başlamaktadır. Kimlik doğrulama işleminin iptal edilip sunucunun öncelikle LCP sorgulamalarına döndürülmesi amacıyla, kullanıcı adı ve şifre istenen küçük paketlerin alınması durumunda boş bir LCP paketi gönderilerek karşı uçtaki PPP sunucunun en başından LCP sorgulamalarına döndürülmesi sağlanmıştır.

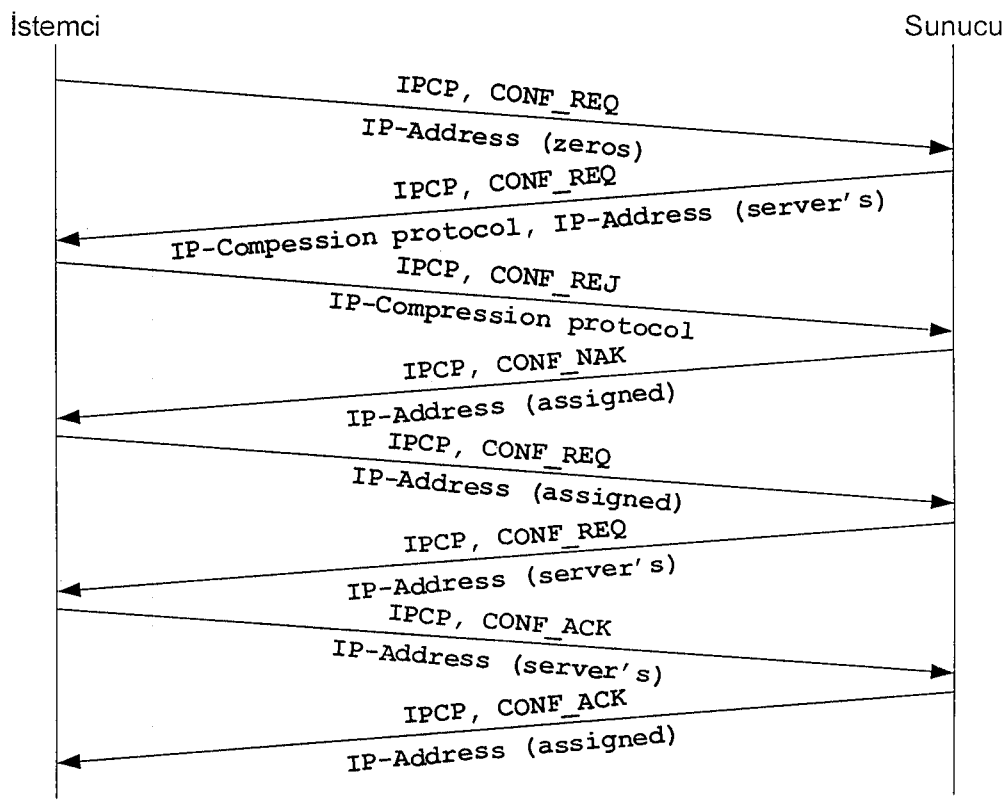
İstemcinin kendi seçeneklerini karşıya göndermesinden sonra karşı uçtaki PPP sunucusundan gelen LCP paketi ele alınmış; gerçeklenmeyen seçeneklere Kur – Reddet (Configure-Reject) paketi ile yanıt verilerek, sunucudan yeniden gönderilecek Kur – İstek (Configure-Request) paketinden, reddedilen seçeneklerin çıkartılması sağlanmıştır. Bu çerçevede, varsayılan değeri kabul edilen seçenek 01 (MRU), genellikle bu çalışmada gerçeklenmeyen bağ bakım paketlerinde kullanılan seçenek 05 (Sihirli Sayı), her biri paketlerde birer baytlık avantaj

sağlayan seçenek 07 (PFC) ve seçenek 08 (ACFC), ve çoklu bağlantılar için kullanılan seçenek 17 (MRRU) ve seçenek 19 (Son Nokta Ayıracı) Kur – Reddet (Configure-Reject) paketi ile reddedilmiştir. Sadece modemler üzerinde veri akışında yazılım kontrolü için kullanılan CTRL-S ve CTRL-Q karakterlerinin kaçış işlemine tabi tutulmasını sağlayacak şekilde seçenek 02 (ACCM) ve bütün ISP'lerle bağlantı kurulumu için gereken seçenek 03 (Doğrulama Protokolü) kabul edilmiştir. Bu seçenekler dışındaki PPP seçenekleri ise genel olarak sorgulanmayan seçenekler olduğundan, farklı İnternet servis sağlayıcıların aranması sırasında karşılaşılmamıştır. PPP protokolünde doğrulama amacıyla kullanılan PAP ve CHAP protokollerinden de gerçeklemesinde hiçbir şifreleme prosedürü gerektirmeyen ve ISP'lerin de çoğu tarafından tercih edilen PAP protokolü tercih edilmiştir. Gerçeklenen bu LCP sorgulama yordamı Şekil 4.4'te gösterilmiştir.



Şekil 4.4. Gerçeklenen LCP sorgulamaları

En gerekli seçeneklerin kabul edilmesi ile bitirilen LCP sorgulamalarından sonra, kullanıcı adı ve şifre içeren bir doğrulama isteği paketi gönderilerek karşılığında gelecek onay paketinin ardından IP protokolü'nün denetimi için kullanılan bir ağ denetim protokolü (NCP) olan IPCP protokolüne geçilmiştir. Bu protokol aracılığıyla yapılan sorgulamalarda ise seçenek 03 (IP - Adresi) dışında sorgulanan sıkıştırma protokolleri ve DNS (Domain Name System) seçenekleri Kur – Reddet (Configure-Reject) paketleri ile reddedilmiştir. Karşılıklı olarak sadece IPCP seçenek 03'ün onaylanmasından sonra çalışmadaki gerçeklemeye bir IP adresi atanması için dört yollu bir sorgulama yapılmıştır. Bu dört yollu sorgulamanın ilk aşamasında IP adres değeri yerinde sıfır yazılı bir Kur – İstek (Configure-Request) paketi gönderilmiş, Karşılığında alınan Kur – Onaylama (Configure-Nak) paketindeki IP adresinin yazıldığı yeni bir Kur – İstek (Configure-Request) gönderilerek bu IP adresi için Kur – Onayla (Configure-Ack) alınmasıyla bir IP adresi alınmıştır. Şekil 4.5'te bu sorgulamaların bir gösterimi verilmiştir.



Şekil 4.5. Gerçeklenen IPCP sorgulamaları

Kurulan bu PPP bağının sonlandırılması ise TCP veri alışverişinin ardından bir PPP Kapat – İstek (Terminate-Request) paketinin gönderilmesiyle olmuştur. Bu pakete karşılık alınan Kapat – Onayla (Terminate-Ack) paketinin ardından karşı uçtaki PPP gerçeklemesinin kurulan telefon bağını da bitirmesiyle işlem sona erdirilmiştir.

Ayrıca PPP protokolleri ailesinden bu çalışmada gerçeklenmemiş bazı protokolleri içeren paketler alındığında ise bu paketler için Kod – Reddet (Code-Reject) paketi gönderilerek gerçeklenmeye bu protokollerin kurulmakta olan bağ üzerinde kullanılmayacağı ifade edilmiştir.

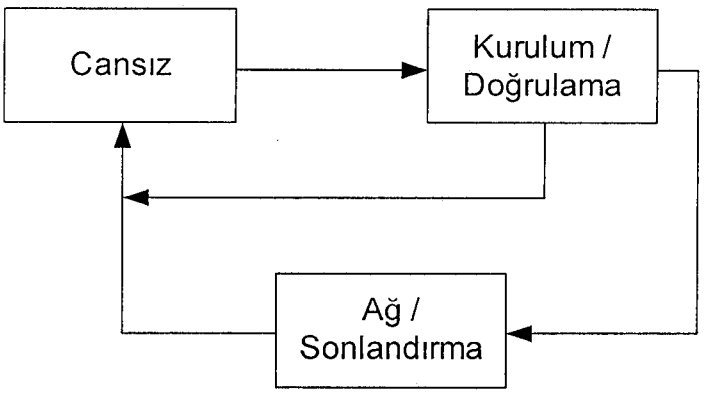
Bahsedilen bu sorgulama ve işlemler, alınan PPP paketlerindeki protokol bilgisine göre uygun fonksiyonların çağırılması Şekil 4.6'de gösterilen kod bloğu aracılığıyla gerçekleştirilmiştir.

```
switch (*(unsigned short*) &PPP_packet [2]) {
case LCP_PACKET:
      // handle LCP
      LCP_Handle ();
      break;
case PAP_PACKET:
      // handle PAP
      // the only possiblity of receiving a pap packet is pap ack
      if (InBuffer [4] = 0x02)
      // upon receipt of a valid a pap ack we send an empty ipcp packet
      // to start the three-way handshake negotiations for our ip addres
      send_first_IPCP ();
      break;
case IPCP_PACKET:
      // handle IPCP
      IPCP_Handle ();
      break;
case IP_DATAGRAM:
      //handle IP
      IP_Receive ((IPDatagram *) &InBuffer [4]);
default:
      break;
}
```

Şekil 4.6. PPP paketlerinin protokol bilgisine göre ele alınması

Bu tasarım ile Şekil 2.4'te gösterilen PPP bağ evrelerinden Kurulum ve Doğrulama evreleri birleştirilerek bu çalışmadaki gerçeklemenin bu şekilde gösterilenden farklı olarak gerek Kurulum gerekse Doğrulama evrelerinin başarısızlığı durumunda *Cansız* evreye geçişi esas tutulmuştur. Aynı zamanda Ağ ve Sonlandırma evreleri de Sonlandırma evresinin üst katman protokolü olan TCP

tarafından tetiklenmesiyle birleştirilmiş ve adı geçen şekil, Şekil 4.7'deki gibi sadeleştirilmiştir.



Şekil 4.7. Sadeleştirilmiş PPP bağ evreleri

Karşı uçtaki PPP sunucu tarafından bu çalışmadaki gerçeklemeye bir IP adresinin atanmasından sonra ise TCP bağlantısı kurulması için gereken paketlerin değişilmesi aşamasına geçilmiştir.

4.4. IP ve ICMP Gerçeklemeleri

Ağ katmanı protokolleri olan IP ve ICMP protokollerinin gerçeklenmesinde de tüm çalışmada olduğu gibi minimum özellikler ele alınmıştır.

IP protokolünün gerçeklemesinde standart bir gerçeklemede bulunan yönlendirme ve parçalama özelliklerinden bu çalışmada sadece yönlendirme özelliği gerçeklenmiştir.

Parçalama özelliği, Donanım katmanındaki ağ yapısına uygun datagramların hazırlanarak Veri Bağı katmanına verilmesi işlevini yürütmektedir. Böyle bir işlev, tasarlanan IP gerçeklemesinin farklı donanımlar üzerinde çalışması şartını yerine getirmesi göz önüne alındığında önemli olduğundan, tüm katmanlardaki protokollerin belli bir kısıtlı platforma uygun olarak basileştirildiği bu çalışmada farklı ağ donanımlarında çalışabilecek esnek bir IP protokolünün gerçeklenmesi gerekli görülmemiştir. Dolayısıyla, gerek TCP protokolünden alınacak verinin belli bir uzunluğu geçmeyeceği; gerekse kullanılacak ağ donanımının özelliklerinin değişmeyeceği varsayımı ile IP protokolünün parçalama özelliğinin

gerçeklenmesinin bu çalışmada gerekmediği ortaya çıkmıştır. IP başlığında da bu özellikle ilgili alanlara bu özelliğin gerçeklenmediği durumlarda geçerli olduğu gibi sıfır değerleri konmuştur.

IP protokolünün yönlendirme özelliği çerçevesinde ise, bir üst katmandan alınan verinin önüne IP başlığının eklenmesi ve bu başlıktaki gerekli alanların doldurularak bir alt katmana verilmesi işlevi esas tutulmuştur. Aslında doldurulması gereken alanlardan protokol ve adres alanları bir üst katman tarafından doldurularak verildiğinden, IP gerçeklemesi tarafından yapılanın sadece başlık denetim toplamını hesaplayarak uygun bir şekilde ilişkili alana yerleştirilmesinden ibaret olduğu söylenebilir.

Başlık denetim toplamı için ise PPP protokolü tarafından kullanılan CRC metodu yerine, RFC 791'de belirtildiği üzere, başlıktaki değerlerin 16-bitin tümleyeni toplamının tümleyeni hesaplanarak bu alana yazılmıştır. Standart gerçeklemelerde IP başlık denetim toplamı hesabı zorunlu olduğundan bu hesaplamanın yapılması kaçınılmazdır. IP başlık denetim toplamının hesaplanması için kullanılan fonksiyon Şekil 4.8'de gösterilmiştir [23].

```
// Calculates the IP header checksum
//
unsigned short IP_GetChecksum (unsigned char *cp, short len) {

        unsigned int sum = 0;

        while (len--) {
                sum += ((unsigned int)((*cp << 8) + *(cp+1)) & 0xFFFF);
                cp += 2;
        }

        sum = (sum >> 16) + (sum & 0xFFFF);
        sum += (sum >> 16);

        return (unsigned short) ~sum;
}
```

Şekil 4.8. IP başlık denetim toplamının hesaplanması

Daha sonra ise hazırlanan bu veri bir alt katman protokolü olan PPP protokolüne verilerek ağ üzerinden gönderilmesi sağlanmıştır.

Ağ üzerinden alınan paketlerin gerekli analizi ve hangi üst katman protokolüne verileceği ise yine IP protokolünün üst katman protokolleri arasında

çoklama (multiplexing) yapma özelliğine uygun olarak gerçekleştirilmiştir. Alınan paketler, IP başlığındaki protokol alanındaki bilgiye göre, Şekil 4.9'da gösterildiği gibi, ICMP veya TCP protokollerinden birine verilmiştir. Standart bir IP gerçeklemesinde bulunan başlık denetim toplamının doğruluğunun test edilmesi ise mevcut iletim hatlarında bit hatalarının oluşmayacağı varsayımıyla, işlem yükü oluşturmaması için ele alınmamıştır.

```
// Receives the IP datagram from the link layer module
// and multiplexes according to the protocol data
//
void IP_Receive (IPDatagram *DatagRecvd) {

        switch (DatagRecvd->Protocol) {
        case ICMP:
                // handle ICMP
                dbg_filePrint ("This datagram contains an ICMP message..\n\n");
                ICMP_Handle (DatagRecvd);
                break;
        case TCP:
                //handle TCP
                dbg_filePrint ("This datagram contains a TCP segment..\n\n");
                TCP_Receive ((TCPSegment *) &InBuffer [24]);
                break;
        }
}
```

Şekil 4.9. Alınan verinin üst katman protokollerine verilmesi

Ayrıca IP protokolü fonksiyonları tarafından gerek alınan, gerekse gönderilecek paketlerin kolaylıkla işlenebilmesi amacıyla Şekil 4.10'da gösterildiği gibi "IPDatagram" isimli bir yapı tanımlanmış ve gerçeklemenin en başında çağırılan "`void IP_Init(void)`" fonksiyonu aracılığıyla IP gerçeklemesinde kullanılacak bu yapıyla tanımlanan işaretçilerin girdi ve çıktı katarlarında uygun yerleri işaret etmeleri sağlanmıştır.

```
typedef struct {
        unsigned char           Ver_IHL;
        unsigned char           TOS;
        unsigned short          Length;
        unsigned short          ID;
        unsigned short          Flag_Frag;
        unsigned char           TTL;
        unsigned char           Protocol;
        unsigned short          Checksum;
        unsigned char           SrcIP [4];
        unsigned char           DstIP [4];
        unsigned char           Payload [1480];
} IPDatagram;
```

Şekil 4.10. IP Datagram yapısı

ICMP protokolünün gerçeklenmesinde ise bu protokolün mesaj tipleri yerine sadece tüm IP protokolü gerçeklemelerinde gerçeklenmesi zorunlu olan Yankı İsteği ve Yankı Yanıtı mesajları gerçeklenmiştir. Bu protokoldeki diğer mesaj tipleri ise her kullanıcı için gerekli olmadığından yapılan gerçeklemenin daha az yer tutması açısından ele alınmamıştır.

