5.HATALAR, ALGILAMA VE DÜZELTME

- Hata (Error) Sınıflandırması
- Hata türleri
- Artıklık (Redundancy) kavramı
- Algılama ve düzeltme (Parity)
- Döngüsel Artıklık Kontrolü (CRC)
- Hamming kodu kavramı
- Sağlama tekniği (Checksum)

5.1 GİRİŞ:

Verilerdeki hatalar temel olarak iletim sürecinde meydana gelen çeşitli aksaklıklardan kaynaklanmaktadır. İletimde kusurlu bir saha veya ortam mevcut olduğunda, orijinal verilerde hatalara eğilimlidir.

Hatalar şu şekilde sınıflandırılabilir:

- Zayıflama (Attenuation)
- Gürültü (Noise)
- Bozulma (Distortion)

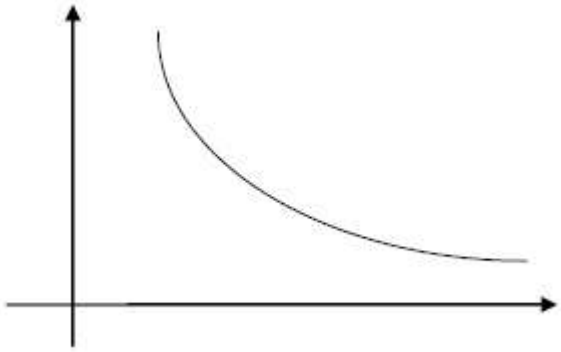
5.2 HATALARIN SINIFLANDIRILMASI

Hatalar aşağıda verilen şekilde sınıflandırılabilir.

1. Zayıflama (Attenuation)

Sinyal iletim ortamdan geçerken mesafe arttıkça gücü azalır, şekil 5.1'de gösterildiği gibi ses işareti buna bir örnektir, mesafe arttıkça zayıflar ve belli bir mesafenin ötesinde içeriğini kaybeder. Mesafe arttıkça zayıflama da artar.

Kuvvet



Mesafe

Şekil 5.1

2. Gürültü (Noise)

Gürültü, istenmeyen veri olarak tanımlanmaktadır. İletim sırasında, ana işarete bozucu bazı elektromanyetik sinyaller girdiğinde, buna genellikle Gürültü denir. Gürültü nedeniyle orijinal verileri veya bilgileri almak zordur.

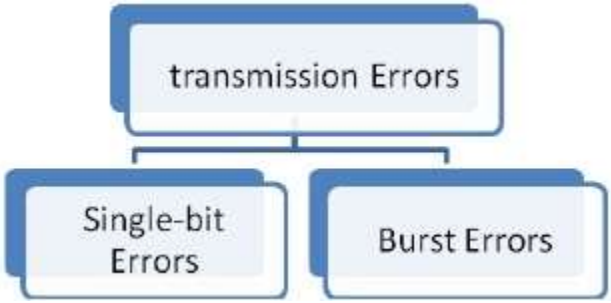
3. Bozulma (Distortion)

Ortamda farklı hızlarla hareket eden farklı frekansların girişimi olduğunda Bozulma meydana gelir. Bu nedenle farklı frekanslar arasında bir boşluk (koruma alanı) olması önemlidir.

5.3 HATA TÜRLERİ:

Sinyal ikili verilerden oluşuyorsa iletim sırasında iki tür hata meydana gelebilir:

1. Tek bit hataları
2. Burst Hataları



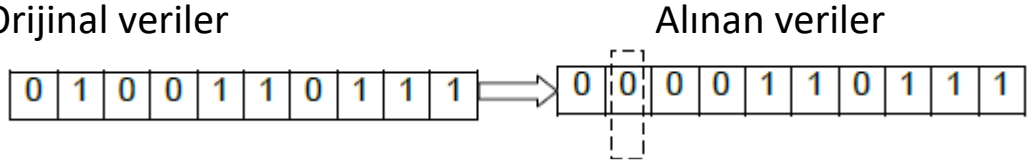
Şekil 5.2

1. Tek bitlik hatalar:

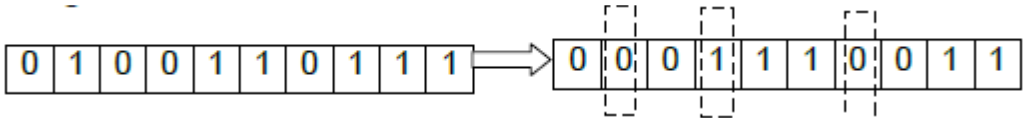
Tek bit hatasında, 0'lık bir bit değeri, 1'lik bir bit değerine dönüşür veya bunun tersi de olabilir. Paralel iletimde tek bit hatalarının meydana gelme olasılığı daha yüksektir. Şekil 5.3 (a)

2. Burst hataları:

Burst (Çoklu bit, darbe katarı/patlaması) hatasında ikili değerin birden fazla biti değişir. Burst hatası, iletimdeki herhangi iki veya daha fazla biti değiştirebilir. Bu bitlerin bitişik bitler olması gerekmez. Seri iletimde Burst hatalarının meydana gelme olasılığı daha yüksektir. Şekil 5.3 (b)



