T.C.
PAMUKKALE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

MOBİL CİHAZLAR ARASI ÖNBELLEKLEMEDE SİLİNTİ KODLARININ GELİŞTİRİLMESİ VE ANALİZİ

YÜKSEK LİSANS TEZİ

ERDİ KAYA

T.C.
PAMUKKALE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

MOBİL CİHAZLAR ARASI ÖNBELLEKLEMEDE SİLİNTİ KODLARININ GELİŞTİRİLMESİ VE ANALİZİ

YÜKSEK LİSANS TEZİ

ERDİ KAYA

DENİZLİ, EYLÜL – 2018

KABUL VE ONAY SAYFASI

ERDİ KAYA tarafından hazırlanan "**MOBİL CİHAZLAR ARASI ÖNBELLEKLEMEDE SİLİNTİ KODLARININ GELİŞTİRİLMESİ VE ANALİZİ**" adlı tez çalışmasının savunma sınavı 07.09.2018 tarihinde yapılmış olup aşağıda verilen jüri tarafından oy birliği / oy çokluğu ile Pamukkale Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı Yüksek Lisans Tezi olarak kabul edilmiştir.

Jüri Üyeleri	İmza
Danışman Dr. Öğr. Ü. Elif Haytaoğlu Pamukkale Üniversitesi	.....
Üye Prof. Dr. Sezai Tokat Pamukkale Üniversitesi	.....
Üye Doç. Dr. Tufan Turacı Karabük Üniversitesi	.....

Pamukkale Üniversitesi Fen Bilimleri Enstitüsü Yönetim Kurulu'nun 12/09/2018 tarih ve 37/20 sayılı kararıyla onaylanmıştır.

.....

Prof. Dr. Uğur YÜCEL

Fen Bilimleri Enstitüsü Müdürü

Bu tez çalışması Pamukkale Üniversitesi Bilimsel Araştırma Projeleri Birimi tarafından 2018FEBE009 nolu proje ile desteklenmiştir.

Bu tezin tasarımı, hazırlanması, yürütülmesi, araştırmalarının yapılması ve bulgularının analizlerinde bilimsel etiğe ve akademik kurallara özenle riayet edildiğini; bu çalışmanın doğrudan birincil ürünü olmayan bulguların, verilerin ve materyallerin bilimsel etiğe uygun olarak kaynak gösterildiğini ve alıntı yapılan çalışmalara atfedildiğine beyan ederim.

ERDİ KAYA

ÖZET

MOBİL CİHAZLAR ARASI ÖNBELLEKLEMEDE SİLİNTİ KODLARININ GELİŞTİRİLMESİ VE ANALİZİ

YÜKSEK LİSANS TEZİ
ERDİ KAYA
PAMUKKALE ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

(TEZ DANIŞMANI: DR. ÖĞR. Ü. ELİF HAYTAOĞLU)

DENİZLİ, EYLÜL - 2018

Küresel mobil veri trafiğinin 2020 yılı itibari ile her ay için 30 eksabaytı aşması tahmin edilmektedir ve bu artış 2015 yılındaki trafiğin 10 katı anlamına geliyor. Kablosuz ağ trafiğinde devam eden bu hızlı artış, ağ performansını olumsuz etkileyen önemli bir sorun olarak görülmektedir. Bu trafikten kaynaklı ağda oluşabilecek tıkanıklıkları aşmak için kullanılan temel teknik ise önbelleklemedir. Veri önbelleklemede, ilgili içerik son kullanıcıların cihazlarında paketler halinde dağıtık olarak saklanır. Bir cihaz bu içeriği talep eder ve bu içerikle ilgili veri komşu cihazlarda bulunuyor ise, kendisine en yakın konumdaki bu cihazların önbelleğinde tutulan paketler indirilerek ilgili içerik tedarik edilmiş olur. Böylece ağ trafiğindeki yük baz istasyonu yerine cihazlara dağıtılarak hız ve maliyet iyileştirilmesi sağlanır. Bu noktada, ağ içerisindeki bir cihazın ağdan ayrılması ya da bu cihaz ile bağlantının kopması durumunda kayıp verinin tamiri önemli bir önbellekleme problemidir. Kayıp verinin tamirini minimal iletişim maliyetiyle gerçekleştirmek için çeşitli silinti düzeltme kodları kullanılır. Bu tez çalışması kapsamında, silinti kodlama şemaları olarak literatürde yer alan *MDS* ile Çeşme kodları kayıp verinin tamirinde baz istasyonundan ve düğümlerden çekilen veri sembolü sayısı, tamir süreleri açısından analiz edilerek karşılaştırılmıştır. Bu amaçla genel bir simülatör geliştirilerek farklı silinti kodlarının kolayca test edilebilmesi sağlanmıştır. Ayrıca her düğümün farklı büyüklüklerde veri tutmasına imkan veren ve daha gerçekçi bir yaklaşım olan artık veri kullanımı da gerçekleştirilmiştir. *MDS* kodu olarak *Reed-Solomon* kod, Çeşme kod olarak ise dış kodu *LDPC*, iç kodu *LT* olan *Raptor* kod kullanılmıştır. Ayni zamanda *LDPC* kodu da tamir sürecindeki performansı açısından diğer silinti düzeltme kodları ile karşılaştırılmıştır. Yapılan simülasyon sonuçlarına göre, iletişim maliyeti açısından baz istasyonundan ve düğümlerden çekilen sembol sayıları *Reed-Solomon* ve *Raptor* kodları için birbirine çok yakın çıkmıştır. Tamir sürecinde ise *Reed-Solomon* kod en uzun tamir süresine sahip olur iken, *LDPC* kodu ise en kısa tamir süresine sahip olarak gözlemlenmiştir. Bu çalışma kapsamında, LDPC tamir sürecinde hem tek düğüm hem de çoklu düğüm tamiri gerçekleştirilmiştir. Aynı zamanda, D2D iletişimde daha önce denenmemiş olan Çeşme kodları diğer silinti düzeltme kodları ile karşılaştırılmıştır.

ANAHTAR KELİMELER: Cihazlar Arası (D2D) İletişim, Veri Önbellekleme, Dağıtık Depolama, Silinti Düzeltme Kodları, Kodlanmış Veri Önbellekleme

ABSTRACT

DEVELOPMENT AND ANALYSIS OF ERASURE CODES IN CACHING BETWEEN MOBILE DEVICES

MSC THESIS
ERDİ KAYA
PAMUKKALE UNIVERSITY INSTITUTE OF SCIENCE
COMPUTER ENGINEERING
(SUPERVISOR: ASST. PROF. ELİF HAYTAOĞLU)

DENİZLİ, SEPTEMBER 2018

Global mobile data traffic is estimated to exceed 30 exabytes per month by 2020, an increase of 10 times that of 2015 traffic. This rapid increase in wireless network traffic is seen as a major problem that negatively affects the network performance. The basic technique used to overcome the bottlenecks that may arise due to this traffic is data caching. In data caching, the related content is stored in packets on end users' devices. When a device requests this content and the data related to this content is present in the neighboring devices, the packets stored in the closest device are downloaded and the related content is supplied. Thus, the load on the network traffic is distributed to the devices instead of the base station, thereby improving the speed and the cost. At this point, if the device in the network is disconnected from the network or if the connection with the device is broken, the loss of the redundancy is an important problem of caching. Various erasure correction codes are used to perform the repair of the lost data with minimal communication cost. Within the scope of this thesis, MDS and Fountain codes in the literature as erasure coding schemes has been analyzed and compared in terms of number of symbols downloaded from base station and storage nodes during repair process and repair times. For this purpose, a general simulator has been developed so that different erasure codes can be easily tested. Furthermore, the use of residual data, which allows each node to retain data at different sizes and is a more realistic approach has been carried out. For *MDS* and *Fountain* codes respectively, *Raptor* code that consists of *LDPC* code as outer code and *LT* code as inner code and *Reed-Solomon* code are used in this study. Also, *LDPC* code was compared with other erasure correction codes in terms of performance in the repair process. According to the simulation results, symbol numbers downloaded from base station and storage nodes in terms of communication cost are very close to each other for *Reed-Solomon* and *Raptor* codes. In the repair process, *Reed-Solomon* code has the longest repair time while *LDPC* code has the shortest repair time. In this study, both single and multiple node repairs were performed in the LDPC repair process. Also, Fountain codes, which have not been tried before in D2D communication, are compared with other erasure correction codes.

KEYWORDS: Device-to-Device (D2D) Communications, Data Caching, Distributed Storage, Error Correction Codes, Coded Caching

İÇİNDEKİLER

Sayfa

ŞEKİL LİSTESİ

Sayfa

TABLO LİSTESİ

Sayfa

KISALTMALAR LİSTESİ

LDPC : Seyrek Eşlik Kontrolü (Low Density Parity Check)

LT : Luby-Transform

MDS : Maximum Distance Separable

RS : Reed-Solomon

BHC : Bose–Chaudhuri–Hocquenghem

LRC : Yerel Tamiredilebilir Kodlar (Local Repairable Codes)

IoT : Nesnelerin İnterneti (Internet of Things)

D2D : Cihazlar Arası (Device-to-Device)

FEC : Gönderim Yönünde Hata Düzeltimi (Forward Error Correction)

KB : Kilobayt (Kilobyte)

ÖNSÖZ

Yüksek lisans tezimin her aşamasında beni yönlendiren, zamanını ve desteğini esirgemeyerek her zaman çalışmamla yakından ilgilenen tez danışmanım sayın Dr. Öğr. Ü. Elif HAYTAOĞLU'na sonsuz teşekkürü bir borç bilirim.

Yüksek lisans tezim boyunca bilgilerini esirgemeyerek çalışmama destek olan sayın Doç. Dr. Şuayb Ş. ARSLAN'a ve Prof. Dr. Serdar İPLİKÇİ'ye teşekkür ederim.

Yüksek lisans eğitimim süresince yardımcı olan çalışma arkadaşlarıma ve aileme teşekkür ederim.

1. GİRİŞ

Klasik hücresel ağlar, her biri baz istasyonu olarak bilinen sabit bir konum alıcı-vericisi içeren hücrelerden oluşan ağlardır ve bu baz istasyonları, hücrenin ses ve veri aktarımı için kullanılabilen şebeke kapsama alanını sağlar. Bu mobil ağlarda, hücreler biraraya gelerek daha büyük coğrafi alanlara yayın sağlar. Öyle ki cep telefonları gibi mobil cihazlar hücreler arasında hareket halinde iken bile iletişim kurmaya devam edebilir. Kısacası, klasik bir hücresel ağda tüm iletişim baz istasyonu aracılığıyla gerçekleştirilir.

Cihazlar arası (*D2D*) ağlarda ise baz istasyonları aradan çıkarılarak iki mobil kullanıcı arasında doğrudan iletişim sağlanır (Asadi ve diğ. 2014). *D2D* iletişim ile verilen bant genişliği üzerinden aktarılan veri oranını ifade eden spektral verimlilik artarken iletişim gecikmesinde de azalma gözlemlenmiştir (Asadi ve diğ. 2014). Diğer bir deyişle, *D2D* iletişim ağ performansını artıran yeni bir model olarak ortaya çıkmıştır.

D2D iletişimde içerik dağıtımı için kullanılan teknik ise önbelleklemedir. Yani cihazların baz istasyonlarını aradan çıkararak birbirleriyle doğrudan iletişim kurabilmeleri için baz istasyonlarındaki verinin bu cihazların önbelleklerinde saklanması gerekir. Bu nedenle *D2D* iletişim dağıtık depolama kavramı ile güçlü bağlara sahiptir (Pedersen ve diğ. 2016).

İlgili içeriği önbelleklerinde tutan mobil cihaz kümesi önbellekleme ağına benzetilebilir. *D2D* bir ağ içerisindeki hücrelere cihazlar rastgele olarak girip çıkar ve yine rastgele olarak ilgili içerik talebinde bulunabilir. Bu nedenle dağıtık depolamada verinin dayanıklı bir şekilde saklanması önemli bir sorundur. Bu noktada silinti düzeltme kodları, dayanıklı veri depolanmasında kolaylık sağlar ve dağıtık veri depolama performansını iyileştirebilir (Weatherspoon ve Kubiatowicz, 2002).

D2D iletişimin uygulama alanlarını, nesnelerin interneti (*Internet of Things*, kısaca *IoT*) olarak adlandırılan fiziksel nesnelerin birbirleriyle ya da daha büyük sistemlerle bağlantılı olduğu iletişim ağlarında görmek mümkündür. Son yıllarda popüler hale gelen nesnelerin interneti kavramıyla beraber cihazlar arası (*D2D*)

iletişim, akıllı sistemler temelinde birçok uygulama alanında kullanılmaktadır. *D2D* iletişim trafik, park yeri, ışıklandırma, atık yönetimi için akıllı şehir uygulamalarında; hava kirliliği, yağış durumu, baraj doluluğu, orman yangını gibi çevresel faktörlerin gözlenmesi için akıllı çevre uygulamalarında ve akıllı ev ve tedarik uygulamalarında görülebilir (Niyato ve diğ. 2011).

Bu tez kapsamında, klasik ağlarda kullanılan MDS kodları ile Çeşme kodlar D2D iletişimde tamir işlemleri açısından test edilerek karşılaştırılmıştır. Bu tezin özgün tarafı, daha önce dağıtık önbellekleme (*distributed caching*) alanında hiç kullanılmayan Çeşme kodları ilk defa bu çalışma ile kullanılmıştır. Bu açıdan literatüre önemli bir katkı sağlanması amaçlanmıştır. Ayrıca optimal silinti kodlarını kendi aralarında karşılaştıran çalışmalar olsa da bu karşılaştırmalara bir Çeşme kodun (*LT* ya da *Raptor*) dahil edildiği herhangi bir çalışma bulunmamaktadır. Bu nedenle bu tezin diğer bir özgünlüğü *MDS* kodları gibi optimal silinti kodlarının hücresel ağlarda önbellekleme alanında ilk defa bir Çeşme kod ile karşılaştırılmış olmasıdır. Paakkonen ve arkadaşları tarafından yapılan çalışmada *D2D* iletişim ile veri paketlerinin düğümlere dağıtılmasında enerji tüketim maliyeti incelenmiştir (Paakkönen ve diğ. 2013). Tekrarlama (*replication*) tekniğinin yenileme (*regenerating*) kodlarının aksine yedekleme (*redundancy*) işlemi için enerji tüketim maliyeti açısından daha iyi performans gösterdiği belirtilmiştir. Bir başka çalışmada, Pedersen ve arkadaşları kablosuz ağlarda hücre içerisindeki düğümlerde veri önbellekleme üzerinden düğüm tamirlerini silinti düzeltme kodları ile test etmişlerdir (Pedersen ve diğ. 2016). Bu çalışmada (Pedersen 2016) iyileştirilmek istenen maliyet parametresi, kablosuz ağlar için dağıtık depolamada ortaya çıkan iletişim maliyetidir ve odak noktası da verilerin depolandığı bir cihazın hücreden ayrılması ile oluşan tamir (*repair*) problemidir. Bu çalışmada (Pedersen 2016) *MDS* kodları, yenileme (*regenerating*) kodları ve yerel olarak tamir edilebilir kodlar (*Locally Repairable Codes*, kısaca *LRC*) iletişim maliyeti açısından karşılaştırılmıştır. Bu iki çalışmadan yola çıkarak Çeşme kodların da tamir işlemleri için bu karşılaştırmalara dahil edilebileceği düşünülmüştür.

Bu tez çalışmasının ikinci kısmında bu çalışma kapsamında kullanılan silinti düzeltme kodları anlatılmıştır. Üçüncü kısımda ise cihazlar arası (*D2D*) iletişim ve kodlama ile veri önbellekleme (*coded caching*) konuları hakkında bilgi verilmiştir. Dördüncü kısımda bu tez çalışması kapsamında yapılan çalışmalar, beşinci kısımda da

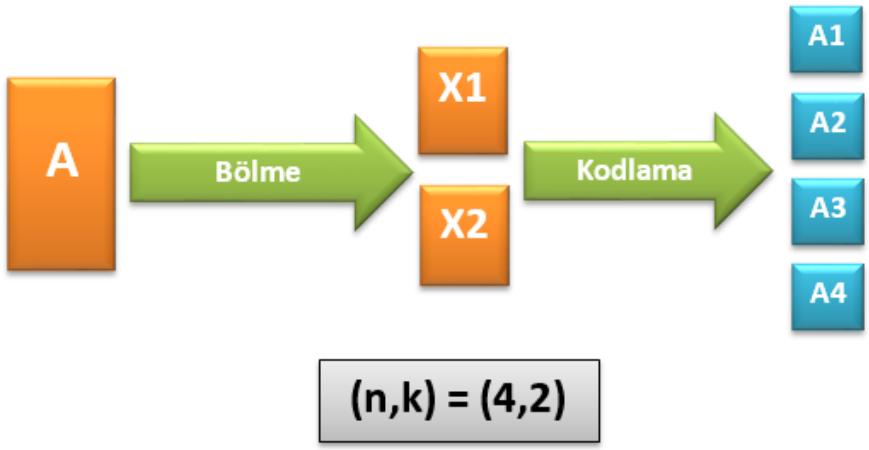
simülasyon sonuçları verilmiştir. Son kısımda ise sonuçlar ve gelecek çalışmalara dair öngörüler bildirilmiştir.

2. SİLİNTİ DÜZELTME KODLARI

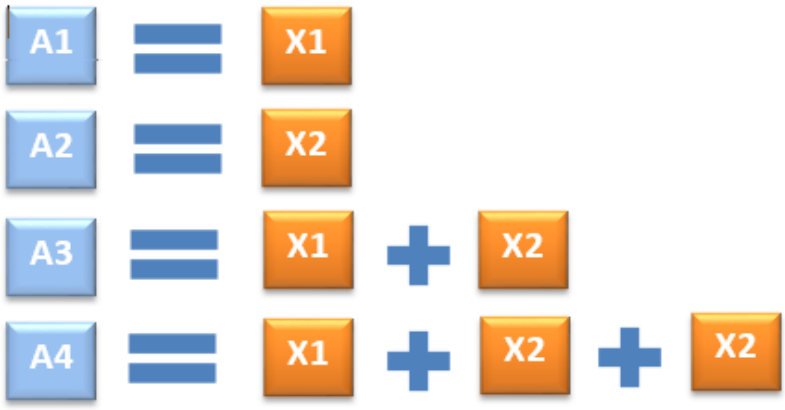
Kodlama teorisinde geçen silinti düzeltme kodları, iletilen veride gürültülü iletişim kanalları (*noisy communication channels*) üzerinde yaşanabilecek aksaklıklar sonucu oluşan hataları tespit edip düzeltmek için kullanılan kodlardır. Silinti düzeltme kodlarındaki temel mantık, iletilecek olan verinin ilgili silinti düzeltme koduna göre yedeklenmesine dayanır. Diğer bir deyişle iletilecek olan mesaj, silinti düzeltme kodunun kendi yöntemlerine göre kodlanır ve içerisinde ilgili mesaj verisine ait paketlerin yedeklenerek tutulduğu ve orijinal veriye göre genişletilmiş olan yeni veri elde edilir. Bu yedekleme özelliği sayesinde mesajın herhangi bir konumunda yaşanacak belli bir sayıdaki hatalar tespit edilip düzeltilebilir. Silinti düzeltme kodları sayesinde verinin yeniden iletimine gerek kalmadan ilgili hata ya da hatalar düzeltilebilir. Bu özelliği sayesinde silinti düzeltme kodları, yeniden iletimin çoklu gönderim (*multicast*) gibi maliyetli ya da tek yönlü iletişim kanallarındaki gibi imkansız olduğu durumlarda büyük bir avantaj sağlamaktadır. Silinti düzeltme kodları ilk olarak 1940'larda Amerikalı matematikçi *Richard Hamming* tarafından geliştirilmeye başlanmıştır. 1950 yılında da *Hamming*, ilk silinti düzeltme kodunu geliştirmiştir (Hamming 1950).

Kodlama teorisinde ileri hata düzeltme (*Forward Error Correction – FEC*) tekniği olarak tanımlanan silinti düzeltme kodlarında (*erasure correction codes*), verinin hata toleranslı bir şekilde depolanmasını ifade eden yedekleme işlemi iletilen verinin kodlanmasıyla (*encoding*) gerçekleştirilebileceği gibi kodlanmamış olarak da sağlanabilir. Aynı zamanda bu yedekleme sürecinde eğer iletilen veri aynı şekilde korunarak kodlanmış ise sistematik, aynı şekilde korunmadan kodlanmış ise de sistematik olmayan (*non-systematic*) kodlama yapılmış demektir. Tekrarlama (*repetition*) mantığına dayanan kodlar, en basit silinti düzeltme kodlarından biridir. Bu silinti düzeltme kodlarında iletilen orijinal veri hiçbir kodlama işlemine tabi tutulmaz. Sadece aynı veriye fazladan aynı şekilde tekrar kopyalanarak yedeklenir. Öte yandan kodlama tekniğine dayanan *MDS* gibi silinti düzeltme kodunun temel çalışma mantığı şu şekildedir: Bir dosya (mesaj) k tane veri parçasına (paket) bölünür ve ardından bu k veri parçası kodlanarak n tane kodlanmış veri parçası elde edilir. Diğer bir deyişle, k parçadan oluşan ilgili dosya, n parçadan oluşan daha büyük bir veriye dönüştürülür. Öyle ki bu n parçanın bir alt kümesinden orijinal dosya yeniden oluşturulabilir. Silinti

düzeltme kodlarının temel mantığı göndericinin bir silinti düzeltme kodu kullanarak ilgili dosyanın yedekli kodlanmasına dayanır. İletişim kanalları üzerinden veri iletimi esnasında bu veri parçalarında hata ortaya çıktığında eldeki fazladan kodlanmış veri çözülerek (*decoding*) kayıp veri parçası ya da parçaları yeniden elde edilebilir. *MDS* sınıfından bir silinti kodu tipik olarak *(n, k)* terminolojisi ile tanımlanır. Burada *n* kodlanmış sembol sayısıdır, *k* ise dosyanın ilk halinin elde edilmesi için gerekli olan paket sayısıdır. Şekil 2.1, 2.2 ve 2.3'te silinti kodlamanın nasıl çalıştığı gösterilmektedir:



Şekil 2.1: A dosyası 2 pakete (X1, X2) bölündükten sonra bu veri parçalarından A1, A2, A3, A4 kodlanmış parçaları elde ediliyor.



Şekil 2.2: Kodlanmış parçaların içerikleri.

Şekil 2.3: Bu kodlanmış parçaların düğümlere dağıtılması.

k adet sembole bölünmüş olan bir kaynak veri üzerinden çalıştırılan bir silinti düzeltme kodunda λ, ilgili kodun yükünü (*overhead*) ifade eder. Herhangi bir $k(1+\lambda)$ adet kodlanmış sembol (*code symbols*), *k* adet sembolden oluşan kaynak veriyi yeniden oluşturmak için yeterli olmaktadır (Gummadi ve Sreenivas, 2011).

Bu noktada, silinti kodlar kendi içerisinde λ olarak ifade edilen yükün değerine göre optimal ve optimale yakın (*near-optimal*) silinti kodlar olarak ikiye ayrılır. Optimal silinti kodlarda orijinal mesajı yeniden elde edebilmek için *n* tane sembolün herhangi bir *k* tanesi teorik olarak en az depolama ile yeterlidir. Diğer bir deyişle, yük sıfıra eşit ($\lambda = 0$) olduğu zaman ilgili silinti kod optimal olarak nitelendirilir. Optimal silinti kodlara *MDS* (*Maximum Distance Separable*) kodları örnek olarak verilebilir. Bu tez çalışmasında da kullanılmış olan *Reed-Solomon (RS)* kodları *MDS* kodlarındandır.

Optimale yakın silinti kodlar ise orijinal mesajı yeniden oluşturabilmek için $k(1+\lambda)$ adet sembole ihtiyaç duyar (burada $\lambda > 0$). kodlar optimale yakın silinti kodların dikkat çeken örneklerindendir. kodlar, *k* tane sembolden oluşan bir mesajı teorik olarak sonsuz sayıda kodlanmış sembol içeren bir biçime dönüştürebilir. Diğer bir deyişle, orijinal mesajdan oluşturulan kodlanmış paket sayısı sınırsız olabilmektedir. Bu özelliklerinden ötürü, kodlar oransız (*rateless*) olarak kabul edilir. kodların bilinen örnekleri ise *Luby-Transform* (*LT*) kodlar (Luby 2002) ile bu tezde de kullanılacak olan *Raptor* kodlardır (Shokrollahi 2006). *Raptor* kodlar, *LDPC* (*Low Density Parity Check*) (Gallager 1962) gibi oransız özelliği taşımayan silinti kodlardan biri (*precode* ya da *outer code*) ile *LT* kodun (*inner code*) birleşiminden elde edilir. Silinti kodlar tablo 2.1'de verildiği gibi sınıflandırılabilir.

Tablo 2.1: Silinti Kodların Sınıflandırılması

Optimale Yakın Silinti Kodlar	**Optimale Yakın Çeşme (Oransız) Silinti Kodlar**	**Optimal Silinti Kodlar**
LDPC Kodlar (Gallager 1962)	Çeşme Kodlar (MacKay 2005)	*MDS* Kodları
Tornado Kodlar (Luby 1999)	*LT* Kodlar (Luby 2002)	*Reed-Solomon* Kodlar (Reed ve Solomon, 1960)
	Raptor Kodlar (Shokrollahi 2006)	
	Çevrimiçi (*Online*) Kodlar (Cassuto ve Shokrollahi, 2015)	

Silinti düzeltme kodları sistematik ve sistematik olmayan şeklinde ikiye ayrılır. Orijinal mesaj sembollerinin kodlanmış sembollerin bir parçası olduğu sistematik form, bir depolama biriminden kod çözmeden mesaj sembollerinin okunmasını sağlar.

Optimal silinti kodlarından olan yenileme kodları, literatürde ikiye ayrılmaktadır: 1. Minimum Depolama Yenileme Kodları (*MSR*), 2. Minimum Bant Genişliği Yenileme Kodları (*MBR*). *MSR* kodları depolama maliyetini düşürürken, *MBR* kodları ise bant genişliği harcamasını düşürmektedir. Dimakis (Dimakis ve diğ. 2010), yaptığı çalışma ile düğüm tamiri konusunda yenileme kodlarının klasik *MDS* kodlarında görülen verimsizlikleri giderdiğini açıklamıştır.

Silinti kodlarının ikinci tipi olan optimale yakın silinti kodlarının öncülü ise katmanlı bir yaklaşıma sahip olan *Tornado* kodlarıdır (Luby 1999). Son katmanı dışındaki diğer katmanlarında hızlı ama başarısızlık ihtimali de olan *LDPC* hata düzeltme kodu kullanılır iken son katmanında ise *LDPC*'ye göre daha yavaş ama optimal çalışan *Reed-Solomon* kodu kullanılmaktadır.

Optimale yakın silinti kodlarının ikinci kolu ise Çeşme (*Fountain)* kodlardır. MacKay (MacKay 2005) çalışmasında Çeşme kodlarının oransızlık (*rateless*) özelliğini *k* tane sembolden oluşan bir mesajı teorik olarak sonsuz sayıda kodlanmış sembol içeren bir biçime dönüştürebilme olarak açıklamıştır. Yine bu çalışmada Çeşme kodlar üç gruba ayrılmıştır: Rastgele Lineer Çeşme kodları, *LT* kodları ve *Raptor* kodları. Luby'nin çalışmasında Çeşme kod sınıfından olan ve kendisinin geliştirdiği *LT* kodlar tanıtılmıştır (Luby 2002). Çeşme kodlar seyrek iki bölümlü çizgelere (*bipartite graph*) dayanır. En belirgin özelliği ise belli bir mesajı kodlama ve kod çözme (*decoding*) işlemlerinde exclusive or (*XOR*) operasyonu temelli basit bir algoritma kullanmasıdır. *LT* kodlar, rastgele lineer çeşme kodlara göre şifreleme ve çözme karmaşıklıklarını düşürmektedir. Çeşme kodların son türü olan *Raptor* kodlar ise *Shokrollahi* tarafından yapılan çalışmada önerilmiştir (Shokrollahi 2006). *Raptor* kodlar, kodlama ve kod çözme işlemleri için bir *LT* kod ile *LT* kodda gözlemlenen eksiklikleri gideren bir dış kodu birleştirir. *Shokrollahi* çalışmasında *Raptor* kodların *LT* kodlardan daha düşük kodlama ve çözme maliyetine sahip olduğunu göstermiştir.