Yankı isteği mesajının kullanımı derleme sırasında belirlenebilmekte; ve gereksiz görüldüğü durumlarda gerçeklemeden çıkartılabilmektedir. Yankı isteği mesajları hazırlanırken ICMP başlık bilgilerinin yerleştirilerek paketin iletim için IP protokolüne verilmesi sağlanmıştır. Alınan ve gönderilen mesajların eşleştirilmesi için kullanılan tanıtıcı değeri için belli bir sabit değer (`0x271D`) derleme anında yerleştirilmiştir. Seri numarası için de yine derleme anında belli bir sabit değer (`0x02FF`) yerleştirilmiş, fakat protokol gereği her ICMP yankı isteği mesajı için bir artırılması sağlanmıştır. Denetim toplamı ise IP başlığındaki başlık denetim toplamıyla aynı algoritmayla hesaplandığından Şekil 4.8'de verilen fonksiyon kullanılarak hesaplanmıştır. Veri alanı ise seçime bağlı olduğundan hiç kullanılmamıştır.

Alınan bir ICMP mesajının ise yankı isteği mesajı olması durumunda mesaj aynen çıktı katarına kopyalanmış; başlıktaki tip alanı değiştirilip yeni mesaj için denetim toplamı hesaplanarak hazırlanan yankı yanıtı mesajı IP protokolüne verilmiş ve ağ üzerinden iletimi sağlanmıştır. Eğer alınan mesaj bir yankı yanıtı mesajı ise sadece seri numarası alanına bakılarak gönderilen bir yankı isteği mesajı için alınıp alınmadığı denetlenmiştir.

4.5. TCP Gerçeklemesi

Bu çalışmanın asıl hedefi olan TCP protokolünün gerçeklenmesinde, bu protokolün iki önemli özelliği olan bağlantılılık (connection-oriented) ve güvenilirlilik (reliability) göz önünde tutulmuştur. Çünkü bir TCP gerçeklemesinin bu iki özellik olmaksızın farklı TCP gerçeklemeleriyle karşılıklı olarak işlem yapılmasına olanak sağlanması mümkün değildir. Tipik bir TCP gerçeklemesinde olduğu gibi bu çalışmada ele alınan gerçeklemede de bağlantılılık, karşılıklı üç-yollu el sıkışma şeklinde yapılan SYN parçalarının değişilmesi ve ACK parçalarıyla onaylanması yoluyla gerçekleştirilmiştir. Aynı zamanda kurulan bağlantının bitirilmesi de FIN mesajlarının karşılıklı değişilmesi ve onaylanması yoluyla gerçeklenmiştir. Bağlantının kurulması ve bitirilmesi aynen Şekil 3.7 ve Şekil 3.8 de gösterildiği gibi sade bir yaklaşımla ele alınmıştır.

Güvenilirlilik ise, hem alınan veri parçalarının uygun ACK parçalarıyla onaylanması, hem de gönderilen veriler için ACK parçalarının alınıp alınmadığının denetimi yapılması suretiyle gerçeklenmiştir.

İstemci tipi bir TCP tasarımının ele alındığı bu gerçeklemede, sadece sunucu gerçeklemeleri için geçerli olan LISTEN evresi hiç düşünülmemiş ve gerçeklemenin Veri Bağı katmanındaki PPP bağı kurulduktan sonra doğrudan IP adresi derleme sırasında belirlenmiş olan bir TCP sunucu ile bağlantı kurması sağlanmıştır. Bu amaçla PPP bağının kurulumundan sonra, bir IP adresinin alınmasıyla birlikte "`void TCP_Conn (unsigned char FLAGStoSET)`" fonksiyonu "`SYN`" parametresi ile çağırılarak bir SYN parçasının belirlenen sunucuya gönderilmesi sağlanmıştır. Sunucudan, gönderilen SYN parçasını onaylayan, hem de bağlantı için sunucunun isteğini belirten ACK ve SYN bayrakları set edilmiş bir parça alınıp karşılığında onay için bir ACK gönderilmesiyle TCP bağlantısının kurulması sağlanmıştır. Bu prosedür sonucunda başlangıçta CLOSED evresinde olan gerçekleme, SYN_SENT evresinden geçirilerek veri alışverişi için hazır olduğu ESTABLISHED evresine getirilmiştir. Bağlantı kurulum prosedürünün daha karmaşık bir yapıda olmaması için, SYN_SENT evresinde alınacak sadece SYN bayrağı set edilmiş bir TCP paketinin işleme konmaması sağlanmıştır.

Gereken veri alışverişi yapıldıktan sonra ise yine kurulan bağlantının sonlandırılması aşamasına geçilmiştir. Bağlantıyı sonlandırma yordamı, sade bir gerçekleme için, bağlantı kurulan TCP sunucunun uygulamasına göre farklılık gösterebilecektir. Çünkü TCP bağlantısı kurulan kimi gerçeklemelerin veri alışverişinin hemen ardından bir FIN parçası gönderdiği, kimi gerçeklemelerin ise FIN parçasını karşıdan gönderilmesini beklediği gözlenmiştir. İlk FIN parçasının karşı uç veya bu çalışmadaki gerçekleme tarafından gönderilmesi durumuna göre TCP bağlantısının sonlandırılma işlemi FIN_WAIT_1 veya CLOSE WAIT evlerine geçerek sürdürülecektir. TCP bağlantısının kurulması ve sonlandırılması işlemi için kullanılan "`void TCP_Conn (unsigned char FLAGStoSET)`" fonksiyonunun girdi parametresine göre yaptığı işlem Şekil 4.11'de gösterilmiştir.

```
switch (FLAGStoSET) {
case SYN:
        IP_2Send->Length = htons (0x002C);
        TCP_2Send->dataOFF = 0x60;
        TCP_2Send->ctrlBITS = SYN;
        TCP_2Send->data [0] = 0x02;
        TCP_2Send->data [1] = 0x04;
        TCP_2Send->data [2] = 0x05;
        TCP_2Send->data [3] = 0xB4;
        chksm_buff = TCP_GetChecksum ( (unsigned char *) &(IP_2Send [0]) );
        TCP_2Send->chksm = htons (chksm_buff);
        conn.state = SYN_SENT;
        break;
case ACK:
        IP_2Send->Length = htons (0x0028);
        TCP_2Send->dataOFF = 0x50;
        TCP_2Send->ctrlBITS = ACK;
        chksm_buff = TCP_GetChecksum ( (unsigned char *) &(IP_2Send [0]) );
        TCP_2Send->chksm = htons (chksm_buff);
        break;
case PSH:
        break;
case FIN:
        IP_2Send->Length = htons (0x0028);
        TCP_2Send->dataOFF = 0x50;
        TCP_2Send->ctrlBITS = (FIN | ACK);
        chksm_buff = TCP_GetChecksum ( (unsigned char *) &(IP_2Send [0]) );
        TCP_2Send->chksm = htons (chksm_buff);
        if (conn.state == ESTABLISHED)
                conn.state = FIN_WAIT_1;
        else if (conn.state = CLOSE_WAIT)
                conn.state = LAST_ACK;
        break;
}
```

Şekil 4.11. TCP_Conn fonksiyonunun yapısı

TCP bağlantısının kurulmasından sonra herhangi bir veri alışverişi için hazır olan gerçekleme, Uygulama katmanı protokollerinden biri olan HTTP (Hyper Text Transfer Protocol) üzerinden test edilmiştir. IP adresi bilinen bir TCP sunucuya, 80 numaralı porttan "HTTP GET" komutu gönderilmiştir. Bu şekilde sunucudaki varsayılan web sayfası istenmiştir.

Standart bir TCP gerçeklemesinde kurulan TCP bağlantısına ait bir kısım durum verilerinin tutan İletim Denetim Bloğu (Transmission Control Block, TCB) bu gerçeklemede de gerçeklemenin durumunu, gönderilen ve alınan parçalarla güncellenen seri ve onay numaralarını, kurulan bağlantının port numaralarını bağlantı süresince saklayacak şekilde derleme anında belirlenerek veri alanına yerleştirilmesi sağlanmıştır. Bu çalışma için kullanılan TCB yapısı Şekil 4.12.'de verilmiştir.

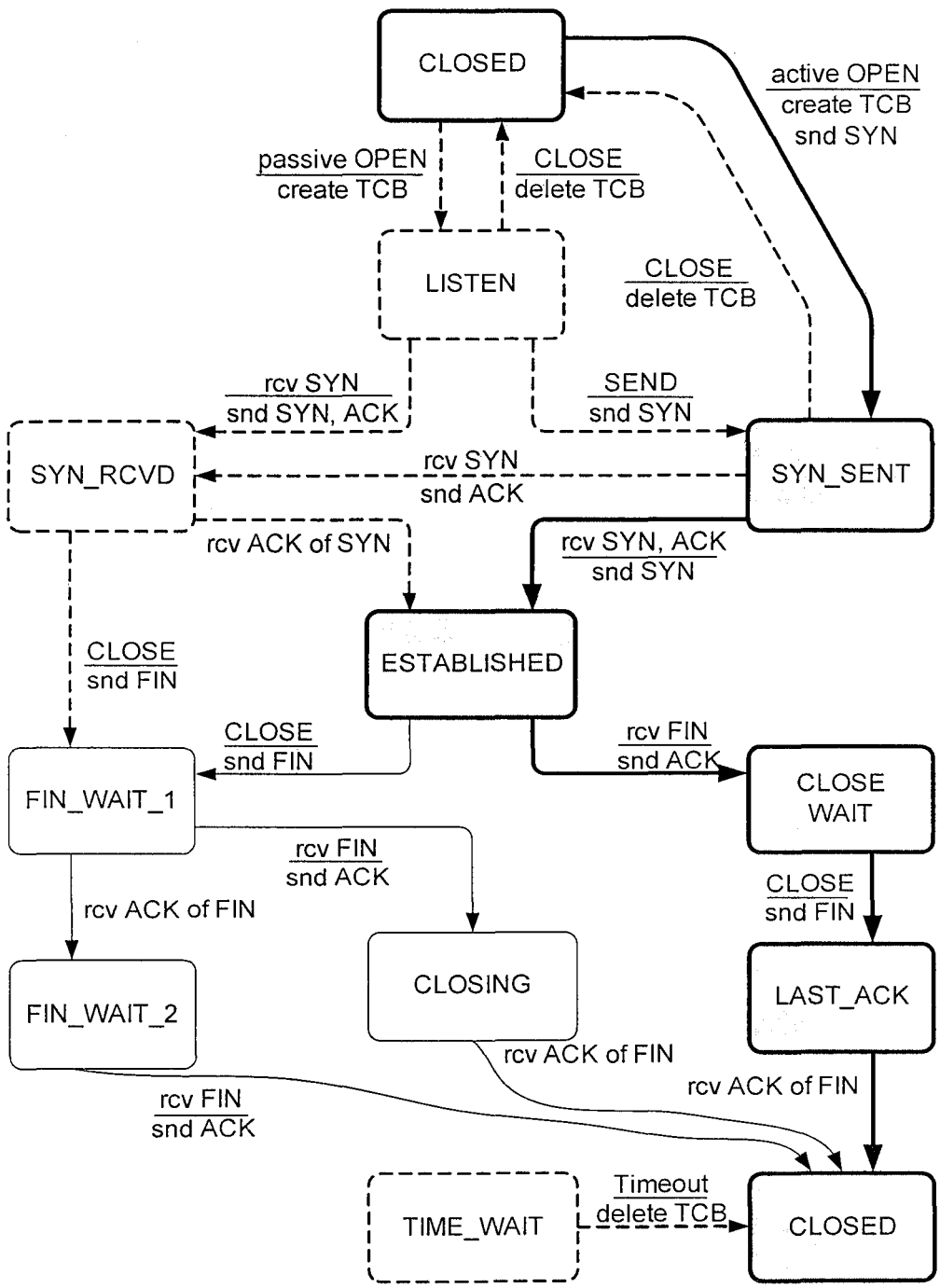
```
struct TCPTCB {
        unsigned short          lPort;
        unsigned short          rPort;
        unsigned char           rIP [4];
        unsigned char           state;
        unsigned int            lSeq;
        unsigned int            rSeq;
};
```

Şekil 4.12. Gerçeklemede kullanılan TCB yapısı

Bu TCB yapısındaki karşıdan gelen verilere bağlı olmayan ilk seri numarası (ISN) ve yerel port numarası için gerçeklemenin en başında "`void TCP_Init(void)`" fonksiyonunun içerisinde derleme anında verilen sabit ilk değerler atanmıştır. Normal bir TCP gerçeklemesinde ise, hem istemci uçta hem de sunucu ucunda bir anda birden fazla TCP bağlantısı başlatılabileceğinden, ISN ve yerel port numarasına, her bağlantı için farklı değerleri alabilmesi amacıyla, sistem saati gibi sürekli güncellenen bir kaynak yardımıyla üretilen rassal değerlerin atanması sağlanmaktadır. Bu yaklaşımın, sadece bir tane TCP bağlantısının düşünüldüğü bu çalışmada, mikrodenetleyici sisteminin düşük işlem hızı kısıtı da gözönüne alındığında gereksiz olacağı açıktır.

Bu çalışmada gerçeklenen TCP protokolü kullanılarak bir HTTP sunucu ile TCP bağlantısının kurulması ve veri alışverişi sonunda HTTP sunucunun FIN

parçası göndermesiyle kurulan bağlantının sonlandırılması, Şekil 4.13'teki TCP akış şemasında kalın çizgilerle gösterilmiştir. Adı geçen şekildeki kesikli çizgiler ise, bu çalışmanın amacı dışında, bir TCP sunucu için gerçeklenmesi gereken akışları göstermektedir. İnce çizgilerle gösterilen akışlar ise, TCP bağlantısının sunucu tarafından değil de istemci tarafından başlatıldığı durumda gerçekleştirilmektedir.



Şekil 4.13. Gerçeklenen TCP bağlantı akışı

Gönderilen tüm TCP parçalarının başlık alanlarındaki denetim toplamı değerinin hesaplanması ise yine IP bağlık denetim toplamımın hesaplanmasında kullanılan Şekil 4.8'de verilen fonksiyon kullanarak, TCP bölümü ve Şekil 3.6'da gösterilen bir Yalancı TCP başlığı üzerinden 16-bitin tümleyeni toplamının tümleyeni yoluyla hesaplanmıştır. TCP parçalarında denetim toplamı hesaplanırken Yalancı TCP başlığının da göz önüne alınması RFC 793'te belirtilen bir gerekliliktir [25].

TCP bağlantısında gelen ve gönderilecek parçalara gereken fonksiyonlar üzerinden kolaylıkla ulaşılabilmesi amacıyla gerçeklemenin en başında girdi ve çıktı katarlarının uygun yerlerini göstermeleri sağlanan global yapı işaretçileri Şekil 4.14te gösterilen "TCPSegment" tipinde tanımlanmıştır.

```
typedef struct {
        unsigned short          srcPort;
        unsigned short          dstPort;
        unsigned int            seqNO;
        unsigned int            ackNO;
        unsigned char           dataOFF;
        unsigned char           ctrlBITS;
        unsigned short          wnd;
        unsigned short          chksm;
        unsigned short          urgPNT;
        unsigned char           data [1460];
} TCPSegment;
```

Şekil 4.14 TCPSegment yapısı

5. SONUÇ

Gömülü sistemlerde çalışan PPP/TCP/IP uygulamaları, genellikle açık kaynak kodu olarak gerçeklemelere dayanmaktadır. BSD Unix ve Linux işletim sistemlerinde çalışan yığıt uygulamaları, gömülü sistemlere uyarlanmaktadır. Her ortamda çalışması için tasarlanan bu yığıt uygulamaları, gömülü sistemlerde kullanılmayan pek çok özelliği de bünyesinde barındırmaktadır. Bu uygulamalarda, veri yapılarının karmaşık olması bir yana, standartlara bire bir uyumlulukları gerektiğinden, protokollerin bütün seçenekleri gerçeklemeye eklenmiştir.