Şekil 5.3 (a) Tek bit hatası

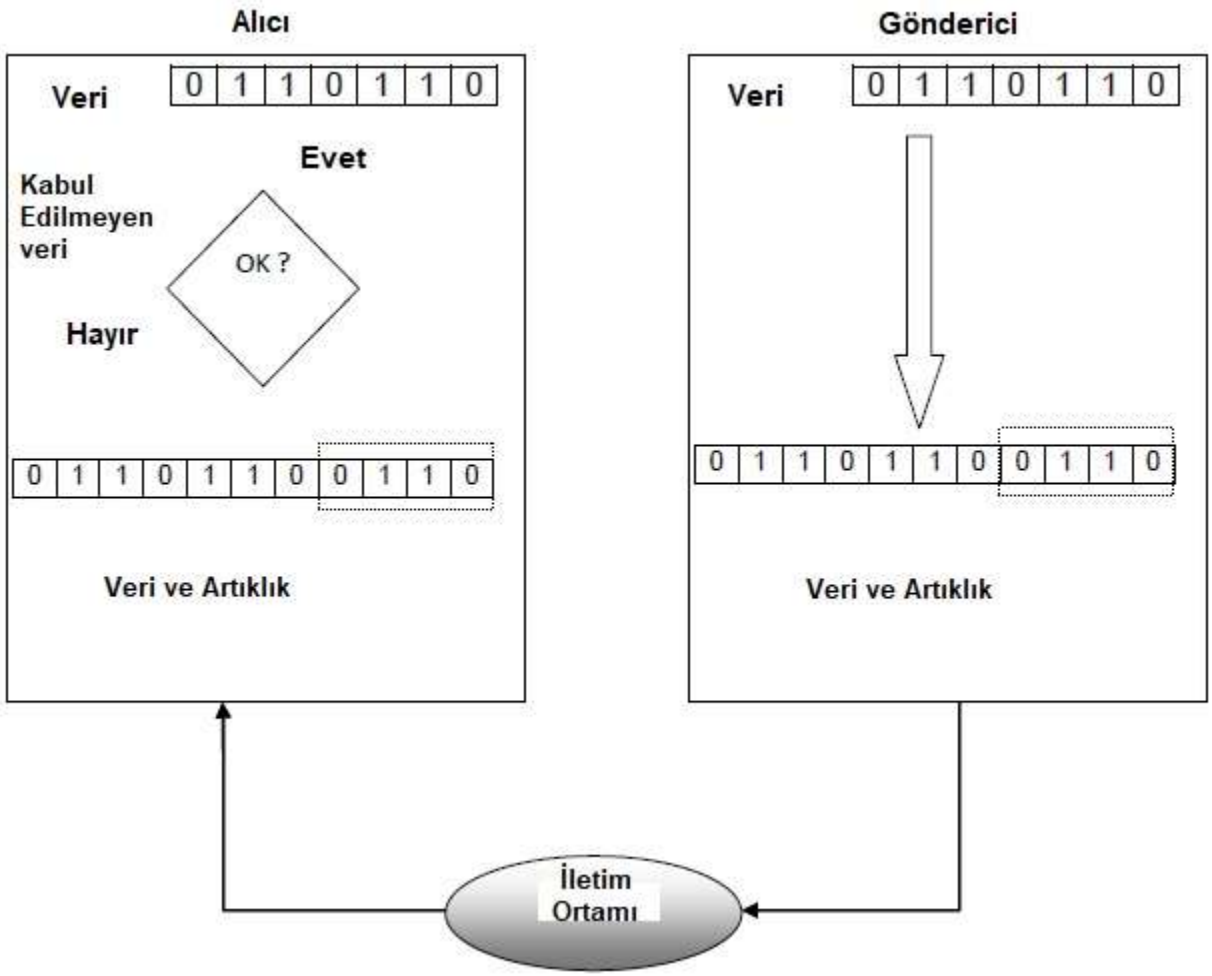


Şekil 5.3 (b) Burst hatası

5.4 ARTIKLIK (REDUNDANCY)

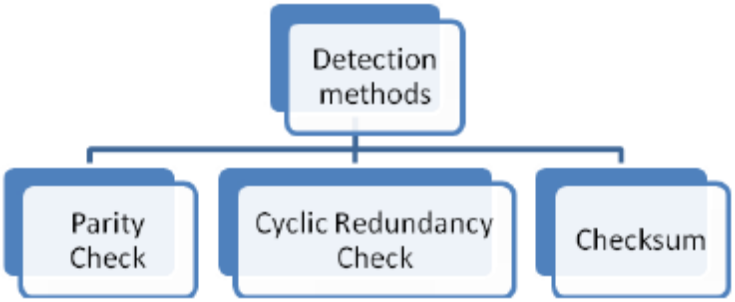
Veri iletişimindeki hataları tespit etmek ve düzeltmek için orijinal verilere bazı ekstra bitler ekleriz. Bu ekstra bitler, veri alındıktan sonra alıcı tarafından kaldırılacak olan fazlalık (artık/redundant) bitlerdir.

Bunların varlığı, alıcının bozuk bitleri tespit etmesine veya düzeltmesine olanak tanır. Tüm veri akışını tekrarlamak yerine, tüm veri akışına kısa bir bit grubu eklenebilir. Bu tekniğe artıklık (redundancy) denir çünkü fazladan bitler bilgi için yedeklidir: iletimin doğruluğu sağlanır sağlanmaz atılırlar.


Alıcı
Gönderici
Veri
0 1 1 0 1 1 0
Evet
Kabul Edilmeyen veri
OK ?
Hayır
0 1 1 0 1 1 0 0 1 1 0
Veri ve Artıklık
Veri
0 1 1 0 1 1 0
0 1 1 0 1 1 0 0 1 1 0
Veri ve Artıklık
İletim Ortamı

