9 Kanal Kodlama

Bir tanıdığınızla kalabalık ve gürültülü bir yerde karşılaştığınızı ve konuşmaya çalıştığınızı varsayalım. Ancak, ortam gürültüsünden dolayı birbirinizin cümlelerini tam olarak anlayamadığınızı göz önüne getirelim. Problemi gidermek için, beraberce daha sakin bir ortama geçmek dışında, karşılıklı iletişimi daha sağlıklı hale getirmek için birkaç seçenek vardır; Daha yüksek sesle konuşmak, cümleyi anlamadığınızda bunu belirtip tekrar edilmesini sağlamak, veya daha kolay anlaşılır şekilde yavaş ama içinde bol tekrar barındıran cümleler kurmak. Tabi ki bu üç önlemi beraber alabilirsiniz. Benzeri durum elektronik haberleşmede de karşımıza çıkabilir. Bahsedilen önlemlerin elektronik haberleşmedeki karşılıkları;

- İşaret gücünü arttırmak,
- Alıcının, işareti tam algılamadığı durumda bunu vericiye ileterek son kısmın tekrar edilmesini sağlamak,
- İletişim hızını düşürmek ve/veya gönderilen işaretin içine tekrarlılık eklemek.

Bunların hepsi, kendisini değiştiremeyeceğimiz ortamın içindeki, işareti bozucu etmenlerin iletişimi kötü yönde etkilemesi ve alıcıda hataya sebep olmasına karşı alınan önlemlerdir. İşaretin gücünü arttırma seçeneği basitçe İşaret/Gürültü oranını (SNR: signal to noise ratio) arttıracağından, işaret gücünün yasalar ve yönetmelikle sınırlandırıldığı durumlar dışında her zaman yapılabileceğinden, bu bölümde diğer iki seçeneği inceleyeceğiz.

Gönderilecek veri içinde her zaman bir tekrarlılık bulunur. Aynı bilgiyi daha az veri ile taşımak/saklamak için 2.*Veri Sıkıştırma* bölümünde gördüğümüz yöntemler kullanılır. Bunlar, veri içindeki tekrarlılığı azaltırlar. Şimdi de, bu tekrarlılığı arttırarak iletişimi daha kolaylaştırmayı düşünüyoruz. Bu bir çelişki gibi görünüyor. Ancak, normalde veri içinde bulunan tekrarlılık, verinin alıcı tarafında anlaşılmasını kolaylaştırıcı şekilde değildir, genellikle işe yaramaz. Bu bölümde değineceğimiz tekrarlılık, veri tarafından taşınan bilgiyi verinin farklı noktalarına yayarak, bir bölümü bozulduğunda diğer bölümlerini kullanarak düzeltmeye ya da en azından bozulduğunu anlamaya yöneliktir. Tekrarlılığı azaltma işlemine kaynak kodlama (source coding) demiştik. Faydalı tekrarlılık ekleme işlemine de Kanal Kodlama (channel coding) ismi verilir.

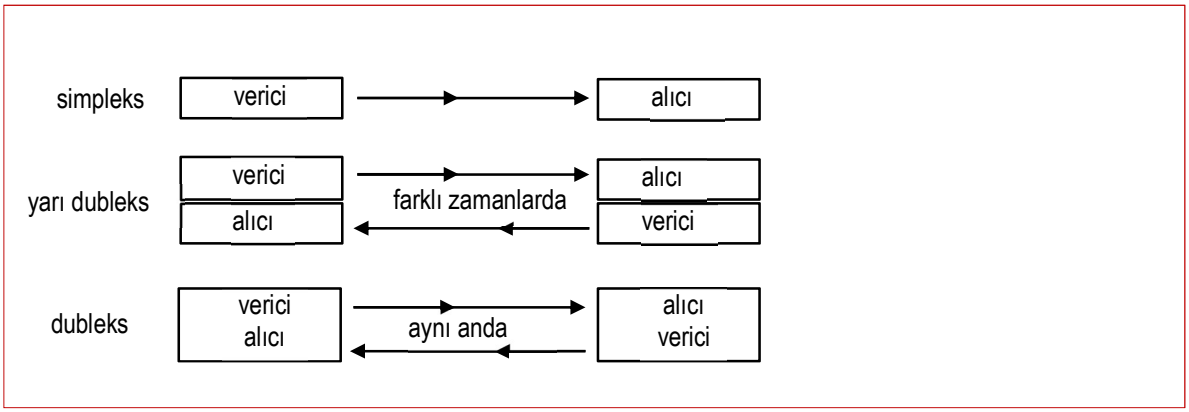
Faydalı tekrarlılık ekleme (ve alıcıda çıkarma) işlemi başlıca şekilde yapılır;

- Alınan veri içinde hata olup olmadığını bulma,
- Veri içinde tespit edilen hataları düzeltme.

Bu iki yaklaşımın zorluk seviyeleri farklıdır ve ikisi beraber de kullanılabilirler. Burada önemli bir varsayımız var. Hatalı veriden kastımız, içindeki bazı bitlerin yanlış değer taşımasıdır. Veri içinde eksik ya da fazladan bit varsa buna yapabileceğimiz pek birşey yok. Bu bölüm sonuna kadar verideki toplam bit sayısının aynı kaldığını, iletişim sırasındaki eşzamanlama bozukluğundan dolayı eksilip artmadığını varsayıyoruz.

Alıcı tarafında, kanaldan elde edilen veri içinde hata olduğu bulunursa izlenebilecek iki yol vardır; verinin hatalı alındığını vericiye bildirip tekrar gönderilmesini istemek ve hatalı kısmı alıcıda düzeltmeye çalışmak. İlk yol göreceli olarak kolaydır. Bu yaklaşımda, gönderilen verinin içine özel

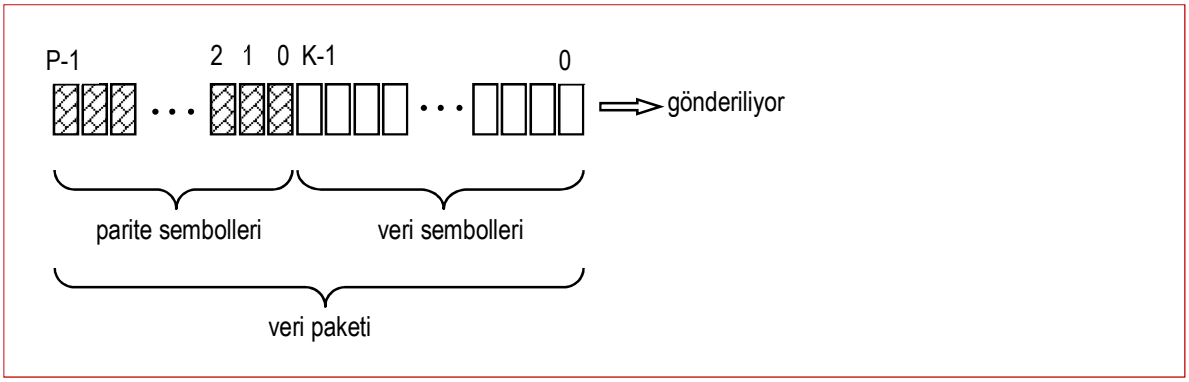
olarak yerleştirilen tekrarlar alıcıda kontrol edilerek sağlaması yapılır. Hata bulunması durumunda bu durumu vericiye bildirmek için iletişim kanalının çift-yönlü (duplex) olması gerekir. Şekil 9.1 iletişim kanalı modlarını açıklamaktadır.



Şekil 9.1. Tek yönlü (simplex), yarı çift yönlü (half duplex), tam çift yönlü (full duplex) iletişim modlarını açıklayıcı şekil.