Örneğin, IP katmanının temel işlevi, gönderilecek bir veri paketinin başına IP başlığını koyan, ardına da veri paketini ekleyen basit bir yordamdır. En fazla 30 satır alan yordam, basit bir uygulamada gerekmeyecek parçalama gibi özelliklerin de eklenmesiyle rahatlıkla, 500 satıra erişebilir.

Bu tez çalışmasında, PPP/TCP/IP protokol yığıtı, basit bir veri alış verişi için gerekli özellikleriyle birlikte gerçeklenmiştir. Bu yaklaşımla, genellikle büyük kod alanı gerektiren İnternet uygulamalarının, kısıtlı bellek içeren gömülü sistemlerde çalıştırılabilmesi amaçlanmıştır.

Bu kısıtlı gerçekleme yaklaşımıyla, çeşitli ISP'lere bağlantıda sorunlar çıkması beklenebilir. Bununla birlikte, bu çalışmada, bir PPP bağının kurulması için yapılan LCP, PAP ve IPCP sorgulamalarında, alınan bir ICMP Yankı İsteği mesajı için Yankı Yanıtı mesajı gönderilmesinde veya derleme anında belirlenen bir TCP sunucudan uygulama verisi yüklenmesi sırasında hiç bir başarısızlıkla karşılaşılmamıştır. Türkiye'de faaliyet gösteren, ulusal çapta çalışan ISP'ler olan, TTNET, Superonline, E-KOLAY gibi ISP'lerin herbiriyle başarılı PPP bağları kurulmuştur.

TCP/IP protokol yığıtındaki protokollerin sadeleştirilerek tasarlanması, genel anlamda, gerek PPP bağı gerekse TCP bağlantısı kurulması sırasında standart iki farklı gerçekleme arasında izlenen prosedür ele alınacak şekilde yapıldığından tasarımın çalışmasında herhangi bir olumsuzlukla karşılaşılması pek olası değildir. Fakat PPP bağı yahut TCP bağlantısı kurulması sırasında gerçeklenmemiş bileşenler tarafından ele alınacak herhangi bir olumsuzlukla

karşılaşılması durumunda sistemin kendini yeniden başlatarak prosedürün en baştan ele alınmasını sağlamak yeterli bir çözüm olacaktır.

Bu çalışmada gerçeklenen, gömülü sistemlerde çalışabilen PPP/TCP/IP protokol yığıtı, çeşitli sensör verilerinin çevrim içi şekilde izlenmesi, mobil cihazların GPS konum bilgilerinin GPRS üzerinden aktarılması, alarm sistemleri gibi uygulamalarda kullanılabilir.

Gerçeklenen PPP/TCP/IP yığıtı, Motorola 68HC812A4 mikroişlemcisinde, 12 KB kod alanı ve 4 KB RAM alanına gerek duymaktadır. Diğer 8 bit'lik mikroişlemcilerde benzer bir hafıza gereksinimi beklenebilir.

Bu çalışmada gerçeklenen tasarımın çalıştırılması sonucu yapılan veri alışverişinin kayıtları EK-1'de, tüm projenin kodu ise EK-2'de verilmiştir.

KAYNAKLAR

[1] STEVENS W. R., *TCP/IP Illustrated,* **Vol. I**, The Protocols, Addison-Wesley, Massachusetts, USA (1999).

[2] CARLSON J., *PPP Design, Implementation and Debugging,* Addison-Wesley, New Jersey, USA (2000).

[3] Net-linking MCUs requires trade-offs, http://www.eetimes.com/story/OEG20010521S0062 (2001).

[4] ALP U., AYAZ H., KARADENİZ M., DİKİCİ Ç. ve BOZMA H.I., *İnternet Üzerinden Uzakta Robot Erişimi,* 11. Sinyal İşleme ve İletişim Uygulamaları Kurultayı Bildiriler Kitabı (Ed: A. Murat TEKALP), Koç Üniversitesi, İstanbul, Türkiye, 540-543 (2003).

[5] CHONG C.Y. ve KUMAR S.P., *Sensor Networks: Evolution, Opportunities, and Challenges,* Proceedings of the IEEE, **91**, **8**, 1247-1255 (2003).

[6] TRENADO R., *Connecting an M68HC08 Family Microcontroller to an Internet Service Provider (ISP) Using the Point-to-Point Protocol (PPP),* Motorola Semiconductor Products Sector Application Note No: AN2120, (2001).

[7] Bilgisayar Terimleri Karşılıklar Kılavuzu, http://www.tdk.gov.tr/bilterim/.

[8] PERKINS D., *RFC 1547, Requiremets foran Internet Standard Point-to-Point Protocol,* Internet Engineering Task Force, (1993).

[9] SIMPSON W., *RFC 1661, The Point-to-Point Protocol (PPP),* Internet Engineering Task Force, (1994).

[10] SIMPSON W., *RFC 1598, PPP in X.25,* Internet Engineering Task Force, (1994).

[11] SIMPSON W., *RFC 1618, PPP over ISDN,* Internet Engineering Task Force, (1994).

[12] SIMPSON W., *RFC 1973, PPP in Frame Relay,* Internet Engineering Task Force, (1996).

[13] GROSS G., KAYCEE M., LI A., MALIS A. ve STEPHENS J., *RFC 2363, PPP over FUNI,* Internet Engineering Task Force, (1998).

[14] GROSS G., KAYCEE M., LI A., MALIS A. ve STEPHENS J., *RFC 2364, PPP over AAL5,* Internet Engineering Task Force, (1998).

[15] MAMAKOS L., LIDL K., EVARTS J., CARREL D., SIMONE D. ve WHEELER R., *RFC 2516, A Method for Transmitting PPP Over Ethernet (PPPoE),* Internet Engineering Task Force, (1998).

[16] SIMPSON W., *RFC 1662, PPP in HDLC-like Framing,* Internet Engineering Task Force, (1994).

[17] SIMPSON W., *RFC 1661, The Point-to Point Protocol (PPP),* Internet Engineering Task Force, (1994).

[18] REYNOLDS J., POSTEL J., *RFC 1700, Assigned Numbers*, Internet Engineering Task Force, (1994).

[19] SIMPSON W., *RFC 1548, The Point-to Point Protocol (PPP),* Internet Engineering Task Force, (1993).

[20] LLOYD B. ve SIMPSON W., *RFC 1334, PPP Authentication Protocols,* Internet Engineering Task Force, (1992).

[21] SIMPSON W., *RFC 1994, PPP Challenge Handshake Authentication Protocol (CHAP),* Internet Engineering Task Force, (1996).

[22] BLUNK L. ve VOLLBRECHT J., *RFC 2284, Extensible Authentication Protocols (EAP),* Internet Engineering Task Force, (1992).

[23] POSTEL J., *RFC 791, Internet Protocol (IP),* Internet Engineering Task Force, (1981).

[24] POSTEL J., *RFC 792, Internet Control Message Protocol (ICMP),* Internet Engineering Task Force, (1981).

[25] POSTEL J., *RFC 793, Transmission Control Protocol (TCP),* Internet Engineering Task Force, (1981).

[26] ImageCraft Embedded C Development Tools, http://www.imagecraft.com/software/ (2004).

[27] Adapt812DXLT MCU Module, http://www.technologicalarts.com/myfiles/812dxlt.html (2004).

[28] PACK D.J., BARRET S.F., *68HC12 Microcontroller, Theory and Applications,* Prentice-Hall, New Jersey, USA (2002).

EK – 1 Kayıtlar

```
Serial port COM1 is opening...
COM1 was opened succesfully...
Press any key to proceed with the necessary modem
initialization...
Success in modem initialization...
Press any key to dial the ISP...
Connection with the remote ISP was established...
TCP module was initialized...
IP module was initialized...
PPP module was initialized... i.e. ReSync status bit is set.
Polling the serial link for the PPP frame...
Polling the serial link for the PPP frame...
Polling the serial link for the PPP frame...
read character is: 3A     :
We send a void lcp packet to force the isp to the
negotiatioins...
Our packet to be sent...
FF 03 C0 21 01 01 00 04 D1 B5
Polling the serial link for the PPP frame...
Polling the serial link for the PPP frame...
Polling the serial link for the PPP frame...
read character is: 3A     :
We send a void lcp packet to force the isp to the
negotiatioins...
Our packet to be sent...
FF 03 C0 21 01 01 00 04 D1 B5
Polling the serial link for the PPP frame...
Polling the serial link for the PPP frame...
END flag is received... Buffering started...
END flag is received. Receipt of a frame is completed...
50 characters were received...
```

```
08 20 08 5E FF 03 C0 21 01 FB 00 2A 02 06 00 0A 00 00 03 04
C0 23 05 06 EF FD 00 3B 07 02 08 02 11 04 05 F4 13 0E 01 75
73 65 72 33 31 34 34 30 31 36
We drop the first received packet and send our first lcp
options...
Our packet to be sent...
FF 03 C0 21 01 00 00 0A 02 06 00 0A 00 00 DF 45
Polling the serial link for the PPP frame...
Polling the serial link for the PPP frame...
Polling the serial link for the PPP frame...
END flag is received... Buffering started...
END flag is received. Receipt of a frame is completed...
16 characters were received...
FF 03 C0 21 02 00 00 0A 02 06 00 0A 00 00 B6 31
This is an LCP packet...
LCP options will be handled...
All of the options we negotiate were acked...
Polling the serial link for the PPP frame...
END flag is received... Buffering started...
END flag is received. Receipt of a frame is completed...
48 characters were received...
FF 03 C0 21 01 FC 00 2A 02 06 00 0A 00 00 03 04 C0 23 05 06
EF FD 00 3B 07 02 08 02 11 04 05 F4 13 0E 01 75 73 65 72 33
31 34 34 30 31 36 70 21
This is an LCP packet...
LCP options will be handled...
Our packet to be sent...
FF 03 C0 21 04 FC 00 20 05 06 EF FD 00 3B 07 02 08 02 11 04
05 F4 13 0E 01 75 73 65 72 33 31 34 34 30 31 36 E6 51

Polling the serial link for the PPP frame...
END flag is received... Buffering started...
END flag is received. Receipt of a frame is completed...
20 characters were received...
FF 03 C0 21 01 FD 00 0E 02 06 00 0A 00 00 03 04 C0 23 AC 3C
This is an LCP packet...
```

```
LCP options will be handled...
Our packet to be sent...
FF 03 C0 21 02 FD 00 0E 02 06 00 0A 00 00 03 04 C0 23 92 BF
Our packet to be sent...
FF 03 C0 23 01 01 00 1C 10 39 30 39 31 34 39 30 35 39 37 30
37 30 31 33 39 06 36 36 31 35 34 31 83 55
Polling the serial link for the PPP frame...
END flag is received... Buffering started...
END flag is received. Receipt of a frame is completed...
11 characters were received...
FF 03 C0 23 02 01 00 05 00 8B 3B
This is a PAP packet...
Our username and password are authenticated...
We send an empty ipcp packet...
Our packet to be sent...
FF 03 80 21 01 01 00 0A 03 06 00 00 00 00 13 28
Polling the serial link for the PPP frame...
END flag is received... Buffering started...
END flag is received. Receipt of a frame is completed...
22 characters were received...
FF 03 80 21 01 EA 00 10 02 06 00 2D 0F 00 03 06 D9 83 48 FE
2E 12
This is an IPCP packet...
Our packet to be sent...
FF 03 80 21 04 EA 00 0A 02 06 00 2D 0F 00 4F B3
Polling the serial link for the PPP frame...
END flag is received... Buffering started...
END flag is received. Receipt of a frame is completed...
16 characters were received...
FF 03 80 21 03 01 00 0A 03 06 D9 83 48 A7 DD CB
This is an IPCP packet...
Our packet to be sent...
FF 03 80 21 01 02 00 0A 03 06 D9 83 48 A7 94 45
Polling the serial link for the PPP frame...
END flag is received... Buffering started...
END flag is received. Receipt of a frame is completed...
```

16 characters were received...
FF 03 80 21 01 EB 00 0A 03 06 D9 83 48 FE 6A 6D
This is an IPCP packet...
Our packet to be sent...
FF 03 80 21 02 EB 00 0A 03 06 D9 83 48 FE 03 19
Polling the serial link for the PPP frame...
END flag is received... Buffering started...
END flag is received. Receipt of a frame is completed...
16 characters were received...
FF 03 80 21 02 02 00 0A 03 06 D9 83 48 A7 FD 31
This is an IPCP packet...
The IP address of the remote ISP: 217.131.72.254
Our IP adress: 217.131.72.167
LinkOn flag was set...
We call TCP connection management function to establish a connection...
A SYN will be sent...
Our packet to be sent...
FF 03 00 21 45 00 00 2C 00 0A 00 00 80 06 46 DE D9 83 48 A7 C1 8C 10 2D 0C 2C 00 50 E7 73 42 37 00 00 00 00 60 02 05 B4 68 67 00 00 02 04 05 B4 EC 79
if statement inside the while was executed...
Polling the serial link for the PPP frame...
END flag is received... Buffering started...
END flag is received. Receipt of a frame is completed...
50 characters were received...
FF 03 00 21 45 00 00 2C D8 7E 40 00 3E 06 70 69 C1 8C 10 2D D9 83 48 A7 00 50 0C 2C F7 29 8B 91 E7 73 42 38 60 12 22 38 C9 16 00 00 02 04 05 B4 7F 2D
This is an IP datagram...
This datagram contains a TCP segment...
We received SYN + ACK from the remote server...
We will send an ACK to perform the last operation of the three way handshake...
An ACK will be sent...
Our packet to be sent...

```
FF 03 00 21 45 00 00 28 00 0B 00 00 80 06 46 E1 D9 83 48 A7
C1 8C 10 2D 0C 2C 00 50 E7 73 42 38 F7 29 8B 92 50 10 05 B4
FD 57 00 00 E5 59
if statement inside the while was executed...
Now the connection is in state ESTABLISHED...
GET / HTTP/1.0
TCP header is filled...
Data is copied to be the payload...
Our packet to be sent...
FF 03 00 21 45 00 00 3A 00 0C 00 00 80 06 46 CE D9 83 48 A7
C1 8C 10 2D 0C 2C 00 50 E7 73 42 38 F7 29 8B 92 50 18 05 B4
1E 9E 00 00 47 45 54 20 2F 20 48 54 54 50 2F 31 2E 30 0D 0A
0D 0A 6B 5E
Polling the serial link for the PPP frame...
END flag is received... Buffering started...
END flag is received. Receipt of a frame is completed...
46 characters were received...
FF 03 00 21 45 00 00 28 DB 7E 00 00 3E 06 AD 6D C1 8C 10 2D
D9 83 48 A7 00 50 0C 2C F7 29 8B 92 E7 73 42 4A 50 10 40 00
C2 F9 00 00 23 DE
This is an IP datagram...
This datagram contains a TCP segment...
We received an ACK from the remote server...
This is an ACK for the data that we sent...
if statement inside the while was executed...
Now the connection is in state ESTABLISHED...
Polling the serial link for the PPP frame...
END flag is received... Buffering started...
END flag is received. Receipt of a frame is completed...
906 characters were received...
FF 03 00 21 45 00 03 84 DC 7E 00 00 3E 06 A9 11 C1 8C 10 2D
D9 83 48 A7 00 50 0C 2C F7 29 8B 92 E7 73 42 4A 50 18 40 00
E9 6F 00 00 48 54 54 50 2F 31 2E 30 20 32 30 30 20 4F 4B 0D
0A 41 67 65 3A 20 32 35 37 33 34 38 0D 0A 41 63 63 65 70 74
2D 52 61 6E 67 65 73 3A 20 62 79 74 65 73 0D 0A 44 61 74 65
3A 20 54 75 65 2C 20 30 32 20 4E 6F 76 20 32 30 30 34 20 30
```