Şekil 5.4

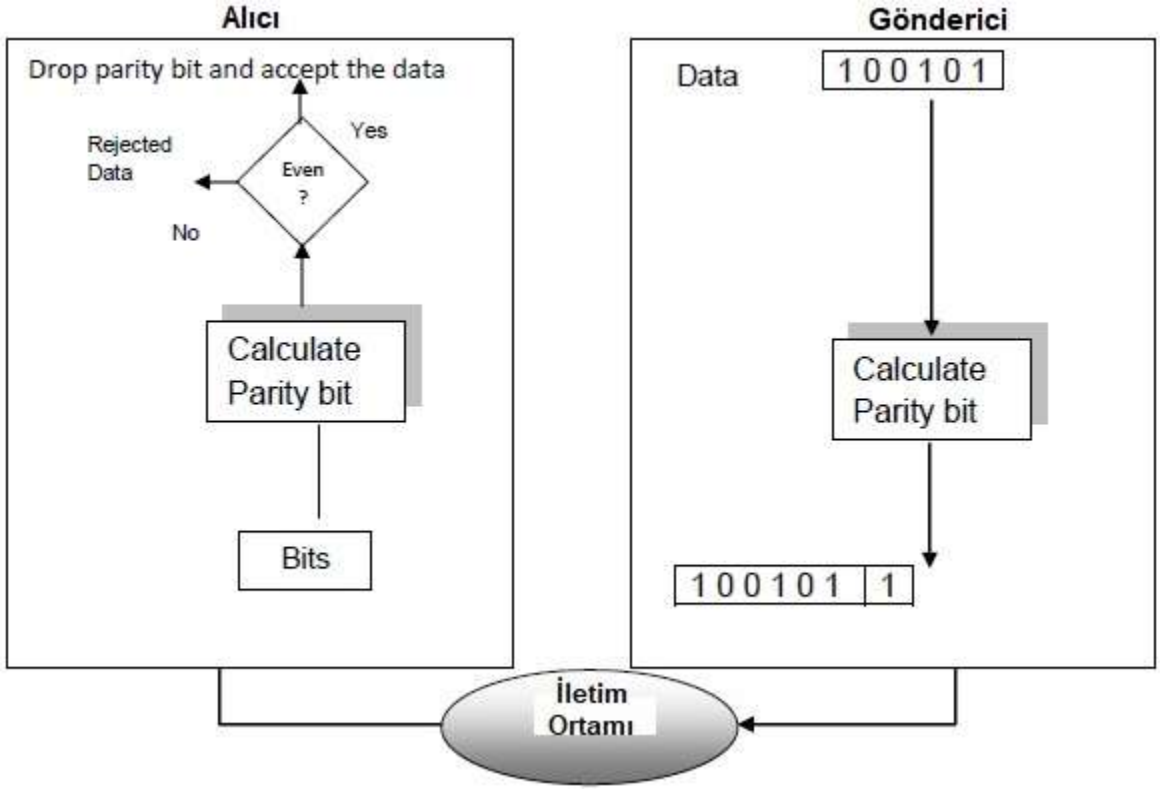
İletim hatası tespiti ve düzeltilmesi için kullanılan farklı teknikler vardır.



Şekil 5.5

1. Tekli Parite Kontrolü:

Bu teknikte, her veri birimine eşlik biti adı verilen yedek bir bit eklenir, böylece birimdeki 1'lerin toplam sayısı (eşlik biti dahil) çift/even (veya tek/odd) olur. Aşağıdaki Şekil ikili verileri iletirken bu kavramın nasıl kullanıldığını göstermektedir.



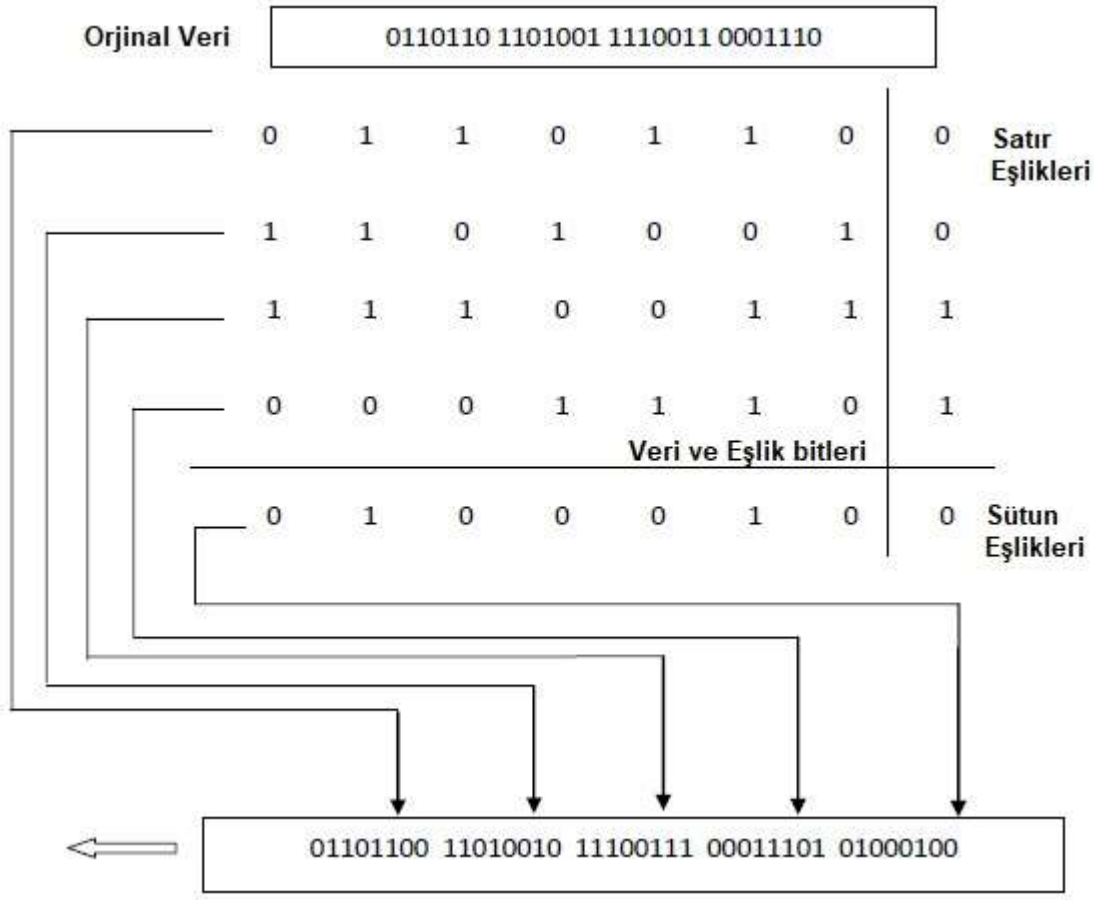
Basit eşlik kontrolü tüm tek bitlik hataları tespit edebilir. Ayrıca, değiştirilen bitlerin toplam sayısı tek olduğu sürece Burst hatalarını da

tespit edebilir. Bu yöntem, değiştirilen toplam bit sayısının iki olduğu durumlarda hataları tespit edemez.

2. İki Boyutlu Parite Kontrolü:

Daha iyi bir yaklaşım iki boyutlu eşlik kontrolleridir. Bu yöntemde, bir bit bloğu bir tabloda (satırlar ve sütunlar) düzenlenir. Öncelikle her veri birimi için eşlik bitini hesaplıyoruz. Daha sonra bunları bir tablo halinde düzenliyoruz. Daha sonra her sütun için eşlik bitini hesaplıyoruz ve 8 bitlik yeni bir satır oluşturuyoruz.

Aşağıdaki örneği düşünün; Göndereceğimiz dört veri birimimiz var. Aşağıda gösterildiği gibi tablo halinde düzenlenmiştir.