Uygulamanın karakteristiklerine göre iletişimin tek ya da çift yönlü (yarı ya da tam) olacağı belirlenir. Eğer alıcıdan vericiye bir geri besleme (onay, hata bildirimi vb) olması gerekiyor ise doğal olarak çift yönlü iletişim olmalıdır. Örneğin yerel ağlar üzerinden yapılan veri haberleşmesi çift yönlüdür. Burada kullanılan paket haberleşmesi yöntemi dolayısı ile, alınan her paketin (belli boyutta veri) doğruluğu kontrol edilir ve varsa hata durumu göndericiye bildirilir. Her paket içinde verinin bütünlüğünü kontrol amacı ile bir çevrimsel artıklık kodu (CRC : cyclic redundancy check code) bulunur. Bu tür kodlar, çoğu paket/dosya/blok verisi için kullanılmakta ve çeşitlilik arzetmektedir. Örneğin bir CRC kodu, veri içindeki kelimelerin (byte/word) toplamını temsil edebilir. Alıcı da aynı toplamı yapar ve CRC ile aynı değer bulunmaz ise verinin hatalı olduğuna karar verilir. Artıklık ya da tekrarlılık kodları veriyi düzeltmeye yaramaz, sadece hata olup olmadığını, belki de kanal üzerinde veriye müdahale edilip edilmediğini bulmaya yarar. Hata olduğunda geri bildirim ve aynı paketin yeniden gönderilmesi hem iletişim hattını hem de cihazları fazladan meşgul edeceğinden, geri bildirimli hata düzeltme yaklaşımı iletişim kanalının az hata ürettiği durumlarda daha doğru bir yaklaşımdır. Aksi halde, daha karmaşık olan ama hatanın alıcıda düzeltilebileceği hata düzeltme yöntemleri kullanılır. Örneğin Mars'tan gönderilen resim verisi paketinin tekrar gönderilmesini istemek, gönderilmiş verilerin Mars'taki cihazda ortalama 40 dk daha tutulmasını gerektirir. Çünkü verinin dünyaya ulaşması ortalama 20 dk sürer.

Hata düzeltme kodları (ECC: error correction codes) da aynı şekilde bir matematiksel ifade ile üretilir. Ancak ECC, öncesinde ya da beraberinde gönderilen veri bloğuna göre yeterince büyüktür. ECC ile her hata düzeltilemese de çoğu küçük hata düzeltilebilir. Bu kodlara ve kodları oluşturan bitlere genellikle parite ve parite bitleri ismi verilmektedir. Şekil 9.2 K adet veri sembolü (bit/kelime) ile tüm veri paketinde bazı hataların bulunup düzeltilmesine olanak veren P adet parite sembolünün birleştirilerek veri paketinin oluşturulması gösteriyor. Tabi normalde K adet sembol gönderilecekken, ECC ile $K + P$ adet veri gönderilmiş oluyor ve bu daha uzun sürdüğü için kanal verimliliği bir miktar düşüyor.



Şekil 9.2. K adet veri biti/kelimesi ile beraber P adet parite biti/kelimesi birleştirilerek veri paketi oluşturulur.

9.1 Basit Parite

Çok basit bir örnek ile parite sembollerinin hata bulmaya yardımını anlamaya çalışalım. Paketlerimiz 2 bit veri 1 bit pariteden oluşsun. Bu durumda, kanal kapasitemizin yaklaşık %66'sını kullanıyor olacağız. Buna karşılık alınan paketlerde en fazla 1 bit hata varsa bunu tespit edebileceğiz. Paketimizin veri bitleri x_1x_0 parite biti ise p olsun. Veri bitlerinin ardından, 3üncü bit olarak $p = x_1 \oplus x_0$ işlemiyle hesaplanan biti gönderelim. Yani veri paketimiz x_1x_0p bit üçlüsüdür. Burada $\oplus$ eldesi olmayan/atılan toplamadır (modulo sum, xor). x_1 ve x_0'ın değerlerine göre p değerleri

x_1	x_0	p
0	0	0
0	1	1
1	0	1
1	1	0

tablosunda görülmekte. Yani, alıcının doğru olarak kabul edeceği bit üçlüleri 000, 011, 101 ve 110'dır. Diğer bit üçlüleri, 010, 100, 001 ve 111 ise, hatalı olarak görülecektir. Çünkü alıcı aynı $p = x_1 \oplus x_0$ işlemini yaptığında farklı bir p biti aldığını görecek, en az 1 bitin yanlış olduğuna karar verecektir. Hangi bitin hatalı olduğunu bilemeyeceği için tüm paketin vericiden yeniden istenmesi gerekecektir.

Aynı anda 2 bitin hatalı okunması durumu ise farkedilmeyecektir, çünkü parite hesabı doğru olduğunu söyleyecektir. Yani bu basit örnekteki sistem en fazla 1 bitin hatalı olması durumunda çalışır. En fazla 1 bitin hatalı olabileceği kabullenmesi ile veri bitlerinin sayısı arttırılabilir. Örneğin

$p = x_6 \oplus x_5 \oplus x_4 \oplus x_3 \oplus x_2 \oplus x_1 \oplus x_0$ ile hesaplanan parite bitiyle oluşturulan

$v = x_6x_5x_4x_3x_2x_1x_0p$ paketi 7 bitlik ASCII karakter kodlarının yakın mesafe kablolu iletiminde oldukça fazla kullanılmıştır. Burada da parite bitinin işlevi aynıdır. Alıcıda parite biti veri bitlerinden tekrar hesaplanır ve alınan parite biti ile karşılaştırılır. Böylelikle paket içinde hata olup olmadığı bulunur. Hangi bitin hatalı olduğu bilinemeyeceğinden paket göndericiden tekrar istenir, ya da göndericinin beklediği onay işareti gönderilmez. Benzeri şekilde, paket içinde çift sayıda hatalı bit varsa, alıcı bunu farkedemez.

K bitlik kelimeye 1 bit parite eklenmesi ($P = 1$) ve $n = K + 1$ bitlik kelimeler oluşturulması durumunda, alıcıda hataların tespit edilme/edilmeme olasılığına bakalım, çünkü bazı hatalar (çift sayıda bitin hatalı olması) tespit edilemiyor. Kanaldan okunan herhangi bir bitin hatalı olma olasılığı p_b olsun. Bu durumda, n bit içinden sadece 1 bitin hatalı olma olasılığı

$$\begin{aligned} p_{1bithata} &= (1-p_b)(1-p_b)\cdots(1-p_b)p_b \\ p_{1bithata} &= (1-p_b)^K p_b = (1-p_b)^{n-1}p_b \end{aligned} \tag{9.1}$$

olur. $(1-p_b)$ ilgili bitte hata olmama olasılığıdır. Benzeri şekilde 2 bitin hatalı olma olasılığı da

$$\begin{aligned} p_{2bithata} &= (1-p_b)(1-p_b)\cdots(1-p_b)p_b p_b \\ p_{2bithata} &= (1-p_b)^{K-1}p_b^2 = (1-p_b)^{n-2}p_b^2 \\ p_{2bithata} &= (1-p_b)^{n-1}\frac{p_b^2}{(1-p_b)} = p_{1bithata}\frac{p_b}{(1-p_b)} \end{aligned} \tag{9.2}$$

şeklinde yazılabilir. Tam olarak k bitte hata olma olasılığı ise

$$\begin{aligned} p_{kbithata} &= (1-p_b)(1-p_b)\cdots(1-p_b)p_b p_b \cdots p_b \\ p_{kbithata} &= (1-p_b)^{K-k}p_b^k = (1-p_b)^{n-k-1}p_b^k \\ p_{kbithata} &= p_{1bithata}\left(\frac{p_b}{(1-p_b)}\right)^{k-1} \end{aligned} \tag{9.3}$$

şeklinde yazılır. Çift sayıda bitte hata yapıldığında tespit edemediğimize göre

$$\begin{aligned} p_{hatakaçağı} &= p_{2bithatakaçağı} + p_{4bithatakaçağı} + \cdots \\ p_{hatakaçağı} &= p_{2bithatasayısı}p_{2bithata} + p_{4bithatasayısı}p_{4bithata} + \cdots \end{aligned} \tag{9.4}$$

şeklinde yazılabilir. Burada $p_{xbithatasayısı}$ kelime içinde olası farklı x bitlik hata sayısıdır. Örneğin x=2 bitlik hatalar (v_0, v_1), (v_0, v_2), (v_0, v_3), ..., (v_1, v_2), (v_1, v_3), ... çiftlerinden birisi olabilir ve çiftlerin sayısı da

$$p_{2bithatasayısı} = \binom{n}{2} = \binom{K+1}{2} = \frac{n!}{2!\,(n-2)!} \tag{9.5}$$

ile gösterilir. Tabi ki, genel olarak, n bit kelime içinde kaç şekilde k bit hata olabileceği de

$$p_{2bithatasayısı} = \binom{n}{k} = \binom{K+1}{k} = \frac{n!}{k!\,(n-k)!} \tag{9.6}$$