2.1 LDPC Kodları

LDPC (Seyrek Eşlik Kontrolü) kod, 1960 yılında Massachusetts Teknoloji Enstitüsü'ndeki doktora çalışması kapsamında R. G. Gallager tarafından geliştirilen optimale yakın bir silinti düzeltme kodudur (Gallager 1962). Bu kodun uygulanması dönemin bilgisayar donanımı açısından pratik olmadığından başlangıçta yeterli ilgiyi görmemiştir. 1993 yılında *LDPC* kodlar gibi Shannon kapasitesine (*Shannon limit*) yakın çalışan Turbo kodlar keşfedilmiştir ve *LDPC* kodlar ile performans açısından karşılaştırılarak incelenmiştir. Yüksek kod oranlarında (*code rate*) *LDPC* kodlar Turbo kodlardan daha performanslı çalışırken, düşük kod oranlarında da Turbo kodların *LDPC* kodlardan daha performanslı çalıştığı gözlemlenmiştir (NASA Deep Space Network, 2013). Böylece 1996 yılından itibaren *LDPC* kodlar yeniden ilgi çekmiştir.

LDPC kodlar, 2003 yılından itibaren Turbo kodlara göre daha düşük kod çözme karmaşıklığı göstermesi nedeniyle uydudan dijital televizyon yayın standardı *DVB-S2* (Morello ve Mignone, 2006)'de kullanılmaya başlanmıştır. *LDPC* kodlar aynı zamanda saniyede 10 gigabit veri gönderen *10GBase-T Ethernet* standardı üzerinde de

kullanılmaktadır. 2009 yılından itibaren de *Wi-Fi 802.11* standardında kullanılmaya başlanmıştır (IEEE Computer Society, 2012).

LDPC kodlar seyrek bir eşlik kontrol matrisi (sparse *parity-check matrix*) tarafından oluşturulur. Eşlik kontrol matrisi kodlanmış veri (*codeword*) bileşenlerinin doğrusal ilişkilerini tanımlayan bir matristir ve herhangi bir satırdaki bütün değerler eşlik kontrol denklemlerinin (*parity-check equations*) katsayılarını ifade eder. Diğer bir deyişle eşlik kontrol matrisindeki her satır bir eşlik kontrol denklemine karşılık gelir. Seyrek ifadesi ile eşlik kontrolü matrisindeki "1" sayısının seyrekliği ifade edilmektedir. LDPC kod iki bölmeli çizge kullanılarak elde edilmektedir.

Eğer matris içerisindeki her bir sütunda aynı sayıda 1 değeri varken aynı zamanda her satır içerisinde de yine sabit bir sayıda 1 var ise bu *LDPC* kod düzenli bir kod olarak kabul edilir. Eğer her satırdaki ya da her sütundaki 1 sayısı değişkenlik gösteriyorsa da bu *LDPC* kod düzensiz (*irregular*) kabul edilir. Bu tez kapsamında ise düzensiz bir *LDPC* kod kullanılmıştır.

LDPC kodları, iki bölmeli çizge temelinde, seyrek Tanner çizgeleri (*Tanner graph*) kullanılarak oluşturulur.

2.1.1 Kodlama

LDPC kod için kodlama işleminde ilk olarak eşlik kontrol matrisinin oluşturulması gerekmektedir. Gallager tarafından sunulan *(N, j, k)* terminolojisi ile Bu tez çalışmasında *Gallager*'ın 1960 tarihinde yayımladığı orijinal *LDPC* kod versiyonu kullanılmıştır. İlk olarak eşlik kontrol matrisi elde edilmiştir. Bu matrisi elde etmek için *Gallager* tarafından belirtilmiş olan *(N, j, k)* terminolojisi kullanılmıştır. Daha açık bir şekilde anlatılırsa, *N* matristeki blok sayısını, *j* matrisin her sütunundaki 1 değerlerinin sayısını, *k* ise matrisin her satırındaki 1 değerlerinin sayısını ifade eder.

Bu noktada $j \geq 3$ ve $k > j$ koşullarının sağlanması gerekmektedir. Ardından ise her sütununda sadece bir adet 1 değeri olacak şekilde *j* adet alt matris oluşturulur. Bu aşamada ilk alt matris için 1 değerleri azalan sırada bütün olarak yani araya 0 değeri almadan yerleştirilir. Diğer bir deyişle *i*. satıra $((i-1)\times k+1)$. sütundan $(i\times k)$. sütuna

kadar 1'ler yerleştirilir. Geriye kalan $j - 1$ adet alt matris ise oluşturulan bu ilk alt matrisin rastgele sütun permütasyonlarından oluşturulur. Bu işlemler sonucunda *H* olarak adlandırılan seyrek eşlik kontrol matrisi elde edilmiş olur.

Eşlik kontrol matrisi *H*'nin elde edilmesinin ardından bu *H* matrisinin sistematik hale dönüştürülme işlemi gerçekleştirilir. Sistematik H_{sys} matrisinden kastedilen ise matris içerisinde birim matris *I*'nın oluşturulmasıdır. Bu birim matris sayesinde de eşlik kontrolü gerçekleştirilebilir hale gelir. Bu noktada *Gauss* eleme yönteminden faydalanılmıştır. *Gauss* eleme yöntemi içerisindeki temel satır işlemleri sayesinde $(n - m)\ x\ m$ boyutlarındaki ilgili *H* matrisi sistematik hale dönüştürülür: $H_{sys} = [P^T | I_{n-m}]$.

Bir sonraki aşamada ise $(n - m)\ x\ m$ boyutlarındaki H_{sys} matrisi $m\ x\ n$ boyutlarındaki üreteç matris *G*'ye dönüştürülür: $G = [I_m | P]$.

Son olarak da elde edilen üreteç *G* matrisinin bileşenleri *m* sembolden oluşan verinin sembolleri ile *XOR* işlemlerine tabi tutularak kodlama işlemi gerçekleştirilir. Bu noktada eldeki *m* sembolden ne kadar daha fazla kodlanmış sembolün üretileceği $(n - m)$ kodlama işlemi öncesinde belirlenen kod oranına göre belirlenir ve bu kod oranı ilgili silinti düzeltme kodunun performansını etkileyen önemli bir parametredir.

2.1.2 Kod Çözme

Kodlanmış bir verinin içerisinde herhangi bir ya da birden fazla sembolün bozulması halinde kayıp sembol ya da sembollerin yeniden oluşturulma sürecine kod çözme (*decoding*) denir. Kod çözme işleminde kullanılan genel algoritma ise inanç yayılma (*belief propagation*) algoritması, mesaj aktarma (*message passing*) algoritması, toplam-çarpım (*sum-product*) algoritması gibi farklı isimlerle adlandırılmaktadır.

İnanç yayılma algoritmaları LDPC için olduğu gibi LT kodlarında da kod çözmek için kullanılmaktadır. Bu algoritmayı anlamak için ilk olarak *hard-decision* (Leiner 2005) daha sonra da daha performanslı çalışan *soft-decision* kod çözme yöntemleri anlatılacaktır.

2.1.2.1 Hard-Decision

Hatasız olarak kodlanan verinin $\mathbf{c} = [1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1]$ olarak kabul edildiği bir senaryoda ikinci bit 0 yerine 1 olarak gelirse alınan hatalı veri $\mathbf{c} = [1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1]$ olacak ve bu hatalı bit yeniden elde edilerek düzeltilmeye çalışılacaktır.

(n,k) = (8,4) parametrelerine sahip bir LDPC kod için *hard-decision* kod çözme yönteminde ilk olarak tüm değişken düğümler, kendileri için doğru olduğuna inandıkları biti içeren bir "mesajı" kontrol düğümlerine gönderir. Bu aşamada herhangi bir değişken düğümü v_i'nin sahip olduğu tek bilgi, karşılık gelen *i. v* bitidir (y_i). Örnek vermek gerekirse şekil 2.4'te belirtilen LDPC kod matrisine göre v_1 , c_2 ve c_4'e 1 değerini içeren mesaj gönderir iken v_2 ise c_1 ve c_2'ye 1 değerini içeren mesaj gönderir.

$$H = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}$$

Şekil 2.4: *(n,k)* = (8,4) parametrelerine sahip bir eşlik kontrolü matrisi örneği (Leiner, 2005).

İkinci aşamada ise her kontrol düğümü c_j, her bağlı değişken düğümüne bir cevap hesaplar. Yanıt mesajı, c_j'ye bağlanan diğer değişken düğümlerin doğru olduğu varsayılarak yapılan hesaplamalar sonucunda, değişken düğüm v_i için doğru olduğuna inanılan biti içerir. Buradaki örneğe göre, her kontrol düğümü c_j dört değişken düğümüne bağlanır. Dolayısıyla bir kontrol düğümü c_j, üç değişken düğümünden alınan mesaja bakar ve eşlik kontrol denklemini (*parity check equation*) yerine getirmek için dördüncü değişken düğümün sahip olması gereken biti hesaplar. Tablo 2.2 bu ikinci aşamayı göstermektedir.

Tablo 2.2: *Hard-decision* kod çözmede ikinci aşama

Kontrol Düğümleri	Alınan / Gönderilen				
c_1	Alınan:	$v_2 = 1$	$v_4 = 1$	$v_5 = 0$	$v_8 = 1$
	Gönderilen:	$v_2 = 0$	$v_4 = 0$	$v_5 = 1$	$v_8 = 0$
c_2	Alınan:	$v_1 = 1$	$v_2 = 1$	$v_3 = 0$	$v_6 = 1$
	Gönderilen:	$v_1 = 0$	$v_2 = 0$	$v_3 = 1$	$v_6 = 0$
c_3	Alınan:	$v_3 = 0$	$v_6 = 1$	$v_7 = 0$	$v_8 = 1$
	Gönderilen:	$v_3 = 0$	$v_6 = 1$	$v_7 = 0$	$v_8 = 1$
c_4	Alınan:	$v_1 = 1$	$v_4 = 1$	$v_5 = 0$	$v_7 = 0$
	Gönderilen:	$v_1 = 1$	$v_4 = 1$	$v_5 = 0$	$v_7 = 0$

Üçüncü aşamada değişken düğümleri kontrol düğümlerinden gelen mesajları alıp bu ek bilgiyi başlangıçta alınan bitin doğru olup olmadığını belirlemek için kullanır. Bunu belirlemek için de çoğunluk oylamasına başvurulur. Yukarıdaki örnek üzerinden devam edilir ise bir değişken düğümünün kendi bit değeri ile ilgili üç bilgi kaynağı bulunmaktadır: alınan bit değeri ve kontrol düğümlerinden gelen iki diğer bit değeri önerisi. Bu aşamanın ardından değişken düğümleri çoğunluk oylaması ile doğru olduğuna karar verilen (*hard-decision*) bit değerini kontrol düğümlerine yeni bir mesaj olarak gönderir. Tablo 2.3 bu aşamayı göstermektedir.

Tablo 2.3: *Hard-decision* kod çözmede üçüncü aşama

Değişken Düğümü	Alınan Mesaj (y_i)	Kontrol Düğümlerinden Mesajlar	Karar
v_1	1	$c_2 = 0$ $c_4 = 1$	1
v_2	1	$c_1 = 0$ $c_2 = 0$	0
v_3	0	$c_2 = 1$ $c_3 = 0$	0
v_4	1	$c_1 = 0$ $c_4 = 1$	1
v_5	0	$c_1 = 1$ $c_4 = 0$	0
v_6	1	$c_2 = 0$ $c_3 = 1$	1
v_7	0	$c_3 = 0$ $c_4 = 0$	0
v_8	1	$c_1 = 1$ $c_3 = 1$	1

Dördüncü aşamada ise ikinci aşamaya giderek süreç doğru kararlara ulaşılana kadar tekrar edilir.

2.1.2.2 Soft-Decision

İnanç yayılma algoritması temelli olan *Soft-Decision* kod çözme yöntemi daha sistematik olmasının verdiği avantaj ile *Hard-Decision* yönteminden daha iyi performans göstermektedir. Soft-Decision yöntemini anlatmadan önce önemli formüller aşağıdaki gibidir.

$$P_i = Pr(c_i = 1 \mid y_i) \tag{2.1}$$

q_{ij} = *değişken düğümü* v_i*'den kontrol düğümü* c_j*'ye gönderilen bir mesaj* (2.2)

r_{ji} = *kontrol düğümü* c_j*'den değişken düğümü* v_i*'ye gönderilen bir mesaj* (2.3)

Birinci aşamada bütün değişken düğümleri onların q_{ij} mesajlarını kontrol düğümlerine gönderir. Bu noktada gönderilen mesajların bit değerinin "1" ya da "0" olma olasılığı şu şekilde formülize edilir.

$$q_{ij}(1) = P_i \tag{2.4}$$

$$q_{ij}(0) = 1 - P_i \tag{2.5}$$

İkinci aşamada kontrol düğümleri kendi cevap mesajları olan r_{ji}'leri hesaplar (Leiner 2005). Aşağıda sırasıyla bu cevap mesajlarının "0" ve "1" olma durumlarındaki formülleri verilmiştir.

$$r_{ji}(0) = \frac{1}{2} + \frac{1}{2} \prod_{i' \epsilon V_j \backslash i} (1 - 2q_{i'j}(1)) \tag{2.6}$$

$$r_{ji}(1) = 1 - r_{ji}(0) \tag{2.7}$$

v_i'nin "0" olma olasılığı olan $r_{ji}(0)$, v_i haricindeki diğer değişken düğümlerinin değerlerinde "1" değerlerinin çift sayıda olma olasılığına karşılık gelir.

Üçüncü aşamada değişken düğümleri kontrol düğümlerine cevap mesajlarını aşağıdaki formüller üzerinden günceller.

$$q_{ij}(0) = K_{ij}(1 - P_i) \prod_{j' \epsilon C_i \backslash j} r_{j'i}(0) \tag{2.8}$$

$$q_{ij}(1) = K_{ij} P_i \prod_{j' \epsilon C_i \backslash j} r_{j'i}(1) \tag{2.9}$$

Buradaki *Konstants* K_{ij}, $q_{ij}(0) + q_{ij}(1) = 1$ temel olasılık kuralı çerçevesinde seçilir. Bu noktada, değişken düğümleri değişken v_i'nin mevcut tahmini $\widehat{v_i}'yi$ günceller. Aşağıdaki denklemler ile "0" ve "1" gelme olasılığı hesaplanır ve büyük değere sahip olan doğru değer olarak seçilir.

$$Q_i(0) = K_{ij}(1 - P_i) \prod_{j \epsilon C_i} r_{ji}(0) \tag{2.10}$$

$$Q_i(1) = K_{ij} P_i \prod_{j \epsilon C_i} r_{ji}(1) \tag{2.11}$$

$$\widehat{v_i} = \begin{cases} 1 & Q_i(1) > Q_i(0) \\ 0 & \end{cases} \tag{2.12}$$

Dördüncü aşamada ise eğer doğru kod sözcüğüne henüz ulaşılamadı ise ikinci aşamaya dönüş yapılır ve tüm bu süreç tekrarlanır.

2.2 Çeşme Kodlar

Oransız silinti kodlar olarak da bilinen Çeşme kodları, bir veriyi oluşturan kaynak sembol dizisinden sınırsız sayıda kodlanmış veri (*codeword*) üretebilen silinti düzeltme kodlarıdır. Öyle ki *k* sayıda kaynak sembol, kodlanmış sembollerin *k* kadar ya da *k*'den biraz daha büyük sayıdaki bir alt kümesinden büyük olasılıkla yeniden oluşturulabilir (MacKay 2005).

Çeşme kodların en basit versiyonu rastgele doğrusal çeşme kodlarıdır (MacKay 2005). Rastgele doğrusal kodların devamı niteliğindeki LT kodlar (Luby 2002) 2002 yılında geliştirilmiştir. Daha sonra ise *Raptor* (Shokrollahi 2006) ve Çevrimiçi kodlar (Cassuto ve Shokrollahi 2015) geliştirilmiştir.

Çeşme kodlar, sabit bir kod oranında esnek bir şekilde uygulanabilir veya sabit bir kod oranının belirlenemediği, büyük miktarlarda verinin kodlanması ve kodlanmış verinin çözülmesi gerektiği durumlarda kullanılabilir.

Hem kodlanan hem de kodu çözülen sembol için çok az sayıda *XOR* işlemi gerektirmesinden ötürü *Raptor* kodlar en etkili Çeşme kodlarından biridir. Şöyle ki *k* adet veri sembolü kullanılarak gerçekleştirilen bir *Raptor* kodlamada bir sembolün kodlanması için $O(\log(1/\varepsilon))$ işlem gerekirken, kodlanmamış bir kaynak sembol ise $O(k\log(1/\varepsilon))$ işlem ile yeniden elde edilebilir (Shokrollahi 2006). Burada ε ile kastedilen ise *Raptor* kodlamanın yüküdür. Aşağıdaki alt bölümlerde, Çeşme kod sınıfında kabul edilen *LT* kodlar ve *LT* kodların da içerisinde kullanıldığı *Raptor* kodlar anlatılacaktır.

2.2.1 *LT* Kodlar

Optimale yakın silinti kodlardan olan Çeşme kodların ilk örneği *Luby-Transform (LT)* kodlar, *Michael Luby* tarafından 1998 yılında geliştirilip 2002 yılında yayınlanmıştır. Diğer Çeşme kodlarda olduğu gibi *LT* kodlarda da kodlama ve kod çözme işlemlerinde seyrek, iki bölümlü çizgeler kullanılmaktadır. *LT* kodlar, Çeşme kod sınıfına girmesi nedeniyle oransızlık özelliğini taşımaktadır. Diğer bir deyişle, kaynak veri paketlerinden sınırsız sayıda kodlanmış veri paketi üretebilmektedir. *LT*

kodlarında dikkat çeken özellik ise *LDPC* kodlarında olduğu gibi veri paketlerini kodlamak ve kod çözmek için *XOR* işlemlerini kullanmasıdır.

δ karakteri, eldeki K adet kodlanmış sembolden orijinal sembollerin (*input symbols*) elde edilmesi sürecindeki başarısızlık olasılığını ifade eder. Genel olarak, bir LT süreci (LT *process*) en az $1-\delta$ olasılıkla başarıyla tamamlanır. k adet orijinal sembolden oluşan bir kaynak verinin kodlanma süreci düşünülür ise, LT kod için kodlanmış bir sembol ortalama olarak $O(In(k/\delta))$ adet sembol işlemi ile oluşturulabilir. Bir orijinal sembol ise herhangi bir $k + O(\sqrt{k}In^2(k/\delta))$ adet kodlanmış sembol ile yeniden elde edilebilir ve bu süreç için ortalama $O(kIn(k/\delta))$ adet sembol işlemine ihtiyaç duyulur (Luby 2002).

2.2.1.1 Kodlama

LT kodlarda kodlama işlemi henüz kodlanmamış verinin eşit boyutlarda *k* adet bloğa bölünmesi ile başlar. Daha sonrasında ise kodlanmış paketler yalancı rastgele sayı üreteci (*pseudorandom number generator*) yardımıyla oluşturulur.

Kodlanmamış mesajdan derece dağılımı (*degree distribution*) ile elde edilen *d* adet *($1 \leq d \leq k$)* blok rastgele olarak seçilir. Kodlanacak olan veri paketini bulunduran bloktaki diğer veri paketleri (sembolleri) *XOR* işlemine tabi tutularak ilgili paket kodlanır. Kodlanmamış olan bütün veri paketleri bu işleme tabi tutulana kadar bu süreç devam eder.

LT kodlarını optimize etmek için genellikle kullanılan parametre, yalancı rastgele sayı üreteci olarak da bilinen derece dağılım fonksiyonlarıdır. Bu tür bir derece dağılımı yönteminin kullanılma nedeni, kod çözme sürecinde genişletilerek üretilecek olan kodlanmış verideki fazladan paket sayısını minimize etmektir.

Luby, orijinal çalışmada *k* adet kodlanmamış paket üzerinden aşağıda denklemleri verilen ideal soliton dağılımını (*ideal soliton distribution*) önermiştir (Luby 2002):

- $P\{d = 1\} = \frac{1}{k}$
- $P\{d = i\} = \frac{1}{i\,(i-1)}$ $\quad (i = 2,3,\ldots,k).$

Ama *ideal soliton* dağılımı uygulamada iyi bir performans gösterememektedir (Luby 2002). *Ideal soliton* dağılımında *ripple* olarak adlandırılan ve tek komşuya sahip kodlanmış paketlerin komşularının tutulduğu kümenin beklenen eleman sayısı 1'dir; fakat kod çözme sürecinde oluşabilecek bir sapma durumunda birinci derece için ($d = 1$) hiç paket kalmayabilir ve bu durumda kod çözme süreci başarısızlıkla sonuçlanacaktır.

Ideal *soliton* dağılımında görülen bu aksaklıklar güçlü *soliton* dağılımı (*robust soliton distribution*) ile iyileştirilmiştir. Güçlü *soliton* dağılımında görülen farklılık ise çok küçük dereceler için ($d = 1$ gibi) daha çok kodlanmış paket üretmesi ve birden büyük dereceler için daha az kodlanmış paket üretmesidir. Böylece kod çözme sürecinin başarıyla tamamlanma olasılığı arttırılmış olur. Aşağıda güçlü *soliton* dağılımının denklemleri görülmektedir (Luby 2002):

- $R = c.In(\,k/\delta\,)\sqrt{k}$
- $\tau(i) = \begin{cases} R/ik\,, & i = 1,\ldots,k/R-1 \\ R\;In(R/\delta)/ik\,, & i = k/R \\ 0, & i = k/R+1,\ldots,k \end{cases}$.

2.2.1.2 Kod Çözme

Kod çözme sürecinde ilk olarak derece dağılım fonksiyonuna göre derecesi 1 olan, diğer bir deyişle sadece 1 adet kodlanmış paket ya da sembole komşu olan orijinal paket ya da semboller *ripple* denilen tek komşulu orijinal sembollerin tutulduğu kümeye eklenir. Daha sonra *ripple* kümesi içindeki her orijinal sembol sırasıyla işlemden (*process*) geçirilir. Bu işlem ise orijinal sembolün komşu olduğu diğer kodlanmış semboller ile olan komşuluklarının sonlandırılması sürecidir. Diğer bir deyişle, *ripple* içerisindeki bu kaynak sembolün tek komşuya sahip olmasından ötürü, komşu sembol herhangi bir *XOR* işlemi gerektirmeden doğrudan kaynak sembol olarak belirlenir. Bunun ardından *ripple* içerisindeki ilgili orijinal sembolün başka

kodlanmış semboller ile komşuluk ilişkisi var ise bu orijinal sembol ile komşu olduğu kodlanmış semboller *XOR* işlemine tabi tutulur. *XOR* işlemi ardından bu orijinal sembolün bütün komşulukları kaldırılır ve bu işlemdeki kodlanmış sembollerin dereceleri bir azaltılır. Eğer bu kodlanmış sembollerin yeni dereceleri hala birden büyük *(d > 1)* ise bu semboller geçici bellek (*buffer*) üzerinde bekletilir.

Bütün bu sürecin ardından tek komşulu orijinal semboller belirlenerek *ripple* kümesi tekrar güncellenir ve yukarıda anlatılan süreç *ripple* kümesi içerisinde eleman kalmayıncaya kadar tekrarlanır. Yani derece düşürme işleminden sonra bir kodlanmış sembolün derecesi bire eşit *(d = 1)* ise bu sembol *ripple* kümesine dahil edilir.

Kodlanmış veri bloğundaki bütün k adet veri paketi ya da sembolü *ripple* kümesine girip çıktığı zaman, kod çözme işlemi başarı ile tamamlanmış olur. b değerinin bir bit değerini ifade ettiği bir durumda $b \oplus b = 0$ koşulundan ötürü, d derecesine sahip bir kodlanmış paket için $d - 1$ adet veri paketi *XOR* işlemine tabi tutulur ve bu işlemin ardından geriye kodlanmamış olan veri paketi kalır.

2.2.2 *Raptor* Kodlar

Amin Shokrollahi tarafından 2001 yılında geliştirilip 2004 yılında yayınlanan *Raptor* kod, *LT* kod üzerinden geliştirilen ve Çeşme kod sınıfına giren bir silinti düzeltme kodudur (MacKay 2005). Bu kodlama yöntemi de k adet sembolden oluşan bir mesajı sınırsız sayıda kodlanmış sembole çevirebilen bir kod türüdür.

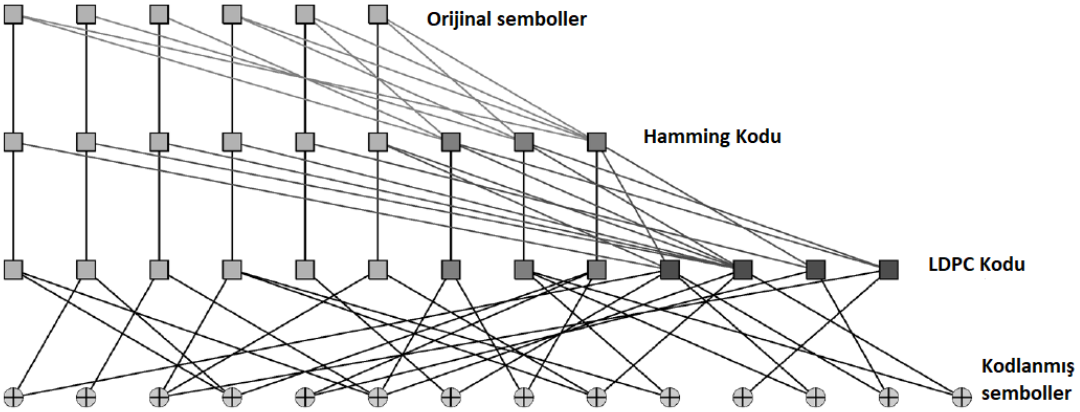
Raptor kodlar sistematik ve sistematik olmayan şeklinde iki sınıfa ayrılabilir. Sistematik *Raptor* kodlarda kodlanmamış mesaj, kodlanmış veri içerisinde aynı şekilde tutulur. Sistematik olmayan *Raptor* kodlarda ise orijinal veri tamamen kodlanarak aynı şekliyle korunmamış olur. Sistematik olmayan *Raptor* kodlara örnek olarak Çevrimiçi kodlar (Cassuto ve Shokrollahi 2015) verilebilir.

Raptor kodların özelliklerinden bahsetmeden önce belirtilmesi gereken temel performans parametrelerinden biri, ε karakteri ile gösterilen ve kod çözme sürecinde ortaya çıkan yük parametresidir. Yük parametresi, orijinal sembollerin yeniden elde edilebilmesi için ihtiyaç duyulan kodlanmış sembol sayısından orijinal sembol

sayısının çıkarılmış hali olarak tanımlanmaktadır. Bir LT kodun yükü $O(log^2(k)/\sqrt{k})$ olarak ifade edilir (Shokrollahi 2006).

Raptor kodların LT kodlara karşı ana avantajlarından biri, sabit bir maliyette kodlama ve kod çözme işlemlerine olanak sağlamasıdır. *Raptor* kodlarının diğer bir avantajı da kod çözme sürecinde LT kodlara göre daha az sayıda kodlanmış sembol ile orijinal sembolleri elde edebilmesidir. *Raptor* kodlar için k adet orijinal sembolü kod çözme süreciyle yeniden elde edebilmek için $k(1+\varepsilon)$ adet kodlanmış sembol yeterli olmaktadır. Bir kodlanmış sembol için $O(log(1/\varepsilon))$ adet işleme ihtiyaç duyulur iken, bir orijinal sembol ise $O(k.log(1/\varepsilon))$ adet işlem ile yeniden elde edilir (Shokrollahi 2006).