Daha sonra her sütun için eşlik bitini hesaplıyoruz ve 8 bitlik yeni bir satır oluşturuyoruz; bunlar tüm bloğun eşlik bitleridir. Beşinci sıradaki ilk eşlik

bitinin tüm birinci bitlere göre hesaplandığına dikkat edin: ikinci eşlik biti tüm ikinci bitlere göre hesaplanır ve bu böyle devam eder. Daha sonra 8 eşlik bitini orijinal verilere bağlarız ve bunları alıcıya göndeririz.

İki boyutlu eşlik kontrolü, Burst hatalarını tespit etme olasılığını artırır. Bu yöntemle n'bitten daha fazla bir Burst hatası da çok yüksek bir olasılıkla tespit edilir.

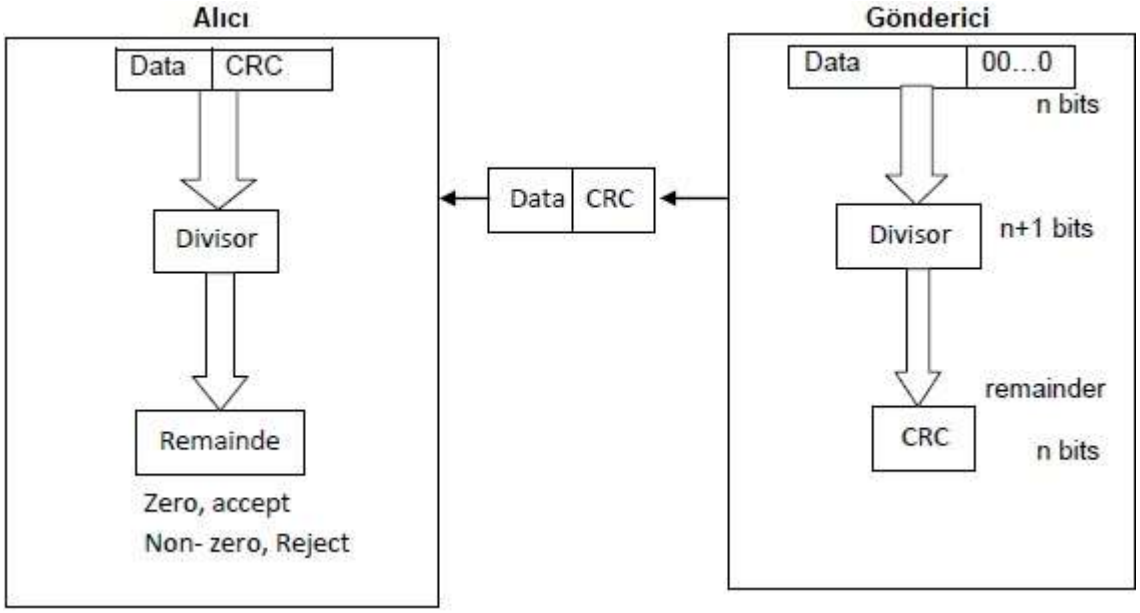
5.5. Döngüsel Artıklık Kontrolü (CRC)

Artıklık kontrol tekniklerinden en güçlüsü döngüsel artıklık (CRC, **C**yclic **R**edundancy **C**heck) kontrolüdür. Bu yöntem ikili bölmeye dayanmaktadır. CRC'de, fazlalık bitlerin istenen dizisi üretilir ve veri biriminin sonuna eklenir. Aynı zamanda CRC hatırlatıcısı olarak da adlandırılır. Böylece ortaya çıkan veri birimi önceden belirlenmiş bir ikili sayıya tam olarak bölünebilir hale gelir.

Varış noktasında gelen veri birimi aynı sayıya bölünür. Bu adımda kalan yoksa veri biriminin doğru olduğu varsayılır ve bu nedenle kabul edilir. Geriye kalan, veri biriminin nakliye sırasında hasar gördüğünü ve bu nedenle reddedilmesi gerektiğini gösterir.

CRC tarafından kullanılan artıklık bitleri, veri biriminin önceden belirlenmiş bir bölene bölünmesiyle elde edilir; geri kalan CRC'dir. Geçerli olması için, bir CRC'nin iki niteliği olmalıdır: Bölene göre tam olarak bir bit eksik olmalı ve bunun veri dizisinin sonuna eklemesi, elde edilen bit dizisini bölen tarafından tam olarak bölünebilir hale getirilmesi gerekir.

Aşağıdaki şekil bu süreci göstermektedir:



Adım 1: Veri birimine 0'lar dizisi eklenir. n bit uzunluğundadır. n sayısı, önceden belirlenmiş bölendeki (desen) n + 1 bit olan bit sayısından 1 eksiktir.

Adım 2: Yeni oluşturulan veri birimi, ikili bölme adı verilen bir işlem kullanılarak bölene bölünür. Bu bölünmeden elde edilen kalan kısım CRC'dir.

Adım 3: Adım 2'de türetilen n bitlik CRC, veri ünitesine eklenen 0'ların yerini alır. CRC'nin tümünün 0'lardan oluşabileceğini unutmayın.

Veri birimi önce alıcı verisine, ardından CRC'ye ulaşır. Alıcı tüm diziyi bir birim olarak ele alır ve onu CRC geri kalanının kullanıldığı aynı bölene böler. Dizi hatasız ulaşırsa, CRC denetleyicisi sıfırın kalanını verir, veri birimi geçer. Dizi geçiş sırasında değiştirilmişse, bölme sıfır kalan verir ve veri birimi geçmez.

Ör1.1: Aşağıdaki şekil CRC kalan oluşturma sürecini göstermektedir:
Mesaj/dizi: 11100 (5-bit)
Desen/bölen: 1001 (4-bit)

CRC Üreteci:

- Bir CRC üreteci mod-2 bölümünü kullanır. Öncelikle bölenin (1001) uzunluğu 4-bit olduğundan verinin (11100) sonuna üç sıfır (n+1=4 ise n=3) eklenir. Eklenecek "0" dizisinin uzunluğunun her zaman bölenin (desen) uzunluğundan bir eksik olduğunu biliyoruz.
- Şimdi dizi "11100000" olur ve elde edilen dizi 1001 bölenine bölünür.
- İkili bölmeden elde edilen kalana CRC kalanı denir. CRC kalanının üretilen değeri burada "111" olur.
- CRC kalanı, veri biriminin sonuna eklenen 0 dizisinin yerini alır ve son mesaj/dizi, ağ üzerinden gönderilen "11100111" olacaktır.