şeklinde yazılabilir. İlgilendiğimiz hatalar (tespit edilemeyecekler) k'nın çift olduğu durumlar olduğundan denklemler (9.3) ve (9.4)'ü takiben

$$\begin{aligned} p_{çiftbithatakaçağı} &= \binom{n}{2}(1-p_b)^{n-2}p_b^2 + \binom{n}{4}(1-p_b)^{n-4}p_b^4 + \cdots \\ p_{çiftbithatakaçağı} &= \sum_{i=1}^{int\{n/2\}} \binom{n}{2i}(1-p_b)^{n-2i}p_b^{2i} \end{aligned} \tag{9.7}$$

ile hesaplanabilir. Buna göre daha önce çok kullanıldığını söylediğimiz K=7, P=1, n=7+1=8 bitlik kelimelerden oluşan iletişim sisteminde tespit edilemeyen hata olasılığını hesaplayalım. Her bitin hatalı okunma olasılığı ise $p_b = 1/10000 = 10^{-4} = 0.0001$ olsun. Denklem (9.7)'den

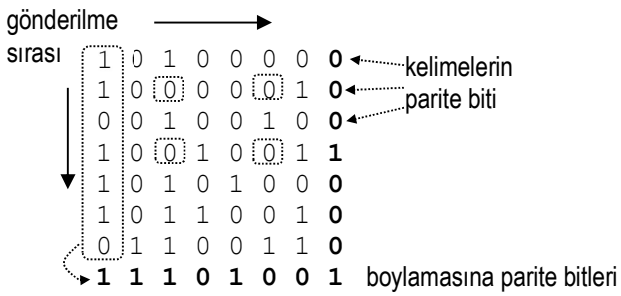
$$p_{çiftbithata\ \ çağı} = \binom{8}{2}(1-10^{-4})^6 10^{-8} + \binom{8}{4}(1-10^{-4})^4 10^{-16} + \binom{8}{6}(1-10^{-4})^2 10^{-24} + \binom{8}{8}(1-10^{-4})^0 10^{-32}$$

$p_{çiftbithatakaçağı} = 2.7983 \times 10^{-7}$ bulunur. Yani, iletilen yaklaşık 3.5 milyon 8 bitlik kelimeden birisinde hata olduğu halde farkedilemez. İlgilenenler için denklem (9.7)'nin Octave kodu Şekil 9.3'te verilmiştir.

```
prob = 0;
n = 8;        % değiştiriniz
pb = 0.0001; % değiştiriniz
for i=1:floor(n/2);
  i2 = 2*i;
  sayi = factorial(n)/(factorial(i2)*factorial(n-i2));
  prob = prob + sayi*(1-pb)^(n-i2) * pb^i2;
end;
```

Şekil 9.3. Bir adet parite biti içeren n bitlik kelimelerde tespit edilemeyen hatalar için Octave kodu.

Birçok iletişim sisteminde tespit edilemeyen hata olması kabul edilemez. O nedenle parite biti sayısını arttırarak ve konfigurasyonunu değiştirerek tespit edilemeyen hata sayısı azaltılmaya çalışılır. Boylamasına parite kelimesi bu yaklaşımlardan birisidir. Burada her kelimeye bir parite biti eklendiği gibi gönderilen her N adet kelime için de bir parite kelimesi eklenir. Parite kelimesinin ilk biti N kelimenin ilk bitleri için pariteyi, 2nci biti ikinci bitlerin paritesini vb. göstermektedir.



Şekil 9.4. Boylamasına parite kullanımı örneği. Her satır ve sütunda çift sayıda 1 var. Parite bitleri kalın gösterilmiş.

Şekil 9.4'teki örnekte yapılacak birkaç deneme ile anlaşılacağı üzere kelimelerde yapılan ama kelimelerin sonundaki parite bitinden tespit edilemeyecek 2 bitlik hatalar boylamasına parite bitleri sayesinde bulunabilir. Burada, bir diktörtgenin köşelerinde olacak şekilde oluşmuş 2'şer bitlik hataların tespit edilemeyeceğini görebiliriz. Şekilde örnek olarak 4 adet bit kesikli küçük diktörtgen ile işaretlenmiştir. Bu bitlerde aynı anda hata olması durumunda hem kelime parite bitleri hem de boylamasına parite bitleri bu durumu gözden kaçırır. Yani, boylamasına parite kontrolü ve bunun için eklenen ilave kelimeler de tüm hataları bulamaz, ama gözden kaçma ihtimalini azaltır.

Hem her kelimede hem de kelime bloğu sonunda parite bitleri eklemek yerine, sadece blok sonlarında, blok içinde olası hataları kontrol etmeyi sağlayacak şekilde, parite biti gibi kontrol bilgisi eklemek bir diğer seçenektir. Böyle bir örnek, gönderilen kelimelerin toplamını ya da toplamından üretilecek bir kodu göndermektir. Toplama Sağlaması (Checksum, sağlama toplaması) ismi verilen bu yöntemde, kelimeler pozitif tamsayı olarak düşünülür ve gönderildiği sırada toplamları da hesaplanır. Tüm bloğun sonunda bu toplam ya da bitlerinin tersi gönderilir. Böylece alıcı da aynı toplamı yapar ve

sonundaki sağlama bitleriyle de kontrol eder. Uyuşmuyor ise hata var demektir. Bu yaklaşım, hata olasılığının düşük olduğu sistemlerde büyük blokları hızlıca gönderebilmek için kullanılır. Örneğin, gönderilecek bloğun 8 bitlik kelimelerden oluştuğu

{12, 40, 05, 80, FB, 12, 00, 26, B4, BB, 09, B4, 12, 28, 74, 11}

verisini ele alalım. Tüm bloğun toplamı,

12+40+05+80+FB+12+00+26+B4+BB+09+B4+12+28+74+11= 4F5

olarak hesaplanır. Son kelime (11) gönderildikten sonra 04 ve F5 kelimeleri de gönderilir. Bazı durumlarda sadece F5 kelimesi gönderilir ve olası hataları tespit etmesi umulur. Bazı durumlarda da kelimeler elde (carry) ihmal edilerek toplanır ve 1 kelimelik sonucun bitlerinin tersinden oluşan kelime gönderilir. Tabi ki, alıcı sağlama kelimesi dahil tüm kelimeleri elde olmadan toplar ve sonucun 00 olmasını bekler. Aksi halde hata oluşmuş demektir.

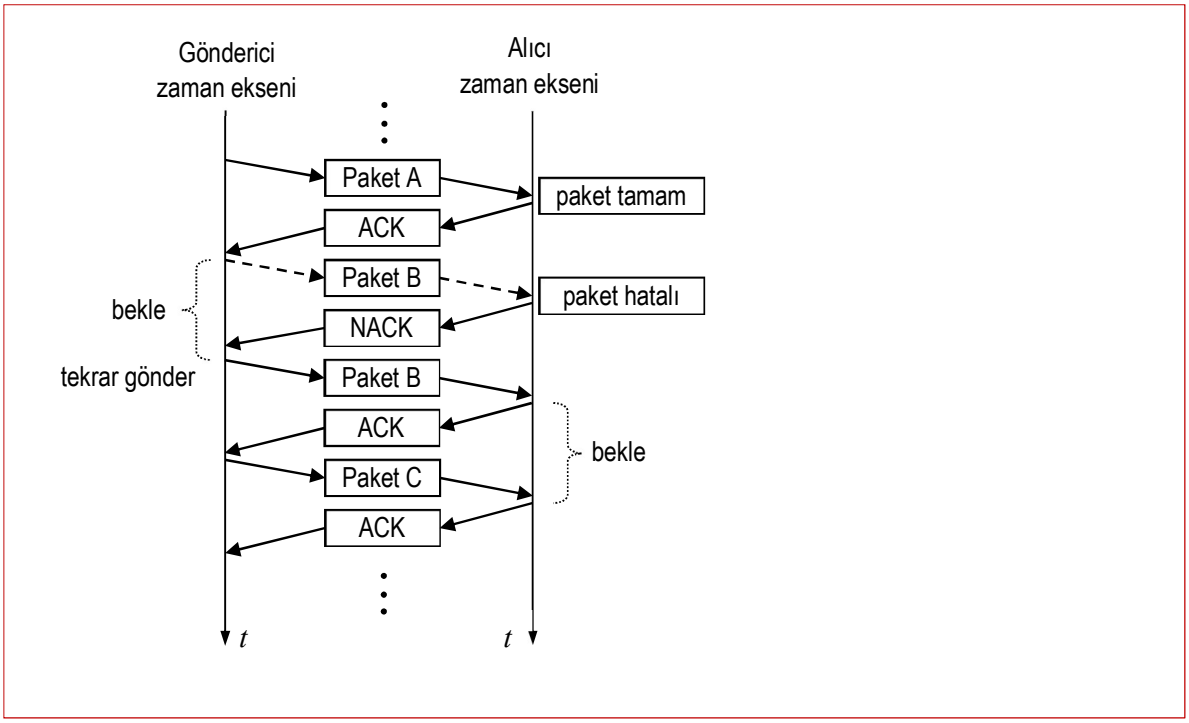
Toplamanın yukarıda anlatıldığı gibi basit toplama olması gerekmez. Pozisyon Duyarlı Toplama da denilen döngüsel tekrarlılık kontrolü (CRC : cyclic redundancy check) çerçeveyi uzun bir ikili polinom P olarak ele alır. Gönderici bu polinomu sabit bir G polinomuna böler ve bölünemeyen kısmını (kalan) çerçevenin ardından gönderir. Alıcı tarafında da aynı bölme gerçekleştirilir ve bölmenin kalanının aynı olması beklenir. Kelimelerin ve bitlerin pozisyonları bölme sonucunu ve kalanı etkilediğinden toplama sağlamasından çok daha etkili bir hata bulma kabiliyetine sahiptir. Ancak, bu etkililiğin oluşması için G polinomunun özenle belirlenmiş polinomlardan olması gereklidir.