38 3A 32 31 3A 34 32 20 47 4D 54 0D 0A 43 6F 6E 74 65 6E 74
2D 4C 65 6E 67 74 68 3A 20 34 39 37 0D 0A 43 6F 6E 74 65 6E
74 2D 54 79 70 65 3A 20 74 65 78 74 2F 68 74 6D 6C 0D 0A 43
6F 6E 74 65 6E 74 2D 4C 6F 63 61 74 69 6F 6E 3A 20 68 74 74
70 3A 2F 2F 31 39 33 2E 31 34 30 2E 31 36 2E 34 35 2F 44 65
66 61 75 6C 74 2E 68 74 6D 0D 0A 4C 61 73 74 2D 4D 6F 64 69
66 69 65 64 3A 20 54 68 75 2C 20 32 35 20 44 65 63 20 32 30
30 33 20 30 38 3A 33 35 3A 32 38 20 47 4D 54 0D 0A 45 54 61
67 3A 20 22 38 63 38 65 64 62 63 63 32 63 61 63 33 31 3A 66
32 66 22 0D 0A 53 65 72 76 65 72 3A 20 4D 69 63 72 6F 73 6F
66 74 2D 49 49 53 2F 36 2E 30 0D 0A 58 2D 50 6F 77 65 72 65
64 2D 42 79 3A 20 41 53 50 2E 4E 45 54 0D 0A 56 69 61 3A 20
31 2E 31 20 73 67 74 2D 6E 65 74 63 61 63 68 65 2D 30 32 20
28 4E 65 74 43 61 63 68 65 20 4E 65 74 41 70 70 2F 35 2E 35
52 32 29 0D 0A 0D 0A 3C 48 54 4D 4C 3E 0D 0A 0D 0A 3C 48 45
41 44 3E 0D 0A 3C 54 49 54 4C 45 3E 41 6E 61 64 6F 6C 75 20
DC 6E 69 76 65 72 73 69 74 65 73 69 20 42 69 6C 67 69 73 61
79 61 72 20 4D FC 68 65 6E 64 69 73 6C 69 3F 69 3C 2F 54 49
54 4C 45 3E 0D 0A 3C 2F 48 45 41 44 3E 0D 0A 0D 0A 3C 46 52
41 4D 45 53 45 54 20 63 6F 6C 73 20 3D 20 22 31 38 30 2C 2A
22 20 42 4F 52 44 45 52 3D 30 20 66 72 61 6D 65 62 6F 72 64
65 72 3D 22 30 22 3E 0D 0A 0D 0A 3C 46 52 41 4D 45 20 53 52
43 20 3D 20 22 73 6F 6C 2E 68 74 6D 6C 22 20 4E 4F 52 45 53
49 5A 45 20 53 43 52 4F 4C 4C 49 4E 47 3D 22 4E 4F 22 20 74
61 72 67 65 74 3D 22 6D 61 69 6E 22 20 6D 61 72 67 69 6E 77
69 64 74 68 3D 22 30 22 20 6D 61 72 67 69 6E 68 65 69 67 68
74 3D 22 30 22 3E 0D 0A 3C 46 52 41 4D 45 20 53 52 43 20 3D
20 22 64 75 79 75 72 75 6C 61 72 2E 68 74 6D 6C 22 20 4E 41
4D 45 3D 22 6D 61 69 6E 22 20 73 63 72 6F 6C 6C 69 6E 67 3D
22 61 75 74 6F 22 3E 0D 0A 3C 6E 6F 66 72 61 6D 65 73 3E 0D
0A 3C 70 20 61 6C 69 67 6E 3D 22 63 65 6E 74 65 72 22 3E 3C
66 6F 6E 74 20 66 61 63 65 3D 22 54 61 68 6F 6D 61 22 20 73
69 7A 65 3D 22 32 22 3E 3C 61 20 68 72 65 66 3D 22 64 75 79
75 72 75 6C 61 72 2E 68 74 6D 6C 22 3E 42 75 72 61 79 61 3C
2F 61 3E 20 74 FD 6B 6C 61 79 61 72 61 6B 0D 0A 73 61 79 66
61 6D FD 7A FD 20 7A 69 79 61 72 65 74 65 20 62 61 FE 6C 61

```
79 61 62 69 6C 69 72 73 69 6E 69 7A 2E 3C 2F 66 6F 6E 74 3E
3C 2F 70 3E 0D 0A 3C 2F 6E 6F 66 72 61 6D 65 73 3E 0D 0A 3C
2F 46 52 41 4D 45 53 45 54 3E 0D 0A 0D 0A 3C 2F 48 54 4D 4C
3E 20 0D 0A AD 9C
This is an IP datagram...
This datagram contains a TCP segment...
if statement inside the while was executed...
Now the connection is in state ESTABLISHED...
Polling the serial link for the PPP frame...
END flag is received... Buffering started...
END flag is received. Receipt of a frame is completed...
46 characters were received...
FF 03 00 21 45 00 00 28 DD 7E 00 00 3E 06 AB 6D C1 8C 10 2D
D9 83 48 A7 00 50 0C 2C F7 29 8E EE E7 73 42 4A 50 11 40 00
BF 9C 00 00 D9 A5
This is an IP datagram...
This datagram contains a TCP segment...
if statement inside the while was executed...
Now the connection is in state ESTABLISHED...
Polling the serial link for the PPP frame...
END flag is received... Buffering started...
END flag is received. Receipt of a frame is completed...
906 characters were received...
FF 03 00 21 45 00 03 84 E0 7E 00 00 3E 06 A5 11 C1 8C 10 2D
D9 83 48 A7 00 50 0C 2C F7 29 8B 92 E7 73 42 4A 50 19 40 00
E9 6E 00 00 48 54 54 50 2F 31 2E 30 20 32 30 30 20 4F 4B 0D
0A 41 67 65 3A 20 32 35 37 33 34 38 0D 0A 41 63 63 65 70 74
2D 52 61 6E 67 65 73 3A 20 62 79 74 65 73 0D 0A 44 61 74 65
3A 20 54 75 65 2C 20 30 32 20 4E 6F 76 20 32 30 30 34 20 30
38 3A 32 31 3A 34 32 20 47 4D 54 0D 0A 43 6F 6E 74 65 6E 74
2D 4C 65 6E 67 74 68 3A 20 34 39 37 0D 0A 43 6F 6E 74 65 6E
74 2D 54 79 70 65 3A 20 74 65 78 74 2F 68 74 6D 6C 0D 0A 43
6F 6E 74 65 6E 74 2D 4C 6F 63 61 74 69 6F 6E 3A 20 68 74 74
70 3A 2F 2F 31 39 33 2E 31 34 30 2E 31 36 2E 34 35 2F 44 65
66 61 75 6C 74 2E 68 74 6D 0D 0A 4C 61 73 74 2D 4D 6F 64 69
66 69 65 64 3A 20 54 68 75 2C 20 32 35 20 44 65 63 20 32 30
```

```
30 33 20 30 38 3A 33 35 3A 32 38 20 47 4D 54 0D 0A 45 54 61
67 3A 20 22 38 63 38 65 64 62 63 63 32 63 61 63 33 31 3A 66
32 66 22 0D 0A 53 65 72 76 65 72 3A 20 4D 69 63 72 6F 73 6F
66 74 2D 49 49 53 2F 36 2E 30 0D 0A 58 2D 50 6F 77 65 72 65
64 2D 42 79 3A 20 41 53 50 2E 4E 45 54 0D 0A 56 69 61 3A 20
31 2E 31 20 73 67 74 2D 6E 65 74 63 61 63 68 65 2D 30 32 20
28 4E 65 74 43 61 63 68 65 20 4E 65 74 41 70 70 2F 35 2E 35
52 32 29 0D 0A 0D 0A 3C 48 54 4D 4C 3E 0D 0A 0D 0A 3C 48 45
41 44 3E 0D 0A 3C 54 49 54 4C 45 3E 41 6E 61 64 6F 6C 75 20
DC 6E 69 76 65 72 73 69 74 65 73 69 20 42 69 6C 67 69 73 61
79 61 72 20 4D FC 68 65 6E 64 69 73 6C 69 3F 69 3C 2F 54 49
54 4C 45 3E 0D 0A 3C 2F 48 45 41 44 3E 0D 0A 0D 0A 3C 46 52
41 4D 45 53 45 54 20 63 6F 6C 73 20 3D 20 22 31 38 30 2C 2A
22 20 42 4F 52 44 45 52 3D 30 20 66 72 61 6D 65 62 6F 72 64
65 72 3D 22 30 22 3E 0D 0A 0D 0A 3C 46 52 41 4D 45 20 53 52
43 20 3D 20 22 73 6F 6C 2E 68 74 6D 6C 22 20 4E 4F 52 45 53
49 5A 45 20 53 43 52 4F 4C 4C 49 4E 47 3D 22 4E 4F 22 20 74
61 72 67 65 74 3D 22 6D 61 69 6E 22 20 6D 61 72 67 69 6E 77
69 64 74 68 3D 22 30 22 20 6D 61 72 67 69 6E 68 65 69 67 68
74 3D 22 30 22 3E 0D 0A 3C 46 52 41 4D 45 20 53 52 43 20 3D
20 22 64 75 79 75 72 75 6C 61 72 2E 68 74 6D 6C 22 20 4E 41
4D 45 3D 22 6D 61 69 6E 22 20 73 63 72 6F 6C 6C 69 6E 67 3D
22 61 75 74 6F 22 3E 0D 0A 3C 6E 6F 66 72 61 6D 65 73 3E 0D
0A 3C 70 20 61 6C 69 67 6E 3D 22 63 65 6E 74 65 72 22 3E 3C
66 6F 6E 74 20 66 61 63 65 3D 22 54 61 68 6F 6D 61 22 20 73
69 7A 65 3D 22 32 22 3E 3C 61 20 68 72 65 66 3D 22 64 75 79
75 72 75 6C 61 72 2E 68 74 6D 6C 22 3E 42 75 72 61 79 61 3C
2F 61 3E 20 74 FD 6B 6C 61 79 61 72 61 6B 0D 0A 73 61 79 66
61 6D FD 7A FD 20 7A 69 79 61 72 65 74 65 20 62 61 FE 6C 61
79 61 62 69 6C 69 72 73 69 6E 69 7A 2E 3C 2F 66 6F 6E 74 3E
3C 2F 70 3E 0D 0A 3C 2F 6E 6F 66 72 61 6D 65 73 3E 0D 0A 3C
2F 46 52 41 4D 45 53 45 54 3E 0D 0A 0D 0A 3C 2F 48 54 4D 4C
3E 20 0D 0A C8 B1
```

This is an IP datagram...

This datagram contains a TCP segment...

We received data from the remote server...

```
We will respond with an ACK to this data...
An ACK will be sent...
Our packet to be sent...
FF 03 00 21 45 00 00 28 00 0D 00 00 80 06 46 DF D9 83 48 A7
C1 8C 10 2D 0C 2C 00 50 E7 73 42 4A F7 29 8E EF 50 10 05 B4
F9 E8 00 00 9D FE
Also the FIN flag of this segment is set...
It means that by sending this ACK we entered CLOSE_WAIT
state...
We will send our FIN also...
A FIN will be sent...
We will enter the LAST_ACK state...
Our packet to be sent...
FF 03 00 21 45 00 00 28 00 0E 00 00 80 06 46 DE D9 83 48 A7
C1 8C 10 2D 0C 2C 00 50 E7 73 42 4A F7 29 8E EF 50 11 05 B4
F9 E7 00 00 F5 C5
if statement inside the while was executed...
Polling the serial link for the PPP frame...
END flag is received... Buffering started...
END flag is received. Receipt of a frame is completed...
46 characters were received...
FF 03 00 21 45 00 00 28 E3 7E 00 00 3E 06 A5 6D C1 8C 10 2D
D9 83 48 A7 00 50 0C 2C F7 29 8E EF E7 73 42 4B 50 10 40 00
BF 9B 00 00 B9 3C
This is an IP datagram...
This datagram contains a TCP segment...
We received an ACK from the remote server...
We received an ACK to the FIN that we sent...
The implementation enters the CLOSED state...
We will terminate the PPP link...
Our packet to be sent...
FF 03 C0 21 05 00 00 04 E1 9D
if statement inside the while was executed...
Polling the serial link for the PPP frame...
END flag is received... Buffering started...
END flag is received. Receipt of a frame is completed...
```

```
10 characters were received...
FF 03 C0 21 06 00 00 04 2C B8
This is an LCP packet...
LCP options will be handled...
if statement inside the while was executed...
Polling the serial link for the PPP frame...
if statement inside the while was executed...
Serial port was closed succesfully...
```