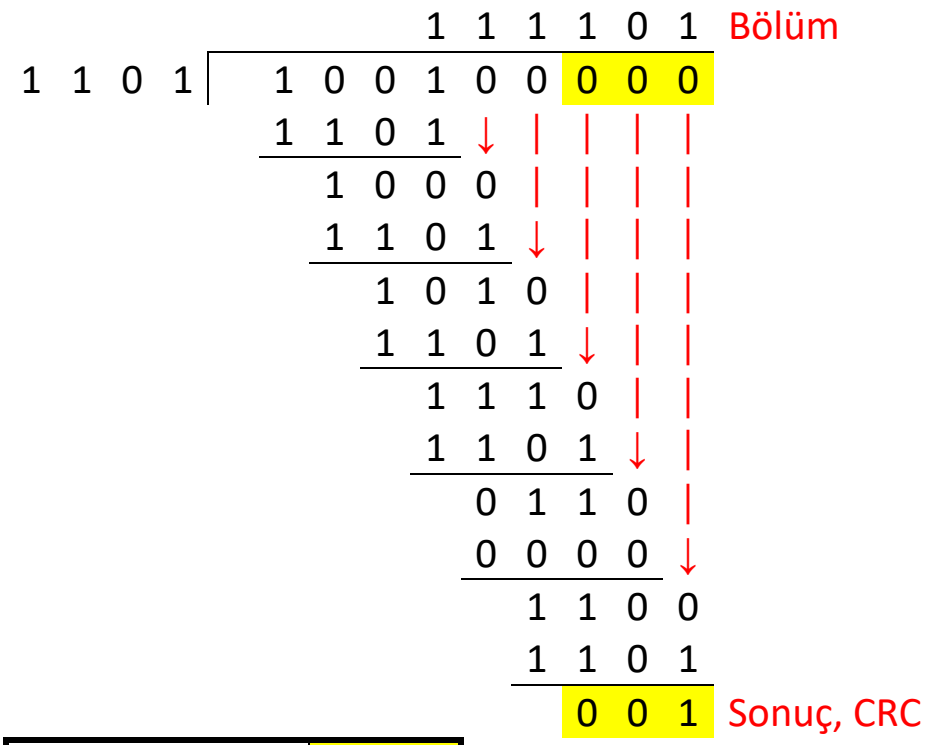
Mesaj/dizi: 11100 (5-bit)
Desen/bölen: 1001 (4-bit)

```
                    1 1 1 1 1   Bölüm
1 0 0 1 | 1 1 1 0 0 0 0 0
          1 0 0 1 ↓ | | |
            1 1 1 0 | | |
            1 0 0 1 ↓ | |
              1 1 1 0 | |
              1 0 0 1 ↓ |
                1 1 1 0 |
                1 0 0 1 ↓
                  1 1 1 0
                  1 0 0 1
                    1 1 1   Sonuç, CRC
```

1	1	1	0	0	1	1	1

Gönderilen dizi (8-bit)

Ör2.1: Aşağıdaki şekil CRC kalan oluşturma sürecini göstermektedir:
Mesaj: 100100 (6-bit)
Desen (bölen) : 1101 (4-bit)



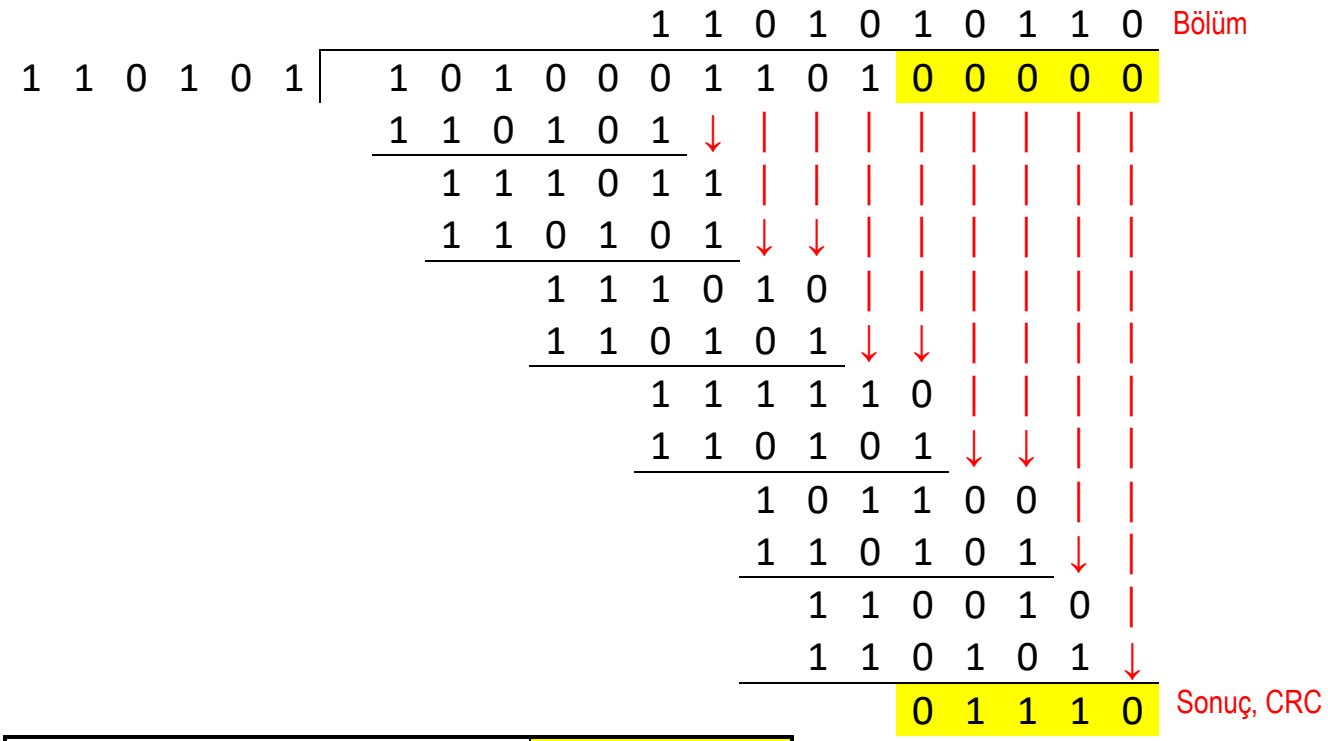
1	0	0	1	0	0	0	0	1

Gönderilen dizi (9-bit)