Raptor kod, iki silinti düzeltme kodunun birleşimiyle elde edilen bir koddur. Önkod ya da dış kod olarak adlandırılan ilk kod *LDPC* (Gallager 1962) gibi sabit oranlı (*code rate*) bir silinti düzeltme kodu olabilir. Burada kullanılan dış kod aynı zamanda başka silinti kodların birleşimiyle de elde edilebilir. İç kod ise dış koddan gelen kodlanmış veriyi alır ve bu veriyi kullanarak yeni kodlanmış veri paketlerini oluşturur. İç kod aşamasında kullanılan silinti düzeltme kodu her zaman LT koddur. Şekil 2.5'te bir *Raptor* kodun birden fazla silinti düzeltme kodunun kullanımına dayanan dış kod süreci gösterilmektedir. Burada orijinal semboller ilk olarak *Hamming* silinti düzeltme kodu kullanılarak kodlanır. Daha sonra ise bu kodlanmış semboller, girdi olarak kabul edilip *LDPC* kodu ile tekrar kodlama işleminden geçirilir. Bu işlemin ardından ortaya çıkan yeni kodlanmış semboller de *Raptor* kodların ikinci aşaması olan iç kod, LT tarafında kullanılır.

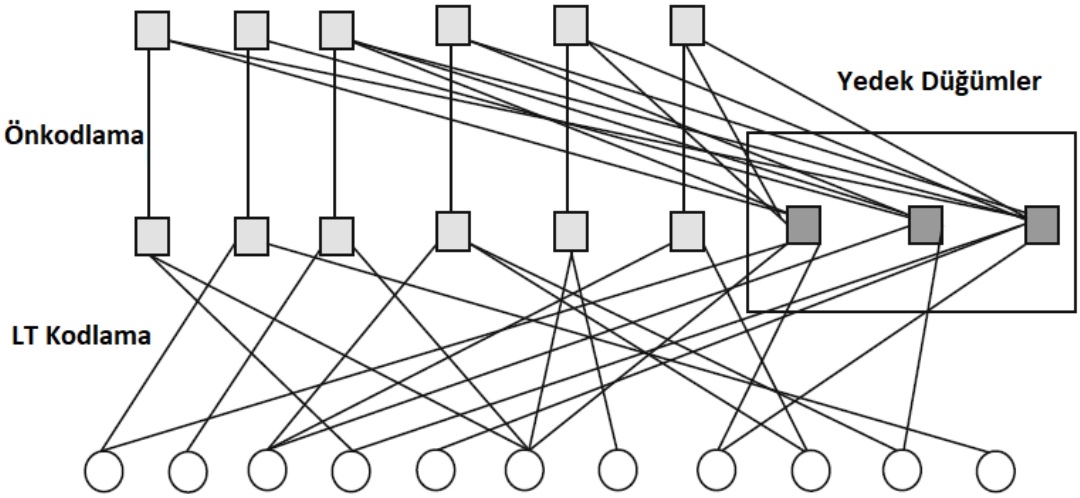


Şekil 2.5: *Raptor* kodlarda birden fazla silinti düzeltme kodunun kullanıldığı bir dış kod süreci (Shokrollahi 2006).

2.2.2.1 Kodlama

Önkodun ürettiği kodlanmış veri paketlerinden özel bir olasılık dağılımı olan derece dağılımına göre rastgele olarak seçtiği veri sembollerini alarak *XOR* işlemine tabi tutar ve kendi kodlanmış veri paketlerini elde etmiş olur. Derece dağılımı, genellikle *ideal soliton* dağılımı ya da onun modifiye edilerek geliştirilmiş hali olan güçlü soliton dağılımı olmaktadır.

Sistematik olmayan *Raptor* kodlarında, henüz kodlanmamış olan veri, önkodun kendi kodlama süreçleri için girdi olarak kullanılır. Sistematik *Raptor* kodlarında ise önkodun kodlama işleminde kullanılacak girdi verisi olan k adet sembol, kodlama işleminin tersi uygulanarak elde edilir. Bu yüzden tersinir kodlama işlemi ile üretilen k adet kodlanmış sembole normal kodlama işlemi uygulandığında *LT* kodun çıktısını oluşturan yeni kodlanmış verideki ilk k adet sembol, orijinal kaynak sembollerini temsil etmiş olur. Şekil 2.6'da *Raptor* kodun genel çalışma mantığı gösterilmektedir.



Şekil 2.6: *Raptor* kodlama süreci (Shokrollahi 2006).

2.2.2.2 Kod Çözme

Raptor kodlarda iki tür kod çözme yöntemi uygulanmaktadır. Bunlardan ilki olan sıralı yöntemde; iç kod olan *LT* kod, inanç yayılma algoritması (*belief propagation algorithm*) kullanarak kod çözme işlemini gerçekleştirir. Yeterli sayıda orijinal kaynak sembol yeniden elde edildikten sonra, geri kalan kaynak semboller de dış kodun kendi kod çözme yöntemi uygulanarak yeniden elde edilir (Shokrollahi 2006).

İkincisi olan birleştirilmiş kod çözme yönteminde ise hem iç kod (örneğin *LT*) hem de dış kod (örneğin *LDPC*) tarafından kodlanan semboller arasındaki ilişkiler genellikle Gauss eleme yöntemi ile elde edilen birleştirilmiş bir eşzamanlı denklemler kümesini temsil eder (Venkiah ve diğ. 2007).

2.3 Reed-Solomon Kodlar

Reed-Solomon kodları *Irving S. Reed* ve *Gustave Solomon* tarafından 1960 yılında geliştirilen silinti düzeltme kodlarıdır (Reed ve Solomon 1960) . *Reed-Solomon* kodları CD, DVD, Blu-Ray diskler, QR kodları, DSL ve WiMAX gibi veri transfer teknolojileri başta olmak üzere geniş bir uygulama alanına sahiptir.

Reed-Solomon kodlarının üzerinde çalıştığı ve semboller olarak adlandırılan veri blokları, sonlu alan elemanları kümesinde tanımlanmaktadır. Örneğin 4096 byte'lık (32768 bit'lik) bir veri bloğu, her biri 12 bitlik 2731 adet sembolden oluşacak şekilde ifade edilir ve her sembol $GF(2^{12})$ sonlu alanına ait bir elemana karşılık gelir (Park ve diğ. 2018).

(n, k) terminolojisi ile ifade edilen bir *Reed-Solomon* kod için n kodlanmış sembol sayısını ifade eder iken k ise henüz kodlanmamış olan ham verideki sembol sayısını ifade etmektedir. Bir *Reed-Solomon* kodu kodlanmamış veri bloğuna $t = n - k$ adet kontrol sembolü ekleyerek t adet hatalı sembolü tespit edebilir ve $t/2$'ye kadar sembolü de düzeltebilir (Reed ve Solomon 1960). Örneğin; *(n,k) = (255,191)* ile gösterilen bir kod için *n = 255* sembollük bir blok içinde *k = 191* adet 8-bitlik sembol ve 64 adet 8-bitlik eşlik sembolü (*parity symbol*) kodlanır. Bu durumda bir *Reed-Solomon* kod, blok başına 16 adet sembole kadar silinti düzeltebilir.

Reed-Solomon kodların iki temel türü vardır: orijinal versiyon ve *BCH (Bose, Chaudhuri and Hocquenghem)* versiyon.

Reed-Solomon kodlarının kullanım alanlarına bakılacak olur ise bu kodlar kompakt disklerde (*CD*) anahtar bir bileşen görevi görür. *CD*'de iki Reed-Solomon katmanı bulunmaktadır (Wicker & Bhargava, 1994). Aynı şekilde DVD'ler de benzer bir şema kullanır. Bu silinti düzeltme kodları *USENET* üzerindeki dosyalarda da kullanılmaktadır. *PDF-417, MaxiCode, Datamatrix, QR Code* ve *Aztec Code* gibi neredeyse tüm iki boyutlu barkodlarda yine Reed-Solomon kodları kullanılır. Barkod tarayıcısı bir barkod sembolünü tanıyamazsa, bunu bir silinti olarak algılar ve barkodun bir kısmı hasar görmüş olsa bile doğru okumaya izin vermek için *Reed-Solomon* silinti düzeltme kodunu kullanır. *Reed-Solomon* kodları aynı zamanda *Voyager* uzay aracı tarafından çekilip geri gönderilen dijital fotoğrafları kodlamak için kullanılmıştır.

Reed-Solomon kodlar temel olarak üç ana parametre tarafından karakterize edilmektedir: alfabe boyutu q, blok uzunluğu n ve $k < n \leq q$ koşulunu sağlayan mesaj uzunluğu k. Alfabe semboller kümesi, q kuvvetine sahip bir sonlu alanı ifade eder ve q de bir asal kuvvettir. *Reed-Solomon* kodlarda kodlanmış veriye karşılık gelen blok uzunluğu henüz kodlanmamış olan mesaj uzunluğunun genellikle sabir bir katı olarak

tayin edilir ve bu noktada kod oranı $R = k / n$ olarak gösterilir. Blok uzunluğu alfabe boyutuna eşit ya da ondan daha az olur ($n = q$ ya da $n = q - 1$).

Derecesi k'den küçük olan herhangi iki farklı polinom en fazla $k - 1$ noktada benzerlik gösterir. Diğer bir deyişle, *Reed-Solomon* kod tarafından kodlanan iki veri bloğu en az $n - (k - 1) = n - k + 1$ noktada farklılık gösterir. Bu nedenle de Reed-Solomon kodun uzaklığı (*distance*) $d = n - k + 1'$dir. Kod oranının $R = {}^{k}/_{n}$ olduğu durumda göreceli uzaklık (*relative distance*) *delta* = ${}^{d}/_{n} = 1 - {}^{k}/_{n} + {}^{1}/_{n} = 1 - R + {}^{1}/_{n} \sim 1 - R$' dir. Göreceli mesafe ve kod oranı arasındaki bu denge nedeniyle *Reed-Solomon* kodlar maksimum mesafe ayırt edilebilir (*maximum distance separable – MDS*) kod sınıfına girmektedir.

Reed-Solomon kodların 2^m elemanlı bir sonlu alan olan F üzerinden kullanılması yaygındır. Bu durumda her sembol *m-bit* üzerinden temsil edilir. Buna göre kodlanmış bir blok içindeki sembol sayısı ise $n = 2^m - 1$. Bu çalışmada da kullanıldığı şekliyle 8-bitlik semboller üzerinden çalışan *Reed-Solomon* kodlarında blok başına $n = 2^8 - 1 = 255$ sembol oluşmaktadır.

2.3.1 Sonlu Alan

Sonlu alan (*Finite Field* ya da *Galois Field*) sınırlı sayıda eleman içeren bir alandır. Matematikte sonlu bir alan; çarpma, bölme, toplama ve çıkarma işlemlerinin tanımlandığı ve belirli temel kuralların karşılandığı bir kümedir. Bir sonlu alanın eleman sayısı onun kuvvetini ifade eder. Sonlu alanlar yalnızca p'nin bir asal sayı ve k'nın da pozitif bir tam sayı olduğu p^k asal kuvveti (*prime power*) için geçerlidir. q kuvvetine sahip sonlu bir alanda $X^q - X$ polinomunun tüm q elemanları bu polinomun kökleri olur. Sonlu alanların en bilinen örnekleri asal kuvvet alanlarıdır. p kuvvetine sahip sonlu bir alanın elemanları $0, \ldots, p - 1$ aralığındaki tam sayılar tarafından temsil edilir. Kısaca, sonlu bir alana ait bir polinom hangi sayı ile dört işleme tabi tutulursa tutulsun sonuç bu sonlu alana ait elemanlardan birine karşılık gelir.

2.3.2 Kodlama

Reed-Solomon kodlarda kodlama süreci genel olarak 1960 tarihli orjinal versiyon ve *BCH* versiyon olarak ikiye ayrılmaktadır. Her iki versiyon da kendi içerisinde klasik ve sistematik kodlama şeklinde ikiye ayrılır. *k* adet orijinal sembolden oluşan bir verinin kullanıldığı bir örnek üzerinden gidilecek olur ise, klasik kodlamada orijinal semboller kodlama sürecinde kullanılan polinomun katsayıları olur. Yine yukarıda bahsedilen örnek düşünülür ise sistematik kodlamada ilgili polinomun ilk *k* adet değeri orijinal sembolleri aynı şekilde tutmuş olur.

2.3.2.1 Orijinal Versiyon

Reed-Solomon kodun 1960 tarihli orijinal versiyonunda her kod sözcüğü, *k*'den daha küçük dereceli bir polinomun fonksiyon değerler dizisini ifade etmektedir. Diğer bir deyişle, bu silinti kod türünde bir mesajı kodlamak için ilgili mesaj, q elemandan oluşan ve *F* ile adlandırılan bir sonlu alan içerisinde *k*'den daha küçük dereceli bir polinom ile işleme tabi tutulur. Bu polinom, *F* alanının *n* farklı noktasında $(a_1,\ldots, a_n)$ değerlendirilir ve ortaya kodlanmış veri çıkar.

2.3.2.1.1 Klasik Kodlama

Orijinal yöntemde mesaj, sonlu alan özelliği taşıyan polinomun katsayıları olarak yorumlanmaktadır. Diğer bir deyişle mesaj $x = (x_1, \ldots, x_k) \in F^k$ olarak tanımlanır ise polinom da $p_x(a) = \sum_{i=1}^{k} x_i a^{i-1}$ olarak tanımlanır. Mesaj *x*, polinom p_x'in sonlu alan *F*'ye ait *n* farklı noktada $(a_1, \ldots, a_n)$ değerlendirilmesi ile kodlanır.

Sonlu alan üzerindeki *n* farklı nokta için ise *Vandermonde* matrisinin devriği olan bir matrisin elemanları kullanılır. Şekil 2.7 üzerinden de bu süreç görülebilir.

$$[x_1\ x_2\ \dots\ x_k]\ .\ \begin{bmatrix} a_1^0 & \cdots & a_k^0 & \cdots & a_n^0 \\ a_1^1 & \cdots & a_k^1 & \cdots & a_n^1 \\ a_1^2 & \cdots & a_k^2 & \cdots & a_n^2 \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ a_1^{k-1} & \cdots & a_k^{k-1} & \cdots & a_n^{k-1} \end{bmatrix} = [c_1\ c_2\ \dots\ c_n]$$

Şekil 2.7: *Reed-Solomon* orijinal versiyonda klasik kodlama süreci (***c*** **=** ***x.A***).

2.3.2.1.2 Sistematik Kodlama

Reed-Solomon kodlar için alternatif bir yöntem olan sistematik kodlamada, polinom p_x'in ilk *k* adet değeri mesaj *x*'i tutar ($p_x(a_i) = x_i$; $i \in \{1,\dots,k\}$). Polinom p_x'i mesaj *x*'e ait değerler üzerinden hesaplamak için *Lagrange* interpolasyonu (*Lagrange interpolation*) kullanılabilir. Bu interpolasyon işleminin ardından mesaj *x*, polinom p_x'in sonlu alan *F*'ye ait *n* farklı noktada $(a_1, \dots, a_n)$ değerlendirilir ve genişletilmiş veri elde edilir. Şekil 2.8'de içerisinde birim matris bulunduran *G* matrisi üzerinden sistematik kodlama gösterilmektedir.

$$[x_1\ x_2\ \dots\ x_k]\ .\ \begin{bmatrix} 1 & 0 & 0 & \dots & 0 & g_{1,k+1} & \cdots & g_{1,n} \\ 0 & 1 & 0 & \dots & 0 & g_{2,k+1} & \cdots & g_{2,n} \\ 0 & 0 & 1 & \dots & 0 & g_{3,k+1} & \cdots & g_{3,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & g_{k,k+1} & \cdots & g_{k,n} \end{bmatrix} = [c_1\ c_2\ \dots\ c_n]$$

Şekil 2.8: *Reed-Solomon* orijinal versiyonda sistematik kodlama (***c*** **=** ***x .G)***.

2.3.2.2 BCH Versiyon

BCH versiyonda mesaj *x* yine polinom p_x'e eşleştirilir (*mapping*) ve orijinal kodlama versiyonunda olduğu gibi kendi içerisinde klasik ve sistematik kodlama şeklinde iki ayrı kola ayrılır.

Bu versiyonun orijinal versiyondan ayrıldığı nokta ise sonlu alandaki *n* farklı nokta yerine $n = q - 1$ koşulu temelinde $n - 1$ dereceli bir polinom *s* hesaplanır ve

bu polinomun *n* adet katsayısı alınarak kodlanmış veri elde edilir. (Bose ve Ray-Chaudhuri 1960).

$$\boldsymbol{c} = \{(s_1, s_2, \dots, s_n) | s(a) = \sum_{i=1}^{n} s_i a^{i-1}\} \tag{2.14}$$

2.3.3 Kod Çözme

Bu kısımda ise bu tez çalışmasında da kullanılan klasik kod çözme yöntemi anlatılmaktadır. Klasik yöntemde, orijinal mesajı yeniden elde edebilmek için *n* tane sembolün herhangi bir *k* tanesinin teorik olarak en az depolama ile yeterli olmaktadır. Bu özellik silinti kodların optimallik özelliğinden faydalanılarak elde edilmektedir. Bu kod çözme yönteminde herhangi bir silintiyi düzeltmek için *n* adet sembolden oluşan kodlanmış bloktan herhangi bir *k* kadarı alınarak polinom yeniden üretilmeye çalışılır. Polinom elde edilebilirse kodlanmış bloktaki herhangi bir silinti düzeltilebilir.

3. CİHAZLAR ARASI İLETİŞİM

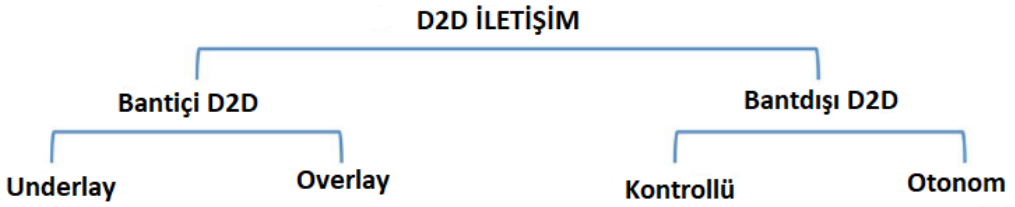
Hücresel ağlarda cihazlar arası (*D2D)* iletişim, klasik hücresel ağlardaki klasik yöntem olan baz istasyonları üzerinden iletişim kurmak yerine iki mobil kullanıcının doğrudan birbirleriyle iletişim kurması olarak tanımlanabilir. D2D iletişim ağ performansını iyileştirmesi nedeniyle son yıllarda yeni bir yöntem olarak ön plana çıkmaktadır.

Klasik bir hücresel ağda bir hücre içerisindeki bütün iletişim baz istasyonu üzerinden kurulur. Diğer bir deyişle bir hücre içerisindeki iki mobil cihaz birbirleriyle ancak baz istasyonu aracılığıyla iletişim kurabilmektedir. Baz istasyonu üzerinden iletişim, mobil kullanıcıların yeterince yakın olmadığı ve düşük veri hızı gerektiren iletişim durumları için uygundur. Fakat günümüzde yüksek veri hızı gerektiren durumlarda (video paylaşımı gibi) D2D iletişim, ağda bant genişliği üzerinden aktarılabilen veri oranı olan spektral verimliliği artırması nedeniyle avantajlı hale gelmiştir. D2D iletişim sadece spektral verimliliği artırmakla kalmaz aynı zamanda enerji verimliliği ve iletişim gecikmesinde de iyileştirmeler sağlayabilir.

Her geçen gün veri trafiği artmaktadır ve telekom operatörleri her geçen gün artan uygulamalar temelindeki bu kullanıcı taleplerini karşılamaya çalışmaktadır. *WiMAX* (IEEE Computer Society and the IEEE Microwave Theory and Techniques Society 2006) ve *LTE-A* (Ghosh ve diğ. 2010) gibi *4G* teknolojiler kullanıcıların artan bu taleplerini karşılamakta zorluk çekmektedir. Bu nedenle de D2D teknolojisi hücresel ağlardaki klasik iletişim yöntemlerine alternatif olarak sunulmuştur.

D2D iletişim ilk olarak Lin ve diğ. (2000) tarafından çok atlamalı (*multihop*) iletim için önerilmiştir. Sonrasında ise Kaufman ve Aazhang (2008) ile Peng ve diğ. (2009) hücresel ağlarda spektral verimliliği ilerletmek adına *D2D* iletişimi incelemişlerdir. Daha sonrasında ise bugün bilinen D2D iletişim kullanım yöntemlerinden çoklu gönderim (multicasting) Du ve diğ. (2012) ile Zhou ve diğ. (2013) tarafından; uçtan uca iletişim (*peer-to-peer communication*) Lei ve diğ. (2012) tarafından; video yayımlama Doppler ve diğ. (2009), Golrezaei ve diğ. (2014) ile Li ve diğ. (2012) tarafından; makineler arası *(machine-to-machine (M2M))* iletişim ise Pratas ve Popovski (2013) tarafından ilk olarak incelenmiştir.

Asadi ve diğ. (2014) tarafından yapılan araştırma çalışmasında D2D iletişim iki ana başlık altında sınıflandırılmıştır. Bunlardan ilki bantiçi (*inband*) D2D olarak, diğeri de bantdışı D2D (*outband D2D*) olarak adlandırılır. Bantiçi D2D iletişim de hücresel ve D2D iletişim için aynı radyo kaynaklarının kullanıldığı *underlay* ile farklı radyo kaynaklarının kullanıldığı *overlay* olarak ikiye ayrılır. Bantiçi D2D iletişimde sıkça karşılaşılan radyo parazitleri sorununa çözüm bulmak amacıyla geliştirilen bantdışı D2D ise kontrollü ve otonom olarak iki ayrı kola ayrılır. Klasik hücresel ağlarda taşıyıcı cihazlar veri alışverişi için tahsis edilen hava dalgalarını kullanırlar. Bu iletişim şekline örnek olarak ağ operatörlerinin veri trafiğini azaltmak adına lisanssız *5 GHz* frekans bandından iletişimine izin veren *LTE (Long Term Evolution)-Unlicenced* verilebilir. Bantdışı D2D, paylaşılan bir bant üzerinde diğer kullanıcıların müdahalesine açık olan lisanssız spektrum üzerinde çalışır. *Wi-Fi, bluetooth* gibi kablosuz ağlar lisanssız spektrum üzerinden çalışır. Şekil 3.9'da D2D iletişimin sınıflandırılması görülmektedir.



Şekil 3.1: D2D iletişimin sınıflandırılması (Asadi ve diğ. 2014).

D2D iletişimi ile mobil iletişim teknolojisinin gelişim süreci arasında güçlü bağlar bulunmaktadır. Birinci nesil (1G) sesin ana odak nokta olduğu analog sistemlerdir. İkinci nesilde (2G) ses aktarımı yine ön planda olmuştur.

Bu noktada D2D iletişimin öncüllerinden olarak 3G gösterilebilir. 3G teknolojisi, bir cep telefonu iletişim protokolü olan GSM ve kablosuz internet erişimi için IEEE 802.16 standartlarını kullanan WiMAX teknolojilerini içeren bir kablosuz telefon standartlar bütünüdür. Öncülleri *2G* ve 2.5G olarak da bilinen *GPRS*'ye göre daha yüksek veri hızını desteklemektedir. Ses aktarımını ana odak noktası yapan 2G teknolojilerinin aksine, 3G teknolojisi veri aktarımını iyileştirmeye çalışmıştır. Bu özelliği nedeniyle D2D iletişimde önemli bir başlangıç noktasıdır. 3G teknolojisinde

veri iletim hızı minimum 2 Mbit/s, maksimum 14.4 Mbit/s'dir. 2009 yılından itibaren Türkiye'de kullanılmaya başlanmıştır.

3G standartlarının devamı olan 4G, bir kablosuz telefon standartlar bütünüdür. 4G teknolojisi ile beraber hem veri iletim hızlarında hem de kapsama alanlarında önemli ilerlemeler sağlanmıştır. Öyle ki veri iletim hızı 100 Mbps ile 1 Gbps arasındadır. 2015 yılından itibaren Türkiye'de kullanılmaya başlanmıştır.

2020 yılında ticari kullanıma girmesi beklenen ve 4G teknolojilerinin yerini tamamen alması planlanan beşinci nesil mobil iletişim teknolojisi olan 5G ile beraber veri iletişim hızının 4G'ye göre daha hızlı olması beklenmektedir. 5G de tıpkı 4G teknolojisinde olduğu gibi 6 GHz bandını kullanmaktadır. Öyle ki maksimum veri iletim hızının 20 Gbps olması beklenmektedir. 5G teknolojisi ile birlikte veri iletim hızındaki büyük ilerlemeler cihazlar arası (D2D) iletişime dayanan nesnelerin interneti (internet of things) teknolojisinin gelişimini de hızlandıracaktır.

3.1 Kodlanmış Veri Önbellekleme

Bu kısımda *D2D* teknolojilerinin temel felsefesi olan iki mobil cihaz arasında doğrudan iletişimde veri trafiğini düşürerek ağ performasında iyileştirmeler sağlayan ve bu tez çalışmasının da ana konusu olan dağıtık depolamada kodlanmış veri önbellekleme (*coded caching*) konusu ilgili literatür çalışmaları üzerinden anlatılacaktır.

Hepsinden önce veri önbellekleme konusuna değinilecek olur ise, mobil cihazlarda veri önbellekleme popüler bir içeriğin mobil cihazların belleklerinde önbelleğe alınarak veri trafiğini düşürme yöntemi olarak tanımlanmaktadır (Maddah-Ali ve Niesen 2014). Bu noktada veri önbellekleme ikiye ayrılmaktadır: Klasik yöntem olan kodlanmamış veri önbelleklemede *(uncoded data caching)* ilgili içerik son kullanıcı dosyalarında kodlanmamış halde parçalar halinde tutulur iken kodlanmış veri önbelleklemede ise son kullanıcıların mobil cihaz belleklerinde ham veri bitleri kodlanmış mesajlar olarak tutulur. Maddah-Ali ve Niesen (2014) tarafından yapılan bu çalışmada kodlanmış veri önbelleklemenin klasik kodlanmamış veri önbelleklemeye göre daha performanslı çalıştığı gözlemlenmiştir.

Kodlama tekniklerinin hücresel ağ uygulamalarında kıyaslandığı çalışmalar da mevcuttur. Paakkonen (2014) tarafından yapılan bir çalışmada, kablosuz ağ kapasitesini artırmak adına mobil cihazlar üzerinde D2D iletişim ile veri dosyalarının önbelleklenmesi fikrinden yola çıkılıyor ve bir hücresel ağda enerji tüketimi açısından yenileme kodlarının performansı analiz ediliyor. Yenileme kodlarının performans iyileştirmeleri sağladığı belirtilen bu makalede, depolama düğümlerinin hücreden ayrılması veya devredışı kalması durumda kullanılan yedekleme yönteminin ilgili verinin popülerliği durumunda sadece faydalı olabileceği öne sürülmektedir. Yine bu çalışmada yenileme kodlarının kodlanmamış veri önbelleklemeye göre daha performanslı çalıştığı gözlemlenmiştir. Bu çalışmadaki (Paakkonen ve diğ. 2014) temel varsayımlardan biri ise bir baz istasyonundan bir mobil kullanıcıya veri iletiminin, iki mobil kullanıcı arasındaki veri iletiminden daha maliyetli olduğudur. Bu varsayımın nedeni olarak da bir baz istasyonu ile kullanıcı arasındaki ortalama uzaklığın iki kullanıcı arasındaki ortalama uzaklıktan genellikle daha fazla olması ve kablosuz sinyallerin mesafe arttıkça kaybolma olasılıklarının artması gösterilmektedir. Bu senaryoda mobil kullanıcıları temsil eden düğümlerin hücre içerisinde bulunma süreçleri *M/M/∞ Markov* modeli ile açıklanmaktadır. Markov zinciri modeline göre, düğümler belli bir oranda hücre içerisine girer ve hücre içerisindeki düğüm sayısı ile orantılı olarak da hücre dışına çıkar. Paakkonen (2014) tarafından yapılan bu çalışmada iki farklı tür yenileme kodu kullanılmıştır: minimum bant genişliği tamir kodları (*MBR*) ve minimum depolama tamir kodları (*MSR*). *MBR* kodları kayıp bir veri bloğunu tamir etmek için gereken trafik miktarını minimize ederken, *MSR* kodları depolama düğümlerinde (*storage nodes*) saklanan veri miktarını minimize eder. Tekrarlama yönteminde ise düğümler ilgili veri dosyasının tam kopyasını taşırlar. Bu yöntemin dezavantajı ise yüksek tamir bant genişliği ve depolama alanıdır.