Örneğin yerel bilgisayar ağlarındaki iletişimde (eternet) paket sonlarına eklenen CRC, verinin 04C11DB7 sayısı ile temsil edilen

$$G = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

polinomuna bölünmesinden kalan ile oluşturulur. Benzeri şekilde, USB haberleşmesinde kullanılan polinom da $G = x^5 + x^2 + 1$'dir. Her ne kadar CRC kullanımıyla hataların oldukça önemli bir kısmı tespit edilse de, hepsi tespit edilemez. Özel polinomları ve bunlarla hata düzeltme (sadece tespit değil) işleminin nasıl yapıldığını hata düzeltme kodları (ECC) konusunda göreceğiz.

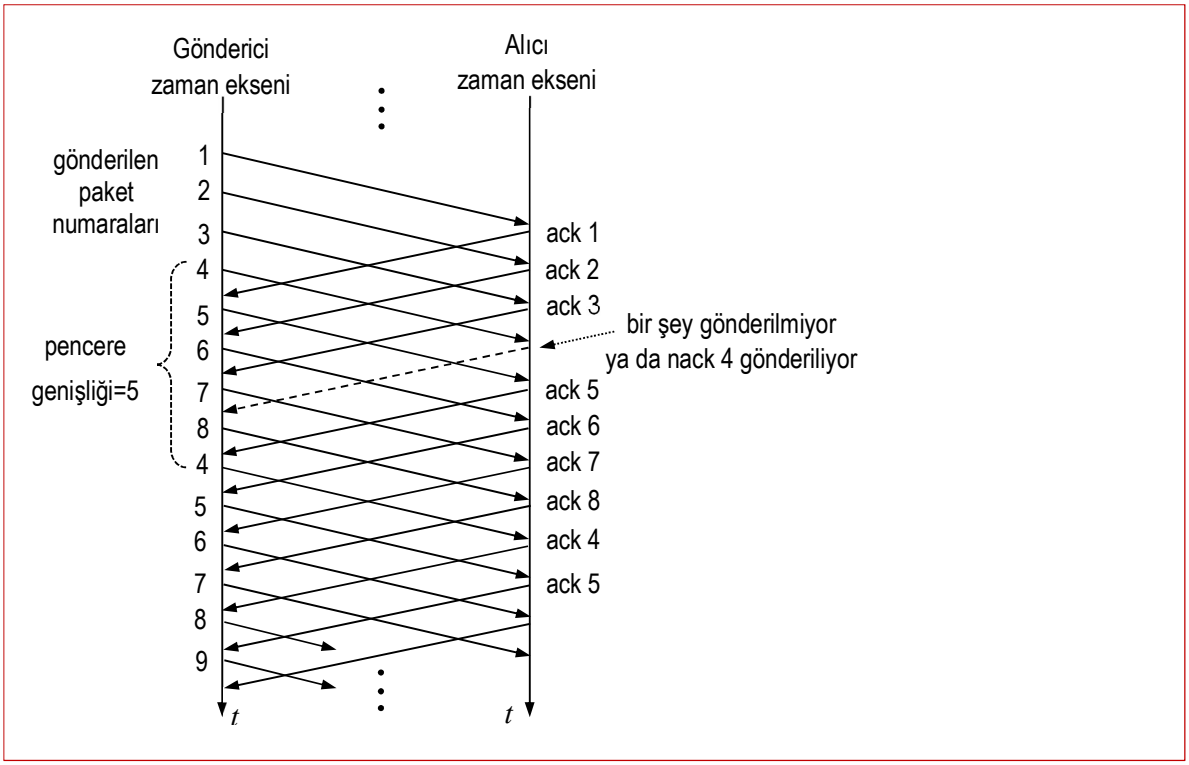
İçinde hata olduğu saptanan veri blokları için göndericiye bir yeniden gönderim isteği iletilir. Bu isteklere Otomatik Tekrar İsteği (ARQ: Automatic Repeat reQuest) denir. İki yönlü iletişim gerektiren ARQ işlemi için 3 yaklaşım vardır. Şekil 9.5'te Dur-Bekle-ARQ (stop-and-wait-ARQ) yaklaşımı açıklanmaktadır. Bu yaklaşım el-sıkışma (handshaking) protokolüne benzer. Gönderici her veri bloğu gönderiminden sonra alıcıdan paketin doğru alındığı (ACK) ya da alınmadığına (NACK) dair bir cevap bekler. Cevap NACK ise ya da önceden belirlenen bir süre (timeout) sonunda herhangi bir cevap gelmemişse gönderici son veri bloğunu tekrar gönderir. Tabi ki bekleme süreleri boşa geçen zamandır. O nedenle bu basit yaklaşım iletişim hızının önemli olmadığı ya da alıcının gönderilen veriyi zaten daha yavaş işlediği/kullandığı uygulamalarda yer bulur. Örneğin bilgisayar ile yazıcı arasındaki iletişimin bu şekilde olmasında bir sakınca yoktur. ACK/NACK bildirimi sadece 1 bit de olabilir, ki 1 ACK, 0 NACK anlamına gelecektir.



Şekil 9.5. Dur-Bekle-ARQ hata bildirim protokolünün açıklaması.

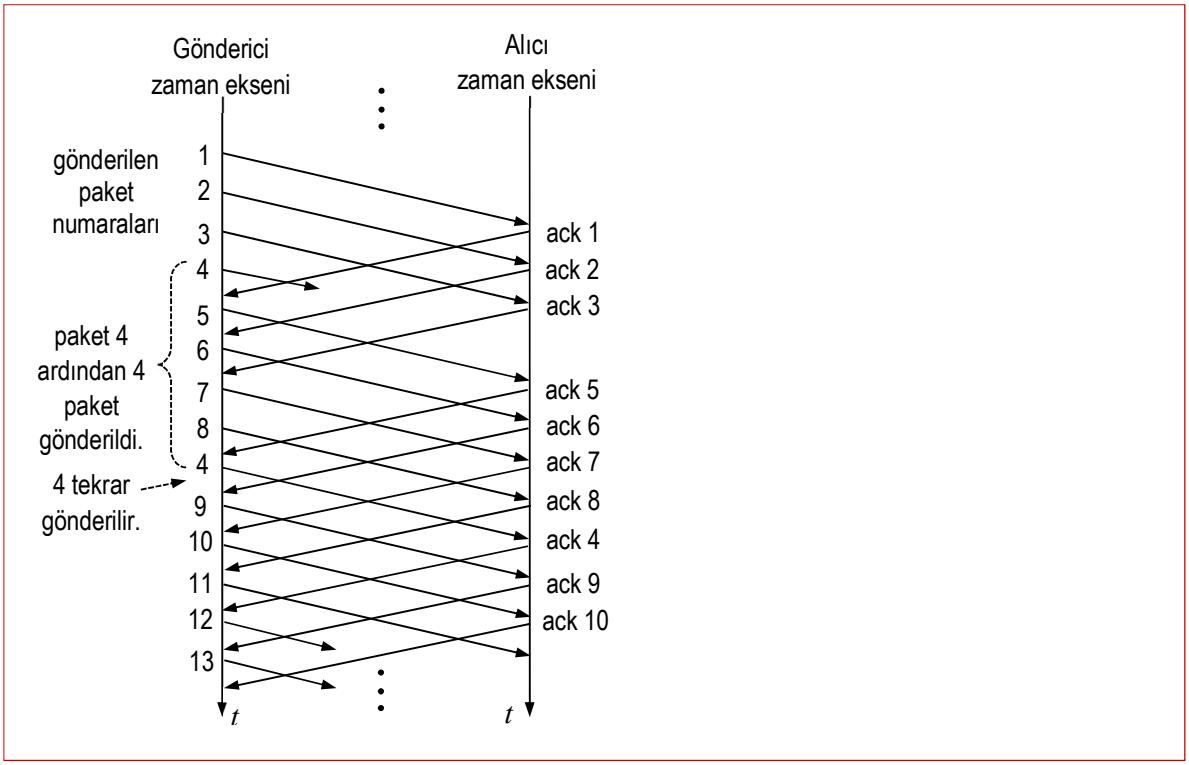
Bu gibi sistemlerde bir zamanlama kontrolü de vardır. Gönderici alıcıdan uzun süre cevap alamadığında ya da devamlı olarak NACK aldığında iletim ortamında problem olduğuna karar verip bir hata işareti üretir. Bu yaklaşımda paketlerin şekilde gösterildiği gibi numaralı/tanımlı olmasına gerek yoktur. Çünkü sadece son gönderilen paket için bir cevap beklenir. Cevap gelinceye kadar da, tekrar gönderilme ihtimali nedeniyle, gönderici hafızasında tutulur.