Ör3.1: Aşağıdaki şekil CRC kalan oluşturma sürecini göstermektedir:

Mesaj: 1010001101 (10-bit)

Desen: 110101 (6-bit)



1	0	1	0	0	0	1	1	0	1	0	1	1	1	0

Gönderilen dizi (15-bit)

CRC Denetleyici:

Bir CRC denetleyici işlevi tam olarak üreticinin yaptığı gibi çalışır. CRC'ye eklenen verileri aldıktan sonra aynı mod-2 bölümünü yapar. Geri kalanın tümü 0'lar ise, CRC atılır ve veri kabul edilir; aksi takdirde, alınan bit akışı kabul edilmez ve veri yeniden gönderilir.

Ör1.2: Aşağıdaki şekil alıcıdaki aynı bölme sürecini göstermektedir:

Mesaj: 11100 (5-bit)

Desen: 1001 (4-bit)

```
                      1  1  1  1  1   Bölüm
1  0  0  1 | 1  1  1  0  0  1  1  1
           | 1  0  0  1  ↓  |  |  |
           |----------------------
           |    1  1  1  0  |  |  |
           |    1  0  0  1  ↓  |  |
           |-------------------|  |
           |       1  1  1  1  |  |
           |       1  0  0  1  ↓  |
           |----------------------|
           |          1  1  0  1  |
           |          1  0  0  1  ↓
           |-------------------------
           |             1  0  0  1
           |             1  0  0  1
           |-------------------------
                         0  0  0   Sonuç, CRC
```

1	1	1	0	0	0	0	0	0

Alınan dizi (9-bit), CRC bitleri atıldığında 6-bit mesaj doğru olarak elde edilir.

- CRC denetleyicisinin işlevselliği CRC oluşturucuya benzer.
- Alıcı uçta 111000111 dizisi alındığında, CRC denetleyicisi mod-2 bölümünü gerçekleştirir.
- Bu dizi aynı desene/bölene (1001) bölünür.
- Sonuç olarak, CRC denetleyicisi CRC kalanı “0” lardan oluştuğu için veri kabul edilir.

Performans:

CRC çok etkili bir hata algılama yöntemidir.

Eğer bölenler daha önce bahsedilen kurallara göre seçilirse,

1. CRC, tek sayıda biti etkileyen tüm Burst hatalarını tespit edebilir.

2. CRC, polinomun derecesine eşit veya daha küçük uzunluktaki tüm Burst hatalarını tespit edebilir

3. CRC, polinomun derecesinden daha büyük uzunluktaki Burst hatalarını ise çok yüksek bir olasılıkla tespit edebilir.

5.6. Hamming Kodu

Hamming kodu herhangi bir uzunluktaki veri birimlerine uygulanabilir ve yukarıda açıklanan veriler ile artıklık bitleri arasındaki ilişkiyi kullanır. Örneğin, 7 bitlik bir ASCII kodu, veri biriminin sonuna eklenebilen veya orijinal veri bitlerinin arasına serpiştirilebilen 4 artıklık biti gerektirir. Aşağıdaki Şekilde, bu bitler 1, 2,4 ve 8 konumlarına (2'nin kuvvetleri olan 11 bitlik bir dizideki konumlar) yerleştirilir. Aşağıdaki örneklerde netlik sağlamak amacıyla bu bitleri r1, r2, r4 ve r8 olarak adlandırıyoruz.

11	10	9	8	7	6	5	4	3	2	1
d	*d*	*d*	*r8*	*d*	*d*	*d*	*r4*	*d*	*r2*	*r1*

Hamming kodunda her r biti, veri bitlerinin bir birleşiminin eşlik (parity) bitidir ve aşağıda gösterilmiştir:

r1: bitler 1,3,5,7,9,11

r2 : bitler 2,3,6,7,10,11

r3 : 4,5,6,7 bitleri

r4 : 8,9,10,11 bitleri

r1 şu bitlerin konumuna atanacaktır:

11		9		7		5		3		1
d	*d*	*d*	*r8*	*d*	*d*	*d*	*r4*	*d*	*r2*	*r1*

r2 şu bitlerin konumuna atanacaktır:

11	10			7	6			3	2	
d	*d*	*d*	*r8*	*d*	*d*	*d*	*r4*	*d*	*r2*	*r1*

r4bu bitlerin konumuna atanacaktır:

				7	6	5	4			
d	*d*	*d*	*r8*	*d*	*d*	*d*	*r4*	*d*	*r2*	*r1*

r8 bu bitlerin konumuna atanacaktır:

11	10	9	8							
d	*d*	*d*	*r8*	*d*	*d*	*d*	*r4*	*d*	*r2*	*r1*

Şimdi gönderilecek 1001101 verimiz olduğunu varsayalım, sonra artık (redundant) bitler aşağıdaki yöntemle hesaplanır:

Veriler: 1 0 0 1 1 0 1

11	10	9	8	7	6	5	4	3	2	1
1	0	0	r8	1	1	0	r4	1	r2	r1

r1 ekleniyor: 1,3,5,7,9,11 “1” lerin sayısı çift eşlikten

11	10	9	8	7	6	5	4	3	2	1
1	0	0		1	1	0		1		**1**

r2 ekleniyor:

11	10	9	8	7	6	5	4	3	2	1
1	0	0		1	1	0		1	**0**	1

r4 ekleniyor:

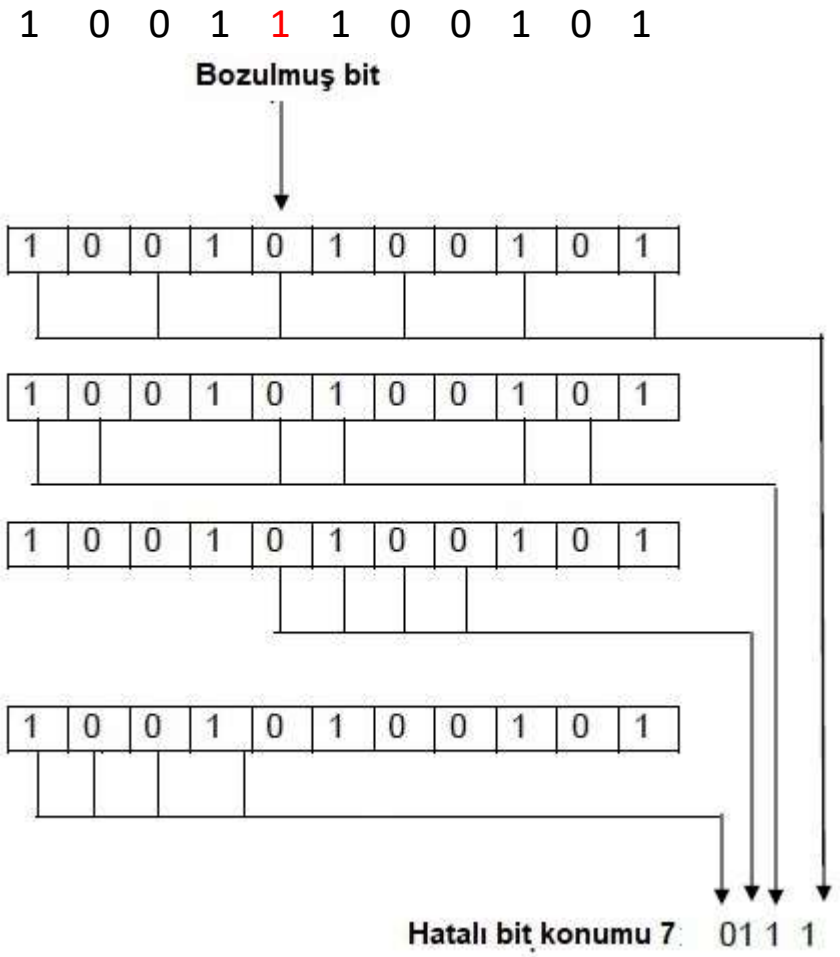
11	10	9	8	7	6	5	4	3	2	1
1	0	0		1	1	0	**0**	1	0	1

r8 ekleniyor:

11	10	9	8	7	6	5	4	3	2	1
1	0	0	**1**	1	1	0	0	1	0	1

Kod: 10011100101

Şimdi, yukarıdaki iletim alındığında, 7 numaralı bitin 1'den 0'a değiştiğini hayal edin. Alıcı, iletimi alır ve gönderici tarafından kullanılan aynı bit setlerini artı her set için ilgili eşlik r bitini kullanarak 4 yeni eşlik bitini yeniden hesaplar (aşağıdaki Şekil'e bakınız). Daha sonra yeni eşlik değerlerini r konumuna (r8, r4, r2, r1) göre ikili bir sayı halinde birleştirir. Örneğimizde, bu adım bize hatalı bitin kesin konumu olan 0111 (ondalık sistemde 7) ikili sayısını verir.



Bit tanımlandıktan sonra alıcı, bu bit değerini tersine çevirebilir ve hatayı düzeltebilir. Tekniğin güzelliği, donanıma kolaylıkla uygulanabilmesi ve alıcının haberi olmadan kodun düzeltilebilmesidir.

5.7. Sağlama Toplamı

Sağlama toplamı (Checksum), göndericiden alıcıya gönderilecek veriler üzerinde belirli işlemlerin yapılması sonucu oluşan sabit uzunluktaki verilerdir. Gönderen, verinin sağlama toplamını hesaplamak için uygun sağlama toplamı algoritmasını çalıştırır ve bunu çeşitli başlıkların yanı sıra gönderilecek verileri içeren pakete bir alan olarak ekler.

Alıcı veriyi aldığında, yeni bir sağlama toplamı hesaplamak için aynı sağlama toplamı algoritmasını çalıştırır. Alıcı, bu yeni hesaplanan sağlama toplamını gönderen tarafından hesaplanan sağlama toplamıyla

karşılaştırır. İki sağlama toplamı eşleşirse, verinin alıcısı, verinin aktarım sırasında değişmediğini anlamış olur.

1975 yılında geliştirilen Intel biçimli onaltılık dosya, veri ve dosya sonu olmak üzere iki ana kısımdan oluşur. Bütün satırlar ":" karakteri ile başlar. Veri kısmında ":" karakterinden sonra Uzunluk (veri değerlerinin sayısı), Adres, Kayıt tipi (0=veri, 1=kayıt sonu, 2=uzatılmış adres, 3=programın başlangıç adresi), Veri değerleri ve satır sonunda kontrol toplamı vardır. Bu tip 8-bit onaltılık çıkış dosyasındaki her satırın sonunda yer alan kontrol toplamı (checksum) aşağıda verilen şekilde hesaplanır.

$$\left(\begin{array}{c}\text{Uzunluk}\\ \text{değeri}\end{array} + \begin{array}{c}\text{Adres bayt}\\ \text{değerleri}\end{array} + \begin{array}{c}\text{Tip}\\ \text{değeri}\end{array} + \begin{array}{c}\text{Veri}\\ \text{Bayt}\\ \text{değerleri}\end{array}\right) = \begin{array}{c}\text{Sonucun}\\ \text{2'ye tümleyeni}\end{array}$$

Bir I8HEX Intel 8-bit onaltılık dosya aşağıda verilmiş ve her bir satırının açıklaması yapılmıştır.