EK – 2 Yazılım Projesi

```
// superonline2.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"                    // for the precompiled headers stuff
#include "dbg.h"                       // for the debugging
#include "serialport.h"                // for the serialport handler
#include "modem.h"                     // for the modem handler
#include "PPP.h"                       // for the PPP implementation
#include "IP.h"                        // for the IP implementation
#include "ICMP.h"                      // for the ICMP implementation
#include "TCP.h"                       // for the TCP implementation
#include <windows.h>
#include <conio.h>
#include <time.h>


// Application specific modem initialization string
//
const char *Modem_Initialization = "ATZE0&C0&D0&S0\r";
// Above string means;
// reset, disable echo, CD always ON, DTR always ON, DSR always ON


// The number of the ISP to dial
// superonline
//
const char *tel_no = "ATDT08223143016\r";


// a FIFO buffer
//
unsigned char Modem_Buffer[20];


int main(int argc, char* argv[])
{
   // a text file is opened for debugging
   dbg_fileOpen ();

   // open the serial port COM1
   serial_Open (1, 9600, NoParity, 8, OneStopBit, NoFlowControl, FALSE);
   // COM1, 9600 BAUD, NoParity, OneStopBit, NoFlowControl, no overlapping

   // modem initializaiton
```

```
        dbg_filePrint ("Press any key to proceed with the necessary modem
initialization...\n\n");
        getch ();

        serial_Write (Modem_Initialization, strlen (Modem_Initialization));

        // wait a little
        Sleep (500);

        // The response function returns 0 for an OK response
        if (Modem_Response_COM (Modem_Buffer, 2) != 0) {
                dbg_filePrint ("Modem initialization is failed...\n\n");
                dbg_filePrint ("Terminating the program...\n\n");
                Sleep (1000);
                dbg_fileClose ();
                exit (1);
        }

        else
                dbg_filePrint ("Success in modem initialization...\n\n");

        // wait a little
        Sleep (1000);

        // dial the ISP
        dbg_filePrint ("Press any key to dial the ISP...\n\n");
        getch ();

        serial_Write (tel_no, strlen (tel_no));

        // after sending the dial command it takes 5 secs for it to dial
        Sleep (5000);

        // and it normally takes 30 secs for it to receive CONNECT message
        // The response function returns 1 for a CONNECT response
        if (Modem_Response_COM (Modem_Buffer, 30) == 1)
                dbg_filePrint ("Connection with the remote ISP was
established...\n\n");

        else {
                dbg_filePrint ("Connection with the remote ISP was
failed...\n\n");
                dbg_filePrint ("Terminating the program...\n\n");
                Sleep (1000);
                dbg_fileClose ();
                exit (1);
        }
```

```
        // protocol intializations
        TCP_Init ();
        IP_Init ();
        PPP_Init ();

        // loop endlessly until a key is pressed
        while (!kbhit ()) {

                PPP_poll ();

                if (PPPStatusByte.LinkOn) {

                        dbg_filePrint ("if statement inside the while was
executed...\n\n");

                        // we may comletely ignore sending ECHO_REQUESTs since
it is not mandatory
                        // in the case of ignoring
                        // we must be careful that in IPCP_Handle we start the
TCP connection

                        // we first send an ICMP_ECHO request to our remote ISP
                        // ICMP_Ping ((unsigned char *)ISPIPAddress);

                        // we then send an ICMP_ECHO request to the remote http
server
                        // ICMP_Ping ((unsigned char *) ISPIPAddress);

                        // send an HTTP GET command to the remote HTTP server
                        if (conn.state == ESTABLISHED) {
                                dbg_filePrint ("Now the connection is in state
ESTABLISHED...\n\n");

                                TCP_Send ((unsigned char *) "GET /
HTTP/1.0\r\n\r\n", 18);

                                // in the case when we initialize the TCP
connection termination
                                // TCP_Conn (FIN);
                        }

                }

        }

        // close the serial port
        serial_Close ();
        // close the debugging text file
        dbg_fileClose ();
```

```
    return 0;

}


// serialport.cpp : includes serial port driver functions
//
#include "stdafx.h"
#include "dbg.h"                        // for the debugging
#include "serialport.h"                 // for the serialport handler


// Handle to the port
//
HANDLE m_hComm;


// Is the port open in overlapped IO
//
BOOL   m_bOverlapped;


void serial_Init(void)
{
   m_hComm = INVALID_HANDLE_VALUE;
   m_bOverlapped = FALSE;
}


void serial_Open(short nPort, DWORD dwBaud, short parity, BYTE DataBits,
short stopbits, short fc, BOOL bOverlapped)
{
   char str[15];
   DCB dcb;

   sprintf (dbg_buff, "Serial port COM%d is opening...\n\n", nPort);
   dbg_filePrint (dbg_buff);

   //Call CreateFile to open up the seril port
   sprintf (str, "\\\\.\\COM%d", nPort);
   m_hComm = CreateFile (str, GENERIC_READ | GENERIC_WRITE, 0, NULL,
OPEN_EXISTING, bOverlapped ? FILE_FLAG_OVERLAPPED : 0, NULL);
   if (m_hComm == INVALID_HANDLE_VALUE) {
            // We just report the error and exit
            sprintf (dbg_buff, "Serial port COM%d could not be
opened...Error %d\n\n", nPort, GetLastError());
            dbg_filePrint (dbg_buff);
            exit (1);
```

```
        }
        else {
                // Give out a success mesage...
                sprintf (dbg_buff, "COM%d was opened succesfully...\n\n",
nPort);
                dbg_filePrint (dbg_buff);
        }

        // Maybe I can check for this later... NOP
        m_bOverlapped = bOverlapped;

        // Get the current state prior to changing it
        serial_GetState(&dcb);

        // Setup the baud rate
        dcb.BaudRate = dwBaud;

        // Setup the Parity
        switch (parity) {
                case EvenParity:  dcb.Parity = EVENPARITY;  break;
                case MarkParity:  dcb.Parity = MARKPARITY;  break;
                case NoParity:    dcb.Parity = NOPARITY;    break;
                case OddParity:   dcb.Parity = ODDPARITY;   break;
                case SpaceParity: dcb.Parity = SPACEPARITY; break;
        }

         // Setup the data bits
        dcb.ByteSize = DataBits;

        // Setup the stop bits
        switch (stopbits) {
                case OneStopBit:           dcb.StopBits = ONESTOPBIT;   break;
                case OnePointFiveStopBits: dcb.StopBits = ONE5STOPBITS; break;
                case TwoStopBits:          dcb.StopBits = TWOSTOPBITS;  break;
                default:
                                           break;
        }

        // Setup the flow control
        dcb.fDsrSensitivity = FALSE;
        switch (fc) {
                case NoFlowControl:
                        dcb.fOutxCtsFlow = FALSE;
                        dcb.fOutxDsrFlow = FALSE;
                        dcb.fOutX = FALSE;
                        dcb.fInX = FALSE;
                        break;
                case CtsRtsFlowControl:
                        dcb.fOutxCtsFlow = TRUE;
```

```
                dcb.fOutxDsrFlow = FALSE;
                dcb.fRtsControl = RTS_CONTROL_HANDSHAKE;
                dcb.fOutX = FALSE;
                dcb.fInX = FALSE;
                break;
        case CtsDtrFlowControl:
                dcb.fOutxCtsFlow = TRUE;
                dcb.fOutxDsrFlow = FALSE;
                dcb.fDtrControl = DTR_CONTROL_HANDSHAKE;
                dcb.fOutX = FALSE;
                dcb.fInX = FALSE;
                break;
        case DsrRtsFlowControl:
                dcb.fOutxCtsFlow = FALSE;
                dcb.fOutxDsrFlow = TRUE;
                dcb.fRtsControl = RTS_CONTROL_HANDSHAKE;
                dcb.fOutX = FALSE;
                dcb.fInX = FALSE;
                break;
        case DsrDtrFlowControl:
                dcb.fOutxCtsFlow = FALSE;
                dcb.fOutxDsrFlow = TRUE;
                dcb.fDtrControl = DTR_CONTROL_HANDSHAKE;
                dcb.fOutX = FALSE;
                dcb.fInX = FALSE;
                break;
        case XonXoffFlowControl:
                dcb.fOutxCtsFlow = FALSE;
                dcb.fOutxDsrFlow = FALSE;
                dcb.fOutX = TRUE;
                dcb.fInX = TRUE;
                dcb.XonChar = 0x11;
                dcb.XoffChar = 0x13;
                dcb.XoffLim = 100;
                dcb.XonLim = 100;
                break;
        default:
                break;
    }

    // Now that we have all the settings in place, make the changes
    serial_SetState(&dcb);

} // serial_Open


void serial_Close(void)
{
    // only attempt to close the serial port if it was opened before
```

```
        if (m_hComm!= NULL) {
                if (!CloseHandle(m_hComm)) {
                        // We just report the error and exit
                        sprintf (dbg_buff, "Serial port could not be closed...
Error = %d.\n\n", GetLastError());
                        dbg_filePrint (dbg_buff);
                        dbg_filePrint ("Teminating the program...\n\n");
                        exit (1);
                }
                else
                        dbg_filePrint ("Serial port was closed
succesfully...\n\n");

                // take the handle to the initial state.
                m_hComm = INVALID_HANDLE_VALUE;
                m_bOverlapped = FALSE;
        }

    } // serial_Close


    DWORD serial_Read(void *lpBuf, DWORD dwCount)
    {
       DWORD dwBytesRead = 0;
       if (!ReadFile(m_hComm, lpBuf, dwCount, &dwBytesRead, NULL))    {
                // We just report the error and exit
                sprintf (dbg_buff, "Serial_Read is failed... Error = %d\n\n",
GetLastError ());
                dbg_filePrint (dbg_buff);
                dbg_filePrint ("Teminating the program...\n\n");
                exit (1);
        }

        return dwBytesRead;

    } // serial_Read


    DWORD serial_Write(const void* lpBuf, DWORD dwCount)
    {
       DWORD dwBytesWritten = 0;

       if (!WriteFile(m_hComm, lpBuf, dwCount, &dwBytesWritten, NULL))        {
                // We just report the error and exit
                sprintf (dbg_buff, "Serial_Write is failed... Error = %d\n\n",
GetLastError ());
                dbg_filePrint (dbg_buff);
                dbg_filePrint ("Teminating the program...\n\n");
                exit (1);
```

```
        }

        return dwBytesWritten;

    }  // serial_Write


    // called to learn if the serial port buffer is full
    //
    void serial_GetStatus(COMSTAT* stat)
    {
        DWORD dwErrors;
        if (!ClearCommError(m_hComm, &dwErrors, stat)) {
                // We just report the error and exit
                sprintf (dbg_buff, "ClearCommState failed with error %d.\n\n",
GetLastError());
                dbg_filePrint (dbg_buff);
                dbg_filePrint ("Teminating the program...\n\n");
                exit (1);
        }
    }  // serial_GetStatus


    // called prior to setting the state of the serial port
    //
    void serial_GetState(DCB* dcb)
    {
        if (!GetCommState(m_hComm, dcb)) {
                // We just report the error and exit
                sprintf (dbg_buff, "GetCommState failed with error %d.\n\n",
GetLastError());
                dbg_filePrint (dbg_buff);
                dbg_filePrint ("Teminating the program...\n\n");
                exit (1);
        }
    }  // serial_GetState


    // called after the port settings are updated as desired
    //
    void serial_SetState(DCB* dcb)
    {
        if (!SetCommState(m_hComm, dcb))
        {
                // We just report the error and exit
                sprintf (dbg_buff, "SetCommState failed with error %d.\n\n",
GetLastError());
                dbg_filePrint (dbg_buff);
                dbg_filePrint ("Teminating the program...\n\n");
```

```
                exit (1);
        }
    }  // serial_SetState


    // modem.cpp : includes modem driver functions
    //
    #include "stdafx.h"
    #include "dbg.h"                        // for the debugging functions
    #include "serialport.h"                 // for the serialport handler
functions
    #include "modem.h"                      // for the modem handler functions
    #include <time.h>

    // Gets the response of the modem when it is in the command mode
    //
    short Modem_Response_COM (unsigned char *buf, short tout)
    {
        COMSTAT stat;
        int top, ret;
        time_t now;

        now = time (NULL);
        top = 0;
        while (top < 1600){
                serial_GetStatus (&stat);
                if (stat.cbInQue > 0) {
                        serial_Read (&buf[top], 1);
                        top++;
                }
                if ((time (NULL) - now) > tout)
                        break;
        }
        buf [top] = 0;

        // since the numeric corresponding of the OK response is 0,
        // we return 0 for success and non-zero for fail
        if (strstr ((char *)buf, "OK"))
                ret = 0;
        else if (strstr ((char *)buf, "CONNECT"))
                ret = 1;

        return ret;
    }


    // PPP.cpp : includes the PPP implementation functions.
    //
    #include "stdafx.h"
```

```
#include "dbg.h"                        // for the debugging
#include "serialport.h"                 // for the serialport handler
#include "modem.h"                      // for the modem handler
#include "PPP.h"                        // for the PPP implementation
#include "IP.h"                         // for the IP implementation
#include "ICMP.h"                       // for the ICMP implementation
#include "TCP.h"                        // for the TCP implementation
#include <windows.h>
#include <time.h>


// Our username and password
//
const char *Username = "xxxxxxxxxxxxxxxx";
const char *Password = " xxxxxx";


// FCS lookup table
//
static const unsigned short fcstab[256] = {
   0x0000, 0x1189, 0x2312, 0x329b, 0x4624, 0x57ad, 0x6536, 0x74bf,
   0x8c48, 0x9dc1, 0xaf5a, 0xbed3, 0xca6c, 0xdbe5, 0xe97e, 0xf8f7,
   0x1081, 0x0108, 0x3393, 0x221a, 0x56a5, 0x472c, 0x75b7, 0x643e,
   0x9cc9, 0x8d40, 0xbfdb, 0xae52, 0xdaed, 0xcb64, 0xf9ff, 0xe876,
   0x2102, 0x308b, 0x0210, 0x1399, 0x6726, 0x76af, 0x4434, 0x55bd,
   0xad4a, 0xbcc3, 0x8e58, 0x9fd1, 0xeb6e, 0xfae7, 0xc87c, 0xd9f5,
   0x3183, 0x200a, 0x1291, 0x0318, 0x77a7, 0x662e, 0x54b5, 0x453c,
   0xbdcb, 0xac42, 0x9ed9, 0x8f50, 0xfbef, 0xea66, 0xd8fd, 0xc974,
   0x4204, 0x538d, 0x6116, 0x709f, 0x0420, 0x15a9, 0x2732, 0x36bb,
   0xce4c, 0xdfc5, 0xed5e, 0xfcd7, 0x8868, 0x99e1, 0xab7a, 0xbaf3,
   0x5285, 0x430c, 0x7197, 0x601e, 0x14a1, 0x0528, 0x37b3, 0x263a,
   0xdecd, 0xcf44, 0xfddf, 0xec56, 0x98e9, 0x8960, 0xbbfb, 0xaa72,
   0x6306, 0x728f, 0x4014, 0x519d, 0x2522, 0x34ab, 0x0630, 0x17b9,
   0xef4e, 0xfec7, 0xcc5c, 0xddd5, 0xa96a, 0xb8e3, 0x8a78, 0x9bf1,
   0x7387, 0x620e, 0x5095, 0x411c, 0x35a3, 0x242a, 0x16b1, 0x0738,
   0xffcf, 0xee46, 0xdcdd, 0xcd54, 0xb9eb, 0xa862, 0x9af9, 0x8b70,
   0x8408, 0x9581, 0xa71a, 0xb693, 0xc22c, 0xd3a5, 0xe13e, 0xf0b7,
   0x0840, 0x19c9, 0x2b52, 0x3adb, 0x4e64, 0x5fed, 0x6d76, 0x7cff,
   0x9489, 0x8500, 0xb79b, 0xa612, 0xd2ad, 0xc324, 0xf1bf, 0xe036,
   0x18c1, 0x0948, 0x3bd3, 0x2a5a, 0x5ee5, 0x4f6c, 0x7df7, 0x6c7e,
   0xa50a, 0xb483, 0x8618, 0x9791, 0xe32e, 0xf2a7, 0xc03c, 0xd1b5,
   0x2942, 0x38cb, 0x0a50, 0x1bd9, 0x6f66, 0x7eef, 0x4c74, 0x5dfd,
   0xb58b, 0xa402, 0x9699, 0x8710, 0xf3af, 0xe226, 0xd0bd, 0xc134,
   0x39c3, 0x284a, 0x1ad1, 0x0b58, 0x7fe7, 0x6e6e, 0x5cf5, 0x4d7c,
   0xc60c, 0xd785, 0xe51e, 0xf497, 0x8028, 0x91a1, 0xa33a, 0xb2b3,
   0x4a44, 0x5bcd, 0x6956, 0x78df, 0x0c60, 0x1de9, 0x2f72, 0x3efb,
   0xd68d, 0xc704, 0xf59f, 0xe416, 0x90a9, 0x8120, 0xb3bb, 0xa232,
   0x5ac5, 0x4b4c, 0x79d7, 0x685e, 0x1ce1, 0x0d68, 0x3ff3, 0x2e7a,
   0xe70e, 0xf687, 0xc41c, 0xd595, 0xa12a, 0xb0a3, 0x8238, 0x93b1,
```

```
        0x6b46, 0x7acf, 0x4854, 0x59dd, 0x2d62, 0x3ceb, 0x0e70, 0x1ff9,
        0xf78f, 0xe606, 0xd49d, 0xc514, 0xb1ab, 0xa022, 0x92b9, 0x8330,
        0x7bc7, 0x6a4e, 0x58d5, 0x495c, 0x3de3, 0x2c6a, 0x1ef1, 0x0f78
    };


    // declare buffers for the PPP input and output
    //
    unsigned char InBuffer        [PPP_BUFF+1];
    unsigned char OutBuffer       [PPP_BUFF+1];


    // pointer to the input packet and a counter for the buffer
    //
    unsigned char *PPP_packet = InBuffer;
    static int top = 0;


    // void lcp to force the isp to the negotiations
    //
    unsigned char lcp_void [] = {0xFF, 0x03, 0xC0, 0x21, 0x01, 0x01, 0x00, 0x04,
                                                          0x00, 0x00};

    // a standart packet to end the PPP link
    //
    unsigned char lcp_term [] = {0xFF, 0x03, 0xC0, 0x21, 0x05, 0x00, 0x00, 0x04,
                                                          0x00, 0x00};

    // our lcp request
    //
    unsigned char lcp_req [] = {0xFF, 0x03, 0xC0, 0x21, 0x01, 0x01, 0x00, 0x0A,
                                                          0x02, 0x06, 0x00,
0x0A,    0x00, 0x00,       0x00, 0x00};


    // our ipcp request
    //
    unsigned char ipcp_req [] = {0xFF, 0x03, 0x80, 0x21, 0x01, 0x01, 0x00, 0x0A,
                                                          0x03, 0x06, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00};


    // a global bit-field of the type PPPStatus to hold the status of the PPP
link
    //