Hata olasılığının daha az olduğu ve alıcının alınan paketleri sıraya koyma kabiliyetinin sınırlı olduğu sistemlerde beklemede geçen süreyi değerlendirmek için farklı bir yaklaşım kullanılabilir. Gönderici, paketleri numaralandırıp (paketin içine gömülü bir sayı) ardışıl paketler arasında herhangi bir ACK beklemeksizin sırasıyla gönderir. Gönderdiği ve henüz ACK almadığı paketleri ve numaralarını da hafızada tutar. Gönderici, ACK almadığı en küçük paket numarası ile göndermek üzere olduğu paketin numarası arasındaki fark önceden belirlenen bir sayıyı aşarsa ACK almadığı paketten itibaren tüm paketleri yeniden ve aynı numaralarla gönderir. Önceden belirlenen bu fark sayısına pencere, bu yönteme de kayan pencere ARQ (sliding window ARQ) denir. Yöntemin zaman diyagramı Şekil 9.6'te gösterilmiştir. Gönderilen her paket için hemen ARQ beklenmemesi iletişimi hızlandırır. Tabi ki ACK bildirimleri de yolda kaybolabilir. En son başarılı gönderilen paketten itibaren yeniden gönderme ile, gönderimin güvenliği sağlanmış olur.



Şekil 9.6. Kayan Pencere ARQ hata bildirim protokolünün açıklaması.

Gönderilen paketlerin farklı yollar izleyip alıcıya farklı zamanlarda ulaşması sözkonusuysa (örneğin internet) ve alıcının gelen paketleri sıraya koyma yeteneği varsa, sadece hatalı paketlerin tekrar gönderilmesinin istenmesi daha pratik olabilir. Bu durumda Kayan Pencereli Seçimli Tekrar ARQ hata bildirim protokolü söz konusudur. Şekil 9.7'de örnek zaman diyagramı gösterilen bu yöntemde, yol üzerinde kaybolan ve alınamayan paketler için bir NACK gönderilmesi de problemli olabilir. O nedenle sağlam alınan paketler için paket numaralı ACK gönderilmesi daha doğrudur. Gönderici gönderdiği ama makul sürede ACK alamadığı paketleri yeniden gönderir. Tabi ki diğerlerinden gecikmeli alınan bu paketlerin alıcıda asıl veri bütünü içinde uygun sıraya konulması gerekmektedir.



Şekil 9.7. Kayan Pencereli Seçimli Tekrar ARQ hata bildirim protokolünün açıklaması. ACK alınmayan paketler bir süre sonra yeniden gönderilir.

9.2 Hamming Kodları

Bazı sistemlerde ise paketlerin yeniden istenmesi ya mümkün değil ya da pratik değildir. O nedenle gönderilen verinin içinde, hata olması durumunda, hataları düzeltmeye yarayacak yeterince tekrarlılık olması istenir. Örneğin bir kayıt ortamına (manyetik disk, katı hal bellek kartı vb) kayıt edilen verilerin tekrar okunmasında hata ile karşılaşıldığında, hatalı kısmı tekrar istemek gibi birşey mümkün değildir. Kayıt ortamından tekrar okunması durumunda yine aynı hatalarla karşılaşılacaktır. Mantıklı olan, kayıt sırasında olası hatalar için tekrarlılık da yerleştirmektir. Bir başka örnek ise, Mars'tan gönderilen resimlerin parçalarını tekrar istemektir. Bu pratik olmaktan uzaktır, çünkü EM dalganın Mars'a gidip gelmesi ortalamada 40dk'ya mal olur. Bu süre zarfında, zaten gönderilmiş olan verinin Mars aracı hafızasında tutulması gerekir. Tabi ki en veri güvenlikli sistem, hem tekrar veri istenmesine izin veren hem de veri içinde faydalı tekrarlılık (düzeltmekta faydası olan) içeren bir sistemdir. Nasıl bir veri koruma istendiği, uygulamaya göre belirlenecek bir şeydir.

Kelime başına tek bir parite bitinin bozulan kelimeyi düzeltmeye yaramadığını, sadece bozulduğunu anlamaya yaradığını görmüştük. Parite biti sayısını arttırma ile yeni olanaklar ortaya çıkar. Burada da iletişim kanalı ile ilgili bazı varsayımlar kullanılır. Örneğin bir kelime içinde sadece 1 bitin hatalı olabileceği varsayımı ile tasarlanan parite biti bloğu, olası tüm tek bitlik hataları (parite bitlerindekiler dahil) düzeltmeye yetecek kadar geniş olmalıdır. Burada bir imada bulunduk; Veri kelimeler halindedir ve her kelimeye parite bitleri eklendiğinde daha büyük kelimeler üretilir. Ancak tüm kelimeler birbirinden bağımsız değerlendirilirler. Bu yaklaşıma blok kodlama ismi verilir. Yine bir

imada bulunduk; Demek ki blok olmayan kodlama yöntemleri de var. Şimdi blok kodlama yöntemlerini inceleyelim.

Kodlamadan önceki kelime uzunluğu k ve kodlamadan sonraki kelime uzunluğu n ise, yani $n-k$ adet parite biti eklenmiş ise, bu blok kodlara (n, k) kodu denir.

$$R = \frac{k}{n} < 1 \tag{9.8}$$

oranına kod oranı denir ve her zaman 1'den küçüktür. R 1'e ne kadar yakın ise verimliliğin o kadar yüksek olduğunu söyleyebiliriz. Daha önce gördüğümüz $(n = 8, k = 7)$ basit 1 bitlik parite sisteminde R=7/8 olduğunu ve bunun bir hata düzeltme sağlamadığını biliyoruz. O nedenle, bilgi biti başına eklenen parite biti sayısını ifade eden

$$tekrarlılık = \frac{n-k}{k} < 1 \tag{9.9}$$

oranının büyütülmesi gerektiğini görüyoruz. Blok kodlama işlemini, giren her k bitlik kelime için n bitlik bir başka kod kelimesi üreten bir blok olarak düşünelim. Olası k bitlik kelime sayısı 2^k ve bunlara karşılık üretilebilecek olası n bitlik kod sayısı ise 2^n'dir. $n > k$ ve dolayısıyla $2^n > 2^k$ olduğuna ve girişte olmayacak kelimeler için kod üretmeyeceğimize göre kod kelimeleri çok daha geniş (2^n) seçenek alanından seçilmiş 2^k adet koddur. Hata bulma ve/veya hata düzeltme için etkili olan şey, bu n bitlik 2^k adet kod kelimesinin seçimi ve kodların birbirleri arasındaki uzaklıktır. O halde önce kod kelimeleri arasındaki uzaklığı tanımlayalım. Kod kelimesi seti içindeki herhangi c_i ve c_j kod kelimesi arasındaki Hamming uzaklığı