Paakkonen ve arkadaşları tarafından yapılan bir başka çalışmada popüler veri dosyalarının düğümlerde önbelleğe alındığı ve silinti düzeltme kodları ile korunduğu bir ağ içerisinde bütün maliyeti minimize eden bir kodlama yöntemi bulunmaya çalışılmıştır (Paakkonen ve diğ. 2016). Bu çalışmada kodlamanın enerji tüketimini önemli oranda düşürdüğü ve D2D veri önbelleklemenin telekomünikasyon operatörleri için yeni ekonomik kazanımlar sağlayabileceği gösterilmiştir. Bu makalede üç farklı veri önbellekleme kullanılmıştır. İlki olan basit veri önbelleklemede (*simple caching*) tek bir düğüm, dosyanın tam bir kopyasını

tutmaktadır ve bu düğümün başarısızlığı durumunda yedekleme kullanılmadığı için ilgili dosya da veri önbellekleyen kümeden ayrılmış olmaktadır. Bu durumda talepte bulunan düğüm, bütün dosyayı baz istasyonundan indirmek zorundadır. İkinci yöntem ise tekrarlama yöntemidir. Bütün dosyanın kopyaları hücre içerisindeki farklı düğümler üzerinde saklanmaktadır. Fakat bu yöntemin dezavantajı ise yüksek depolama maliyetidir. Kullanılan üçüncü veri önbellekleme yöntemi ise yenileme kodlarıdır. Yine bu çalışmada da minimum bant genişliği tamir kodları (*MBR*) ve minimum depolama tamir kodları (*MSR*) olarak iki farklı tür yenileme kodu kullanılmıştır. Orijinal veri dosyasından her biri α büyüklüğünde n adet kodlanmış veri parçası üretilir. Bu kodlama yönteminin *MDS* özelliğinden ötürü n adet depolama düğümünün herhangi bir k tanesinden ilgili dosya yeniden tedarik edilebilir. Burada kullanılan küme sisteminde hücre içerisinde belli bir kümeye dahil düğümler baz istasyonu yerine kendi aralarında iletişim kurarlar. Yine ilgili kümeye düğümlerin giriş çıkışları Markov zinciri (*Markov chain*) modeline göre düzenlenir. Diğer bir deyişle hücre içerisinde küme adı verilen baz istasyonsuz bir alt hücre oluşturulur. İletişim ve depolama maliyetleri üzerinden ağı optimize etmeyi amaçlayan bu çalışmada, kodlanmış veri önbelleklemenin yedeklemesiz veri önbelleklemeye göre %90'dan fazla oranlarda enerji tüketimini azalttığı gözlemlenmiştir. Depolama maliyetlerinin yüksek olduğu durumlarda kodlanmış önbellekleme yöntemi önerilir iken depolama maliyetlerinin düşük olduğu durumlarda ise tekrarlama yönteminin tercih edilebileceği belirtilmiştir.

Wei (2015) tarafından yapılan bir çalışmada ilk olarak dağıtık depolamada silinti düzeltme kodlarını, kopyalama temelli yöntem ile kıyaslamışlardır. Silinti kodların tekrarlama yöntemine göre iki önemli avantajının olduğu belirtilmiştir. Bunlardan ilki aynı yedekleme ya da kod oranlarında veri dayanıklılığının (*reliability*) silinti düzeltme kodlarında daha yüksek olduğu belirtilmiştir. İkinci avantajın da veri dayanıklığının sağlanması noktasında kopyalama yönteminin daha fazla depolama alanına ihtiyaç duyması olduğu belirtilmiştir.

Kazez ve diğ. (2017) tarafından yapılan çalışmada D2D iletişime dayanan hücresel ağlarda önbellekleme politikasını optimize etme fikrinden yola çıkılan bellek yerleştirme (*cache placement*) problemi incelenmiştir. D2D veri iletişiminde hücresel bant genişliğinin yeniden kullanımı söz konusu olduğu için çakışma kontrolüne ihtiyaç

duyulmaktadır. Bu sorundan yola çıkarak baz istasyonun hücre içerisindeki düğümlere içerik önbellekleme ve yakınlarındaki diğer düğümlere yardım etmeleri karşılığında ödüllendirilmesine dayanan bir algoritma sunulmuştur. Önerilen yeni algoritmanın hem baz istasyonu ile yaşanabilecek çakışma ihtimalini düşürdüğü hem de veri trafiğini düşürdüğü simülasyon sonuçları ile gözlemlenmiştir.

Zheng ve diğ. (2010) tarafından yapılan çalışmada ise dağıtık depolama sistemlerinde *Raptor* kodlar incelenmiştir. *n* adet düğümden oluşan uçtan uça (*peer-to-peer*) ağlarda verinin $e > 0$ olduğu durumlar için $(1+e)k$ kadar düğümden yeniden elde edilebildiği bu kodlama şeması önkod olarak *B-J* kodun kullanımıyla modifiye edilerek iyileştirilmiştir. *NS2* üzerinden yapılan simülasyon sonuçlarına göre bu *Raptor* kod şeması yüksek kod çözme oranları sağlamıştır.

MDS kodlar olarak da bilinen optimal silinti kodlarda orijinal mesajı yeniden elde edebilmek için *n* tane sembolün herhangi bir *k* tanesi yeterli olmaktadır. Bu algoritmalara örnek olarak Reed ve Solomon tarafından geliştirilen *Reed-Solomon* kodları verilebilir (Reed ve Solomon 1960). Plank ise *Reed-Solomon* kodlarını kullanarak verilerin nasıl kodlandığını, nasıl güncellendiğini ve cihaz arızalanmalarında sistemin nasıl ilk haline getirilebileceğini açıklamıştır (Plank 1997).

Kodlama tekniklerinin karşılaştırıldığı daha kapsamlı bir çalışma da Pedersen ve diğ. (2016) tarafından yapılan mobil kablosuz ağlarda dağıtık depolama konulu çalışmadır. Bu çalışmada iyileştirilmek istenen maliyet parametresi, kablosuz ağlar için dağıtık depolamada ortaya çıkan iletişim maliyetidir ve odak noktası da verilerin depolandığı bir cihazın hücreden ayrılması ile oluşan tamir (*repair*) problemidir. Bu çalışmanın Paakkonen' in çalışmasından (Paakkönen ve diğ. 2013) ayrıldığı nokta ise tamir işlemini periyodik olarak gerçekleştiren toplu bir tamir zamanlaması (*repair scheduling*) sunmasıdır. Oysa bu çalışmada Paakkonen anlık (*instantaneous*) tamir gerçekleştirmektedir ve bu sebeple içerik baz istasyonu yerine her zaman mobil cihazlardan indirilmektedir. Pedersen'in çalışması (Pedersen ve diğ. 2016) bu açıdan daha gerçekçi bir görüntü sunmaktadır. Çünkü tembel (*lazy*) tamir mantığının yanında baz istasyonları da veri indirme işlemlerinde kullanılabilmektedir. Bu periyodik tamir işlemi tamir bant genişliği (*repair bandwidth*) olarak da bilinen tamir sürecinde iletilen veri miktarını düşürmek için kullanılmıştır. Bu çalışmada *MDS* kodları, yenileme kodları ve yerel olarak tamir edilebilir kodlar (*Locally Repairable Codes*, kısaca *LRC*)

(Papailiopoulos ve Dimakis 2014) iletişim maliyeti açısından karşılaştırılmıştır. Eğer periyodik tamir sık bir şekilde tekrarlanıyor ise içeriğin sadece baz istasyonu üzerinden indirilmesinin daha düşük bir iletişim maliyeti sağladığı gözlemlenmiştir. Ayrıca, istenilen içeriğe sahip cihazların ağ içerisine girmesinin iletişim maliyeti açısından yararlı olduğu gösterilmiştir. Aynı zamanda klasik dağıtık depolamada iyi bir performans gösteremeyen *MDS* kodlarının hücresel ağlarda veri önbellekleme alanında düşük bir iletişim maliyeti sağladığı da gözlemlenmiştir.

4. KODLANMIŞ VERİ ÖNBELLEKLEME SİMÜLATÖRÜ

Klasik hücresel ağlarda düğümler arasındaki iletişim tamamen baz istasyonu aracılığı ile gerçekleştirilir iken, *D2D* iletişimde ise düğümler baz istasyonunu aradan çıkararak birbirleriyle doğrudan iletişim kurabilir ve veri alışverişinde bulunabilirler. Kablosuz ağlardaki veri trafiğini düşürmek amacıyla geliştirilen *D2D* iletişim yöntemiyle beraber ağ performansında da iyileştirmeler görülmektedir.

D2D iletişimde hücre içerisindeki düğümlerin birbirleriyle doğrudan veri alışverişinde bulunma noktasında içeriğin dağıtımı için önbellekleme yöntemi kullanılmaktadır. Diğer bir deyişle, klasik ağlarda baz istasyonunda saklanan veri, D2D ağlarda aynı zamanda hücre içerisindeki mobil cihazlara parçalar veya bütün halinde dağıtılır ve bu cihazların önbelleklerinde tutulur. Bu noktada, kodlanmış veri önbellekleme kullanılabilir. D2D iletişime dayanan bir dağıtık depolama sisteminde, hücre içerisindeki cihazların önbelleklerinde orijinal veri yedekli olarak saklanır. İşte bu yedeklilik, kodlama işlemi ile gerçekleştirilebilir. Bu cihazlardan bir veya birkaçının hücreden ayrılması durumunda, bu düğümlerin önbelleklerinde tutulan kayıp verilerin yeniden elde edilebilmesi için hücre içerisindeki diğer cihazların önbelleklerinde kodlanmış olarak tutulan veriler kullanılır. Baz istasyonundan veri indirme genellikle düğümlerden daha fazla maliyetlidir. Kodlanmış veri önbellekleme sayesinde baz istasyonu kullanılmamış olur ve daha az maliyetle tamir işlemi tamamlanır.

Bu tez çalışmasında, klasik ağlarda genellikle kullanılan *MDS* silinti düzeltme kodlarından olan *Reed-Solomon* kodu ile Çeşme kodlar sınıfına giren *Raptor* kod, *D2D* iletişim ile düğümlerin tamir işlemleri açısından test edilerek çeşitli maliyet parametreleri üzerinden karşılaştırılmıştır. Aynı zamanda bir Çeşme kod olan *LT* kod ile optimale yakın silinti kodlarından *LDPC* kod, bu tamir sürecinde karşılaştırılmıştır.

Bu kapsamda öncelikle hücre içerisindeki düğümlerin *D2D* iletişimi ve tüm bu testlerin gerçekleştirilebilmesi için simülatör hazırlanmıştır. Diğer bir deyişle sistem modeli ortaya çıkarılmıştır. Ardından *Raptor* kod içerisinde dış kod olarak *LDPC*, iç kod olarak da *LT* kodlar kodlama ve kod çözme işlemlerini gerçekleştirecek şekilde hazırlanmıştır. Aynı şekilde *Reed-Solomon* kodu için de önce sistem modeli tasarlanarak simülatör oluşturulmuş, ardından da kodlama ve tamir işlemlerinde

kullanılacak olan kod çözme işlemleri simülatörde hazırlanmıştır. Simülatör ve silinti düzeltme kodları hazır hale getirildikten sonra da hücre içerisindeki düğüm ya da düğümlerin kaybı durumunda kaybedilen verilerin yeniden elde edilmesi amacıyla gerçekleştirilen tamir süreci analiz edilmiştir. Bu tamir sürecinde baz istasyonlarından ve düğümlerden çekilen veri sembol sayısı, tamir süreleri ve kodlama süreleri elde edilerek karşılaştırılmıştır.

4.1 Sistem Modeli

Simülasyon için kullanılan sistem modeli hücresel bir ağ içerisinde tek bir baz istasyonu tarafından hizmet verilen ve içerisinde düğüm olarak da adlandırılan mobil cihazların bulunduğu tek bir hücre şeklinde tasarlanmıştır. Bu sistem modelinde hücre içerisindeki başlangıç düğüm sayısı M olarak ifade edilmiştir. Basitlik sağlaması amacıyla simülatörde önbellekleme işlemleri için F baytlık tek bir dosya kullanılmıştır ve bu dosya aynı zamanda baz istasyonunda kodlanmış (*encoded*) olarak tutulmuş olup her sembol 1 bayt olarak alınmıştır. Simülasyonda hücresel ağdaki düğümlerin veri saklayabildikleri ve *D2D* teknolojisini kullanarak birbirleri ile iletişim kurabildikleri varsayılmıştır. Simülasyonun çalışması süresi, başlatıldıktan sonra N adet düğümün hücre içerisine giriş yaptığı toplam süre olarak ayarlanmıştır. Burada N değeri 1000 olarak alınmıştır ($N = 1000$). Bu sistem modeli Pedersen ve diğ. (2016) tarafından yapılan çalışmada kullanılan sistemi referans alarak hazırlanmıştır.

Mobil cihazlar *Poisson* rastgele sürecine *(Poisson random process)* göre hücre içerisine giriş çıkış yapmaktadır. Bu noktada *Poisson* dağılımından bahsetmek gerekir ise bir olayın belli bir sabit birim zaman aralığında meydana gelme sayısının olasılığı olarak tanımlanabilir. Bu sabit aralıkta ortaya çıkan olayların beklenen sayısı α olarak ifade edilir ise k sayıda olayın gerçekleşme olasılığı aşağıdaki formül ile elde edilir.

$$f(k, \alpha) = \frac{\alpha^k e^{-\alpha}}{k!} \tag{4.1}$$

Düğümler aynı şekilde dağıtılmış rastgele ulaşma zamanları ile üstel bağımsız (*exponential independent*) Poisson sürecine göre hücreye ulaşmaktadır. Bu ulaşma sürecinin olasılık yoğunluk fonksiyonu şu şekildedir.

$$f_{T_a}(t) = M\lambda e^{-M\lambda t}, \qquad \lambda \geq 0, t \geq 0 \tag{4.2}$$

Burada *t*, birim zamanda ölçülen zamanı ifade eder iken; λ bir düğümün beklenen hücreye ulaşma oranını (*arrival rate*) ifade eder.

Diğer yandan, mobil cihazlar üstel, rastgele bir yaşam süresi T_1 kadar aşağıdaki olasılık yoğunluk fonksiyonuna göre hücre içerisinde kalırlar.

$$f_{T_1}(t) = \mu e^{-\mu t}, \qquad \mu \geq 0, \ t \geq 0 \tag{4.3}$$

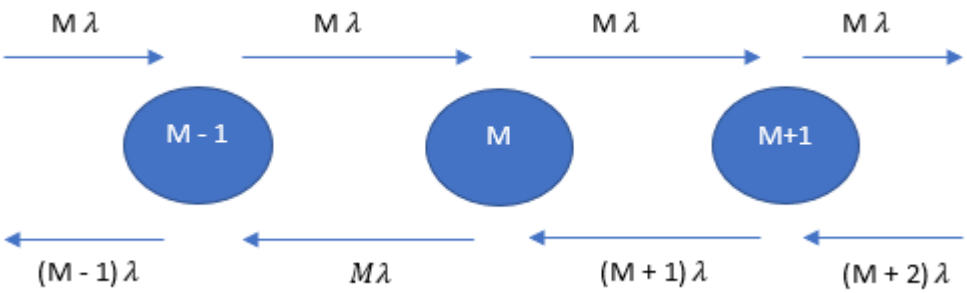
Burada μ bir düğümün beklenen hücreden çıkma oranını (*departure rate*) ifade etmektedir.

Bu çalışma kapsamında kullanılan sistem modelinde bir mobil cihazın hücreye ulaşma oranı ile hücreden çıkma oranı eşit olarak kabul edilmiştir ($\lambda = \mu$). Öyle ki hücre içerisindeki beklenen mobil cihaz sayısı başlangıçta hücre içerisindeki mobil cihaz sayısını ifade eden *M* değerine eşit olarak sabit kalmaktadır.

Hücre içerisindeki mobil cihaz sayısı M/M/∞ *Markov* modeli ile açıklanmaktadır. Aşağıda bir hücre içerisinde *i* adet mobil cihazın bulunma olasılığı gösterilmektedir.

$$\pi(i) = \frac{(M\lambda/\mu)^i}{i!} e^{-(M\lambda/\mu)} \tag{4.4}$$

Hücre içerisindeki mobil cihaz sayısının Markov modeline göre değişimi Şekil 4.10'da görülmektedir:



Şekil 4.1: Hücredeki düğüm sayısının giriş-çıkış durumlarına göre M/M/∞ *Markov* modeli ile gösterimi.

Markov sürecinde (*Markov process*), bir durum (*state*) belirli bir istatiksel değere göre ve geçmiş durumlardan bağımsız olarak değişmektedir. Bu süreçte yalnızca şimdiki durum gelecek durumları etkilemektedir. Buna *Markov* özelliği (*Markovian property*) denir. $X(t)$'nin rastgele sürecinin $t_1 < t_2 < \cdots < t_n < t_{n+1}$ zaman anlarında temsil edildiği düşünülür ise *Markov* süreci aşağıdaki gibi formülize edilebilir (Miller ve Childers 2012).

$$F_X(X(t_{n+1}) \leq x_{n+1} \mid X(t_n) = x_n,\ X(t_{n-1}) = x_{n-1}, \dots,\ X(t_1) = t_1) = F_X(X(t_{n+1}) \leq x_{n+1} \mid X(t_n) = x_n) \quad (4.5)$$

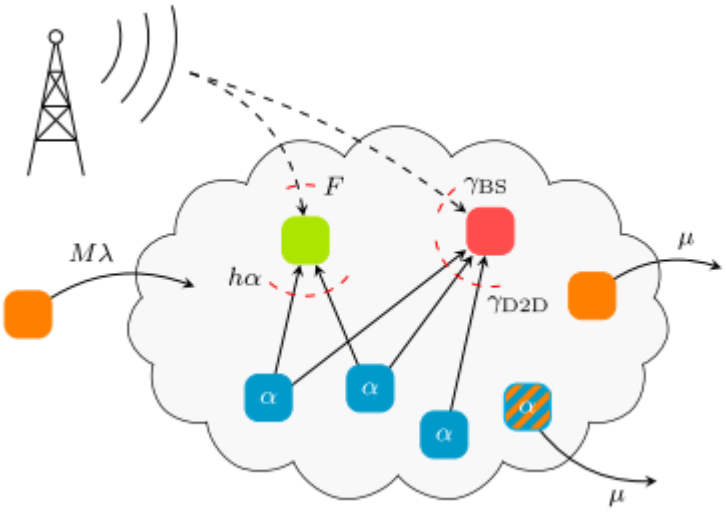
Bu formüle göre t_{n+1} zamanındaki olaylar yalnızca t_n zamanındaki olaylara bağlıdır.

Bu özellikler doğrultusunda *şekil 4.1* incelenecek olur ise her olayın bir sonraki olaya belirli bir olasılık değeriyle geçiş yapabileceği ya da kendi durumunu koruyacağı görülebilir. Örneğin; t_n anında hücre içerisinde M adet düğümün bulunması olayı gerçekleşirken, t_{n+1} anında hücre içerisinde $M + 1$ adet düğümün bulunması olayına ya da $M - 1$ adet düğümün bulunması olayına geçiş yapılabilir. Bu özellik açısından *Markov* zincirleri yönlü çizge (*directed graph*) özelliği göstermektedir. Diğer bir deyişle, hücre içerisindeki düğümlerin hücre içerisinde bulunma ya da hücreden ayrılma gibi belli durumları vardır ve bu durumlar arası geçişler istatiksel olarak yönlü çizgeler üzerinden belirlenir. Markov zincirleri sık sık kuyruklama sistemlerinde (*queueing systems*) kullanılır.

Zaman değişkeninin süreklilik (*continuous*) özelliği gösterdiği *Markov* süreçleri de vardır (*continuous time Markov processes*). Bu süreçlerin en yaygın olanlarından biri de doğum-ölüm sürecidir (*birth-death process*). Bu tez çalışmasında kullanılan sistem modelinde düğümlerin hücre içerisinde kalıp kalmama süreçleri de doğum-ölüm süreci olarak tanımlanabilir. Kuyruklama sistemlerinde yaygın olarak kullanılan bu modeller M/M/1 ya da M/M/∞ olarak ifade edilebilmektedir. M/M/1 üzerinden gidilecek olunur ise kullanılan ilk *M* harfi ile varış sürecini (*arrival process*), ikinci *M* harfi çıkış sürecini (*departure process*), 1 değeri de sunucu sayısını ifade etmektedir (Miller ve Childers 2012). Bir taksi durağı örneği üzerinden açıklanacak olur ise tek bir taksinin hizmet verdiği bir taksi durağında müşterilerin kuyrukta bekleyip hizmet almayı bekledikleri bir süreç M/M/1 modeli olarak düşünülebilir (Miller ve Childers 2012). *M/M/*∞ Markov modelinde ise sonsuz sayıda sunucu

bulunmaktadır ve bu özelliği nedeniyle taksi durağı örneği düşünülür ise kuyrukta bekleyen müşteri yoktur. Bu çalışmada kullanılan sistem modelinde de ağ içerisinde sonsuz sayıda hücrenin hizmet verdiği varsayılır ise bir hücre içerisine düğümlerin girip çıkması M/M/∞ özelliğinde bir doğum-ölüm süreci (*birth-death process*) olarak tanımlanmaktadır.

Kullanılan sistem modelinin gösterildiği Şekil 4.2'de hücreye mobil cihaz girişi ve çıkışı görülmektedir. Aynı zamanda kırmızı olarak gösterilen mobil cihazın tamir sürecinde duruma göre hem baz istasyonundan hem de diğer mobil cihazlardan ilgili veriyi çekmesi gösterilmektedir. Çizgili olarak gösterilen düğümün hücreden ayrılmasının ardından başlayan bu tamir sürecinde γ_{BS} tamir için baz istasyonundan çekilen bayt sayısını, γ_{D2D} ise hücre içerisinde tamire dahil olan mobil cihazlardan çekilen bayt sayısını temsil etmektedir.



Şekil 4.2: Sistem modeli (Pedersen ve diğ. 2016).

Sistem modelindeki örnek hücre içerisinde F baytlık tek dosyanın veri sembollerini kodlanmış olarak yedekli halde taşıyan düğümler, depolama düğümleri olarak adlandırımaktadır. Başlangıçta m adet depolama düğümünün hücre içerisinde bulunduğu varsayılmıştır. Bu depolama düğümlerinden herhangi biri hücreden ayrıldığında, bu düğümdeki veri de kaybedilmektedir. Bu noktada, ağ içerisindeki veri dayanıklılığını sağlamak adına depolama düğüm sayısını m olarak sabit tutmak gerekmektedir. Bunun için de ilgili kayıp verinin hücre içerisinde boş olan

düğümlerden birinde yeniden depolanmasına ihtiyaç duyulmaktadır. Hücre içerisinde veri taşımayan düğümlerden herhangi birinin rastgele seçilerek hücreden ayrılan düğümdeki kayıp verinin bu boş düğümde yeniden oluşturulması sürecine tamir süreci denir. Bu tez çalışmasının ana odak noktası da bu tamir süreci ve bu tamir sürecinin maliyetidir. Maliyete etki eden faktörler ise tamir sürecinin ne kadar sürdüğü ile tamir süreci boyunca baz istasyonundan ve diğer düğümlerden indirilmesi gereken sembol sayısıdır.

Bu tez kapsamında kullanılan sistem modelinde tamir işlemini gerçekleştirme zaman aralığı Δ olarak gösterilmiş ve bu değişken *0.2* ve *0.4* olarak kabul edilmiştir. Bu noktada anlık tamir süreci kullanılmak istendiğinde Δ değişkeninin değeri 0 olarak atanmalıdır ($\Delta = 0$). Fakat Paakkonen ve diğ. (2013) tarafından yapılan çalışmada olduğu gibi anlık tamir yöntemi kullanıldığında baz istasyonları devredışı kalmakta ve bütün tamir süreci tamamen hücre içerisindeki diğer düğümler üzerinden gerçekleştirilmektedir. Tıpkı Pedersen ve diğ. (2016) tarafından yapılan çalışmada olduğu gibi daha gerçekçi bir senaryo için belli aralıklarla devreye giren tembel tamir yöntemi bu sistem modelinde tercih edilmiştir.

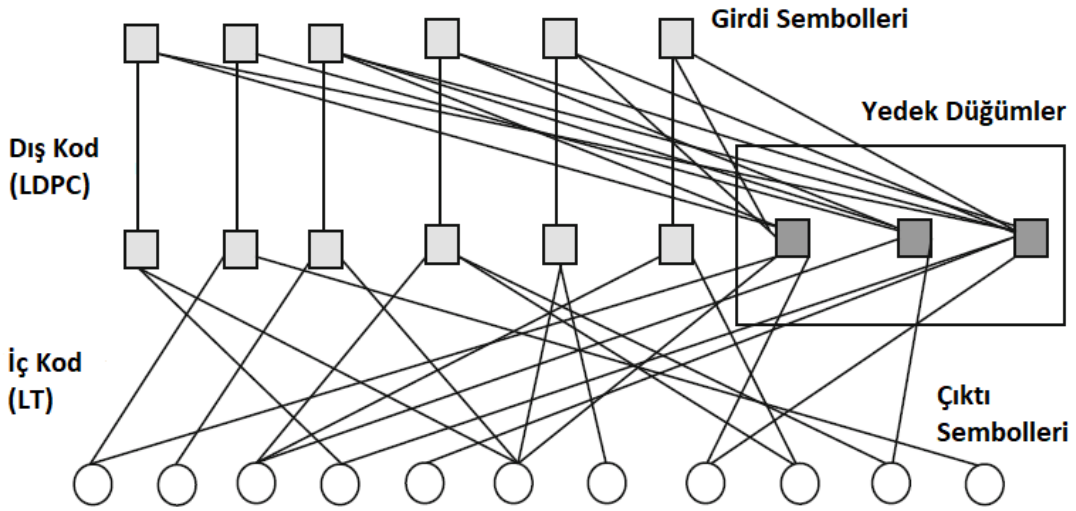
Bu tez kapsamındaki katkılar şu şekilde özetlenebilir:

1. Hücresel ağlarda dağıtık veri önbellekleme için genel bir simülatör geliştirilmiştir. Öyle ki farklı yedekli depolama sistemlerinin performansı bu simülatör ile kolayca ölçülebilir.
2. Veri önbelleklemede daha önce denenmemiş olan *Raptor* kodlarının performansı ölçülerek diğer standart kodlama şemaları ile karşılaştırılmıştır.
3. Bu çalışmada diğer çalışmalarda bulunmayan artık veri uygulaması bulunmaktadır. Hücresel ağlarda bulunan düğüm sayısı her zaman sabit olmayacağından ötürü semboller her zaman depolama düğümlerine eşit dağıtılamaz. Bu yüzden artık veri kullanımı gerekebilir.

4.2 Raptor Kod Süreci

Raptor kodlar 2001 yılında *Amin Shokrollahi* tarafından *LT* kodlar üzerinden geliştirilen, çeşme kodlar sınıfına giren iki katmanlı bir silinti düzeltme kodudur (Shokrollahi 2006). Çeşme kod olması nedeniyle de *k* adet sembolden sınırsız sayıda

kodlanmış paket üretebilmektedir. *LT* kodlardan farklı olarak sabit maliyette kodlama ve kod çözme olanağı sağlayan *Raptor* kodlar iki silinti kodun birleşiminden elde edilir. İç kod her zaman *LT* koddan oluşur. Bu tez kapsamında da *Raptor* kodların dış kod olarak *LDPC*, iç kod olarak da *LT* kod kullanılmıştır. *LDPC*'den kodlanarak elde edilen veri paketleri *LT* kodun girdisi olarak kullanılmıştır. Şekil 4.3'te *Raptor* kodlardaki kodlama süreci gösterilmektedir:



Şekil 4.3: *Raptor kodlarda kodlama süreci* (Shokrollahi, 2006).