    struct PPPStatus PPPStatusByte;


    // initializes the PPP module by setting the ReSync bit
```

```
//
void PPP_Init (void)
{
   PPPStatusByte.ReSync = TRUE;
   dbg_filePrint ("PPP module was initialized... i.e. ReSync status bit is
set.\n\n");
}


// calculates the fcs with the given fcs and the data to be sent
//
static unsigned short PPP_fcs16 (unsigned short fcs, unsigned char *cp,
short len)
{
   while (len--)
            fcs = (fcs >> 8) ^ fcstab[(fcs ^ *cp++) & 0xff];
   return (fcs);
}


// returns the fcs of the PPP packet
//
unsigned short PPP_GetChecksum (register unsigned char *cp, register short
len)
{
   return ~PPP_fcs16( PPPINITFCS16, cp, len );
}


// gets the PPP frame from the modem when it is in the data mode
//
void PPP_Receive (void)
{
   COMSTAT stat;
   unsigned char ch;
   int i;

   while (top < 1600) {

            serial_GetStatus (&stat);

            if (stat.cbInQue > 0) {

                     serial_Read (&ch, 1);

                     // set the byte received status bit
                     PPPStatusByte.ByteRx = TRUE;
```

```
                        // remove the flags and de-escape the escaped characters
on receipt

                        // if a frame is recently received but not processed
                        // just return from the receive function without
                        // further buffering...
                        if (PPPStatusByte.IsFrame)
                                return;

                        // in the case of receipt of a character other than END
                        if (PPPStatusByte.ReSync) {

                                if (ch != END) {

                                        // if the received ch is not the end
of a username prompt
                                        // just return...
                                        if (ch != 0x3A)
                                                return;

                                        else if (ch == 0x3A) {

                                                sprintf (dbg_buff, "read
character is: %X\t%c\n\n", ch, ch);
                                                dbg_filePrint (dbg_buff);

                                                // upon receipt of username
promt we send a void lcp packet
                                                // to force the remote ISP to
the negotiations...
                                                dbg_filePrint ("We send a
void lcp packet to force the isp to the negotiatioins...\n\n");
                                                send_void_LCP ();
                                                return;
                                        }
                                }

                                // if the character is END
                                PPPStatusByte.ReSync = FALSE;       // reset
the synchronization bit
                                top = 0;
        // and the frame pointer (i.e. top)
                                dbg_filePrint ("END flag is received...
Buffering started...\n\n");
                        }


                        // if the previously received character is ESC
                        // de-escape this character
```

```
                        // and reset the ESC status bit
                        if (PPPStatusByte.IsESC) {
                                PPP_packet [top++] = 0x20 ^ ch;
                                PPPStatusByte.IsESC = FALSE;
                        }


                        else {
                                switch (ch) {

                                case END:
                                        // handle END character received at
the end of the packet...
                                        // avoid null PPP packets
                                        if (top > 0) {
                                                PPP_packet [top] = 0;
                                                // set frame bit of the
PPPstatus field
                                                PPPStatusByte.IsFrame = TRUE;

                                                // this portion of the code
prints the received
                                                // packet and it is for
debugging
                                                dbg_filePrint ("END flag is
received. Receipt of a frame is completed...\n\n");
                                                sprintf (dbg_buff, "%d
characters were received...\n\n", top);
                                                dbg_filePrint (dbg_buff);
                                                for (i = 0; i < top; i++) {
                                                        if (PPP_packet [i] <
0x10) {
                                                                sprintf
(dbg_buff, "0%X ", PPP_packet [i]);
        dbg_filePrint (dbg_buff);
                                                        }
                                                        else {
                                                                sprintf
(dbg_buff, "%X ", PPP_packet [i]);
        dbg_filePrint (dbg_buff);
                                                        }
                                                }
                                                dbg_filePrint ("\n\n");
                                                // ...

                                                return;
                                        }
```

```
                                        break;

                                case ESC:
                                        // handle ESC
                                        // only set ESC status bit
                                        PPPStatusByte.IsESC = TRUE;
                                        break;
                                default:
                                        PPP_packet [top++] = ch;
                                        break;
                                }               // end of switch

                        }               // end of else

                }               // end of the greatest if

        }               // end of while

}  // end of receive function


// sends the PPP packet byte by byte
//
void PPP_send (unsigned char *packet, char len)
{
   unsigned short chksm = 0;

   // get the checksum HDLC FCS
   chksm = PPP_GetChecksum (packet, packet[7]+4);
   packet [packet [7] + 4] = chksm & 0xFF;
   packet [packet [7] + 5] = (chksm >> 8) & 0xFF;


   // this portion of the code prints the packet to be sent
   // and it is for debugging...
   {
           int i;

           dbg_filePrint ("Our packet to be sent...\n\n");
           for (i = 0; i < (packet [7] + 6); i++) {
                   if (packet [i] < 0x10) {
                           sprintf (dbg_buff, "0%X ", packet [i]);
                           dbg_filePrint (dbg_buff);
                   }
                   else {
                           sprintf (dbg_buff, "%X ", packet [i]);
                           dbg_filePrint (dbg_buff);
                   }
           }
```

```
            dbg_filePrint ("\n\n");
        }
        // ...


        // start and end flags are inserted
        // and necessary escaping is performed...
        serial_TransmitChar(0x7E);

        while (len--) {

            // if ACCM is not still negotiated escape all ACCM
            if (!PPPStatusByte.IsACCM && *packet < 0x20) {
                serial_TransmitChar (0x7D);
                serial_TransmitChar (*packet ^ 0x20);
            }

            // if ACCM is negotiated, escape only ^S and ^Q
            else if (PPPStatusByte.IsACCM && (*packet == 0x11 || *packet ==
0x13)) {
                serial_TransmitChar (0x7D);
                serial_TransmitChar (*packet ^ 0x20);
            }

            // transmit the others without any processing
            else {

                switch (*packet) {
                case END:
                        serial_TransmitChar (0x7D);
                        serial_TransmitChar (0x5E);
                        break;
                case ESC:
                        serial_TransmitChar (0x7D);
                        serial_TransmitChar (0x5D);
                        break;
                default:
                        serial_TransmitChar (*packet);
                        break;
                }       // end of switch

            }

            packet++;

        }       // end of while

        serial_TransmitChar(0x7E);
```

```
}  // end of send function


// handles ipcp options
//
void IPCP_Handle (void)
{
   // declare two pointers to the options of the input and output buffers
   unsigned char *opt_src = (unsigned char *) &InBuffer[8];
   unsigned char *opt_dst = OutBuffer;

   switch (PPP_packet [4]) {
   case CONF_REQ:

            // when the first and only option is not IP address
            // and the length of the packet is more than 10 bytes
            // a conf_rej answer will be prepared
            if ((InBuffer [8] != 0x03) && (InBuffer [7] > 0x0A)) {

                     // a counter to keep the size of the all of the options
                     unsigned char all_options;
                     // an identifier to the options
                     unsigned char option;
                     // a counter for the size of an option
                     unsigned char size;

                     // copying the first 8 bytes of the received packet to
the output buffer
                     MYstrncpy (InBuffer, OutBuffer, 8);
                     OutBuffer [4] = CONF_REJ;
                     opt_dst += 8;

                     // only 7th element of the buffer holds the length
                     // subtract code, ID and length fields, i.e. 4 bytes
                     all_options = InBuffer [7] - 4;

                     while (all_options) {

                              // get the available option
                              option = *opt_src;
                              // get the size of the option
                              size = *(opt_src + 1);
                              // update the size of the all options
                              all_options -= size;

                              // the option not to be rejected is not copied
                              if (option == 0x03) {
                                       OutBuffer[7] -= size;
                                       opt_src += size;
```

```
                                        }

                                        // and the options those will be rejected are
copied
                                        else {
                                                while (size--)
                                                        *opt_dst++ = *opt_src++;
                                        }
                                }
                        }

                        // when the first and only option is IP address
                        // and the length of the packet is 10 bytes
                        // a CONF_ACK packet is prepared
                        else if ((InBuffer [8] == 0x03) && (InBuffer [7] == 0x0A)) {

                                ISPIPAddress [0] = InBuffer [10];
                                ISPIPAddress [1] = InBuffer [11];
                                ISPIPAddress [2] = InBuffer [12];
                                ISPIPAddress [3] = InBuffer [13];

                                MYstrncpy (InBuffer, OutBuffer, InBuffer [7] + 6);
                                OutBuffer [4] =CONF_ACK;
                        }


                        // packet is sent
                        PPP_send ((unsigned char *) OutBuffer, OutBuffer [7] + 6);
                        break;

                case CONF_ACK:

                        // IP Address is assigned with a CONF_ACK packet
                        // assigned address is taken
                        if (InBuffer [8] == 0x03 ) {
                                OurIPAddress [0] = InBuffer [10];
                                OurIPAddress [1] = InBuffer [11];
                                OurIPAddress [2] = InBuffer [12];
                                OurIPAddress [3] = InBuffer [13];
                        }

                        // IP addresses are printed for debugging
                        try_print ();

                        // at last PPPLink is ON
                        PPPStatusByte.LinkOn = TRUE;

                        if (PPPStatusByte.LinkOn)
                                dbg_filePrint ("LinkOn flag was set...\n\n");
```

```
                // with the IPCP CONF_ACK received PPP negotiations end...

                // we call TCP_Conn function with SYN parameter to establish a
TCP connection
                dbg_filePrint ("We call TCP connection management function to
establish a connection...\n\n");
                TCP_Conn (SYN);

                break;

        case CONF_NAK:

                // CONF_NAK is received to the first ipcp CONF_REQ packet that
we sent
                if ((InBuffer [5] == 0x01) && (InBuffer [8] == 0x03)) {

                        MYstrncpy (InBuffer, OutBuffer, InBuffer [7] + 6);

                        // we request the ip address that we were NAKed of
                        OutBuffer [4] = CONF_REQ;
                        OutBuffer [5] = OutBuffer [5] + 1;

                        PPP_send ((unsigned char *) OutBuffer, OutBuffer [7] +
6);
                }
                break;

        case CONF_REJ:
                // a CONF_REJ is never received in normal conditions
                break;
        }
    }


    // handles LCP options in an incoming packet
    //
    void LCP_Handle (void)
    {
        // two pointers to the options of the input and output buffers
        unsigned char *opt_src = (unsigned char *) &InBuffer[8];
        unsigned char *opt_dst = OutBuffer;

        switch (PPP_packet[4]) {

        case CONF_REQ:

                // first options requested by the server are rejected
                // excluding the option 2, ACCM and option 3, auth. portocol
```

```
                if ((InBuffer [7] > 0x0E)) {
                        // a counter to keep the size of the all of the options
                        unsigned char all_options;
                        // an identifier to the options
                        unsigned char option;
                        // a counter for the size of an option
                        unsigned char size;

                        // copying the first 8 bytes of the received packet to
the output buffer
                        MYstrncpy (InBuffer, OutBuffer, 8);
                        OutBuffer [4] = CONF_REJ;
                        opt_dst += 8;

                        // only 7th element of the buffer holds the length
                        // subtract code, ID and length fields, i.e. 4 bytes
                        all_options = InBuffer [7] - 4;

                        while (all_options) {

                                // get the available option
                                option = *opt_src;
                                // get the size of the option
                                size = *(opt_src + 1);
                                // update the size of the all options
                                all_options -= size;

                                // the options not to be rejected are not
copied
                                if ((option == 0x02) || (option == 0x03)) {
                                        OutBuffer[7] -= size;
                                        opt_src += size;
                                }
                                // and the options those will be rejected are
copied
                                else {
                                        while (size--)
                                                *opt_dst++ = *opt_src++;
                                }
                        }

                        PPP_send ((unsigned char *) OutBuffer, OutBuffer [7] +
6);
                }

                // when options 2 and 3 are negotiated
                // a CONF_ACK packet is prepared
                else if ((InBuffer [7] == 0x0E)) {
```

```
                        MYstrncpy (InBuffer, OutBuffer, InBuffer [7] + 6);
                        OutBuffer [4] =CONF_ACK;

                        PPP_send ((unsigned char *) OutBuffer, OutBuffer [7] +
6);

                        // indicate that we accepted the request of ACCM
                        PPPStatusByte.IsACCM = TRUE;

                        // send the PAP_REQ
                        PPP_PAP_send (CONF_REQ, 1, (unsigned char *)Username,
(unsigned char *)Password);

                }
                break;

        case CONF_ACK:

                // a CONF_ACK only for ACCM option is received
                if (InBuffer [8] == 0x02)
                        dbg_filePrint ("All of the options we negotiate were
acked...\n\n");
                break;

        case CONF_NAK:
                // a CONF_NAK is never received in normal conditions
                break;

        case CONF_REJ:
                // a CONF_REJ is never received in normal conditions
                break;

        }       // end of switch

    } // end of LCP_Handle


    // sends the username and password for PAP authentication
    //
    void PPP_PAP_send (unsigned char code, unsigned char id, unsigned char
*username, unsigned char *password)
    {
       char user_len, pass_len;

       user_len = strlen ((const char *)username);
       pass_len = strlen ((const char *)password);

       // preparing the packet
```

```