:0ED00000CE00004FA7004C088C010026F7015F
:02FFFE00D00031
:00000001FF

:0E D000 00 CE 00 00 4F A7 00 4C 08 8C 01 00 26 F7 01 5F

		Adres	Tip	Değerler	Kontrol Toplamı
:	0E	D0 00	00	CE 00 00 4F A7 00 4C 08 8C 01 00 26 F7 01	**5F** = 00 – ~~**4A1**~~
	Uzunluk →			1 2 3 4 5 6 7 8 9 10 11 12 13 14	

:02 FFFE 00 D0 00 31

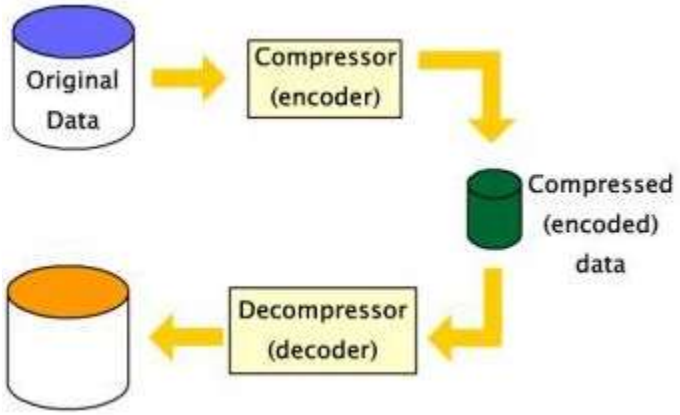
		Adres	Tip	Veriler	Kontrol Toplamı
:	02	FF FE	00	D0 00	**31** = – (02+FF+FE+00+D0+00) = 00 – ~~**2**~~**CF**
	Uzunluk →			1 2	

:00 0000 01 FF

		Adres	Tip	Kontrol Toplamı
:	00	00 00	01	**FF** = – (00+01+00+00) = 00 – **01**
	Uzunluk →			

5.8. Veri Sıkıştırma (DC)

DC, "Data Compression" anlamına gelir. DC, iletilecek verilerin bit cinsinden depolama miktarını azaltmak için sıkıştırıldığı bir dijital sinyal işlemidir. Yani DC uygulandıktan sonra veri depolama alanının normale göre azaldığını söyleyebilirsiniz. Veri iletimi, veri depolama alanını ve iletim kapasitesini büyük ölçüde azaltır. Kaynak kodlama veya bit hızı azaltma olarak da bilinir. Veritabanı yönetim sistemi, yedekleme yardımcı programları vb. veri sıkıştırma yöntemini yaygın olarak kullanır. Birçok dosya sıkıştırma yöntemi vardır ancak ZIP ve ARC çoğunlukla bilinen dosya formatlarıdır.



Kaynak: Notes_of_Data_Communication.pdf

Veri Sıkıştırma İki Ana gruba ayrılır:

1. **Kayıplı**: Kayıplı sıkıştırma yönteminde verinin bir kısmı silinir veya kaybolur. Çünkü gereksiz bilgileri iletimden önce tespit edip siler.
2. **Kayıpsız**: Kayıpsız sıkıştırma yönteminde sıkıştırma, istatistiksel artıklıkların belirlenmesi ve ortadan kaldırılması yoluyla yapılır. Örneğin, bir veri kaynağını iletmeden önce kodladığımızda, boyutu etkili bir şekilde küçültülür ve veriler bozulmadan ve değişmeden kalır.

Veri Sıkıştırmanın Faydaları

- Daha hızlı dosya aktarımı: Sıkıştırılmış dosyaları indirmek için daha az bant genişliği gerektiğinden dosya aktarımının hızını artırır
- Daha Fazla Depolama Kapasitesi: Mevcut depolama alanında daha fazla dosya saklamanıza olanak tanır; Kayıpsız sıkıştırma, bir dosyayı orijinal boyutunun %50'sine kadar küçültebilir.
- Maliyeti Düşürür: Sıkıştırma sonrasında veri depolama maliyetini azaltmanıza olanak tanır, verilen depolama alanında daha fazla dosya depolayabilirsiniz.
- Gecikmeyi azaltır: Kayıt üzerindeki küçük dosya görüntüleri, gecikmeyi azaltan belirli bir dosyaya ulaşmak için daha hızlı taranabilir.

Kayıplı Sıkıştırma ve Kayıpsız Sıkıştırma arasındaki farklar:

1. Kayıplı sıkıştırma, fark edilmeyen verileri ortadan kaldıran yöntemdir. Kayıpsız Sıkıştırma fark edilmeyen verileri ortadan kaldırmaz.

2. Kayıplı sıkıştırmada, dosya orijinal biçiminde geri yüklenmez veya yeniden oluşturulmaz. Kayıpsız Sıkıştırmadayken bir dosya orijinal biçiminde geri yüklenebilir.

3. Kayıplı sıkıştırmada Verinin kalitesi tehlikeye girer. Ancak Kayıpsız Sıkıştırma veri kalitesinden ödün vermez.

4. Kayıplı sıkıştırma veri boyutunu azaltır. Ancak Kayıpsız Sıkıştırma veri boyutunu küçültmez.

5. Kayıplı sıkıştırmada kullanılan algoritmalar şunlardır: Dönüşüm kodlaması, Ayrık Kosinüs Dönüşümü, Ayrık Dalgacık Dönüşümü, fraktal sıkıştırma vb. Kayıpsız sıkıştırmada kullanılan algoritmalar, Çalışma Uzunluğu Kodlaması, Lempel-Ziv-Welch, Huffman Kodlaması, Aritmetik kodlama vb.'dir.

6. Görüntülerde, seste, videoda kayıplı sıkıştırma kullanılır. Metinde, görsellerde ve seste ise Kayıpsız Sıkıştırma kullanılır.

7. Kayıplı sıkıştırmanın veri tutma kapasitesi daha fazladır. Kayıpsız Sıkıştırma, Kayıplı sıkıştırma tekniğine göre daha az veri tutma kapasitesine sahiptir.

SORULAR

1. Hata sınıflandırmasını açıklayınız?
2. Hata türlerini bir örnekle açıklayınız?
3. Eşlik kontrolü nedir?
4. CRC nedir? bir örnekle açıklayınız?
5. Hamming kodunu açıklayınız?
6. Sağlama toplamı bir örnekle açıklayınız??

KAYNAKLAR

1. Data and Computer Communications, Eighth Edition, William Stallings 2007.
2. Computer networks and internets, Sixth edition, Douglas E. Comer, Pearson Education Limited, 2015.
3. Digital Communication Systems, Simon Haykin, John Wiley & Sons, 2014.
4. Fundamentals of Telecommunications. Roger L. Freeman, John Wiley & Sons, Inc., 1999.
5. Data Communications and Networking, Behrouz A Forouzan, McGraw-Hill, 2006.
6. Mikroişlemci Sistemleri, Tuncay UZUN, Nobel Yayın., 2022.