4.2.1 LDPC Kod

Bu tez çalışmasında *Raptor* kodun dış kod katmanında *LDPC* kod kullanılmıştır. *LDPC* kodlar 1960 yılında R. G. Gallager tarafından doktora çalışması kapsamında geliştirilen optimale yakın bir silinti düzeltme kodudur. Optimale yakın silinti düzeltme kodların özelliği ise orijinal mesajı yeniden oluşturabilmek için $\varepsilon > 0$ koşulu altında $(1 + \varepsilon)\, k$ adet sembole gereksinim duyulmasıdır. Bu k değeri çok küçük bir pozitif sayıdır.

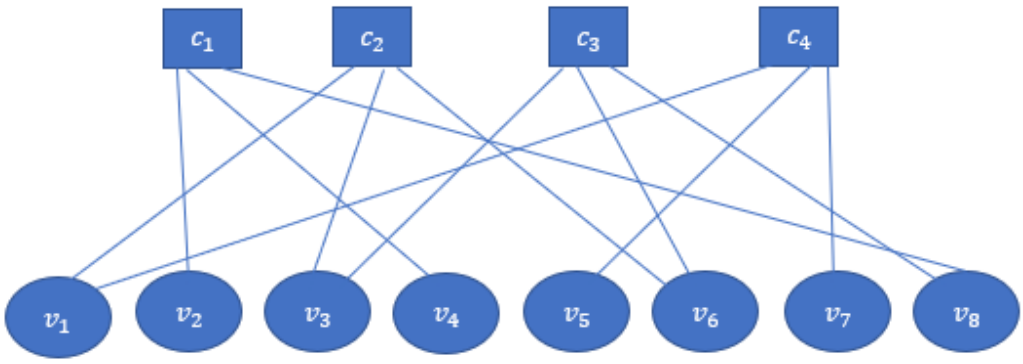
LDPC kodlar seyrek (burada matristeki 1 sayısının azlığı kastedilir) bir eşlik kontrol matrisi *(parity-check matrix)* tarafından oluşturulmaya başlanmaktadır. Matrisin herhangi bir satırındaki elemanlar katsayı olarak kabul edilir ise bu matristeki her satır bir eşlik kontrol denklemine *(parity-check equation*) karşılık gelmektedir.

Eğer matriste her bir sütun için sabit sayıda 1 var iken her bir satır için de yine sabit sayıda 1 var ise bu *LDPC* kod düzenli olarak sınıflandırılmaktadır. Eğer eşlik kontrolü matrisindeki satır veya sütunlarda sabit bir oranda 1 yok ise bu *LDPC* kod düzensiz olarak sınıflandırılmaktadır. Şekil 4.4'te gösterilen eşlik kontrolü matrisinde her satır için 3 tane 1 olsa da her sütundaki 1 sayısı sabit bir sayıda değildir ve bu nedenle de düzensizdir.

$$H = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

Şekil 4.4: Eşlik kontrolü matrisi örneği.

LDPC kodları, iki taraflı çizge (*bipartite graph)* temelinde, seyrek *Tanner* çizgeleri kullanılarak oluşturulan doğrusal kodlardır (*linear code*). Bu iki bölümlü çizge içerisinde iki tip düğüm vardır: Kontrol düğümleri (*check nodes*) ve değişken düğümleri (*variable nodes*). Bu iki tip düğüm kenarlar (*sets*) üzerinden bağlanır. Eğer bu iki düğüm arasında kenar mevcut ise bu bağlantı matris üzerinde 1 olarak temsil edilir. Bu iki düğüm tipi arasında bağlantı yok ise de matris üzerinde 0 olarak temsil edilir. Şekil 4.5'te kontrol düğümleri c_1, c_2, c_3, c_4 satırlar, değişken düğümleri $v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8$ ise sütunlar üzerinden temsil edilmektedir.



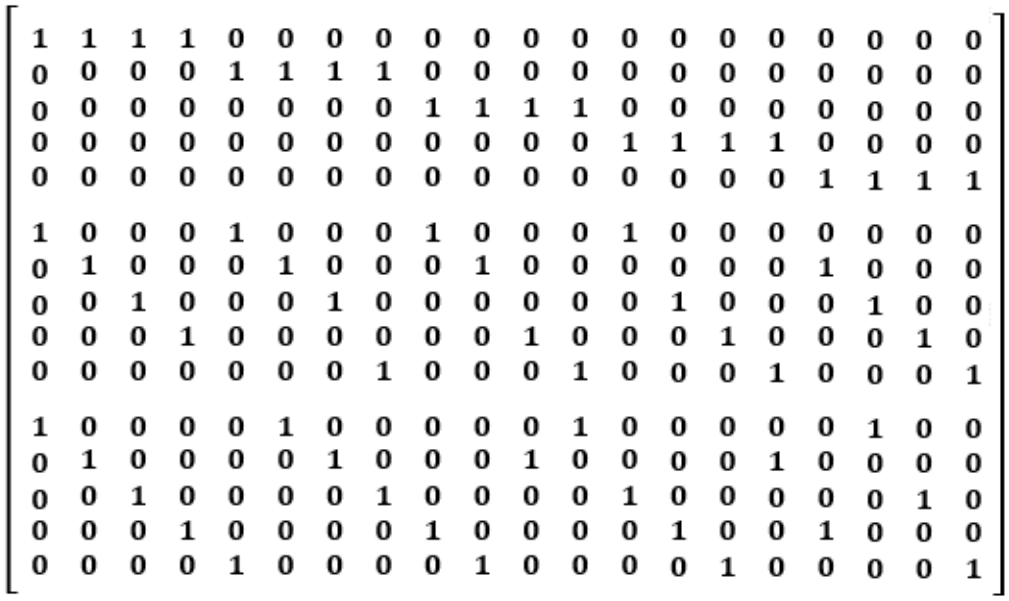
Şekil 4.5: Şekil 4.4'teki matrisin çizge teoremi üzerinden gösterimi.

4.2.1.1 Kodlama

LDPC kod için kodlama işleminde ilk olarak eşlik kontrol matrisinin oluşturulması gerekmektedir. *Gallager* (1962) tarafından sunulan *(N, j, k)* terminolojisi ile *LDPC* kod ifade edilecek olur ise *N* matristeki blok sayısı, *j* matrisin her sütununda bulunan 1'lerin sayısı, k ise matrisin her satırında bulunan 1'lerin sayısıdır. Bu matris öncelikle *j* adet alt matrise bölünmektedir. Öyle ki herhangi bir alt matrisin her sütununda sadece bir adet 1 olmalıdır. Bu alt matrislerin ilki için 1'ler azalan sırada yerleştirilir. Örneğin, i. satıra $\big((i-1)\times k+1\big)$. sütundan $(i\times k)$. sütuna kadar 1'ler yerleştirilir. Şekil 4.6'da verilen örnek matriste en üstte yer alan alt matris incelendiği zaman, bu alt matrisin ilk satırı $(i=1)$ için ilk $k=4$ kadar sütuna kadar 1 değerleri yerleştirilir. İkinci satırda $(i=2)$ 5. sütundan 8. sütuna kadar 1 değerleri yerleştirilir. Bu şekilde alt matrisin her satırı için yukarıda verilen kurala göre 1'ler yerleştirilir. Geriye kalan $j-1$ adet alt matris ise oluşturulan bu birinci alt matrisin rastgele sütun permütasyonlarından oluşturulur. Bu eşlik kontrol matrisinde aynı zamanda $j\geq 3$ ve $k>j$ koşulları sağlanmalıdır. Bu üç temel parametreden yola çıkarak aşağıdaki formül ile eşlik kontrol matrisinin satır sayısı aşağıdaki gibi elde edilir.

$$N*(j/k) \tag{4.6}$$

Şekil 4.6'da, *Gallager*'ın kullandığı seyrek eşlik kontrol matrisi örneği gösterilmektedir.



Şekil 4.6: Seyrek eşlik kontrol matrisi örneği: *N*=20, *j*=3, *k*=4.

Bu tez çalışması kapsamında ikinci aşamada bu eşlik kontrol matrisi (*H*) sistematik hale dönüştürülmüştür. Bu dönüştürme işlemi için Gauss eleme yöntemi kullanılır. Temel satır işlemleri (*elementary row operations*) ile sistematik H_{sys} elde edilir. *H* matrisinin sistematik hale dönüştürülmesinden kasıt içerisinde birim matris (*I*) oluşturarak eşlik kontrol denklemlerini meydana getiren eşlik kontrol kümesi (*parity-check set*) içerisindeki kontrol hanelerini (*check digits*) oluşturmaktır. $k\ x\ n$ boyutlarında bir eşlik kontrol matrisi *H* ile ifade edilecek olur ise sistematik hali H_{sys} ve bu sistematik matristen elde edilen $n-k\ x\ n$ boyutlarında üretici matris (*generator matrix*) *G* aşağıdaki denklemler ile ifade edilebilir (Wei ve diğ. 2015^{b}). Buradaki *k* henüz işlenmemiş veri sembol sayısını ifade eder iken *n* ise kodlama işlemi neticesinde elde edilen genişletilmiş verinin sembol sayısını ifade etmektedir. Özetle sistematik H_{sys} matrisi ve bu matristen de üretici *G* matrisi elde edilmiş olur. Buradaki alt matris *P*'nin genellikle seyrek olmamasından kaynaklı olarak kodlama karmaşıklığı yüksektir (Leiner, 2005). Ayrıca *LDPC* kodlar için $\bar{d} = [d_1, d_2, \dots, d_n]^T$ ise $H\bar{d} = \bar{0}$ kuralı sağlanmalıdır. Sistematik H_{sys} matrisi ve bu matristen türetilen üretici matris G'nin yapısı aşağıdaki gibidir.

$$H_{sys} = [P^T \mid I_k] \tag{4.7}$$

$$G = [I_{n-k} \mid P] \tag{4.8}$$

Son aşamada ise k sembolden oluşan veri, üretici matris G üzerinden *XOR* işlemlerinden geçirilerek $(n-k)$ kadar genişletilmiştir. Bu noktada silinti düzeltme kodlarının performansını etkileyen önemli bir faktör (k/n) ile ifade edilen kod oranıdır (*code rate*). Şekil 4.7'de bu *XOR* işlemi gösterilmektedir. Henüz kodlanmamış veri yığını $\bar{b} = [b_1, \quad b_2, \quad \cdots \quad , b_k]$ içindeki j. sembol b_j ile temsil edilir iken kodlanmış veri vektörü $\bar{d} = [d_1, \quad d_2, \quad \cdots \quad , d_n]$ içindeki j. kodlanmış sembol d_j ile temsil edilmektedir. Bu noktada eğer kod sistematik ise $1 \leq i \leq k$ değerleri için $d_i = b_i$ koşulu sağlanmış olur.

$$[b_1\ b_2 \ldots b_k] \bigoplus \begin{bmatrix} g_{11} & \cdots & g_{1n} \\ g_{21} & \ddots & g_{2n} \\ \vdots & & \vdots \\ g_{k1} & \cdots & g_{kn} \end{bmatrix} = [d_1\ d_2 \ldots d_n]$$

$$\bar{b} \bigoplus \mathrm{G} = \bar{d}$$

Şekil 4.7: *k* sembolden oluşan ham veri ***b*** ile üretici matris *G*'nin *XOR* operasyonunun sonucu olarak *n* sembollü genişletilmiş yeni veri ***d*** elde edilir.

Yukarıda anlatılan *LDPC* ile kodlama süreci ardından kodlanmış *n* adet sembol elde edilmiş olur. Daha sonra ise elde edilen *n* adet kodlanmış sembol, bu tamir süreci için hücre içerisinde hiçbir veri taşımayan boş düğümlerden rastgele olarak seçilmiş düğümlere belirlenen belli bir oranda kaydedilmeye başlanır. Bu süreç bir örnek üzerinden anlatılacak olur ise 24 adet kodlanmış sembol 5 adet düğüme aktarılmak istenecektir. Bu işlemde her düğüme kaydedilecek sembol sayısı $24/5 = 4$ olarak belirlenir. Her düğüme sırasıyla 4 adet sembol kaydedilir ve ilk 20 kodlanmış sembol ilgili düğümlere aktarılmış olur. Bu noktadan itibaren ise artık veri aktarımı başlar. Bu örnek üzerinden devam edilecek olur ise 21. kodlanmış sembolden 24. kodlanmış sembole kadar olan ve her sembolünün farklı düğümlere kaydedildiği veri kısmı artık veri olarak tanımlanır. 21'den 24'e kadar olan 4 sembol sırasıyla ilk 4 düğüme kaydedilir. Burada 5. Düğüme herhangi bir artık veri aktarımı yapılmadığına dikkat

edilmelidir. Böylece her 24'lük kodlanmış sembolde ilk 4 düğüme 5 sembol, 5. sembole ise 4 sembol kaydedilmiş olur. Tablo 4.1'de ilk 3 döngüde artık veri durumu bahsedilen örnek üzerinden gösterilmektedir.

Tablo 4.1: 24 kodlanmış sembol üzerinden 3 döngüde düğümlere sembol aktarımı. Her sütun 24 satırdan oluşmaktadır ve her satır bir sembol indeksini ifade etmektedir. 21'den 24'e kadar olan semboller artık verileri temsil etmektedir.

Sembol İndeksi	1. Döngü	2. Döngü	3. Döngü
1	1. düğüm	1. düğüm	1. düğüm
2	1. düğüm	1. düğüm	1. düğüm
3	1. düğüm	1. düğüm	1. düğüm
4	1. düğüm	1. düğüm	1. düğüm
5	2. düğüm	2. düğüm	2. düğüm
6	2. düğüm	2. düğüm	2. düğüm
7	2. düğüm	2. düğüm	2. düğüm
8	2. düğüm	2. düğüm	2. düğüm
9	3. düğüm	3. düğüm	3. düğüm
10	3. düğüm	3. düğüm	3. düğüm
11	3. düğüm	3. düğüm	3. düğüm
12	3. düğüm	3. düğüm	3. düğüm
13	4. düğüm	4. düğüm	4. düğüm
14	4. düğüm	4. düğüm	4. düğüm
15	4. düğüm	4. düğüm	4. düğüm
16	4. düğüm	4. düğüm	4. düğüm
17	5. düğüm	5. düğüm	5. düğüm
18	5. düğüm	5. düğüm	5. düğüm
19	5. düğüm	5. düğüm	5. düğüm
20	5. düğüm	5. düğüm	5. düğüm
21	1. düğüm	5. düğüm	4. düğüm
22	2. düğüm	1. düğüm	5. düğüm
23	3. düğüm	2. düğüm	1. düğüm
24	4. düğüm	3. düğüm	2. düğüm

4.2.1.2 Tamir Süreci

Bu tez çalışması kapsamında *LDPC* tamir süreci anlatılmadan önce belirtilmesi gereken ve bu çalışmayı özgün kılan özelliklerden biri de *LDPC* tamir sürecinde hem tek düğüm tamiri (*single repair*) (Wei ve diğ. 2015^{b}) hem de tek seferde çoklu düğüm tamiri (*multiple repair*) (Wei ve diğ. 2015^{a}) gerçekleştirilebilinmesidir. Diğer bir deyişle hücre içerisinden Δ zaman aralıkları içerisinde birden fazla depolama düğümü ayrılmış olsa bile tek seferde ayrılan bütün düğümlerdeki kayıp veriler iki farklı tamir yöntemi ile de elde edilebilmektedir.

4.2.1.2.1 Tek Düğüm Tamiri

Wei ve diğ. (2015^{b}) tarafından yapılan çalışmada depolama sistemi içerisinde bulunan bir *LDPC* kodlanmış sembolünün ayrılması senaryosu temelinde bu sembolün yeniden elde edilmesi süreci anlatılmıştır. Bu tez çalışması kapsamında gerçekleştirilen tek düğüm tamir süreci de bu çalışma referans alınarak gerçekleştirilmiştir.

Wei ve diğ. (2015^{b}) tarafından yapılan bu çalışmada tek bir kodlanmış sembolün tamirine dayanan süreç şu şekilde tanımlanmıştır:

1. İlk olarak kurtarma denklemleri (*recovery equations*) belirlenir.
2. İkinci aşamada tamir işleminin gerçekleştirileceği düğüm, belirlenen kurtarma denklemindeki kodlanmış veri sembollerini ilgili düğümlerden çeker.
3. Son aşamada ise hücre içerisindeki diğer düğümlerden kurtarma denklemine göre çekilen kodlanmış veri sembolleri *XOR* işlemlerine tabi tutularak kayıp olan kodlanmış sembol yeniden elde edilmiş olur.

İlgili çalışmada örnek olarak kullanılan sistematik *LDPC* eşlik kontrol matrisi H_{sys} Şekil 4.8'de gösterilmektedir.

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Şekil 4.8: Örnek bir *LDPC* eşlik kontrol matrisi (*n=21, k=15).*

Kurtarma denklemleri ise şu şekilde belirlenir:

- Kayıp olan kodlanmış veri sembolüne ait indeksin eşlik kontrol matrisinde karşılık gelen sütununa gidilir ve bu sütun içerisindeki 1 sayısına sahip olan satırlar belirlenir.
- Bu sütundaki 1 sayısı kadar kayıp kodlanmış sembol için kurtarma denklemi oluşturulur. Diğer bir deyişle, ilgili sütunda 1 değerine sahip olan satırlara tek tek gidilir ve her satır boyunca kayıp sembol sütunu haricinde 1 değerine sahip diğer sütun indeklerine denk gelen kodlanmış semboller denkleme eklenir.
- Bu noktada oluşturulan kurtarma denklemlerinin tamamı tamir süreci için kullanılabilir. Fakat tercih edilen tamir karmaşıklığını minimuma çekmek olduğundan ötürü, oluşturulan kurtarma denklemlerinden en az *XOR* işlemi gerektiren denklem seçilir. Eğer eşlik değer matrisi düzenli bir matris olsaydı o zaman satır ve sütunlardaki 1 sayısı belli oranlarda sabit olacağı için bütün denklemlerdeki eleman sayısı aynı olacaktı. Bu durumda oluşturulan kurtarma denklemlerinden herhangi biri rastgele seçilebilir.

Burada yine ilgili makaledeki (Wei ve diğ. 2015^{b}) örnek üzerinden devam edilir ise, yukarıdaki şekilde verilen matriste ilk kodlanmış veri sembolü olan d_0'ın tamir süreci için kurtarma denklemleri şu şekilde olur:

$$d_0 = d_2 \oplus d_3 \oplus d_4 \oplus d_6 \oplus d_{10} \oplus d_{13} \oplus d_{16} \text{ (2. satıra göre)}$$

$$d_0 = d_6 \oplus d_7 \oplus d_{10} \oplus d_{11} \oplus d_{12} \oplus d_{13} \oplus d_{14} \oplus d_{17} \text{ (3. satıra göre)}$$

$$d_0 = d_1 \oplus d_3 \oplus d_5 \oplus d_8 \oplus d_{12} \oplus d_{19} \text{ (5. satıra göre).}$$

Bu üç denklem içerisinde en az elemana sahip olan ve dolayısıyla en az *XOR* işlemi gerçekleştirecek olan denklem üçüncü denklemdir. Bu nedenle üçüncü denklem kayıp veri sembolünün tamiri için seçilir.

Bu tez çalışması kapsamında ise bu tamir süreci şu şekilde gerçekleştirilmiştir:

- Kayıp düğümdeki verinin tamiri sürecinde hücre içerisindeki LDPC depolama düğümlerinden belli oranlarda semboller okunur. Daha sonrasında aynı işlem artık veri var ise o kısım için de gerçekleştirilir.
- Ardından kullanılan sistematik eşlik kontrol matrisi H_{sys} üzerinden kayıp düğüm tarafından tutulan sembol indeksleri belirlenir ve her kayıp sembol indeksi için kurtarma denklemleri oluşturulur ve içerisinden en az *XOR* işlemi gerektiren denklem ilgili kayıp sembolün tamiri için seçilir.
- Eğer kurtarma denklemindeki yardımcı semboller (*helper symbols*) hücre içerisindeki diğer depolama düğümlerinde bulunuyor ise bu düğümlerden alınır. Eğer gereken yardımcı semboller de kayıp düğüm ile beraber kaybolmuş ise ilgili yardımcı semboller baz istasyonundan tedarik edilir. Böylece tıpkı Pedersen ve diğ. (2016) tarafından yapılan çalışmada olduğu gibi hem hücre içerisindeki depolama düğümlerinin hem de baz istasyonunun kullanılması dağıtık depolamada dayanıklılığı artırır.
- Daha sonra kurtarma denklemi uygulanarak düğümlerden (duruma göre baz istasyonundan da olabilir) çekilen kodlanmış veri sembolleri *XOR* işlemine tabi tutularak kayıp semboller yeniden elde edilir.
- Son aşamada ise tamir edilen kayıp semboller hücre içerisinde hiçbir içeriğe sahip olmayan düğümlerden rastgele biri seçilerek bu düğüme gönderilir. Böylece tek bir düğüm için LDPC tamir süreci tamamlanmış olur.

4.2.1.2.2 Çoklu Düğüm Tamiri

Wei ve diğ. (2015[a]) yapılan devam makalesinde ise birden fazla sembolün aynı anda tamir süreci üzerinde çalışılmıştır. Bu tez çalışması kapsamında gerçekleştirilen hücre içi çoklu düğüm tamir süreci de bu makale referans alınarak gerçekleştirilmiştir.

Bu tamir süreci Wei ve diğ. (2015[a]) tarafından yapılan çalışmada da verilen örnek üzerinden anlatılacaktır. Bu örnekte ise $\overline{\mathbf{D}} = [d_0, d_1, \dots, d_{20}]^T$ ve $\mathbf{H}\overline{\mathbf{D}} = \overline{\mathbf{0}}$ koşullarının sağlandığı 21 adet kodlanmış veri sembolüne sahip bir dosya bulunmaktadır. Burada kullanılan $\overline{\mathbf{D}}$ ile kodlanmış veri sembolleri ifade edilir. $\overline{\mathbf{D_{kayıp}}}$ ile kayıp olan kodlanmış semboller kastedilir iken, $\overline{\mathbf{D_{gereken}}}$ ifadesi ile de tamir sürecinde kullanılan kodlanmış semboller kastedilir. Bu dosya için tercih edilen eşlik kontrol matrisi **H** ise Şekil 4.9'da verilmiştir.

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

Şekil 4.9: Çoklu düğüm tamiri için örnek *LDPC* eşlik kontrol matrisi (Wei ve diğ. 2015[a]).

Hücre içerisindeki *LDPC* ile kodlanmış sembollerin birden fazlasının kaybolması halinde devreye girecek olan bu tamir sürecinde kayıp kodlanmış semboller $\overline{\mathbf{D_{kayıp}}}$ matrisi üzerinden ifade edilir. Bu kayıp semboller aşağıdaki denklem kullanılarak yeniden elde edilir.

$$\overline{\mathbf{D_{kayıp}}} = \mathbf{P^{-1}R}\overline{\mathbf{D_{gereken}}} \tag{4.9}$$

Burada **H** matrisinin bir alt matrisi olan **P** matrisi, sadece kayıp kodlanmış sembol indekslerine göre belirlenir ve lineer bağımsızlık özelliğine sahiptir. **H** matrisinin bir başka alt matrisi olan **R** matrisi ise tamir sürecinde talep edilen kodlanmış veri sembollerine göre belirlenir.

Örnek üzerinden devam edilir ise, ilk üç kodlanmış sembolü saklayan bir düğümün kaybolduğu varsayımından yola çıkarak birinci, ikinci ve dördüncü satırlar temelinde **P** ve **R** matrisleri aşağıdaki gibi oluşturulur:

$$P = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix},$$

$$R = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}.$$

P matrisinin tersi ile **R** matrisinin çarpımı sonucu elde edilen yeni matristeki her satır, kayıp olan bir sembol için kurtarma denklemini temsil eder. İlgili satırlardaki 1 değerine sahip indekslerdeki kodlanmış veri sembolleri ilgili denklem içerisinde *XOR* işlemine tabi tutularak kayıp olan kodlanmış sembol tamir edilir.

Bu tez çalışması kapsamında gerçekleştirilen birden fazla düğümün kaybı sonrasında kayıp kodlanmış sembollerin tamir süreci aşağıdaki gibidir:

- Öncelikle hücre içerisinde bulunan *LDPC* depolama düğümlerindeki kodlanmış semboller sırasıyla okunarak bir vektör üzerinde tutulur.
- Ardından **P** matrisi sistematik eşlik değer matrisi olan $\mathbf{H_{sys}}$ üzerinde kayıp sembol indekslerine göre sütunlar seçilerek lineer bağımsız bir matris elde edilmeye çalışılır. Bu noktada matrisin boyutları çerçevesinde bütün kombinasyonlar denenir. Burada eğer bütün kombinasyonlar sonucunda lineer bağımsız bir **P** matrisi oluşturulamaz ise tamir süreci tek düğüm tamiri ile gerçekleştirilir.
- Eğer **P** matrisi sorunsuz bir şekilde oluşturulmuş ise yine $\mathbf{H_{sys}}$ matrisi üzerinden **P** matrisinin oluşturulmasında kullanılan aynı satırlar üzerinden **R** matrisi oluşturulur. Bu sefer ise **P** matrisinin kullanmadığı diğer bir deyişle depolama düğümlerinden okunabilen sembollerin indekslerine karşılık gelen sütunlar seçilir. Böylece **R** matrisi de oluşturulmuş olur.
- Daha sonra eldeki **P** matrisinin tersi ile **R** matrisi sonlu alan özellikleri çerçevesinde çarpılır ve her satırının bir kurtarma denklemini temsil ettiği nihai matris elde edilmiş olur. Elde edilen bu matriste bir kurtarma denklemine karşılık gelen her satırdaki 1 değerine sahip olan sütun indeksleri, $\overline{\mathbf{D_{gereken}}}$ vektörünün elemanları olan ve bu tamirlerde kullanılacak olan kodlanmış sembol indekslerini tutar.

- Ardından bu matris ile başlangıçta düğümlerden çekilen kodlanmış semboller *XOR* işlemine tabi tutularak kayıp kodlanmış semboller elde edilir. Böylece $\overline{\mathbf{D}_{\mathbf{kayıp}}}$ matrisinin içeriği olan kayıp semboller yeniden elde edilmiş olur.
- Son aşamada ise hücre içerisinde hiç bir içerik bulundurmayan ve rastgele olarak seçilmiş olan boş düğümlerin önbelleklerine tamir edilen kodlanmış semboller yeniden yüklenir. Böylece tamir işlemi tamamlanmış olur.

Çoklu düğüm tamirinin dikkat çeken bir özelliği ise tamir sürecinde kayıp sembol indekslerinin kullanılması söz konusu olmadığından baz istasyonuna gerek duyulmamasıdır.

4.2.2 LT Kod

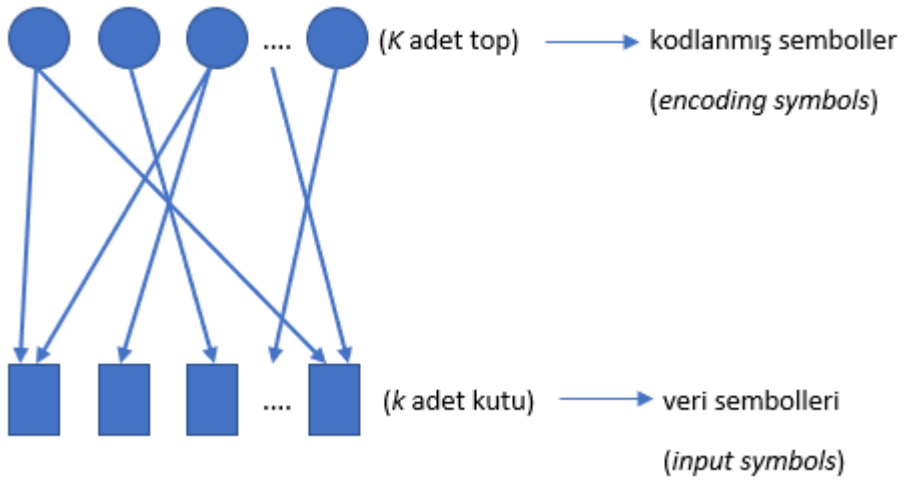
Michael Luby tarafından 2002 yılında geliştirilen *LT* kodlar (Luby 2002) oransızlık özelliğine sahip, çeşme kodları grubuna giren bir silinti düzeltme kodudur. Oransız kodlarda eldeki mevcut veri sembollerinden sonsuz sayıda kodlanmış sembol oluşturulabilmektedir.