        OutBuffer [0] = 0xFF;
        OutBuffer [1] = 0x03;
        OutBuffer [2] = 0xC0;
        OutBuffer [3] = 0x23;
        OutBuffer [4] = code;
        OutBuffer [5] = id;
        OutBuffer [6] = 0;
        OutBuffer [7] = user_len + pass_len + 6;
        OutBuffer [8] = user_len;
        MYstrncpy (username, &OutBuffer [9], user_len);
        OutBuffer [9 + user_len] = pass_len;
        MYstrncpy (password, &OutBuffer [10 + user_len], pass_len);

        // sending the packet
        PPP_send ((unsigned char *)OutBuffer, OutBuffer [7] + 6);
    }

    // copies from source to the destination of the given length
    //
    void MYstrncpy (unsigned char *src, unsigned char *dest, register numBYTEs)
{
        if ( numBYTEs <= 0 ) return;
        if ( src < dest ) {
                src += numBYTEs;
                dest += numBYTEs;
                do {
                *--dest = *--src;
                } while ( --numBYTEs > 0 );
        } else
                do {
                     *dest++ = *src++;
                } while ( --numBYTEs > 0 );
    }


    // sends a void LCP packet when a "Username:" prompt is received from the
remote ISP
    //
    void send_void_LCP (void)
    {
        MYstrncpy (lcp_void, OutBuffer, lcp_req [7] + 6);
        PPP_send ((unsigned char *)OutBuffer, OutBuffer [7] + 6);
    }


    // sends our LCP options after receiving a valid conf_req from the remote
ISP
    //
    void send_first_LCP (void)
```

```
{
   static unsigned char id = 0;

   MYstrncpy (lcp_req, OutBuffer, lcp_req [7] + 6);
   // increasing the ID by one since least one packet is sent before
   // also necessary for retransmissions
   OutBuffer [5] = id++;
   PPP_send ((unsigned char *)OutBuffer, OutBuffer [7] + 6);
}


// sends an IPCP packet with only option IP address to start the four way
// handshake for IP address assignment
//
void send_first_IPCP (void)
{
   MYstrncpy (ipcp_req, OutBuffer, ipcp_req [7] + 6);
   PPP_send ((unsigned char *)OutBuffer, OutBuffer [7] + 6);
}


// ends the PPP link
//
void PPP_Terminate (void) {
   MYstrncpy ( lcp_term, OutBuffer, lcp_term [7] + 6);
   PPP_send ((unsigned char *)OutBuffer, OutBuffer [7] + 6);
}


// continiously polls for a new PPP packet
// it is called from the endless while in the main function
//
void PPP_poll (void)
{
   // to drop the first packet until an ack is received
   static unsigned char first_rcv = FALSE;

   dbg_filePrint ("Polling the serial link for the PPP frame...\n\n");
   // and buffer the received PPP packet into the PPP_packet array
   PPP_Receive ();

   // continue with processing only if a frame is ready
   if (PPPStatusByte.IsFrame) {

            // never executes after the first packet is received...
            if (!first_rcv) {
                     // indicate that the first packet is received...
                     first_rcv = TRUE;
                     // and send our first options
```

```
                        dbg_filePrint ("We drop the first received packet and
send our first lcp options...\n\n");
                        send_first_LCP ();
                        // and drop the first received packet...
                        // reset the frame flag
                        PPPStatusByte.IsFrame = FALSE;
                        // set the synchronization flag
                        PPPStatusByte.ReSync = TRUE;
                        // just to receive another PPP frame
                        return;
                }

                switch (*(unsigned short*) &PPP_packet [2]) {

                case LCP_PACKET:
                        // handle LCP
                        dbg_filePrint ("This is an LCP packet...\n");
                        dbg_filePrint ("LCP options will be handled...\n\n");
                        LCP_Handle ();
                        break;
                case PAP_PACKET:
                        // handle PAP
                        dbg_filePrint ("This is a PAP packet...\n\n");
                        // the only possiblity of receiving a pap packet is pap
ack
                        if (InBuffer [4] = 0x02)
                                dbg_filePrint ("Our username and password are
authenticated...\n\n");

                        // upon receipt of a valid a pap ack an empty ipcp
packet is sent
                        // to start the four handshake negotiations for our ip
address...
                        dbg_filePrint ("We send an empty ipcp packet...\n\n");
                        send_first_IPCP ();

                        break;
                case IPCP_PACKET:
                        // handle IPCP
                        dbg_filePrint ("This is an IPCP packet...\n\n");
                        IPCP_Handle ();
                        break;
                case IP_DATAGRAM:
                        //handle IP
                        dbg_filePrint ("This is an IP datagram...\n\n");
                        IP_Receive ((IPDatagram *) &InBuffer [4]);

                default:
                        break;
```

```
            }
        }

        // reset the frame flag
        PPPStatusByte.IsFrame = FALSE;
        // set the synchronization flag
        PPPStatusByte.ReSync = TRUE;
        // just to receive another PPP frame

}


// IP.cpp : includes the IP implementation functions.
//
#include "stdafx.h"
#include "superonline3.h"     // fow network to host conversions
#include "dbg.h"                        // for the debugging
#include "PPP.h"                        // for the PPP implementation
#include "IP.h"                                 // for the IP implementation
#include "ICMP.h"                       // for the ICMP implementation
#include "TCP.h"                        // for the TCP implementation
#include <windows.h>
#include <time.h>


// two arrays for IP addresses
//
unsigned char OurIPAddress [4];
unsigned char ISPIPAddress [4];


// two pointers to structures for the input and output IP datagrams
//
IPDatagram *IP_Recvd;
IPDatagram *IP_2Send;


// initializes the IP module by initializing the pointers to buffers
//
void IP_Init (void) {

    IP_Recvd = (IPDatagram *) &InBuffer [4];
    IP_2Send = (IPDatagram *) &OutBuffer [4];
    dbg_filePrint ("IP module was initialized...\n\n");

}


// calculates the IP header checksum
```

```
//
unsigned short IP_GetChecksum (unsigned char *cp, short len) {

   unsigned int sum = 0;

   while (len--) {
            sum += ((unsigned int)((*cp << 8) + *(cp+1)) & 0xFFFF);
            cp += 2;
   }

   sum = (sum >> 16) + (sum & 0xFFFF);
   sum += (sum >> 16);

   return (unsigned short) ~sum;
}


// sends the IP datagram to the link layer module
//
void IP_Send (IPDatagram *Datag2Send) {

   static unsigned short ID = 0x0A;

   Datag2Send->Ver_IHL = 0x45;
   Datag2Send->TOS = 0x00;
   // disables the next line when an ICMP ECHO_REPLY is about to be sent
   if (Datag2Send->Payload[0] != 0x00)
            Datag2Send->ID = htons (ID);

   ID = ID + 1;

   Datag2Send->Flag_Frag = 0x00;
   Datag2Send->TTL = 0x80;
   Datag2Send->Checksum = 0x0000;
   // since IP header is always 20 Bytes long the length parameter will
always be 10
   unsigned short chksm_buff = IP_GetChecksum ( (unsigned char *)
&(Datag2Send [0]), 10);
   Datag2Send->Checksum = htons (chksm_buff);

   OutBuffer [0] = 0xFF;
   OutBuffer [1] = 0x03;
   OutBuffer [2] = 0x00;
   OutBuffer [3] = 0x21;

   PPP_send (OutBuffer, OutBuffer [7] + 6);
}
```

```
// receives the IP datagram from the link layer module
// and multiplexes according to the protocol
//
void IP_Receive (IPDatagram *DatagRecvd) {

   switch (DatagRecvd->Protocol) {
   case ICMP:
            // handle ICMP
            dbg_filePrint ("This datagram contains an ICMP message...\n\n");
            ICMP_Handle (DatagRecvd);
            break;
   case TCP:
            //handle TCP
            dbg_filePrint ("This datagram contains a TCP segment...\n\n");
            TCP_Receive ((TCPSegment *) &InBuffer [24]);
            break;
   }
}


// ICMP.cpp : includes the ICMP implementation functions.
//
#include "stdafx.h"
#include "superonline3.h"     // fow network to host conversions
#include "dbg.h"                        // for the debugging
#include "PPP.h"                        // for the PPP implementation
#include "IP.h"                                 // for the IP implementation
#include "ICMP.h"                       // for the ICMP implementation
#include "TCP.h"                        // for the TCP implementation
#include <windows.h>
#include <time.h>


// a global variable to hold the seq no for control
//
static unsigned short sent_seq;


// prints the IP addresses of us and the remote ISP...
//
void try_print () {

   dbg_filePrint ("The IP address of the remote ISP: ");
   sprintf (dbg_buff, "%d.%d.%d.%d\n\n", ISPIPAddress [0], ISPIPAddress [1],
ISPIPAddress [2], ISPIPAddress [3]);
   dbg_filePrint (dbg_buff);
   dbg_filePrint ("Our IP adress: ");
   sprintf (dbg_buff, "%d.%d.%d.%d\n\n", OurIPAddress [0], OurIPAddress [1],
OurIPAddress [2], OurIPAddress [3]);
```

```
        dbg_filePrint (dbg_buff);
    }


    // prepares am ICMP_ECHO message and sends it to the IP module
    //
    void ICMP_Ping (unsigned char *RemoteIP) {

        unsigned short chksm = 0;
        static unsigned short seq = 0x02FF;

        if (seq == 0x0300)
                return;

        dbg_filePrint ("The IP address that is going to be pinged :");
        sprintf (dbg_buff, "%d.%d.%d.%d\n\n", RemoteIP [0], RemoteIP [1],
RemoteIP [2], RemoteIP [3]);
        dbg_filePrint (dbg_buff);

        IP_2Send->Protocol = ICMP;
        IP_2Send->Length = htons (0x001C);

        // modifying the IP addresses
        IP_2Send->SrcIP[0] = OurIPAddress [0];
        IP_2Send->SrcIP[1] = OurIPAddress [1];
        IP_2Send->SrcIP[2] = OurIPAddress [2];
        IP_2Send->SrcIP[3] = OurIPAddress [3];

        IP_2Send->DstIP [0] = RemoteIP [0];
        IP_2Send->DstIP [1] = RemoteIP [1];
        IP_2Send->DstIP [2] = RemoteIP [2];
        IP_2Send->DstIP [3] = RemoteIP [3];

        IP_2Send->Payload [0] = ECHO;                   // Type
        IP_2Send->Payload [1] = 0x00;                   // Code
        IP_2Send->Payload [2] = 0x00;                   // Checksum
        IP_2Send->Payload [3] = 0x00;
        IP_2Send->Payload [4] = 0x27;                   // ID
        IP_2Send->Payload [5] = 0x1D;

        sent_seq = ++seq;
        IP_2Send->Payload [6] = (seq >> 8) & 0xFF;
        IP_2Send->Payload [7] = seq & 0xFF;

        chksm = IP_GetChecksum ( (unsigned char *) &(IP_2Send->Payload [0]), 20);
        IP_2Send->Payload [2] = (chksm >> 8) & 0xFF;
        IP_2Send->Payload [3] = chksm & 0xFF;

        IP_Send (IP_2Send);
```

```
}

// handles incoming ICMP_ECHO messages and sends a reply when a request is
received
//
void ICMP_Handle (IPDatagram *DatagRecvd) {

   unsigned short chksm;

   // switch according to the first byte (ICMP type) of the payload
   switch (DatagRecvd->Payload[0]) {
   case ECHO:
           dbg_filePrint ("We received an ICMP ECHO message...\n\n");
           MYstrncpy ((unsigned char *) IP_Recvd, (unsigned char *)
IP_2Send, ntohs (IP_Recvd->Length));

           // swap source and destination IP addresses
           IP_2Send->DstIP [0] = IP_2Send->SrcIP [0];
           IP_2Send->DstIP [1] = IP_2Send->SrcIP [1];
           IP_2Send->DstIP [2] = IP_2Send->SrcIP [2];
           IP_2Send->DstIP [3] = IP_2Send->SrcIP [3];

           IP_2Send->SrcIP [0] = OurIPAddress [0];
           IP_2Send->SrcIP [1] = OurIPAddress [1];
           IP_2Send->SrcIP [2] = OurIPAddress [2];
           IP_2Send->SrcIP [3] = OurIPAddress [3];

           IP_2Send->Payload [0] = ECHO_REPLY;
           IP_2Send->Payload [2] = 0x00;
           IP_2Send->Payload [3] = 0x00;

           chksm = IP_GetChecksum ( (unsigned char *) &(IP_2Send->Payload
[0]), (htons (IP_2Send->Length) - 20) >> 1);
           IP_2Send->Payload [2] = (chksm >> 8) & 0xFF;
           IP_2Send->Payload [3] = chksm & 0xFF;

           IP_Send (IP_2Send);

           break;

   case ECHO_REPLY:
           dbg_filePrint ("We received an ICMP ECHO_REPLY message...\n\n");

           if ( (sent_seq & 0xFF) == IP_Recvd->Payload[7] ) {
                   dbg_filePrint ("This ICMP ECHO_REPLY message is a reply
to our ECHO message...\n\n");
```

```
                        // TCP_Conn function is called with SYN parameter to
open a TCP connection
                        // dbg_filePrint ("We call TCP connection management
function to establish a connection...\n\n");
                        // TCP_Conn (SYN);
                }

                break;
        }

    }


    // TCP.cpp : includes the TCP implementation functions.
    //
    #include "stdafx.h"
    #include "superonline3.h"     // fow network to host conversions
    #include "dbg.h"                        // for the debugging
    #include "PPP.h"                        // for the PPP implementation
    #include "IP.h"                                 // for the IP implementation
    #include "ICMP.h"                       // for the ICMP implementation
    #include "TCP.h"                        // for the TCP implementation
    #include <windows.h>
    #include <time.h>


    // a structure to hold the TCB of the TCP connection
    //
    struct TCPTCB conn;

    // two pointers to structures for the input and output TCP segments
    //
    TCPSegment *TCP_Recvd;
    TCPSegment *TCP_2Send;

    // Initializes the TCP module by initializing the pointers to buffers
    //
    void TCP_Init (void) {

       // anything for the local port number
       conn.lPort = 0x0C2C;
       // remote port is http port
       conn.rPort = 0x0050;
       // IP address of the remote server is initialized here
       // ceng.anadolu.edu.tr
       conn.rIP [0] = 0xC1;
       conn.rIP [1] = 0x8C;
       conn.rIP [2] = 0x10;
       conn.rIP [3] = 0x2D;
```

```
        // initial state is CLOSED
        conn.state = CLOSED;
        // anything for the sequence number
        conn.lSeq = 0xE7734237;
        conn.rSeq = 0x00000000;

        // initialize the tcp pointers
        TCP_Recvd = (TCPSegment *) &InBuffer [24];
        TCP_2Send = (TCPSegment *) &OutBuffer [24];

        dbg_filePrint ("TCP module was initialized...\n\n");

    }

    // calculates the TCP checksum with the pseudo header
    //
    unsigned short TCP_GetChecksum (unsigned char *IPandTCP) {

        unsigned int chksm = 0;

        static unsigned char count = 1;

        chksm = IP_GetChecksum ( (unsigned char *) &(IPandTCP [12]), (IPandTCP
[3] - 12) >> 1);

        chksm = ~chksm + 0x06;

        chksm += (IPandTCP [3] - 20);

        chksm = (chksm >> 16) + (chksm & 0xFFFF);
        chksm += (chksm >> 16);

        return ~chksm;

    }

    // manages the TCP connection...
    //
    void TCP_Conn (unsigned char FLAGStoSET) {

        unsigned short chksm_buff;

        IP_2Send->Protocol = TCP;

        // modifying the IP addresses
        IP_2Send->SrcIP[0] = OurIPAddress [0];
        IP_2Send->SrcIP[1] = OurIPAddress [1];
        IP_2Send->SrcIP[2] = OurIPAddress [2];
```

```
        IP_2Send->SrcIP[3] = OurIPAddress [3];

        IP_2Send->DstIP [0] = conn.rIP [0];
        IP_2Send->DstIP [1] = conn.rIP [1];
        IP_2Send->DstIP [2] = conn.rIP [2];
        IP_2Send->DstIP [3] = conn.rIP [3];


        TCP_2Send->srcPort = htons (conn.lPort);
        TCP_2Send->dstPort = htons (conn.rPort);

        TCP_2Send->seqNO = htonl (conn.lSeq);
        TCP_2Send->ackNO = htonl (conn.rSeq);

        TCP_2Send->wnd = htons (0x05B4);
        TCP_2Send->chksm = 0x0000;

        TCP_2Send->urgPNT = 0x0000;

        switch (FLAGStoSET) {

        // we always send the first packet to establish a TCP connection
        case SYN:
                dbg_filePrint ("A SYN will be sent...\n\n");
                // in case of sending the first SYN total length of the IP
datagram will be 0x2C bytes
                IP_2Send->Length = htons (0x002C);


                TCP_2Send->dataOFF = 0x60;
                TCP_2Send->ctrlBITS = SYN;

                // maximum segment size option
                TCP_2Send->data [0] = 0x02;
                TCP_2Send->data [1] = 0x04;
                TCP_2Send->data [2] = 0x05;
                TCP_2Send->data [3] = 0xB4;

                chksm_buff = TCP_GetChecksum ( (unsigned char *) &(IP_2Send [0])
);
                TCP_2Send->chksm = htons (chksm_buff);

                conn.state = SYN_SENT;

                break;

        // this is sent only once to be the last packet
        // of the three-way handshake of the connection establishment
        case ACK:
```

```
                dbg_filePrint ("An ACK will be sent...\n\n");
                // in case of sending the an ACK, total length of the IP
datagram will be 0x28 bytes
                IP_2Send->Length = htons (0x0028);

                TCP_2Send->dataOFF = 0x50;
                TCP_2Send->ctrlBITS = ACK;

                // no options in the ack packet...

                chksm_buff = TCP_GetChecksum ( (unsigned char *) &(IP_2Send [0])
);
                TCP_2Send->chksm = htons (chksm_buff);

                break;

        // PSH case is handled to send a data
        case PSH:
                dbg_filePrint ("A PSH will be sent...\n\n");
                break;

        case FIN:
                dbg_filePrint ("A FIN will be sent...\n\n");

                IP_2Send->Length = htons (0x0028);

                TCP_2Send->dataOFF = 0x50;
                TCP_2Send->ctrlBITS = (FIN | ACK);

                // no options in the ack packet...