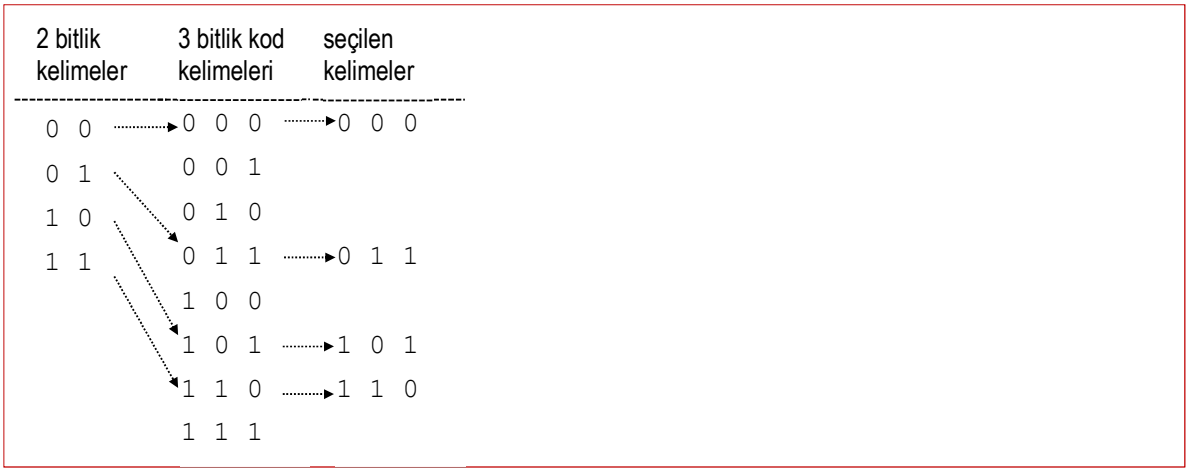
$$d(c_i, c_j) = adet\{c_i[b] \neq c_j[b], b = 0, \cdots, n-1\} \tag{9.10}$$

şeklinde tanımlanır. Yani karşılıklı bitler arasında aynı olmayan bitlerin sayısıdır. Örneğin, 1010011 ve 0111011 kelimeleri arasındaki Hamming uzaklığı, x karşılığı farklı olan bit pozisyonları olmak üzere xx1x011 içindeki x sayısı, yani 3'tür.

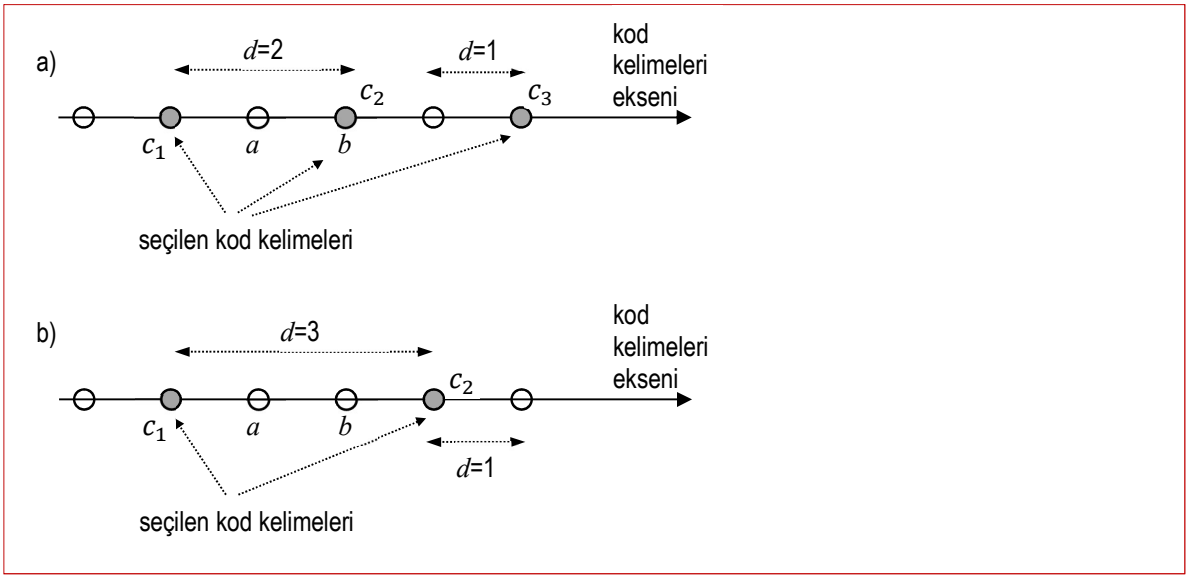
6.*Tabanbant İletişim* bölümünde semboller arası uzaklık arttırıldığında hatanın azaldığını görmüştük. Burada da benzeri bir yaklaşım sözkonusu. Kod kelimeleri arasındaki uzaklığı, yani Hamming uzaklığını, arttırmaya çalışıyoruz. Örnek olarak (n, k)=(3, 2) durumunu ele alalım. Yani 2 bitlik kelimeye 1 bit parite ekleniyor, sonuçta 3 bitlik kod kelimeleri üretiyoruz. 2 bitlik 2^2=4 adet kelimeye karşılık 3 bitlik 2^3=8 adet kod kelimesi içinden seçilecek. Tabi ki kod kelimeleri arasında en fazla uzaklık olacak şekilde seçim yapmalıyız. Bu seçim Şekil 9.8'de açıklanıyor.

1 adet parite biti eklenmesi dolayısıyla kod kelimesi uzayındaki kelime sayısı 2 katına çıkıyor. Böylece seçimler arasında Hamming uzaklığı olarak 2 elde edilebiliyor. Alıcı tarafından kanaldan okunan 3 bitlik kodun 1 biti hatalı ise, tabloda olmayan bir kod kelimesi oluşuyor. Böylece alıcı, alınan kelimenin hatalı olduğuna karar verebiliyor. Aralarında 1 bit fark olan kod kelimelerinin seçimi Şekil 9.9a'da gösteriliyor. Alıcı, örneğin a kelimesini alırsa hatalı olduğunu anlıyor.

Kod kelimeleri arasındaki Hamming uzaklığı 3'e çıkarılırsa, yapılabilecek 1 bitlik hatalar en yakın kod kelimesi seçilerek düzeltilebilir. Bu durum, Şekil 9.9b'de açıklanıyor. Şekle göre c_1 kod kelimesinin gönderildiğini, ancak 1 bitlik hata dolayısı ile a kelimesinin alındığını varsayalım. a'ya en yakın geçerli kod kelimesi seçildiğinde c_1 bulunur ve yapılan 1 bitlik hata düzeltilmiş olur. Benzeri şekilde, c_2 kod kelimesi gönderildiğinde ve hata dolayısı ile b alındığında, b geçerli bir kod kelimesi olmadığından ve en yakın geçerli kod kelimesi c_2 olduğundan gönderilen kod kelimesinin c_2 olduğu kabul edilir ve hata düzeltilmiş olur. Tabi 2 bit birden hatalı olursa hata tespit edilemez. Şekil 9.9'da 1 boyutlu eksen üzerinden anlatılmaya çalışılan kullanılacak kod kelimeleri seçimi aslında çok boyutlu uzayda yapılıyor.



Şekil 9.8. Sekiz adet 3 bitlik kod kelimesi içinden 4 adet kod kelimesi seçimi.



Şekil 9.9. Aralarında 1 bit fark olan (Hamming uzaklığı 1) olan kod kelimeleri içinden a) d=2 olarak seçilmiş kod kelimeleri b) d=3 olacak şekilde seçilmiş kod kelimeleri.

Anlatılan yaklaşımla 2 bitlik hata tespiti için kod kelimelerinin arasındaki uzaklığın 2, 2 bitlik hataların düzeltilmesi için ise bu uzaklığın 5 olması gerektiğini görebiliyoruz. Genel olarak b bitlik hataların tespiti için

$$min\{d(c_i, c_j)\} \geq b \tag{9.11}$$

b bitlik hataların düzeltilmesi için ise

$$min\{d(c_i, c_j)\} \geq 2b + 1 \tag{9.12}$$

olması gerekiyor.

Tabi parite bitleriyle beraber kod kelimelerini Şekil 9.8'deki gibi alt alta yazarak aralarından d uzaklığı olanları seçmek yerine kod kelimeleri arasında olmasını istediğimiz bağıntılardan yola

çıkabiliriz. Örneğin kod kelimelerinin, $c_i = [x_i\ p_i]$ şeklinde yani (n, k) kodu için ilk k bitin orijinal mesaj kelimesinin x_i bitleri, ardından $n - k$ bitin ise parite bitleri olmasını tercih ederiz. Böylece, alınan kodda hata olmadığının tespiti halinde asıl mesaj x_i bitleri doğrudan kullanılabilir. Bu yaklaşım için biraz inceleme yapalım.

Kod kelimeleri arasında bulunacak en küçük uzaklık

$$d_{min} = min\{d(c_i, c_j)\}, i \neq j \tag{9.13}$$

en küçük Hamming uzaklığı olarak anılır. c_i kod kelimesinin Hamming ağırlığı da

$$w(c_i) = adet\{c_i[b] \neq\ 0, b = 0, \cdots, n-1\} \tag{9.14}$$

şeklinde, kod kelimesi içindeki 1 sayısıdır. Bir kodeword seti içindeki en küçük $w(c_i)$'ye

$$w_{min} = min\{w(c_i)\} = min\{d(c_i, 0)\}, i = 0{,}1 \ldots \tag{9.15}$$

en küçük Hamming ağırlığı ismi verilir. Doğal olarak $w_{min} = d_{min}$'dir.