Klasik silinti kodlarında sabit bir kod oranı uygulanmaktadır. Şöyle ki k adet sembolden $(n-k)$ adet yedek sembol oluşturulur ve bu kodlamada kullanılan kod oranı k/n olarak sabittir. *LT* kod ise kodlama işlemi esnasında sabit bir kod oranı uygulanmaz. Sabit bir kod oranı olmadan kodlama, gelen talep artışlarını karşılamada avantaj sağlar. Bu özelliği ile düğüm sayısı dinamik bir şekilde değişen hücre sistemlerine uygundur.

4.2.2.1 Kodlama

LT kodlama süreci, olasılık teorisindeki *balls into bins* problemine benzetilmektedir. Bu problemde *m* kadar top ve *n* kadar kutu vardır ve bu toplar birbirinden bağımsız, rastgele olarak kutulara atılır. Yani, bir topun herhangi bir kutu içerisine düşme olasılığı $1/n$ olarak ifade edilir. Bütün toplar kutulara atıldıktan sonra kutulardaki top sayısı her bir kutunun yükü olarak adlandırılır. Bu noktada, *LT* kodlar için kodlanmış semboller topları ifade ederken, kodlanmamış olan veri sembolleri ise

kutuları ifade eder (Luby, 2002). Henüz kodlanmamış olan veri sembollerinin sayısı k ile ifade edilecek olur ise kodlanmış sembollerin sayısı da k'den biraz daha büyük olan $K = kIn(k/\delta)$ ile ifade edilebilir. LT kodlama sürecinde, k adet kutunun her birinde en azından bir topun olması $1 - \delta$ olasılıkla beklenmektedir. Burada δ sembolü ile kodlama sürecindeki başarısızlık olasılığı ifade edilir. Bütün veri sembolleri en azından bir kodlanmış sembol içerisinde kullanıldığı zaman süreç başarıyla tamamlanmış kabul edilir.



Şekil 4.10: *Balls into bins problemi üzerinden LT kodlama süreci.*

LT kodlarda kodlama süreci ise şu şekilde gerçekleşmektedir (Luby 2002):

- Öncelikle N uzunluğundaki veri $k = N/l$ adet sembole (*input symbol*) parçalanır. Öyle ki her sembol l uzunluğunda olur.
- Ardından kodlama sonucu oluşacak sembol için derece dağılım yöntemi ile d olarak ifade edilen derecesi belirlenir.
- Daha sonra ise k adet sembol içerisinden rastgele olarak d adet farklı sembol seçilir ve bu seçilen semboller kodlanacak olan sembolün komşuları olarak adlandırılır.
- Son aşamada ise seçilen d adet sembol (*input symbol*), *XOR* işlemine tabi tutularak kodlanmış sembolü (*encoding symbol*) oluşturur.

Yukarıdaki süreç tek bir kodlanmış sembolün oluşturulması için geçerlidir. Bu tez çalışmasında ise sistem modelinde kullanılan dosyanın bütün olarak daha rahat

kodlanması amacıyla üretici matris G kullanılmıştır. Diğer bir deyişle, bütün kodlanmış semboller üretici matris G'nin kullanımı ile sistematik olarak elde edilir. Öyle ki üretici matris G, veri sembolleri ile kodlanmış semboller arasındaki ilişkiyi temsil eder. İki taraflı bir çizge üzerinden düşünülecek olur ise bu iki tür sembol arasındaki her kenar G matrisinde 1 olarak temsil edilir. G matrisi içerisindeki her sütun, kodlanacak olan bir sembolü temsil eder iken, her satır da henüz kodlanmamış olan bir sembolü temsil eder. Sütun içerisindeki 1'lerin sayısı da ilgili derece dağılımı ile elde edilmiştir. Bu tez çalışmasında, *LT* kodlama süreci aşağıdaki gibi gerçekleştirilmiştir:

- İlk olarak *Robust Soliton* derece dağılım yöntemi kullanılarak bir sembolün *d* derecesine sahip olma olasılığını ifade eden p(d) değerleri belirlenir. Burada dereceden kastedilen, ilgili kodlanacak sembolün *d* adet veri sembolü ile elde edileceğidir.
- Ardından bu p(d) değerlerine göre üretici matris G oluşturulur.
- Üretici matrisin boyutları doğrultusunda hücre içerisinde bulunan depolama düğümlerinden ilgili veri sembolleri okunur. Bu noktada belirtmek gerekir ki bu veri sembolleri henüz kodlanmamış veriler olabileceği gibi *Raptor* kodun yapısı gereği bir silinti düzeltme kodu sonucu elde edilen kodlanmış semboller de olabilir. Örneğin, bu tez kapsamında *LDPC* kodlama sürecinde elde edilen kodlanmış veriler bu aşamada girdi (*input*) olarak kullanılmıştır.
- Ardından üretici matris G ile veri sembolleri *XOR* işlemine tabi tutularak kodlanmış semboller elde edilir.

k adet henüz kodlanmamış olan veri sembolleri $\{S_1, S_2, \dots, S_k\}$ ile ifade edilecek olur ise kodlanmış bir sembol şu şekilde elde edilmiş olur.

$$c_i = S_{i,1} \oplus S_{i,2} \oplus \dots \oplus S_{i,d} \tag{4.10}$$

Bu çalışmada da kullanılmış olan üretici matris üzerinden kodlama sürecinin gösterimi ise şu şekildedir.

$$\mathbf{c} = \mathbf{s} \times \mathbf{G} \tag{4.11}$$

Burada *s* ile kastedilen veri sembolleri vektörüdür. *c* ise kodlanmış sembolleri temsil eden vektördür.

4.2.2.2 Derece Dağılımları

p(d) kodlanacak olan sembolün *d* derecesine sahip olma olasılığını ifade eder. Bu noktada; klasik *balls into bins* problemi, *LT* kodlama sürecinin özel bir durumu olarak ifade edilebilir. Aynı anda dağılım (*all-at-once distribution*) olarak adlandırılan bu özel durumda **p(1) = 1** durumu sağlanmış olur. Diğer bir deyişle, *p(1) = 1* durumu ile kodlanacak olan sembollerin hepsinin derecesi *1* kabul edilir. Böylece kodlanmamış sembollerden rastgele biri seçilir ve bu seçilen sembol direkt olarak kodlanmış sembol olarak kabul edilir. Bu dağılımda da *K* adet kodlanacak sembolün *k* adet veri sembolünü $1 - \delta$ olasılıkla kapsaması beklenmektedir. Böylece kodlanacak sembollerin dereceleri toplamının en azından $K = k.In(k/\delta)$ olması beklenmektedir. Bu özel dağılım durumunda kodlama sürecindeki sembol operasyonlarının sayısı minimum olmasına rağmen, *k* adet veri sembolünün kapsanabilmesi için gereken kodlanmış sembol sayısı, minimum olan bu sayıdan $In(k/\delta)$ kat daha büyük olmaktadır. Bu durumu iyileştirmek adına *Soliton* dağılımları kullanılmaktadır. Öyle ki *k* adet veri sembolünün kapsanması için *k* adet kodlanmış sembol yeterli olur. Böylece *Soliton* dağılımları sayesinde hem kodlanmış sembol sayısı hem de kodlama sürecindeki *XOR* işlem sayısının minimuma yakın olması sağlanmış olur (Luby 2002).

Klasik bir *balls into bins* probleminde, bütün toplar kutulara aynı anda atılır ve bu durumda bir kutu içerisine birden fazla topun düşmesi yüksek bir olasılıktır. Bu nedenle, bütün kutularda en azından bir adet top olabilmesi için mevcut top sayısından daha fazlasına gereksinim duyulacaktır (Luby, 2002). *LT* kodlamada ise *k* adet kutunun her birinin en azından bir adet topa sahip olması amaçlanır. Diğer bir deyişle, her veri sembolünün kodlanmış semboller içerisinde kullanılmış olması amaçlanır. Bu noktada derece dağılım yöntemleri *LT* kodların performansını etkileyen önemli bir faktör olarak ortaya çıkmaktadır. Çünkü derece dağılımı ile mümkün olduğu kadar minimum depolama alanı kullanımı amaçlanır iken aynı zamanda kodlanmış sembollerin dereceleri mümkün olduğu kadar düşük tutularak *XOR* işlemi sayısının minimuma

çekilmesi amaçlanmaktadır. Burada *LT* kodlarda iki tür derece dağılımı ön plana çıkmaktadır. Bunlar aşağıdaki gibidir:

- *Ideal Soliton* Dağılımı
- *Robust Soliton* Dağılımı.

Ideal soliton dağılımı aşağıdaki gibi ifade edilir.

$$p(1) = 1 / k \tag{4.12}$$

$$p(i) = 1 / i\,(i - 1)\,,\ i = 2, \ldots, k. \tag{4.13}$$

Fakat tek komşuya sahip orijinal verilerin tutulduğu *ripple* kümesindeki eleman sayısını mümkün olduğunca küçük tutmaya çalışan bu dağılım, *ripple*'ın içinde hiç eleman kalmaması durumunda kod çözme süreci duracağı için pratikte pek başarılı değildir. Bu nedenle de ikinci derece dağılım yöntemi olan *Robust Soliton* Dağılımı geliştirilmiştir. Bu derece dağılım yönteminde ise *Ideal Soliton* Dağılımı içerisinde kullanılan *p(d)'ye ripple*'ın içerdiği eleman sayısının olasılığını tutan τ parametresi eklenerek *Ideal Soliton* Dağılımı iyileştirilmeye çalışılmıştır. Bu dağılımda kullanılan parametrelere bakılacak olur ise; δ, *K* adet kodlanmış sembol üzerinden verinin yeniden elde edilmesi sürecinde yaşanabilecek başarısızlık olasılığını; *c* ilgili derece dağılımının ortalaması olan sıfırdan büyük bir sabiti; *k* ise henüz kodlanmamış sembol sayısını ifade eder. *Robust Soliton* dağılım sürecinde beklenen *ripple* büyüklüğü (*expected ripple size*) yaklaşık $In(\,k/\delta\,)\sqrt{k}$ olarak kabul edilir. Bu tez kapsamında da kullanılan *Robust Soliton* Dağılımı, aşağıdaki gibi ifade edilir (Luby 2002).

$$R = c.In(\,k/\delta\,)\sqrt{k} \tag{4.14}$$

$$\tau(i) = \begin{cases} R/ik\,, & i = 1, \ldots, k/R - 1 \\ R\ In(R/\delta)/ik\,, & i = k/R \\ 0, & i = k/R + 1, \ldots, k \end{cases} \tag{4.15}$$

Bu tez kapsamında *Raptor* kod iki silinti düzeltme kodunun birleşiminden elde edilmiştir. Dış kod olarak *LDPC* kod kullanılmıştır. Dış koddan gelen kodlanmış

verinin kullanıldığı ve iç kod olarak adlandırılan LT kod ise ikinci aşamadaki silinti düzeltme kodu olmuştur.

4.2.2.3 Tamir Süreci

İnanç yayılımlı kod çözme (*belief propagation decoding*) olarak da bilinen tamir sürecinde ise kodlanmış sembollerden tam olarak bir adet komşu işlenmemiş sembole sahip olan seçilir ve bu kodlanmamış olan komşu sembol orijinal sembol olarak çekilir. Bu işlemin ardından seçilen bu komşu sembol ile komşuluk ilişkisine sahip diğer kodlanmış semboller ayrı ayrı *XOR* işlemine tabi tutulur ve *XOR* işleminin gerçekleştiği kodlanmış sembol ile orijinal sembol arasındaki komşuluk kaldırılıp bu kodlanmış sembollerin dereceleri birer düşürülür.

Yukarıda anlatılan kod çözme sürecinin başlangıcında ise bütün orijinal semboller kapsanmamış (*uncovered*) olarak işaretlenir. Daha sonrasında tek komşuya sahip kodlanmış semboller, orijinal semboller ile eşleştirilir (*covered*). Bu noktada eşleştirilmiş olup henüz işlenmemiş olan orijinal sembollerden oluşan bu küme *ripple* olarak adlandırılır. Her bir sonraki adımda ise *ripple* içerisindeki orijinal sembollerden biri seçilir ve seçilen bu sembolün komşuluk ilişkileri ortadan kaldırılır. Bu komşulukların kaldırılması sürecine işlem (*process*) denir. Bu işlemin ardından tek komşuluk ilişkisine sahip olan semboller seçilerek kod çözme süreci devam ettirilir. Tüm bu kod çözme süreci *ripple* içerisinde eleman kalmayıncaya kadar devam eder.

Bu tez çalışmasında kullanılan tamir (*repair*) yöntemi Gummadi ve diğ. (Gummadi ve Sreenivas 2011) tarafından yapılan çalışma referans alınarak gerçekleştirilmiştir. *Gummadi* tarafından gerçekleştirilen bu çalışmada LT kodlardaki klasik kod çözme yönteminden farklı olarak sadece kodlanmış semboller değil, aynı zamanda kodlanmamış olan semboller de tamir sürecine dahil edilmiştir. Bu çalışmada kaynak mesajın k adet sembolden oluştuğu varsayılmıştır. Bu k adet sembolün tutulduğu vektör S olarak ifade edilir iken, D ile ise tamir edilecek olan kayıp sembollerin indeksleri tutulmuştur. Bütün kodlanmış sembollerin indeks sayısı n olarak ifade edilir ise $D \subseteq [n]$ ve $R \subseteq [n]$ koşulları sağlanmış olur. Bu durumda kurtarma kümesi (*recovery set*) ise $R = [n]/D$ olarak ifade edilmiştir.. Bu süreçte klasik bir tamir işlemi iki aşamadan oluşmaktadır:

1. İlk aşamada kodlanmış ve kodlanmamış mevcut sembollerden oluşan kurtarma kümesi (*R)* kullanılarak kod çözme (*decoding*) işlemi gerçekleştirilir ve kaynak semboller (*S)* elde edilir.
2. Sonraki aşamada ise kodlanmamış sembolleri tutan *S* kullanılarak yeniden kodlama (*encoding*) gerçekleştirilir ve kayıp semboller (*D)* elde edilir.

Kaynak sembollerin $S = \{s_1, \dots, s_k\}$ olarak, kodlanmış sembollerin de $c_1, c_2, \dots$ olarak ifade edildiği bu LT kod türünde bu iki farklı sembol türü birleştirilerek $\{s_1, \dots, s_k, c_1, c_2, \dots\}$ kullanılmıştır. Bu *LT* kod genişletilmiş *LT* kod (*augmented LT code*) olarak adlandırılmıştır. Tamir sürecinde kullanılan parametreler ise şu şekildedir (Gummadi & Sreenivas, 2011).

$$R_c = R \cap \{c_1, c_2, \dots\} \tag{4.16}$$

$$R_s = R_c \cup S \qquad (R = R_c \cup R_s) \tag{4.17}$$

$$D_s = D \cap S \tag{4.18}$$

$$D_c = D/S = D \cap \{c_1, c_2, \dots\} \tag{4.19}$$

Eldeki mevcut kodlanmış sembolleri temsil eden R_c kümesi üzerinden kod çözme süreci gerçekleştirilerek elde edilen kaynak semboller; $s_{\pi(1)}, s_{\pi(2)}, \dots, s_{\pi(k)}$ olarak ifade edilir. Bu kod çözme sürecinde *ripple* içerisinde işleme tabi tutulan kodlanmış semboller ise $c_{\varepsilon(1)}, c_{\varepsilon(2)}, \dots, c_{\varepsilon(k)}$ olarak ifade edilir (Gummadi ve Sreenivas 2011). Böylece bu tamir sürecinin ilk aşamasındaki kod çözme süreci aşağıdaki denklem ile ifade edilebilir.

$$s_{\pi(i)} = c_{\varepsilon(i)} \oplus \sum_{j \in S_i} s_j, \qquad S_i \subseteq \{s_{\pi(1)}, s_{\pi(2)}, \dots, s_{\pi(i-1)}\} \tag{4.20}$$

Diğer bir deyişle, mevcut kaynak sembollerinin sırası gözetilerek bu denklem uygulanır ve D_s içerisindeki kayıp kodlanmamış semboller yeniden elde edilmiş olur.

Tamir sürecinin ikinci aşamasında ise ilk aşamadaki kod çözme yöntemi ile eksik olanlarının da tamir edildiği bütün kaynak semboller (yani **S** vektörü)

kullanılarak kayıp kodlanmış semboller için kodlama işlemi gerçekleştirilir ve D_c elde edilir. Böylece tamir süreci tamamlanmış olur. Bu süreç aşağıdaki gibi gösterilebilir.

$$D_c = S \times G \tag{4.21}$$

Bu tez kapsamında yapılan çalışmada ise kodlama ve tamir süreçleri şu şekilde geliştirilmiştir:

1. Hücre içerisinde LT kod ile kodlanmış sembolleri tutan depolama düğümlerindeki veriler, tuttukları kodlanmış sembollerin sırası gözetilerek, belirlenen bir oranda okunup bir vektöre gönderilir. Bu noktada okunan sembollerin indeksleri tutulur.
2. Depolama düğümlerinden gerçekleştirilen bu okuma işlemi artık veri dediğimiz sembollerin de ilgili düğümlerden sırası ile okunması ile tamamlanır. Artık veri kavramını bir örnek üzerinden anlatmak gerekir ise 15 tane depolama düğümünden 14. sıradaki düğümün kaybolduğu bir tamir senaryosunda, belli bir oranda sembolleri okuyarak 160 baytlık bir vektör doldurulmak isteniyor. Bu noktada 10'ar byte halinde sırasıyla eldeki depolama düğümlerinden veriler okunur. Kayıp olan düğümün sembol indeksleri bu vektörde sıfır olarak tanımlanır. 14x10 = 140 baytlık bir okuma işlemi sonrasında kayıp düğüm için eklenen 10 adet sıfır değeri ile beraber 160 baytlık vektörün 150'lik kısmı doldurulmuş olur. Bu noktadan itibaren okunacak olan bütün veriler ise artık veri olarak adlandırılır. Öyle ki ilgili vektör her depolama düğümünden sırayla 1'er byte okunarak 160'a tamamlanır. Diğer bir deyişle, ilk 10 depolama düğümünden 1 bayt okunur ve bu 10 bayt artık veriyi temsil eder. Bir sonraki döngüde ise hangi düğümde kalındı ise o düğümden devam ederek artık veri kısmı doldurulmaya devam edilir. En son düğümden de 1 bayt okunduktan sonra ise artık veri için okuma işlemi, mevcut depolama düğümleri içerisindeki ilk düğümden tekrar devam edilir. Böylece her döngüde 10 baytlık artık veri sırası ile bu depolama düğümlerinden sağlanmış olur.
3. Bu okuma işlemleri boyutu önceden tanımlanan geçici bellek (*buffer*) kapasitesi boyunca devam ederek bütün bu okunan veri değerleri bir matris üzerinde tutulur.

4. Daha sonra ise genişletilmiş *LT* kod özelliğinden ötürü kaynak semboller hücre içerisindeki diğer depolama düğümlerinden okunarak bir başka matriste toplanır. Buradaki okuma işlemi de tıpkı 2. ve 3. sıradaki işlemlerde olduğu gibi artık veri durumu gözetilerek gerçekleştirilir. Bu noktada belirtmek gerekir ki bu tez kapsamında *Raptor* kod kullanılmasından ötürü buradaki girdi semboller, *LDPC* ile kodlanmış olan depolama düğümlerinden çekilmiştir.
5. Üretici matris *G* üzerindeki her sütun bir kodlanmış sembolü, her bir satır da bir kaynak sembolü temsil eder. Bu matristeki sütunların üzerindeki 1 sayısı, ilgili kodlanmış sembolün komşuluklarını gösterir. Bu sütunlardaki 1'lerin toplamı alınarak kodlanmış sembollerin her birinin kaç komşuya sahip olduğu bilgisi elde edilmiş olur. Bu bilgiyi taşıyan vektör, eşleştirilmiş olup henüz işlenmemiş olan kaynak sembollerden oluşan *ripple* kümesi hakkında da bilgi vermektedir.
6. Kodlanmış sembollerin komşu sayılarını veren bu vektörde 1 değerine sahip olan indeks, sadece bir komşuya sahip olan kodlanmış sembol indeksini göstermektedir. Bu nedenle, ilgili vektörde 1 değerine sahip olan indeksler çekilerek *ripple* kümesine eklenir.

Bu aşamadan sonra ise tamir süreci başlamaktadır. Burada *Gummadi* ve diğ. (2011) tarafından yapılan çalışmadaki tamir sürecinin bir adaptasyonu kullanılmıştır. Önce hayatta olan ve *LDPC* ile kodlanmış veriyi tutan düğümlerden semboller çekilip *repaired* olarak adlandırılan vektöre atılır. Bu vektör orijinal sembollerin değerlerini tutmaktadır. Ardından ise *ripple* tabanlı inanç yayılımlı kod çözme işlemi gerçekleştirilir. Yani hem *LDPC* yardımcı düğümleri hem de *LT* yardımcı düğümleri kullanılarak bu kod çözme işlemi gerçekleştirilir. Bu noktada geçici bellek kullanımı çok önemlidir. Çünkü geçici bellek sayesinde büyük miktarda veri aynı *XOR* işlemlerinden toplu halde geçirilebilir ve bu yöntem çok hız kazandırır. Bu tamir süreci aşağıdaki detaylı olarak anlatılmaktadır.

7. Bu aşamada *LDPC* depolama düğümlerinden çekilen kaynak semboller, tamir edilen kaynak sembollerin tutulduğu *repaired* olarak adlandırılan bir matrise doğrudan eklenir. Ardından da ilgili kodlanmış sembolün tek komşusu olan kaynak sembolün diğer kodlanmış

semboller ile olan komşulukları kaldırılır. Sonrasında ise bu tek komşu sembol, kodlanmış diğer semboller ile *XOR* işlemine tabi tutulur ve yeni değer bu kodlanmış sembollere eklenir. Bu yapılan XOR operasyonu ile $A \oplus A = 0$ kuralı uygulanmış olur ve bu kodlanmış sembolden komşuluğu kaldırılan orijinal sembolün içeriği de kaldırılmış olur.

8. Bu aşamada ise *ripple* kümesindeki tek komşuya sahip kodlanmış sembollerin değerleri sırasıyla alınarak yine *repaired* adlı matrise doğrudan eklenir. Bunun nedeni, zaten tek komşuya sahip olmasından ötürü aslında bu kodlanmış sembolün doğrudan bir kaynak sembolü temsil etmesidir. Sırayla *ripple* içerisinden alınan her sembolün değeri ile *LT* depolama düğümlerinden çekilen semboller *XOR* işlemine tabi tutulur ve yeni değer bu kodlanmış sembollere eklenir.
9. *Ripple* içerisinde hiç eleman kalmayıncaya kadar 6, 7 ve 8. işlemler tekrarlanır. *Ripple* kümesinde eleman kalmadığı zaman tamir işlemindeki ilk süreç tamamlanmış olur. Tamir edilemeyen semboller ise baz istasyonundan çekilmektedir. Böylece bütün kaynak semboller yeniden elde edilmiş olur.
10. Bu noktadan itibaren ise ikinci aşamaya geçilir ve elde edilen kaynak sembolleri üzerinden *LT* kodlama ile kayıp olan kodlanmış semboller yeniden oluşturulur. Diğer bir deyişle, *repaired* olarak adlandırılan ve kaynak sembollerin tutulduğu matris ile üretici matris *G* arasında *XOR* işlemleri gerçekleştirilerek kayıp kodlanmış semboller tamir edilir.

4.3 Reed-Solomon Kod Süreci

Reed-Solomon kodları, 1960 yılında Reed ve Solomon tarafından geliştirilen optimal bir silinti düzeltme kodudur (Reed ve Solomon 1960). Optimal silinti kod olması nedeniyle *Reed-Solomon* kodlarda *k* adet sembolden kodlanarak elde edilen *n* adet kodlanmış sembolün herhangi bir *k* tanesi ile orijinal *k* sembollük veri yeniden elde edilebilmektedir.

4.3.1 Kodlama

Reed-Solomon kodlarda *k* adet sembolden oluşan bir veriyi kodlamak için ilgili veri, q elemandan oluşan ve *F* ile adlandırılan bir sonlu alan içerisinde *k*'den daha küçük dereceli bir polinom ile işleme tabi tutulur. Daha sonrasında ise bu polinom, *F* sonlu alanındaki *n* farklı noktada $(a_1,..., a_n)$ işleme tabi tutulması ile kodlanmış veri elde edilir. Diğer bir deyişle, bu kodlanmış veri aslında *k*'den daha küçük dereceli bir polinomun fonksiyon değerler dizisini ifade eder. Bu işlem aşağıdaki gibi gösterilebilir.

$$C(x) = (p_x(a_1), \dots, p_x(a_n)) \tag{4.22}$$

Sonlu alan *F* üzerindeki *n* farklı nokta için ise *Vandermonde* matrisinin devriği olan matris *A*'nın elemanları kullanılır. *Vandermonde* matrisi her satıra eşçarpanlı dizi (*geometric progression*) üzerinden değerlerin eklendiği bir matris türüdür. $c = x \,.\, A$ işlemi Şekil 4.11'deki gibi gösterilebilir.

$$[x_1\ x_2\ \dots\ x_k] \,.\, \begin{bmatrix} a_1^0 & \cdots & a_k^0 & \cdots & a_n^0 \\ a_1^1 & \cdots & a_k^1 & \cdots & a_n^1 \\ a_1^2 & \cdots & a_k^2 & \cdots & a_n^2 \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ a_1^{k-1} & \cdots & a_k^{k-1} & \cdots & a_n^{k-1} \end{bmatrix} = [c_1\ c_2\ \dots\ c_n]$$

$$x.A = c$$

Şekil 4.11: *Reed-Solomon* orijinal versiyonda klasik kodlama (**c = x . A**).

Bu tez kapsamında kullanılan sistematik *Reed-Solomon* kodlamada, polinom p_x ilk *k* adet sembole karşılık gelen kaynak veri (*source data*) *x*'i tutar ($p_x(a_i) = x_i$; $i \in \{1,...,k\}$). Polinom p_x'i kaynak veri *x*'e ait değerler üzerinden hesaplamak için *Lagrange* interpolasyonu kullanılabilir. Bu interpolasyon işleminin ardından kaynak veri *x*, polinom p_x 'in sonlu alan *F*'ye ait *n* farklı noktada $(a_1, \dots, a_n)$ değerlendirilir ve genişletilmiş veri elde edilir. Şekil 4.12'de *A* matrisinin sistematik versiyonu olan (yani içerisinde birim matris bulunduran) *G=[I,A]* matrisi üzerinden sistematik kodlama gösterilmektedir.

$$[x_1\ x_2\ \dots\ x_k].\begin{bmatrix} 1 & 0 & 0 & \dots & 0 & g_{1,k+1}^{0} & \dots & g_{1,n}^{0} \\ 0 & 1 & 0 & \dots & 0 & g_{2,k+1}^{1} & \dots & g_{2,n}^{1} \\ 0 & 0 & 1 & \dots & 0 & g_{3,k+1}^{2} & \dots & g_{3,n}^{2} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 & g_{k,k+1}^{k-1} & \dots & g_{k,n}^{k-1} \end{bmatrix} = [c_1\ c_2\ \dots\ c_n]$$

Şekil 4.12: *Reed-Solomon* orijinal versiyonda sistematik kodlama (***c*** **=** ***x*** **.** ***G***).

4.3.2 Kod Çözme

Kod çözme aşamasında ise bu simülatörde *Reed-Solomon* kodların optimal özelliğinden faydalanılmıştır. Bir veya birden fazla düğüm ağ içerisindeki hücreden ayrıldığı zaman, hücre içerisinde bulunan diğer depolama düğümlerinden *k* adet kodlanmış sembol okunarak *G* matrisi ile sonlu alanlar üzerinden çarpım işlemine tabi tutulur ve kodlanmamış olan *k* adet orijinal sembol elde edilir (Plank 1997).