                chksm_buff = TCP_GetChecksum ( (unsigned char *) &(IP_2Send [0])
);
                TCP_2Send->chksm = htons (chksm_buff);

                if (conn.state == ESTABLISHED) {
                        // in the case when we send the first FIN we enter
FIN_WAIT_1 state
                        dbg_filePrint ("We will enter the FIN_WAIT_1
state...\n\n");
                        conn.state = FIN_WAIT_1;
                }
                else if (conn.state = CLOSE_WAIT) {
                        // when we send the FIN after we receive FIN first we
enter LAST_ACK state
                        dbg_filePrint ("We will enter the LAST_ACK
state...\n\n");
                        conn.state = LAST_ACK;
                }
```

```
                break;

        }

        IP_Send (IP_2Send);

    }

    // handles the incoming TCP segment
    //
    void TCP_Receive (TCPSegment *SegRecvd) {

        // we first update the sequnce number of our implementation with the
acknowledge number
        // that we receive from the remote implementation
        conn.lSeq = ntohl (SegRecvd->ackNO);
        // we must modify the received sequence number according to the data
length
        unsigned char offset_buff = SegRecvd->dataOFF >> 4;
        conn.rSeq = ntohl (SegRecvd->seqNO) + (ntohs (IP_Recvd->Length) - 20 -
offset_buff * 4 );

        switch (SegRecvd->ctrlBITS) {

        case (FIN | ACK):
                /*
                // in the case when we receive a FIN with the segment that
contains data
                // from the remote server this case will not hold...
                dbg_filePrint ("We have received FIN from the remote
server...\n\n");
                switch (conn.state) {
                case ESTABLISHED:
                        // we will silently discard this FIN, this is a rare
case...
                        /*
                        dbg_filePrint ("We received a FIN while we are
ESTABLISHED state...\n\n");
                        dbg_filePrint ("WE just ACK and enter the CLOSE_WAIT
state...\n\n");
                        // since FIN flag consumes 1 sequence number we
increment the acknowledge number by 1
                        conn.rSeq++;
                        TCP_Conn (ACK);

                        break;

                case FIN_WAIT_1:
```

```
                        dbg_filePrint ("We will send an ACK to this
FIN...\n\n");
                        dbg_filePrint ("And we will enter CLOSING
state...\n\n");
                        conn.state = CLOSING;
                        // since FIN flag consumes 1 sequence number we
increment the acknowledge number by 1
                        conn.rSeq++;
                        TCP_Conn (ACK);
                        break;

                case FIN_WAIT_2:
                        dbg_filePrint ("We will send an ACK to this
FIN...\n\n");
                        // since FIN flag consumes 1 sequence number we
increment the acknowledge number by 1
                        conn.rSeq++;
                        TCP_Conn (ACK);
                        dbg_filePrint ("In fact a typical TCP connection enters
to TIME_WAIT state, but we ignore that state...\n\n");
                        dbg_filePrint ("We will terminate the PPP link...\n\n");
                        PPP_Terminate ();
                        break;

                }
        */
                break;

        // this case is valid only when we receive a reply to our connection
request packet
        case (SYN | ACK):
                // we call connection management function to send an ACK
                if (conn.state != SYN_SENT) {
                        dbg_filePrint ("We received a SYN when we are not in the
SYN_SENT state...\n\n");
                        dbg_filePrint ("We will drop this SYN...n\n");
                        return;
                }
                else
                        conn.state = ESTABLISHED;

                dbg_filePrint ("We received SYN + ACK from the remote
server...\n\n");
                dbg_filePrint ("We will send an ACK to perform the last
operation of the three way handshake...\n\n");
                // since SYN flag consumes 1 sequence number we increment the
acknowledge number by 1
                conn.rSeq++;
                TCP_Conn (ACK);
```

```
                break;

        case RST:
                dbg_filePrint ("We received RST from the remote server...\n\n");
                dbg_filePrint ("We will terminate the connection...\n\n");

                PPP_Terminate ();
                break;

        case (FIN | PSH | ACK):
                // when the FIN flag of the TCP segment that carries data is set
                dbg_filePrint ("We received data from the remote
server...\n\n");
                dbg_filePrint ("We will respond with an ACK to this
data...\n\n");

                // we first update the acknowledgement number
                conn.rSeq++;
                TCP_Conn (ACK);
                conn.state = CLOSE_WAIT;

                dbg_filePrint ("Also the FIN flag of this segment is
set...\n\n");
                dbg_filePrint ("It means that by sending this ACK we entered
CLOSE_WAIT state...\n\n");
                dbg_filePrint ("We will send our FIN also...\n\n");
                TCP_Conn (FIN);

                break;

        case ACK:
                dbg_filePrint ("We received an ACK from the remote
server...\n\n");
                switch (conn.state) {

                case ESTABLISHED:
                        dbg_filePrint ("This is an ACK for the data that we
sent...\n\n");
                        break;

                case LAST_ACK:
                        dbg_filePrint ("We received an ACK to the FIN that we
sent...\n\n");
                        dbg_filePrint ("The implementation enters the CLOSED
state...\n\n");
                        dbg_filePrint ("We will terminate the PPP link...\n\n");
                        PPP_Terminate ();
                        break;
```

```
                /*
                // in the case when we receive a FIN with the segment that
contains data
                // from the remote server these cases will not hold...

                case FIN_WAIT_1:
                        dbg_filePrint ("This is the ACK to our FIN...\n\n");
                        dbg_filePrint ("We enter FIN_WAIT_2 state...\n\n");
                        conn.state = FIN_WAIT_2;
                        break;

                case CLOSING:
                        dbg_filePrint ("We received an ACK to our FIN,
connection is ended...\n\n");
                        dbg_filePrint ("In fact a typical TCP connection enters
to TIME_WAIT state, but we ignore that state...\n\n");
                        dbg_filePrint ("We will terminate the PPP link...\n\n");
                        PPP_Terminate ();
                        break;
                */

                }
                break;

        }

    }  // end of TCP_Receive


    // sends the data that it is called with
    //
    void TCP_Send (unsigned char *data, unsigned char len) {

        static unsigned char sent_once = FALSE;

        if (sent_once)
                return;

        sprintf (dbg_buff, "%s", data);
        dbg_filePrint (dbg_buff);

        unsigned short chksm_buff;

        IP_2Send->Protocol = TCP;

        // total length in the IP header is the sum of the both IP and TCP
headers length
        // and the length of the data to be sent
```

```
    IP_2Send->Length = htons (0x0028 + len);

    // modifying the IP addresses
    IP_2Send->SrcIP[0] = OurIPAddress [0];
    IP_2Send->SrcIP[1] = OurIPAddress [1];
    IP_2Send->SrcIP[2] = OurIPAddress [2];
    IP_2Send->SrcIP[3] = OurIPAddress [3];

    IP_2Send->DstIP [0] = conn.rIP [0];
    IP_2Send->DstIP [1] = conn.rIP [1];
    IP_2Send->DstIP [2] = conn.rIP [2];
    IP_2Send->DstIP [3] = conn.rIP [3];


    TCP_2Send->srcPort = htons (conn.lPort);
    TCP_2Send->dstPort = htons (conn.rPort);

    TCP_2Send->seqNO = htonl (conn.lSeq);
    TCP_2Send->ackNO = htonl (conn.rSeq);

    TCP_2Send->wnd = htons (0x05B4);
    TCP_2Send->chksm = 0x0000;

    TCP_2Send->urgPNT = 0x0000;

    TCP_2Send->dataOFF = 0x50;
    TCP_2Send->ctrlBITS = (PSH | ACK);

    dbg_filePrint ("TCP header is filled...\n\n");

    MYstrncpy ( data, (unsigned char *) TCP_2Send->data, len);

    dbg_filePrint ("Data is copied to be the payload...\n\n");

    chksm_buff = TCP_GetChecksum ( (unsigned char *) &(IP_2Send [0]) );
    TCP_2Send->chksm = htons (chksm_buff);

    IP_Send (IP_2Send);

    sent_once = TRUE;

}


// dbg.cpp: includes print functions for debugging
//
#include "stdafx.h"
#include "dbg.h"                        // for the debugging functions
#include <windows.h>
```

```
FILE *hFile;

char dbg_buff [100];

void dbg_fileOpen (void)
{
   if ( !(hFile = fopen  ("C:\\Program Files\\Microsoft Visual
Studio\\MyProjects\\dbg.txt", "w+"))) {
            printf ("Debugging file could not be opened...");
            exit(1);
   }
}

void dbg_fileClose (void)
{
   fclose (hFile);
}

void dbg_filePrint (const char *str_print)
{
   printf (str_print);
   fprintf (hFile, str_print);
}

// serialport.h : includes function prototypes and macro definitons of the
serilaport driver
//

#ifndef SERIALPORT_H
#define SERIALPORT_H

//Enums
enum FlowControl {
   NoFlowControl,
   CtsRtsFlowControl,
   CtsDtrFlowControl,
   DsrRtsFlowControl,
   DsrDtrFlowControl,
   XonXoffFlowControl
};

enum Parity {
```

```
      EvenParity,
      MarkParity,
       NoParity,
       OddParity,
       SpaceParity
   };

   enum StopBits {
      OneStopBit,
      OnePointFiveStopBits,
       TwoStopBits
   };

   // functions...
   //
   #include <windows.h>

   void        serial_Init (void);
   void        serial_Open (short nPort, DWORD dwBaud, short parity, BYTE
DataBits, short stopbits, short fc, BOOL bOverlapped);
   void        serial_Close (void);
   DWORD       serial_Read (void* lpBuf, DWORD dwCount);
   DWORD       serial_Write (const void* lpBuf, DWORD dwCount);
   void        serial_TransmitChar (unsigned char cChar);
   void        serial_GetStatus (COMSTAT* stat);
   void        serial_GetState (DCB* dcb);
   void        serial_SetState (DCB* dcb);

   #endif              //__SERIALPORT_H


   // modem.h : includes function prototypes and macros for the modem driver
   //
   #ifndef __MODEM_H
   #define __MODEM_H

   // funtions...
   //
   short Modem_Response_COM (unsigned char *buf, short tout);

   #endif      // __MODEM_H


   // PPP.h : includes function prototypes and macros for the PPP
implementation
   //

   #ifndef PPP_H
   #define PPP_H
```

```
#define TRUE        1
#define FALSE       0

// escape and end characters
#define ESC         0x7D
#define END         0x7E

// PPP buffer size
#define PPP_BUFF    1506

// LCP packet code definitions
#define CONF_REQ    0x01
#define CONF_ACK    0x02
#define CONF_NAK    0x03
#define CONF_REJ    0x04
#define TERM_REQ    0x05
#define TERM_ACK    0x06
#define IDENT               0x0C

// initial fcs value
#define PPPINITFCS16 0xffff

// good final fcs value
#define PPPGOODFCS16 0xf0b8

#define LITTLE_ENDIAN
// or
// #define BIG_ENDIAN

// protocol definitions for a PPP packet
#if defined (LITTLE_ENDIAN)

#define LCP_PACKET          0x21C0
#define PAP_PACKET          0x23C0
#define IPCP_PACKET         0x2180
#define IP_DATAGRAM         0x2100

#elif defined (BIG_ENDIAN)

#define LCP_PACKET          0xC021
#define PAP_PACKET          0xC023
#define IPCP_PACKET         0x8021
#define IP_DATAGRAM         0x0021

#endif

// data structures
```

```
// a bit-field to hold the status of the PPP link
//
struct PPPStatus {
   unsigned IsACCM:  1;
   unsigned IsESC:           1;
   unsigned ReSync:  1;
   unsigned IsFrame: 1;
   unsigned ByteRx:  1;
   unsigned LinkOn:  1;
};

// two buffers for the PPP input and output frames
// here extern, originally defined in PPP module
//
extern unsigned char InBuffer [PPP_BUFF+1];
extern unsigned char OutBuffer        [PPP_BUFF+1];

// a global bit-field to hold the PPP link status
extern struct PPPStatus PPPStatusByte;


// functions
//
void                                          PPP_Init (void);
static unsigned short        PPP_fcs16 (unsigned short fcs, unsigned char
*cp, short len);
unsigned short                          PPP_GetChecksum (register unsigned
char *cp, register short len);
void                                          PPP_Receive (void);
void                                          PPP_send (unsigned char
*packet, char len);
void                                          IPCP_Handle (void);
void                                          PAP_Handle (void);
void                                          LCP_Handle (void);
void                                          PPP_PAP_send (unsigned char
code, unsigned char id, unsigned char *username, unsigned char *password);
void                                          MYstrncpy (unsigned char
*src, unsigned char *dest, register numBYTEs);
void                                          send_void_LCP (void);
void                                          send_first_LCP (void);
void                                          send_first_IPCP (void);
void                                          PPP_poll (void);
void                                          PPP_Terminate (void);
void                                          PPP_ProtocolReject (void);

#endif      // PPP_H


// IP.h : includes function prototypes and macros for the IP implementation
```

```
//

#ifndef IP_H
#define IP_H

#define ICMP        0x01
#define TCP         0x06

typedef struct {
   unsigned char    Ver_IHL;
   unsigned char    TOS;
   unsigned short   Length;
   unsigned short   ID;
   unsigned short   Flag_Frag;
   unsigned char    TTL;
   unsigned char    Protocol;
   unsigned short   Checksum;
   unsigned char    SrcIP [4];
   unsigned char    DstIP [4];
   unsigned char    Payload [1480];                    // 1500 - IHL
} IPDatagram;

// two arrays to hold the IP address that is assigned to our implementation
// and the IP address of the remote ISP
//
extern unsigned char OurIPAddress [4];
extern unsigned char ISPIPAddress [4];

// two pointers to structure for the IP datagrams
//
extern IPDatagram *IP_Recvd;
extern IPDatagram *IP_2Send;

// functions
//
void                        IP_Init (void);
unsigned short      IP_GetChecksum (register unsigned char *cp, register
short len);
void                        IP_Send  (IPDatagram *Datag2send);
void                        IP_Receive (IPDatagram *DatagRecvd);


#endif      // IP_H


// ICMP.h : includes function prototypes and macros for the ICMP
implementation
//
```

```
#ifndef ICMP_H
#define ICMP_H

#define ECHO_REPLY   0x00
#define ECHO                 0x08

// functions
//
void        ICMP_Ping (unsigned char *RemoteIP);
void        ICMP_Handle       (IPDatagram *DatagRecvd);
void        try_print (void);

#endif      // ICMP_H


// TCP.h : includes function prototypes and macros for the TCP
implementation
//

#ifndef TCP_H
#define TCP_H


typedef struct {
   unsigned short    srcPort;
   unsigned short    dstPort;
   unsigned int      seqNO;
   unsigned int      ackNO;
   unsigned char     dataOFF;
   unsigned char     ctrlBITS;
   unsigned short    wnd;
   unsigned short    chksm;
   unsigned short    urgPNT;
   unsigned char     data [1460];
} TCPSegment;

enum ctrlFLAGS {
   FIN = 0x01,
   SYN = 0x02,
   RST = 0x04,
   PSH = 0x08,
   ACK = 0x10,
   URG = 0x20
};

enum TCPState {
   LISTEN,
   SYN_SENT,
   SYN_RECVD,
```

```
        ESTABLISHED,
        FIN_WAIT_1,
        FIN_WAIT_2,
        CLOSE_WAIT,
        CLOSING,
        LAST_ACK,
        TIME_WAIT,
        CLOSED
    };

    struct TCPTCB {
        unsigned short    lPort;
        unsigned short    rPort;
        unsigned char     rIP [4];
        unsigned char     state;
        unsigned int      lSeq;
        unsigned int      rSeq;
    };

    // two pointers to structures for the input and output TCP segments
    //
    extern TCPSegment *TCP_Recvd;
    extern TCPSegment *TCP_2Send;

    // a structure to hold the TCB of the TCP connection
    //
    extern struct TCPTCB conn;


    // functions
    //
    void                        TCP_Init (void);
    unsigned short      TCP_GetChecksum (unsigned char *IPandTCP);
    void                        TCP_Conn (unsigned char FLAGStoSET);
    void                        TCP_Receive (TCPSegment *SegRecvd);
    void                        TCP_Send (unsigned char *data, unsigned char
len);

    #endif      // TCP_H


    // dbg.h : includes print function prototypes and macros for debugging
    //

    #ifndef DBG_H
    #define DBG_H

    extern char dbg_buff [100];
```

```
// functions
//
void dbg_fileOpen (void);
void dbg_fileClose (void);
void dbg_filePrint (const char *str_print);

#endif      // DBG_H

// superonline.h : defines network to host conversions of the  storage types
//
#ifndef SUPERONLINE1_H
#define SUPERONLINE1_H

#define LITTLE_ENDIAN

// #define BIG_ENDIAN

#if defined(BIG_ENDIAN)

#define htons(A)     (A)
#define htonl(A)     (A)
#define ntohs(A)     (A)
#define ntohl(A)     (A)

#elif defined(LITTLE_ENDIAN)

#define htons(A)     ((((A) & 0xFF00) >> 8) | \
                      (((A) & 0x00FF) << 8))

#define htonl(A)     ((((A) & 0xFF000000) >> 24) | \
                      (((A) & 0x00FF0000) >> 8) | \
                      (((A) & 0x0000FF00) << 8) | \
                      (((A) & 0x000000FF) << 24))

#define ntohs        htons
#define ntohl        htonl

#else

#error "Either LITTLE_ENDIAN or BIG_ENDIAN must be defined!!!"

#endif


#endif      // SUPERONLINE1_H
```