Mesaj kelimesi x_i ve karşıgelen (x_i yerine gönderilen) kod kelimesi c_i olsun. $x_i \oplus x_j$ için gönderilecek kod kelimesi $c_i \oplus c_j$ ise bu koda doğrusal blok kod ismi verilir;

$$x_i \rightarrow c_i \text{ ise } x_i \oplus x_j \rightarrow c_i \oplus c_j. \tag{9.16}$$

Yani, her mesaj ve kod kelimesi diğerlerinin doğrusal toplamından (modulo) oluşturulabilir. Örneğin, her biri k bitlik mesaj kelimeleri $e_0 = 00 \ldots 001$, $e_1 = 00 \ldots 010$, $e_2 = 00 \ldots 100$, ... $e_{k-1} = 10 \ldots 000$ gibi her kelimede sadece 1 biti 1 olup diğerleri sıfır olan kelimelerden oluşan bir set olsun. Yani,

$$E = \{e_i | e_i[i] = 1\} \tag{9.17}$$

şeklinde bir taban kelime setindeki kelimelerin doğrusal toplamından diğer tüm olası kelimeler üretilebilir. E setindeki kelimelere karşı gelen k adet n bitlik kod kelimeleri de G setini oluştursun. Doğrusallık gereği herhangi bir x mesaj kelimesi

$$x = \sum_{i=0}^{n-1} x[i] e_i \tag{9.18}$$

şeklinde taban kelimelerinin toplamı şeklinde yazılabildiği gibi, x'e karşı gelen kod kelimesi de

$$c = \sum_{i=0}^{n-1} x[i] g_i \tag{9.19}$$

şeklinde yazılabilir. $x[i]$, yani kelimenin i'inci biti 0 ya da 1 olduğundan i'inci taban kelimesinin toplamaya dahil olup olmayacağını belirtiyor. x toplamına dahil olan e_i'lere karşı gelen kod kelimelerinin toplamı da x'in kod kelimesi c'yi veriyor. Taban kelimelere karşı gelen kod kelimelerini matris şeklinde yazacak olursak

$$G = \begin{bmatrix} g_{k-1} \\ \vdots \\ g_0 \end{bmatrix} \tag{9.20}$$

herhangi bir kod kelimesi de, x mesaj kelimesinin satır matris şekli olmak üzere

$$\mathrm{c} = \mathrm{x}G \tag{9.21}$$

şeklinde yazılabilir.

Örnek: (n, k)=(5,2), yani 2 bitlik mesaj kelimeleri yerine 5 bitlik kod kelimesi gönderen bir iletişim sistemi olsun. Örneğin, X={00, 01, 10, 11} kelimeleri, taban mesaj kelimeleri E={01,10}'in doğrusal toplamları ile ifade edilebilirler. X kelimelerini temsil eden kod C={00000, 10100, 01111, 11011} olsun. Yani, E'deki kelimeleri temsil eden 10100 ve 01111 kod kelimeleri kullanılarak $G = \begin{bmatrix} 10100 \\ 01111 \end{bmatrix}$ yazılabilir. Diğer tüm mesaj kelimelerine karşı gelen kod kelimeleri G'nin elemanlarının doğrusal toplamı şeklinde yazılabiliyor. O halde bu kod doğrusaldır;

$$\begin{aligned} c_{00} &= [0\ 0]G = [00000] \\ c_{01} &= [0\ 1]G = [01111] \\ c_{10} &= [1\ 0]G = [10100] \\ c_{11} &= [1\ 1]G = [11011] \end{aligned}$$

G'ye üretici matris (generator matrix) ismi verilir, rankı k'dır. Örneğini yaptığımız kodun bir özelliği daha vardır. Kod kelimelerinin soldaki 2 biti mesaj kelimesi ile aynıdır. Yani herhangi bir kod kelimesini, p mesaj kelimesinin sonuna eklenecek bitler olmak üzere, $c = [x \quad | \quad p]$ şeklinde yazabiliriz. Örneğin, yukarıdaki örneğe göre, 11 kelimesinin kod kelimesini bulmak için 11'e 011 ekleriz ve 11011'i elde ederiz. Kelimenin p kısmının rastgele olmadığını, mesaj kelimesinden üretildiğini görebiliriz.

G'nin de $k \times k$ kısmı, E'nin tüm değerlerini içermek üzere birim matris, kalan $k \times (n-k)$ kısmı ise parite matrisidir. Yani

$$\mathrm{G} = \begin{bmatrix} g_{k-1} \\ \vdots \\ g_0 \end{bmatrix} = [I| \quad P] = \left[\begin{array}{ccccc|ccc} 1 & 0 & \cdots & 0 & 0 & p_{k-1,n-k-1} & \cdots & p_{k-1,0} \\ 0 & 1 & & 0 & 0 & & & \\ \vdots & & \ddots & & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & & 1 & 0 & & & \\ 0 & 0 & \cdots & 0 & 1 & p_{0,n-k-1} & \cdots & p_{0,0} \end{array}\right] \tag{9.22}$$

Bu şekildeki kodlara sistematik kod ismi veriliyor. Gönderilecek kod kelimesinin ilk k biti mesaj kelimesinden, kalan $n-k$ biti ise sistematik olarak üretilen parite bitlerinden oluşmakta.

Şimdi asıl soruyu soralım; b bitlik hataların düzeltilebilmesi için (n, k) ne olmalıdır, parite bitleri nasıl hesaplanmalıdır? Bu soruya denklem (9.12) yardımıyla cevap verebiliriz. k bitlik kodlarda b bitlik hataların düzeltilebilmesi için $n-k$'nın (parite biti sayısı), dolayısıyla n'nin (kod kelimesi bit sayısı) kelimeler arasındaki uzaklığı en az $2b+1$ yapacak şekilde seçilmesi gerekir.

Bir başka bakış açısıyla, parite biti sayısının her bir hata kombinasyonunu ve hatasızlık durumunu adresleyebilecek kadar sayıda olması gerekiyor. Örneğin n bitlik kod içinde 1 bitlik n farklı hata olabilir. Bu durumda parite bitlerinin sayısının, yani $n-k$'nın $2^{n-k} = 2^p \geq n+1$ eşitsizliğini sağlaması gerektiği açıktır. Bir başka söyleyişle, mesaj kelimelerimiz k bit ise ve p bit parite ekleyip $n = k+p$ bitlik kod kelimeleri üretecek isek, bu kodda 1 bitlik hatanın bulunup düzeltilebilmesi için

$$2^p \geq k + p + 1 \tag{9.23}$$

ya da $2^p - p - 1 \geq k$ olması gerekir.

Tabi soru tersine de sorulabilir. Yani p bitlik parite ile 1 bitlik hataları bulup düzeltmeyi düşünüyor isek mesaj kelimeleri $k \leq 2^p - p - 1$ uzunluğunda olabilir. Örneğin p=3 ile 1 bitlik hataları düzelteceksek k en fazla $2^3 - 3 - 1 = 4$ bit olabilir. Yani kodumuz (4+3, 4)=(7, 4) kod olur. Ayrıca

mesaj bitlerinin ardına parite bitlerini ekleyeceğimiz (mesaj bitlerini değiştirmeden) için kodumuza (7, 4) sistematik kod diyebiliriz.

Örnek: Kendimize bir test sorusu daha soralım. p=5 iken 2 bitlik hatalar düzeltilebilir mi? Düzeltilirse k en fazla kaç olabilir? Önce, 4 parite biti ile kaç farklı hatayı işaret edebiliriz, bulalım; $2^5 = 32$. Yani $\binom{n}{2} < 32$ olmalı. Ayrıca, 1 bitlik hataları da görmezden gelemeyiz. Yani aslında $\binom{n}{2} + n < 32$ olmalı ve $n > 5$ olmalı ki kodumuz bir işe yarasın, sadece parite bitlerini göndermeyelim. Otuzikinci seçenek ise hatasızlık durumunu bildirmek için kullanılacaktır. Bu durumda, $\frac{n!}{2!(n-2)!} + n < 32$ eşitsizliğini sağlayan en büyük n sayısını 7 olarak buluruz. Bunun 5 bitini pariteye ayırdığımıza göre mesaj bitleri sayısı 2 olur, kodumuza da (7,2) ismini verebiliriz. Yani 2 mesaj bitine 5 parite biti eklediğimizde, herhangi 1 ve 2 bitlik hatayı düzeltme şansımız, en azından teorik olarak, vardır. Ama diyebiliriz ki, mesaj kelimeleri zaten 2 bit. 5 bitlik parite ekleyerek sadece parite bitlerini düzeltmeye fayda sağlıyoruz. Olası 2 bitlik hata sayısı $\frac{n!}{2!(n-2)!}$=21 oldukça fazla. Evet, bu kod oldukça verimsiz olurdu.