Reed-Solomon (*RS*) kodlar için geçerli tamir süreci de tıpkı *LT* kodlarda olduğu gibi iki aşamadan oluşur:

- İlk aşamada hücre içerisindeki mevcut RS depolama düğümlerindeki kodlanmış semboller okunarak geçici bellekte depolanır ve daha sonrasında bu geçici bellekte tutulan kodlanmış sembollerden herhangi bir k tanesi rastgele olarak seçilir.
- İkinci aşamada ise depolama düğümlerinden okunup rastgele olarak seçilen *k* adet kodlanmış sembol, üretici matris *G* ile sonlu alan özellikleri temelinde çarpım işlemine tabi tutulur. Bu işlem sonucunda kayıp kodlanmış semboller yeniden elde edilir ve sonrasında bu tamir edilmiş semboller hücre içerisinde tamir için rastgele seçilmiş olan boş düğümlerden bir veya birkaçına gönderilir.

Burada belirtmek gerekir ki eğer hücreden ayrılan depolama düğümü sayısı yüksek olursa ve buna bağlı olarak diğer depolama düğümlerinden çekilen kodlanmış sembol sayısı *k* tanenin altına düşerse eldeki kodlanmış sembol sayısını *k* adete

tamamlamak için gereken sembol sayısı baz istasyonundan çekilerek sağlanır. Örneğin, *k=12* adet kodlanmış sembol ihtiyacı var iken eldeki depolama düğümlerinden sadece 8 adet kodlanmış sembol çekilebildiyse, geri kalan 4 adet kodlanmış sembol de baz istasyonu üzerinden tedarik edilir. Aşağıda *RS* kodlarda gerçekleştirilen tamir işlemi gösterilmektedir:

$$[c_1\ c_2\ \dots\ c_k] \cdot \begin{bmatrix} 1 & 0 & 0 & \dots & 0 & g_{1,k+1}^{0} & \dots & g_{1,n}^{0} \\ 0 & 1 & 0 & \dots & 0 & g_{2,k+1}^{1} & \dots & g_{2,n}^{1} \\ 0 & 0 & 1 & \dots & 0 & g_{3,k+1}^{2} & \dots & g_{3,n}^{2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & g_{k,k+1}^{k-1} & \dots & g_{k,n}^{k-1} \end{bmatrix} = [c_1\ c_2\ \dots\ c_n]$$

Şekil 4.13: *Reed-Solomon* sistematik kod çözme.

Burada sistematik bir üretici matris olan *G*'nin kullanılmasından ötürü elde edilen kodlanmış sembollerin ilk *k* tanesi orijinal veri sembollerini temsil etmektedir.

Bu tez kapsamında gerçekleştirilen kodlanmış veri önbellekleme yönteminde ilk defa olarak *D2D* iletişim içerisinde *LT, LDPC* ve *Raptor* kodları test edilmiştir. Bu tez çalışması bu yönüyle özgün bir yere sahiptir. Ayrıca hem Pedersen ve diğ. (2016) hem de Paakkonen ve diğ. (2013) tarafından yapılan çalışmalarda hücre içerisindeki düğümlerin hep sabit ve eşit bir miktarda sembol tuttuğu varsayılmıştır. Bu çalışmada ise artık veri kullanımı gerçekleştirilmiştir. Simülatör bu özelliği ile testler için daha gerçekçi bir ortam sağlamıştır.

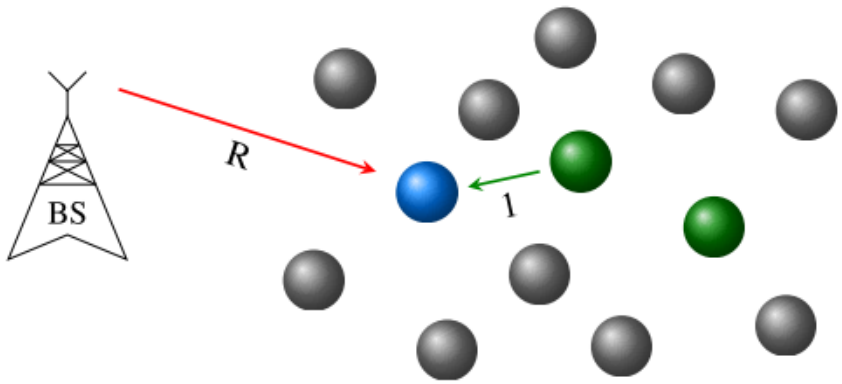
4.4 3-Kopyalama Süreci

Kopyalama temeline dayanan yedekleme yöntemi en basit silinti düzeltme yöntemlerinden biridir. Bu yöntemde yedeklilik için orijinal sembollerin kodlanmasına gerek yoktur. Aynı şekilde kayıp orijinal sembollerin yeniden elde edilmesi için bir kod çözme yöntemine de gerek yoktur. Kopyalamaya dayalı silinti düzeltme kodları 2-Kopyalama (*2-Replication*), 3-Kopyalama (*3-Replication*) gibi orjinal dosyanın kaç adet kopyasının dağıtık olarak tutulacağına göre çeşitlenmektedir.

3-Kopyalama yöntemine göre, orijinal dosya hücre içerisinde 3 farklı düğümde kopyalanarak aynı şekilde tutulur.

Kopyalama yöntemi genel olarak şu şekilde açıklanmaktadır: kopyalama şemasında bir dosya içeriği tekrar edecek şekilde saklanır. İlgili dosya k adet sembole bölünür ve her sembol r adet farklı depolama düğümünde tekrarlanarak yedeklenir. Bu işlemin bir sonucu olarak k adet sembol, $n = k \times r$ adet sembole çoğaltılarak yedeklenmiş olur (Friedman ve Kantor 2013). Bu yöntemde $k \geq 1$ ve $r \geq 1$ koşulları gerekmektedir.

Kopyalamaya dayalı silinti düzeltme kodlarında en basit versiyon ise 2-Kopyalama kodudur. Paakkonen ve diğ. (2013) tarafından kullanılan 2-Kopyalama yönteminde bütün dosyanın 2 adet kopyası hücre içerisindeki iki farklı düğümde saklanır. Bu kopyalama yönteminde tamir süreci ise şöyle olur: eğer bu düğümlerden biri hücre içerisinden ayrılır ise hücre içerisinde hiçbir veriye sahip olmayan bir düğüm rastgele olarak seçilir ve kayıp verideki dosya diğer kopyaya sahip düğümden direkt olarak kopyalanarak seçilen boş düğüme kaydedilir. Böylece tamir işlemi tamamlanmış olur. Şekil 4.14'de 2-Kopyalama yöntemi hücre içerisinde gösterilmektedir. Bu şekilde yeşil olarak gösterilen düğümler ilgili dosyanın bir kopyasını bulundurmaktadır. Gri renkteki düğümler boş düğümleri göstermektedir. Mavi düğüm ise kayıp düğümdeki verinin aktarılacağı düğümdür. Kopyaya sahip yeşil düğümlerden ilgili dosyanın tedarik edilmesinin maliyeti 1 olarak gösterilir iken, aynı dosyanın baz istasyonu üzerinden yeniden sağlanmasının maliyeti ise R olarak temsil edilmiştir.



Şekil 4.14: 2-Kopyalama yöntemi (Paakkönen ve diğ. 2013).

3-Kopyalama yöntemini tasarlamak için ise hücre içerisindeki dosya, orijinal veri sembolleri olarak da adlandırılan k adet veriye bölünür. Hücre içerisindeki düğüm sayısının da s adet olduğu varsayılır ise bu k adet veri sembolü belirlenen belli bir oranda s adet depolama düğümünden ilk $s/3$ adet depolama düğümünde yedeklenir. Bu yedekleme işleminde k adet veri sembolü bu $s/3$ adet düğüme mümkün olduğunca eşit miktarda dağıtılacak şekilde yapılır. Belli bir oran belirlenir ve k adet sembol sırasıyla, belirlenen orana göre ilgili $s/3$ adet düğüme kaydedilir. Ardından dosya içeriğinin kopyalandığı $s/3$ adet depolama düğümündeki içerikler, geriye kalan $2s/3$ adet düğüme de aynı şekilde kopyalanır. Böylece 3-Kopyalama tekniği hücre içerisindeki depolama düğümlerine uygulanmış olur. Dosya içeriğini oluşturan veri sembollerinin her biri 3'er kopyaya sahip olur. Bu kopyalama süreci bir örnek üzerinden anlatılacak olur ise 48 baytlık veriye sahip bir dosyanın içeriği hücre içerisindeki 24 adet depolama düğümüne 3-Kopyalama yöntemine göre yedeklenmek isteniyor. Öncelikle 24 adet depolama düğümü 3'e bölünerek 8 depolama düğümünden oluşan 3 ayrı grup oluşturulur. Daha sonra ise ilk 8 depolama düğümüne bu 48 baytlık veri 8 parçaya bölünerek sırasıyla kaydedilir. Böylece seçilen bu 8 düğümün her biri bu dosyaya ait 6 baytlık içeriği önbelleğinde tutmuş olur. Bu işlemin ardından bu 8 adet düğümün sakladığı içerikleri diğer 16 depolama düğümüne aynı şekilde kopyalanır. Böylece 24 depolama düğümünün her biri 6 bayt içerik saklar ve her içerik 3 farklı düğümde tutulmuş olur.

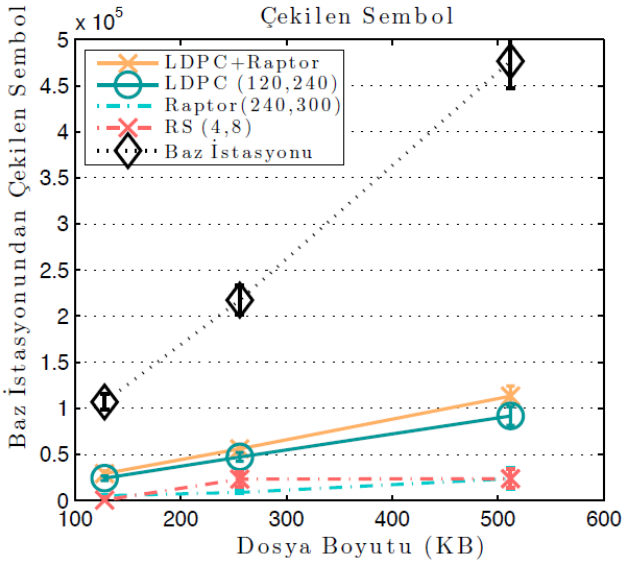
Tamir sürecinde ise öncelikle kayıp düğüm ya da düğümler için hücre içerisinde hiçbir içeriğe sahip olmayan düğümlerden kayıp düğüm sayısı kadar olacak sayıda rastgele olarak seçilir. Hücre içerisinde kayıp düğümün içeriğine sahip başka düğüm olup olmadığına bakılır. Eğer kayıp düğümün içeriğinin kopyalarına sahip diğer 2 düğümden herhangi biri hayatta ise ilgili içerik bu düğümden indirilerek seçilen boş düğümlerden birine kaydedilir. Eğer kayıp düğümün içeriğine sahip diğer 2 depolama düğümü de hücreden ayrılmış ise ilgili içerik baz istasyonu üzerinden indirilir. Bu işlemler her kayıp düğüm için tekrarlanır. Diğer silinti düzeltme kodlarında olduğu gibi 3-Kopyalama yönteminde de hem hücre içerisindeki depolama düğümleri hem de depolama düğümlerinden ilgili içeriğin çekilemediği durumlarda devreye giren baz istasyonu kullanılmış olur.

5. SİMÜLASYON SONUÇLARI

Simülasyondaki testler bir dizi silinti düzeltme kodunun tamir sürecinde kullandıkları çeşitli maliyet parametrelerinin karşılaştırılması ile gerçekleştirilmiştir. Bu kapsamda *Reed-Solomon* kodu, *LDPC* ve *Raptor* kodların birleşiminden oluşan *Raptor* kodu, *LDPC* kodu ve LDPC ile Raptor kodun beraber çalıştırıldığı kod bu çalışmada kullanılmıştır. Ayrıca sadece *LDPC* kod da aynı maliyet parametreleri üzerinden tamir sürecinde performans analizleriyle karşılaştırılmıştır. Bu simülasyon testlerinde kullanılan maliyet parametreleri ise tamir sürecinde baz istasyonundan çekilen sembol sayısı, depolama düğümlerinden çekilen sembol sayısı, kullanılan yardımcı düğüm sayısı, tamir süreleri ve bunların normalize edilmiş halleri olmuştur. Normalizasyondan kastedilen ise tamir sürecinde kullanılan bütün sembollerin tek bir sembole indirgenmesidir. Böylece tamir edilen tek bir sembol için maliyet parametrelerinin sonuçları alınabilmiştir. Bu testler için sistem modelinde tasarlandığı gibi basitlik sağlaması amacıyla tek bir dosyanın yedekli bir şekilde depolandığı varsayılmıştır. Bu dosya, boyutu *128 KB*, *256 KB* ve *512 KB* olacak şekilde 3 farklı versiyon olarak testlerde kullanılmıştır. Bu testler boyunca tembel tamir yöntemi kullanılmıştır. Diğer bir deyişle anlık tamirde olduğu gibi depolama düğümlerinden bir veya birkaçının hücreden ayrılmasının hemen ardından tamir gerçekleştirilmemiştir. Anlık tamir durumunda tamir aralığını ifade eden Δ değişkeni sıfır olmaktadır. Tembel tamir yöntemi için bu çalışmada tamir aralığı olarak $\Delta = 0.2$ ve $\Delta = 0.4$ ardışık değerleri kullanılmıştır. Yani Δ arttıkça düğüm tamiri aralığı artmakta, bu süre zarfında daha çok düğüm kaybı olmaktadır.

Şekil 5.1'de tamir aralığının $\Delta = 0.2$ olarak gerçekleştirildiği tamir sürecinde baz istasyonundan çekilen sembol sayısı açısından ilgili silinti düzeltme kodlarının kullanılan dosya boyutuna göre sonuçları gösterilmektedir. Bu şekildeki baz istasyonu çizgisi ise bütün tamir işleminin sadece baz istasyonu üzerinden gerçekleştirilmesi durumunda baz istasyonundan indirilmesi gereken sembol sayısını göstermektedir. Baz istasyonu çizgisine bakarak silinti düzeltme kodları sayesinde tamir işleminin büyük çoğunluğunun hücre içerisindeki düğümlerden çekilen semboller ile sağlandığı görülmektedir. Dosya boyutu en küçük versiyon olan *128 KB* iken bütün silinti kodların neredeyse aynı miktarda baz istasyonundan veri çektiği görülmektedir. Fakat dosya boyutu arttıkça silinti kodları farklı performans göstermeye başlamıştır.

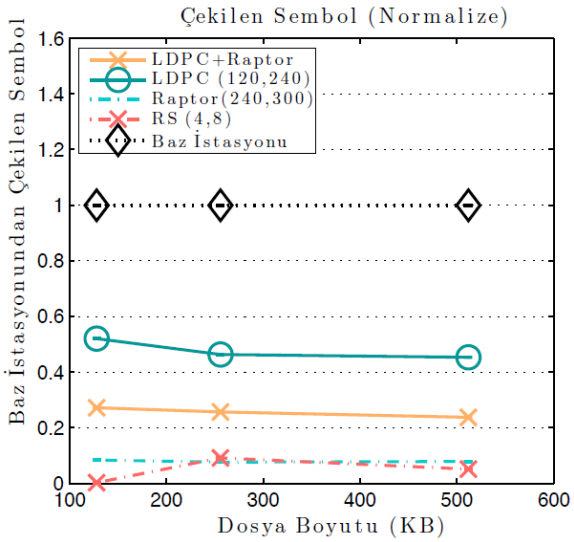
Şekil 5.1 dosya boyutları üzerinden incelenir ise dosya boyutları arttıkça baz istasyonundan en az sembol indirme eğilimi *Raptor* kod ve *Reed-Solomon* (*RS*) kodda gözlemlenmektedir. Bu durum $\Delta = 0.2$ aralıklarla gerçekleştirilen tamir süreçleri için düğümler arasında veri dayanıklılığının (*reliability*) en yüksek *Raptor* ve *Reed-Solomon* kodlarda gözlemlendiği anlamına gelmektedir. Diğer yandan baz istasyonundan en çok veri indiren silinti düzeltme kodu ise *LDPC* ve *Raptor* kodların beraber çalıştırıldığı hali olmuştur. Baz istasyonundan en çok veri indiren ikinci kod ise *LDPC* kod olmuştur. Bu sonuçlardan yola çıkarak aslında dağıtık depolamada veri dayanıklılığı açısından *LDPC* kodların en kötü performansa sahip olduğu söylenebilir.



Şekil 5.1: Baz istasyonundan çekilen sembol sayısı açısından silinti düzeltme kodlarının dosya boyutuna göre sonuçları ($\Delta = 0.2$).

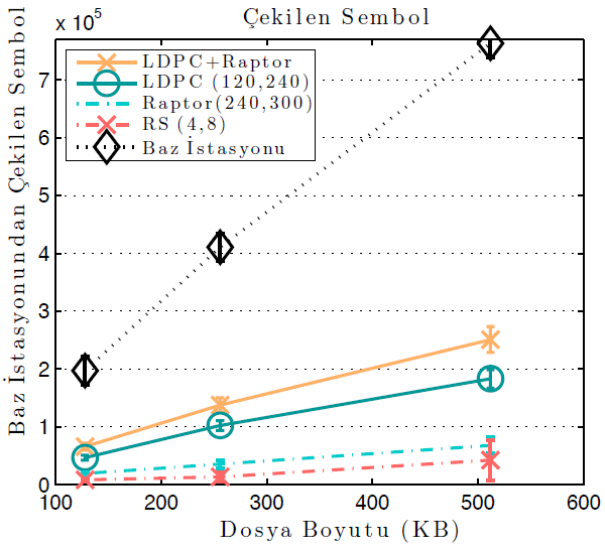
Şekil 5.2'de ise tamir edilecek sembol başına baz istasyonundan çekilen sembol sayısı açısından silinti düzeltme kodlarının dosya boyutuna göre sonuçları gösterilmektedir. Diğer bir deyişle, bir sembolün tamiri için silinti düzeltme kodları tarafından baz istasyonundan çekilen sembol sayısı analiz edilmiştir. Yani sonuçlar normalize edilmiştir. Buradaki baz istasyonu çizgisi kayıp bir adet sembolün tamiri için yine bir adet sembolün baz istasyonundan çekilmesi gerektiğini göstermektedir. Silinti düzeltme kodlarına bakıldığında ise kayıp bir sembol başına baz istasyonundan en çok sembol çeken silinti kodu *LDPC* olmuştur. Onu *LDPC* ve *Raptor* kodların beraber çalıştırıldığı versiyon takip etmiştir. Kayıp sembol başına baz istasyonundan

en az sembol çeken silinti kodu ise *Reed-Solomon* kodu olmuştur. *Raptor* kodu da *Reed-Solomon* koda çok yakın performans göstermektedir. Bu açıdan dağıtık depolamada veri dayanıklığı açısından *Reed-Solomon* ve *Raptor* kodlar *LDPC* koda göre daha iyi performans göstermiştir.



Şekil 5.2: Tamir edilecek sembol başına baz istasyonundan çekilen sembol sayısı açısından silinti düzeltme kodlarının dosya boyutuna göre sonuçları (Δ = 0.2).

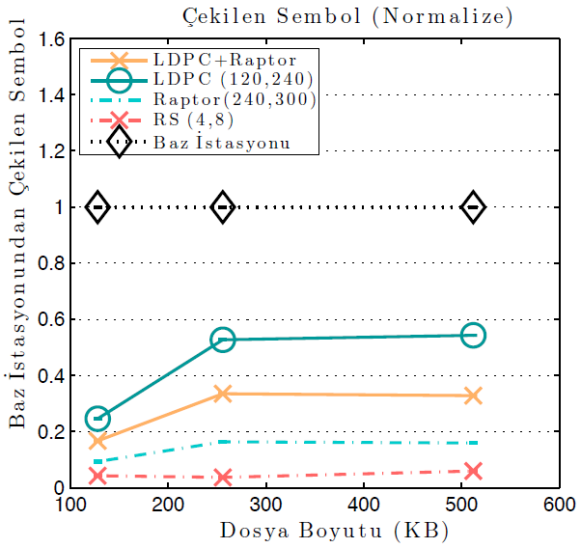
Tamir aralığının yükseltildiği durumda ise (Δ = *0.4*) tamir süreci boyunca silinti düzeltme kodları tarafından baz istasyonundan çekilen sembol sayısı Şekil 5.3'te gösterilmektedir. Tamirin tamamen baz istasyonundan gerçekleştirilmesi durumunda indirilmesi gereken sembol sayısını gösteren baz istasyonu çizgisine bakarak silinti düzeltme kodlarının tamir işlemini bu tamir aralıklarında da büyük oranda hücre içerisindeki düğümlerden gerçekleştirdiği gözlemlenmektedir. Dosya boyutları büyüdükçe silinti düzeltme kodlarının performansları arasındaki farkın açıldığı görülen Şekil 5.3'te yine *LDPC* ve *Raptor* kodun beraber çalıştırıldığı silinti kodunun baz istasyonundan en çok sembol indirme eğilimi gösterdiği gözlemlenmiştir. Bu silinti kodunu sırasıyla *LDPC, Raptor* ve *Reed-Solomon* kodları izlemiştir. Bu sonuçlardan yola çıkarak *LDPC* kodların baz istasyonundan en çok sembol indiren kod olduğu görülmüştür. Baz istasyonundan en az sembol indiren silinti kodu ise *Reed-Solomon* kodu olmuştur.



Şekil 5.3: Baz istasyonundan çekilen sembol sayısı açısından silinti düzeltme kodlarının dosya boyutuna göre sonuçları (Δ = 0.4).

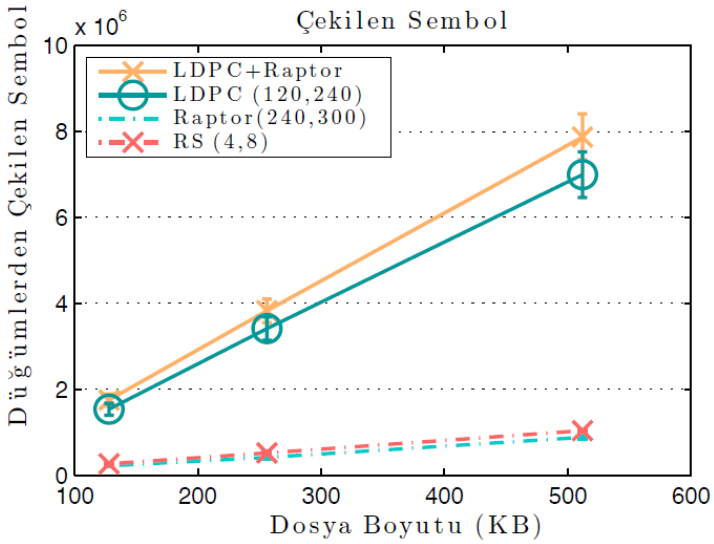
Tamir aralığının yükseltildiği durum (Δ = *0.4*) için tamir edilecek semboller normalize edilerek sembol başına silinti kodları tarafından baz istasyonundan indirilen sembol miktarları şekil 5.4'te gösterilmektedir. Burada sembol başına baz istasyonundan en çok sembol indiren silinti düzeltme kodu *LDPC* olmuştur. Onu sırasıyla *LDPC* ve *Raptor* kodun beraber çalıştırıldığı versiyonu, *Raptor* kodu ve *Reed-Solomon* kodu izlemiştir. Diğer bir deyişle, tamir edilecek sembol başına baz istasyonundan en çok veri çeken *LDPC* olur iken baz istasyonudan en az veri çeken silinti kodu ise *Reed-Solomon* kodu olmuştur.

Şekil 5.1, 5.2, 5.3 ve 5.4'teki sonuçlara genel olarak bakılacak olur ise tamir aralıkları artırılsa da silinti kodların baz istasyonundan çektikleri sembol sayısı kıyaslamalarında belirgin bir farkındalık görülmemiştir. Dağıtık depolama anlamında veri dayanıklılığı en yüksek kod, baz istasyonundan en az veri çeken *Reed-Solomon* kodu olmuştur. *Raptor* kod da *Reed-Solomon* koduna çok yakın sonuçlar elde etmiştir. Veri dayanıklılığı en zayıf silinti kod ise bu sonuçlara göre *LDPC* kod olmuştur.



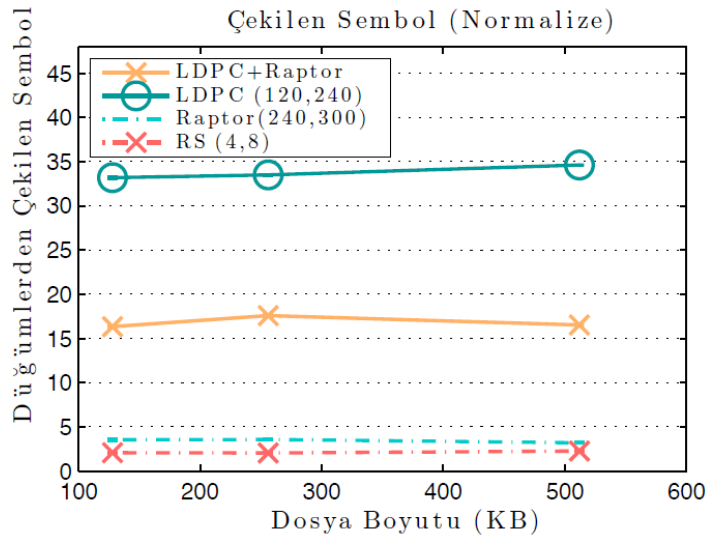
Şekil 5.4: Tamir edilecek sembol başına baz istasyonundan çekilen sembol sayısı açısından silinti düzeltme kodlarının dosya boyutuna göre sonuçları (Δ = 0.4).

Şekil 5.5'te ise tamir aralığının *0.2* olarak belirlendiği (Δ = *0.2*) tamir süreçleri için silinti düzeltme kodları tarafından hücre içerisindeki depolama düğümlerinden çekilen sembol sayısına göre karşılaştırma sonuçları gösterilmektedir. Tamir sürecinde ağ içerisinde iletilmek zorunda olunan veri miktarı tamir bant genişliği (*repair bandwidth*) olarak tanımlanmaktadır (Pedersen ve diğ. 2016). Bu açıdan tamir süresince düğümlerden ne kadar az veri çekilirse, tamir bant genişliği maliyeti de daha az olur. Şekil 5.5'teki sonuçlara göre, 128 *KB*'lık dosya için alınan sonuçlarda *Raptor* ve *Reed-Solomon* kodları neredeyse aynı miktarlarda düğümlerden veri çekerken, *LDPC* ve *LDPC* ile *Raptor* kodun beraber çalıştırıldığı kodlarda ise birbirlerine yakın miktarlarda düğümlerden veri çektikleri gözlemlenmiştir. Dosya boyutu büyüdükçe ise silinti kodlarının düğümlerden çektiği veri miktarları da değişkenlik göstermeye başlamıştır. Genel olarak ise tamir süresince düğümlerden en az veri çeken kod *Raptor* kod olmuştur. *Raptor* kodu çok az bir farkla *Reed-Solomon* kodu takip etmiştir. *LDPC* ise bu maliyet parametresi açısından *Raptor* ve *Reed-Solomon* kodlara göre daha kötü bir performans göstermiştir. Düğümlerden en çok sembol çeken ve dolayısıyla en yüksek tamir bant genişliği değerine sahip olan kod ise *LDPC* ve Raptor kodun beraber çalıştırıldığı versiyon olmuştur.



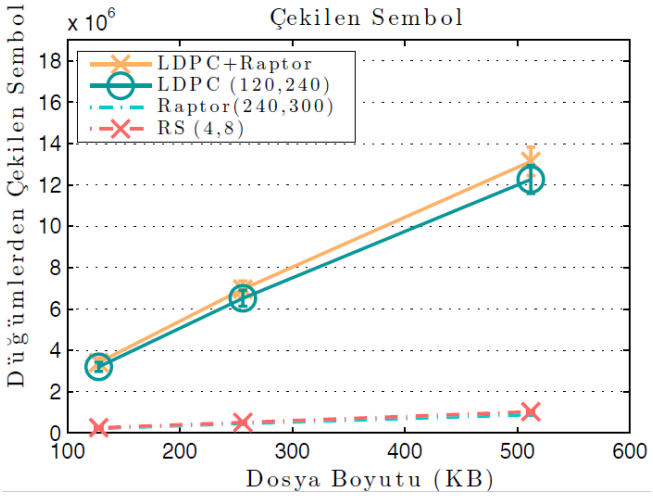
Şekil 5.5: Düğümlerden çekilen sembol sayısı açısından silinti düzeltme kodlarının dosya boyutuna göre sonuçları (Δ = 0.2).

Şekil 5.6'da ise 0.2 tamir aralıklarında *(Δ = 0.2)* devreye giren tamir işlemleri için bu silinti kodların düğümlerden çektikleri sembol sayısı normalize edilerek birim sembol üzerinden performans sonuçları verilmiştir. Bu şekildeki sonuçlara göre *LDPC* kodu sembol başına düğümlerden en çok sembol çeken silinti kodu olmuştur. *LDPC*'yi *LDPC* ve *Raptor* kodlarının beraber çalıştırıldığı versiyon takip etmiştir. Sembol başına düğümlerden en az sembol çeken kod ise *Reed-Solomon* kodu olmuştur. *Reed-Solomon* kodu az bir farkla *Raptor* kod takip etmiştir.



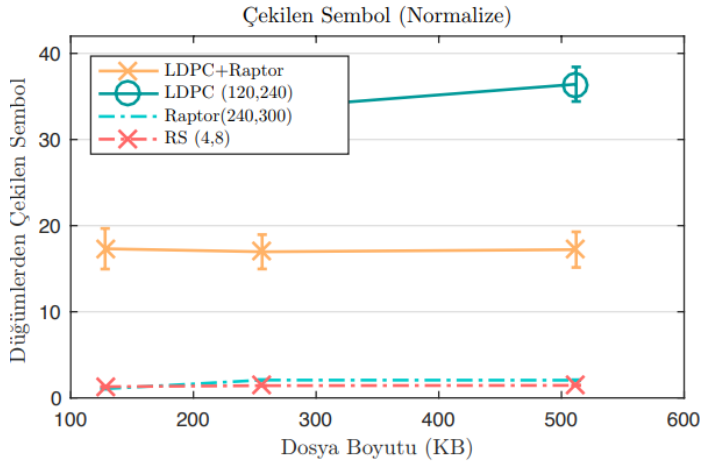
Şekil 5.6: Tamir edilecek sembol başına düğümlerden çekilen sembol sayısı açısından silinti düzeltme kodlarının dosya boyutuna göre sonuçları (Δ = 0.2).

Şekil 5.7'de tamir aralığı büyütülerek $(\Delta = 0.4)$ bu silinti düzeltme kodları tarafından tamir sırasında düğümlerden çekilen sembol sayısı gösterilmektedir. Şekil 5.7'deki sonuçlara göre *Raptor* kodu çok az bir farkla *Reed-Solomon*'dan daha az sembol çekmiştir. *LDPC* ve *Raptor* kodun beraber çalıştırıldığı yöntem ise sadece *LDPC* kodunun çalıştırıldığı duruma göre düğümlerden daha fazla sembol çekerek en yüksek tamir bant genişliğine sahip silinti kodu olmuştur.



Şekil 5.7: Düğümlerden çekilen sembol sayısı açısından silinti düzeltme kodlarının dosya boyutuna göre sonuçları (Δ = 0.4).

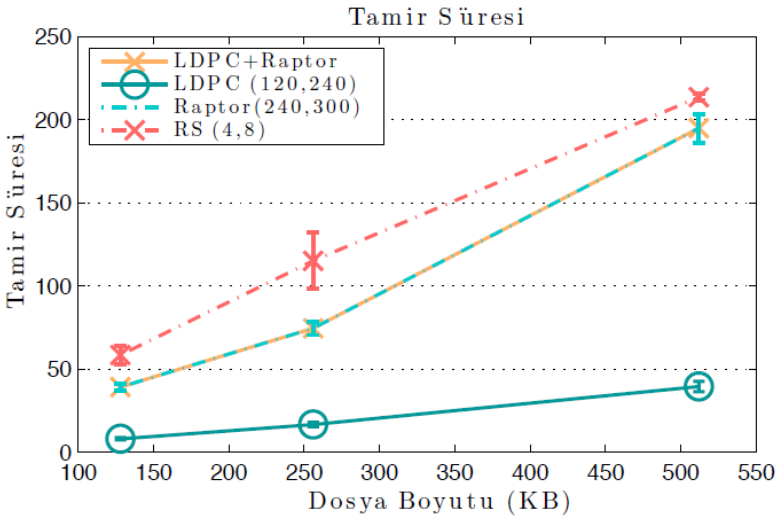
Şekil 5.8'de ise tamir edilecek sembol başına ilgili silinti kodlarının düğümlerden çekmiş olduğu sembol sayısı, 0.4 tamir aralıklarıyla gerçekleştirilen bir tamir süreci $(\Delta = 0.4)$ üzerinden gösterilmektedir. Bu grafiğe göre *Reed-Solomon* kodu, *Raptor* kodundan çok az bir farkla düğümlerden daha az veri çekmiştir ve bu nedenle en iyi performansı göstermiştir. Diğer yandan *LDPC* kod ise tamir edilecek sembol başına açık ara düğümlerden en çok sembol çeken silinti kodu olmuştur.



Şekil 5.8: Tamir edilecek sembol başına düğümlerden çekilen sembol sayısı açısından silinti düzeltme kodlarının dosya boyutuna göre sonuçları ($\Delta = 0.4$).

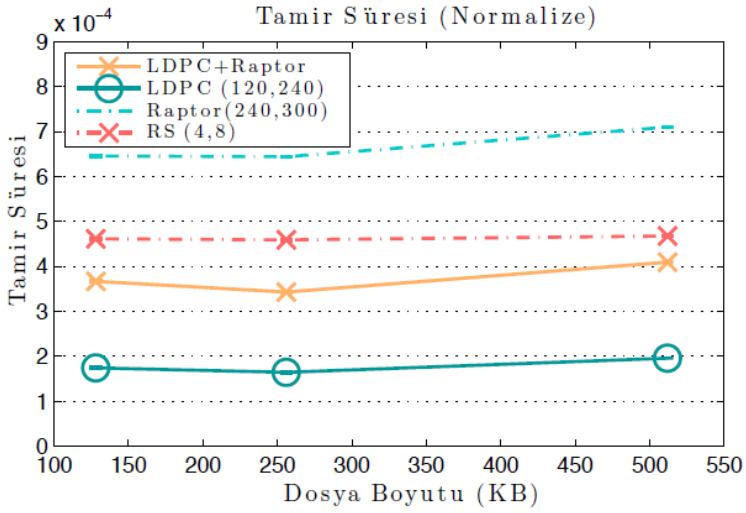
Şekil 5.5, 5.6, 5.7 ve 5.8'de genel olarak tamir sürecinde silinti kodları tarafından depolama düğümlerinden çekilen sembol sayısı gözlemlenmiştir. Bu sonuçlara göre her iki tamir aralığında da *Raptor* kod çok az bir farkla *Reed-Solomon* koddan daha az sembol çekmiştir. Bu açıdan en az tamir bant genişliği (*repair bandwidth*) maliyetini *Raptor* kod sağlamıştır. Normalize edilen grafiklere göre ise yine her iki tamir aralığı için de *Reed-Solomon* kod sembol başına en az sembol çeken silinti kodu olmuştur. Onu çok az bir farkla *Raptor* kodu takip etmiştir. Sembol başına düğümlerden çekilen sembol miktarında en kötü performans *LDPC* kodunda gözlemlenmiştir.

Şekil 5.9'da 0.2 tamir aralığında gerçekleştirilen tamir süreçleri üzerinden kullanılan dosya boyutlarına göre ilgili silinti düzeltme kodlarının ortalama tamir süreleri gösterilmektedir. Bu sonuçlara göre dosya boyutlarındaki değişimler 128 *KB* ile 256 *KB* arasında tamir sürelerinde ufak artışlara neden olurken, 256 *KB* ile 512 *KB* arasındaki geçişlerde ise *LDPC* kod haricindeki diğer kodların tamir sürelerinde önemli yükselişlere neden olmuştur. Tamir sürelerine bakıldığında ise *LDPC* kodun en kısa tamir süresine sahip olduğu görülmektedir. *LDPC* kodu, *LDPC* ile *Raptor* kodun beraber çalıştığı yöntem ile *Raptor* kodu takip etmiştir. En uzun tamir süresi ise *Reed-Solomon* kodunda gözlemlenmiştir.



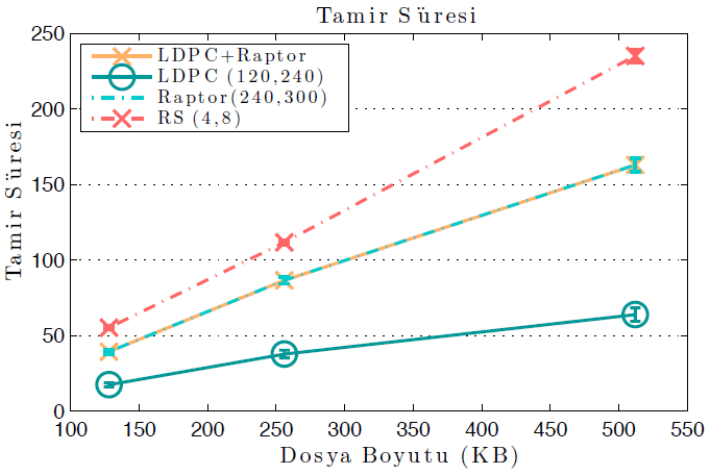
Şekil 5.9: Tamir süresi açısından silinti düzeltme kodlarının dosya boyutuna göre sonuçları (Δ = 0.2).

Şekil 5.10'da silinti düzeltme kodlarının bir sembolün tamiri için 0.2 aralıklı gerçekleştirilen tamirlerdeki normalize edilmiş tamir süreleri karşılaştırılmıştır. Dosya boyutlarındaki değişim bütün silinti kodlarının tamir sürelerinde dikkat çekici bir değişiklik yapmamıştır. Kayıp bir sembolün tamiri için gereken tamir sürelerinde en iyi performansın *LDPC* kodlarda olduğu gözlemlenmiştir. *LDPC* kodu, sırasıyla *LDPC* ile *Raptor* kodun beraber çalıştırıldığı versiyonu, *Reed-Solomon* kodu ve son olarak da *Raptor* kodu takip etmiştir.



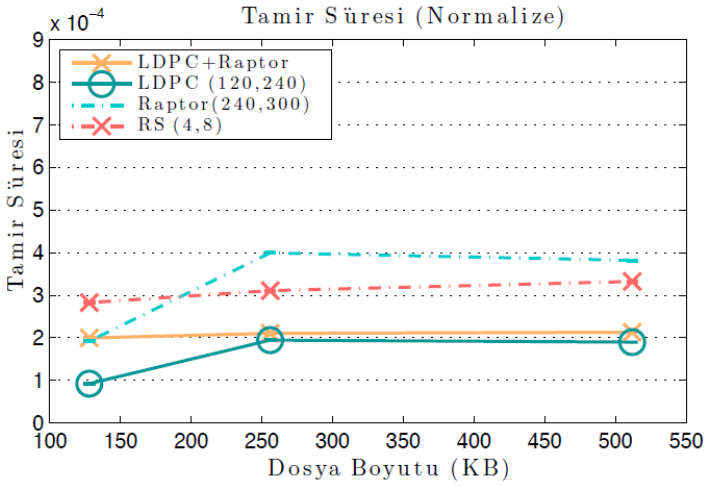
Şekil 5.10: Bir kayıp sembolün tamir süresi açısından silinti düzeltme kodlarının dosya boyutuna göre sonuçları (Δ = 0.2).

Tamir aralığının daha geniş alındığı ($\Delta = 2$) senaryoda ise en kısa tamir süresi *LDPC* kodunda gözlemlenmiştir. *LDPC* ile *Raptor* kodun beraber çalıştırıldığı kod ile *Raptor* kodu neredeyse aynı sürelerde tamiri tamamlamaktadır. Kayıp sembolleri en uzun sürede tamir eden silinti düzeltme kodu ise *Reed-Solomon* kodu olmuştur.



Şekil 5.11: Tamir süresi açısından silinti düzeltme kodlarının dosya boyutuna göre sonuçları (Δ = 0.4).

Şekil 5.12'de bir adet kayıp sembolün tamiri senaryosu üzerinden silinti kodların ortalama tamir süreleri kullanılan 3 farklı boyuttaki dosyalara göre gösterilmiştir. En küçük dosya olan 128 *KB*'lık dosyada sembol başına en kısa tamir süresi *LDPC*'de gözlenirken, *LDPC* ve *Raptor* kodun beraber çalıştırıldığı silinti kodu ile *Raptor* kodun neredeyse aynı tamir sürelerine sahip olduğu görülmüştür. Bu dosya için en uzun tamir süresi ise *Reed-Solomon* kodlarında gözlemlenmiştir. Dosya boyutu 256 *KB*'a çıktığında ise *Raptor* kodun en uzun tamir süresine sahip olduğu görülmektedir. Yine bu dosya için en kısa tamir süresinin ise *LDPC* kodu tarafından gerçekleştirildiği görülmektedir. 512 *KB*'lık dosyada da sonuçlar 256 *KB*'lık dosyadakine benzer gelmiştir. Buna göre en kısadan en uzun tamir sürelerine sırasıyla *LDPC, LDPC* ve *Raptor* kodunun beraber çalıştırıldığı kod, *Reed-Solomon* kodu ve *Raptor* kod olmuştur.



Şekil 5.12: Bir kayıp sembolün tamir süresi açısından silinti düzeltme kodlarının dosya boyutuna göre sonuçları ($\Delta = 0.4$).

Her iki tamir aralığında gerçekleştirilen tamirler incelendiğinde en uzun tamir süresi *Reed-Solomon* kodda gözlemlenirken, en kısa tamir süresi ise *LDPC*'de gözlemlenmiştir. Sembol başına gerçekleştirilen tamir sürelerine bakıldığında ise dosya boyutlarına göre değişkenlik görülmekle beraber ağırlıklı olarak en uzun tamir süresi *Raptor* kodda görülür iken, en kısa tamir ise *LDPC* kodu tarafından sağlanmıştır.

6. SONUÇ VE GELECEK ÇALIŞMALAR

Bu tez çalışması kapsamında Çeşme kod sınıfından *Raptor* kod, optimal silinti düzeltme kodlarından *Reed-Solomon* kod ve optimale yakın silinti düzeltme kodlarından *LDPC* kod kullanılarak hücresel ağ içerisinde tek bir hücredeki kodlanmış veri ön bellekleme süreci simüle edilmiştir. Simülasyon ortamı sadece silinti düzeltme kodlarının matrisleri değiştirilerek farklı silinti düzeltme kod şemaları rahatlıkla bu simülatör üzerinden çalıştırılabilecek şekilde genel olarak hazırlanmıştır. Yine bu çalışma kapsamında daha gerçekçi olarak artık veri kullanılmıştır. Diğer bir deyişle, düğümlerde tutulan sembol sayılarında farklılıklar görülebilmektedir. Bu tezin diğer bir özgün tarafı ise *LDPC* tamir aşamasında hem çoklu düğüm hem de tekli düğüm tamiri için farklı tamir yöntemlerinin kullanılmasıdır. Bu tez çalışmasındaki en önemli özgün değer ise *LT, Raptor* kodların *D2D* iletişimde kodlanmış veri önbellekleme alanında ilk kez test edilmiş olmasıdır.

Bu çalışmadaki temel kontrol noktası *D2D* iletişimde tamir işlemleridir. Bu noktada baz istasyonundan çekilen sembol sayısı, tamir süreleri, kodlama süreleri ve bu tamir süreçlerinde kullanılan yardımcı düğüm sayıları kontrol edilerek gözlemlenmiştir. *LDPC+Raptor* baz istasyonundan ya da düğümlerden sembol indirme konusunda *Reed-Solomon* (*RS*) kodundan daha kötü sonuç vermesine karşın tamir süresinde *RS*'den daha avantajlı çıkmaktadır. Tamir süreci optimize edilerek *LDPC+Raptor* kodlarının daha az sembolle tamir gerçekleştirebilmesi açık bir konudur.

Geleceğe dair çalışmalarda ise kod oranında yapılacak değişikliklerde bu silinti düzeltme kodlarının nasıl bir performans gösterdiği yine bu silinti düzeltme kodları ile test edilebilir. Ayrıca bu tezde kullanılan silinti düzeltme kodlarına başka silinti düzeltme kodları da dahil edilebilir. 3-Kopyalama yöntemi bu tez çalışması kapsamında kullanılan diğer silinti kodlarıyla performans karşılaştırmasına tabi tutulabilir.

SÖZLÜK

Bipartite Graph: İki ayrışık düğüm kümesine ayrılabilen ve her bir kümedeki düğümden diğer kümedeki bir düğüme bir bağlantının bulunduğu, ancak aynı küme içinde hiçbirinin birbirine komşu olmadığı çizge, iki taraflı çizge

Caching: Son olarak kullanılmış olan verinin önbellekte geçici olarak saklanması

Code distance: Derecesi k'den küçük olan herhangi iki farklı polinom en az $n - k + 1$ noktada farklılık gösterir. Buradaki $d = n - k + 1$ bir silinti kodunun uzaklığını ifade eder

Codeword: Orijinal verinin bir silinti düzeltme kodu ile kodlanması sonucu ortaya çıkan genişletilmiş veri

Erasure code: Ağ üzerinden iletişim esnasında veride oluşabilecek bit hatalarını kodlama, kopyalama gibi yöntemler ile yedekleyerek yeniden düzelten silinti kodları

Fountain code: Verilen bir dizi orijinal sembolden sınırsız sayıda kodlanmış sembol oluşturabilen silinti düzeltme kodu

Multicasting: Bir cihazın (düğümün), seçilen birden fazla cihaz ile iletişim kurduğu iletim yöntemi

Repair bandwidth: Bir düğüm tamiri süreci boyunca işlemden geçirilmek zorunda olunan veri miktarı, tamir bant genişliği

Replication: Bir ağdaki hücrede bulunan düğümler arasında orijinal verinin kodlanmadan aynı şekliyle, tekrarlı olarak tutulması mantığına dayanan dağıtık depolama yöntemi

Ripple: LT kod çözme sürecinde kullanılan ve tek komşuya sahip kodlanmış sembollerin komşuları olan orijinal sembollerin tutulduğu küme

Storage Node: Belli bir veriye sahip olan cihaz, depolama düğümü

Inband D2D: Cihazlar arası iletişimin hücresel iletişim ile beraber hücresel spektrumu kullanması

Outband D2D: Cihazlar arası iletişim ile hücresel iletişimin farklı spektrumlar üzerinden gerçekleştirilmesi

7. KAYNAKLAR

Asadi, A., Wang, Q. and Mancuso, V., "A survey on device-to-device communication in cellular networks", *IEEE Communications Surveys and Tutorials*, 16(4), 1801-1819, (2014).

Bose, R. C. and Ray-Chaudhuri, D. K., "Further results on error correcting binary group codes", *Information and Control*, *3*(3), 279–290, (1960).

Cassuto, Y. and Shokrollahi, A., "Online fountain codes with low overhead", *IEEE Transactions on Information Theory*, 61(6), 3137–3149, (2015).

Chen, Z. and Wang, X. J., "Modified Raptor Code for Distributed Storage Systems", *Applied Mechanics and Materials*, 20(23), 52–57, (2010).

Dimakis, A. G., Godfrey, P. B., Member, S., Wu, Y., Wainwright, M. J., Member, S. and Ramchandran, K., "Network Coding for Distributed Storage Systems", *56*(9), 4539–4551, (2010).

Doppler, K., Rinne, M., Wijting, C., Ribeiro, C. B. and Hug, K., "Device-to-device communication as an underlay to LTE-advanced networks", *IEEE Communications Magazine*, 47(12), 42–49, (2009).

Du, J., Zhu, W., Xu, J., Li, Z. and Wang, H., "A compressed HARQ feedback for device-to-device multicast communications", *2012 IEEE Vehicular Technology Conference (VTC Fall)*, 1-5, (2012).

Friedman, R. and Kantor, Y., "Combining Erasure-Code and Replication Redundancy Schemes for Increased Storage and Repair Efficiency in P2P Storage Systems [online]", (3 September 2018), http://www.cs.technion.ac.il/users/wwwb/cgi-bin/tr-get.cgi/2013/CS/CS-2013-03.pdf, (2013).

Gallager, R. G., "Low-density parity-check codes", *IEEE Trans. Info. Theory*, 8, 21–28, (1962).

Ghosh, A., Ratasuk, R., Mondal, B., Mangalvedhe, N. and Thomas, T., "LTE-advanced: Next-generation wireless broadband technology", *IEEE Wireless Communications*, 17(3), 10–22, (2010).

Golrezaei, N., Mansourifard, P., Molisch, A. F. and Dimakis, A. G., "Base-station assisted device-to-device communications for high-throughput wireless video networks", *IEEE Transactions on Wireless Communications*, 13(7), 3665–3676, (2014).

Gummadi, R. and Sreenivas, R. S., "Erasure Codes with Efficient Repair [online]", (3 September 2018), http://ramki-gummadi.github.io/papers/repair.pdf, (2011).

Hamming, R. W., "Error Detecting and Error Correcting Codes", *Bell System Technical Journal*, 29(2), 147-160, (1950).

IEEE Computer Society, "Part 11 : Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications IEEE Computer Society [online]", (3 September 2018), https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6178212, (2012).

IEEE Computer Society and the IEEE Microwave Theory and Techniques Society, "IEEE Standard for Local and metropolitan area networks Part 16 : Air Interface for Fixed and Mobile Broadband Wireless Access Systems Amendment 2 : Physical and Medium Access Control Layers for Combined Fixed and Mobile Operation in Licensed Bands and Corrigendum 1 [online]", (3 September 2018) https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1603394, (2006).

Kaufman, B. and Aazhang, B., "Cellular networks with an overlaid device to device network", *2008 42nd Asilomar Conference on Signals, Systems and Computers*, 1537–1541, (2008).

Kazez, A. C. and Girici, T., "Interference-aware distributed device-to-device caching", *2017 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom)*, 1–5, (2017).

Lei, L., Zhong, Z., Lin, C. and Shen, X., "Operator controlled device-to-device communications in LTE-advanced networks", *IEEE Wireless Communications*, 19(3), 96–104, (2012).

Leiner, B., "LDPC Codes–a brief Tutorial [online]", (3 September 2018), http://www.bernh.net/media/download/papers/ldpc.pdf, (2005).

Li, J. C. F., Lei, M. and Gao, F., "Device-to-device (D2D) communication in MU-MIMO cellular networks", *GLOBECOM - IEEE Global Telecommunications Conference*, 3583–3587, (2012).

Lin, Y., Hsu, Y. and Tung, C., "Multihop Cellular : A New Architecture for Wireless Communications", *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064)*, 3, 1273–1282, (2000).

Luby, M., "Tornado Codes: Practical Erasure Codes Based on Random Irregular Graphs", *Randomization and Approximation Techniques in Computer Science Lecture Notes in Computer Science*, 1518, (1999).

Luby, M., "LT codes", *The 43rd Annual IEEE Symposium on Foundations of Computer Science*, 271–280, (2002).

MacKay, D. J. C., "Fountain codes", *IEE Proceedings - Communications*, 152(6), 1062–1068, (2005).

Maddah-Ali, M. A. and Niesen, U., "Fundamental limits of caching", *IEEE Transactions on Information Theory*, 60(5), 2856–2867, (2014).

Miller, L. S. and Childers, D., *Probability and Random Processes*, 2, Amsterdam, Boston, Heidelberg, London, New York, Oxford, Paris, San Diego, San Francisco, Singapore, Sydney, Tokyo, Elsevier, 383-428, (2012).

Morello, A., and Mignone, V., "DVB-S2: The second generation standard for satellite broad-band services", *Proceedings of the IEEE*, 94(1), 210–226, (2006).

Niyato, D., Xiao, L., and Wang, P., "Machine-to-machine communications for home energy management system in smart grid", *Communications Magazine, IEEE*, 49(4), 53–59, (2011).

O'Dea, A., "Telemetry Data Decoding [online]", (3 September 2018), https://deepspace.jpl.nasa.gov/dsndocs/810-005/208/208B.pdf, (2013).

Paakkonen, J., Barreal, A., Hollanti, C. and Tirkkonen, O., "Coded Caching Clusters with Device-to-Device Communications", *IEEE Transactions on Mobile Computing*, 1–26, (2016).

Paakkonen, J., Hollanti, C. and Tirkkonen, O., "Device-to-device data storage with regenerating codes", *Multiple Access Communications Lecture Notes in Computer Science*, 9305, 57–69, (2014).

Paakkönen, J., Hollanti, C. and Tirkkonen, O., "Device-to-Device Data Storage for Mobile Cellular Systems", *2013 IEEE Globecom Workshops (GC Wkshps)*, 671–676, (2013).

Papailiopoulos, D. S. and Dimakis, A. G., "Locally repairable codes", *IEEE Transactions on Information Theory*, 60(10), 5843–5855, (2014).

Park, H., Lee, D. and Moon, J., "LDPC Code Design for Distributed Storage: Balancing Repair Bandwidth, Reliability, and Storage Overhead", *IEEE Transactions on Communications*, 66(2), 507–520, (2018).

Pedersen, J., Amat, A. G. I., Andriyanova, I. and Brannstrom, F., "Distributed Storage in Mobile Wireless Networks with Device-to-Device Communication", *IEEE Transactions on Communications*, 64(11), 4862-4878, (2016).

Peng, T., Lu, Q., Wang, H., Xu, S. and Wang, W., "Interference avoidance mechanisms in the hybrid cellular and device-to-device systems", *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC*, 617–621, (2009).

Plank, J. S., "A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems", *Software: Practice and Experience*, 27(9), 995–1012, (1997).

Pratas, N. K. and Popovski, P., "Low-Rate Machine-Type Communication via Wireless Device-to-Device (D2D) Links [online]", (3 September 2018), https://arxiv.org/pdf/1305.6783.pdf, (2013).

Reed, I. S. and Solomon, G., "Polynomial Codes Over Certain Finite Fields", *Journal of the Society for Industrial and Applied Mathematics*, 8(2), 300–304, (1960).

Shokrollahi, A., "Raptor codes", *IEEE Transactions on Information Theory*, 52(6), 2551–2567, (2006).

Venkiah, A., Poulliat, C. and Declercq, D., "Analysis and design of raptor codes for joint decoding using information content evolution", *IEEE International Symposium on Information Theory - Proceedings*, 421–425, (2007).

Weatherspoon, H. and Kubiatowicz, J. D., "Erasure Coding Vs. Replication: A Quantitative Comparison", 328–337, (2002).

Wei, Y., Chen, F. and Lim, K. C., "Large LDPC Codes for Big Data Storage", *Proceedings of the ASE BigData & SocialInformatics 2015*, 1, 1-6, (2015).

Wei, Y., Foo, Y. W., Lim, K. C. and Chen, F., "Auto-configurable LDPC codes for distributed storage", *2014 IEEE 17th International Conference on Computational Science and Engineering*, 1332–1338, (2015).

Wicker, S. B. and Bhargava, V. K., "*Reed-Solomon Codes and Their Applications"*, 1, New York, IEEE Press, 41-58, (1994).

Zhou, B., Hu, H., Huang, S. Q. and Chen, H. H., "Intracluster device-to-device relay algorithm with optimal resource utilization", *IEEE Transactions on Vehicular Technology*, 62(5), 2315–2326, (2013).

8. ÖZGEÇMİŞ

Adı Soyadı	: Erdi KAYA
Doğum Yeri ve Tarihi	: Keçiören / Ankara, 20.01.1988
Lisans Üniversite	: Çankaya Üniversitesi – Bilgisayar Mühendisliği
Elektronik posta	: eerdikayaa@gmail.com