Örnek: p=5 iken 1 bitlik hataları düzeltmek istersek $2^5 - 5 - 1 = 26$ bitlik mesaj kelimesi olur. Yani kodumuz (31,26) olurdu.

Şimdi, yukarıda bahsettiğimiz (7, 4) doğrusal sistematik kodu inceleyelim. Bu kodun 1 bitlik hataları düzeltebileceğinden bahsetmiştik. Sistematik olduğuna göre ilk 4 bit mesaj biti, son 3 bit ise parite biti olacak;

$$\begin{aligned} c_6 &= x_3 \\ c_5 &= x_2 \\ c_4 &= x_1 \\ c_3 &= x_0 \\ c_2 &= p_2 \\ c_1 &= p_1 \\ c_0 &= p_0 \end{aligned} \tag{9.24}$$

Parite bitleri x_i bitlerinden hesaplanacak ve 000'dan 111'e kadar değerler alacak. Herhangi bir hata olmaması durumunu 000 ile işaretlediğimizi varsayalım. Parite bitlerindeki tek bitlik hatalar sadece ilgili parite bitini ilgilendirdiğinden (düzeltmeye ihtiyaç olmadığından) 001, 010 ve 100 değerleri de bu durumları işaretlesin. Bu durumda geriye kalan 110, 101, 011 ve 111 değerleri sırasıyla (şimdilik keyfi) x_0, x_1, x_2 ve x_3 bitlerindeki hataları gösterebilir. Yani x_3 değeri, 110'dan yola çıkarak hem p_2 hem p_1 bitinin hesabında (onu ters çevirecek şekilde) yer alacak. Benzeri şekilde x_2 p_2 ve p_0, x_1 p_1 ve p_0, x_0 ise tüm parite bitlerinin hesabında yer alacak. Bu durumda, parite bitlerinin hesabındaki denklemlerimiz;

$$\begin{aligned} p_2 &= x_3 \oplus x_2 \oplus x_0 \\ p_1 &= x_3 \oplus x_1 \oplus x_0 \\ p_0 &= x_2 \oplus x_1 \oplus x_0 \end{aligned} \tag{9.25}$$

şeklinde yazılabilir. Tabi ki mesaj ve parite bitlerinin sırasını bu şekilde seçmek zorunda değildik.

Yukarıda, mantık yürüterek yaptığımız hesaplar ve atamalar zaten Hamming kodları için yapılmış durumda. Kod verimliliği olarak tanımlayabileceğimiz k/n'nin (iletilen bit sayısı başına düşen mesaj bit sayısı) yüksek olduğu olası Hamming kodlarının bir üyesi olan (7, 4) kodu için de ailenin diğer üyeleri gibi üretici matrisinin yeniden hesaplanmasına gerek yok. Ancak, mantığı yukarıdaki örnekten anlayabiliyoruz. (7, 4) Hamming kodunun üretici matrisi, (9.24) ve (9.25) denklemleri ile

$$G = \begin{bmatrix} g_3 \\ \vdots \\ g_0 \end{bmatrix} = \left[\begin{array}{cccc|ccc} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{array}\right] \tag{9.26}$$

olmakta. Buna göre, örneğin 0011 mesaj kelimesine karşılık kod kelimesi, g_0 ve g_1 satırlarının modulo toplamı olan $0001\mathbf{111} \oplus 0010\mathbf{011} = 0011\mathbf{100}$ şeklinde bulunur. Kalın semboller sona eklenen parite bitleridir. Üretici matrisiyle olası tüm mesaj kelimelerine karşılık kod kelimeleri üretildiğinde Şekil 9.10 ile verilen liste elde edilir. Liste detaylı bir şekilde incelenirse Hamming ağırlığı ve en küçük Hamming uzaklığının 3 olduğu görülür. Şekil 9.9b'ye göre de geçerli kelimeler arasında kalan herhangi bir geçersiz kelime alıcı tarafından alındığında en yakın geçerli kod seçilerek hatalı alınan bitin düzeltileceği görülebilir.

```
0 0 0 0 0 0 0
0 0 0 1 1 1 1
0 0 1 0 0 1 1
0 0 1 1 1 0 0
0 1 0 0 1 0 1
0 1 0 1 0 1 0
0 1 1 0 1 1 0
0 1 1 1 0 0 1
1 0 0 0 1 1 0
1 0 0 1 0 0 1
1 0 1 0 1 0 1
1 0 1 1 0 1 0
1 1 0 0 0 1 1
1 1 0 1 1 0 0
1 1 1 0 0 0 0
1 1 1 1 1 1 1
```

Şekil 9.10. Bir (7, 4) Hamming kodunda geçerli tüm kod kelimeleri.

Örneğin, gönderilen kod kelimesi 0111001 olsun (mesaj kelimesi 0111). Alıcıya ulaşan kod kelimesi ise, x_0 bitindeki okuma hatası ile, 011**0**001 olsun (hata koyu renk ile gösteriliyor). Tüm liste aranıp 011**0**001'e en yakın (Hamming uzaklığı ile) kelime olarak 0111001 bulunur ve 1 bitlik hata düzeltilmiş olur. Tabi ki her alınan kelime için listeden arama yapmak algoritmik açıdan doğru olmaz. O nedenle hangi bitin hatalı olduğunu gösterecek bir sendrom vektöründen faydalanmak daha pratik olacaktır. Denklem (9.25)'teki eşitliklerde p_i bitlerini eşitliklerin sağ tarafına atarak

$$\begin{aligned} 0 &= p_2 \oplus x_3 \oplus x_2 \oplus x_0 = s_2 \\ 0 &= p_1 \oplus x_3 \oplus x_1 \oplus x_0 = s_1 \\ 0 &= p_0 \oplus x_2 \oplus x_1 \oplus x_0 = s_0 \end{aligned} \tag{9.27}$$

yazılır. $s_2 s_1 s_0$ vektörü, eğer hiç hata yok ise 000 olacaktır. Hata durumunda olası sendrom vektörleri Şekil 9.10'da olmayan kelimeler denklem (9.25)'e uygulanarak bulunabilir. Ele aldığımız kod sistematik olduğundan, parite bitlerindeki hataları görmezden gelebilir ve sendrom vektörünün sadece x_i'lerdeki hatalarda aldığı değerleri inceleyebiliriz. Kod kelimesi 0000000 geçerli olduğuna göre 1 bitlik hatalar, bu kelimenin tüm bitlerini sırasıyla 1 yaparak sendrom vektörünün aldığı değerler incelenip bulunabilir. Örneğin x_3=1 diğerleri 0 yapılarak 1000000 kelimesi elde edilir ve denklem (9.25)'ten sendrom vektörünün 110 olduğu gözlenir. Bu durumda, alıcıda sendrom vektörü 110 bulunursa x_3 bitinin hatalı olduğu görülür ve o bitin tersi alınır (düzeltilir). Benzeri işlem x_2, x_1 ve x_0 bitleri için tekrar edildiğinde sırasıyla 101, 011 ve 111 sendrom vektörleri bulunacaktır. Yani alıcıda hesaplanan sendrom vektörünün

110, 101, 011 ve 111 değerleri için sırasıyla x_3, x_2, x_1 ve x_0 bitlerinin tersi alınıp hata düzeltilir. Sendrom vektörünün olası diğer değerleri için birşey yapmaya gerek yoktur. Çünkü, 000 hatasızlığı, 100, 010 ve 001 değerleri de parite bitlerinin hatalı olduğunu söyler.

(31,26) kodunu önceki örnekte incelemiştik. Benzeri şekilde, zorunlu $2^p - p - 1 \geq k$ ilişkisinin eşitlik durumunu ve en az parite bitiyle en çok mesaj biti iletimini sağlayıp (yüksek k/n) 1 bit hataları düzeltebilen (15,11), (63,57), (127,120)...vb. kodlarını sayabiliriz. Tabi, liste uzatılabilir ancak burada bir sıkıntı vardır; Hata oluşmasına sebep olacak şekilde gürültülü bir iletişim sisteminde 63 bitlik kelimelerde en fazla 1 bitin hatalı olmasını garanti edebilir miyiz? Bu soru önemli, çünkü 1'den fazla bit hatalıysa hata bu kodlarla düzeltilemez, hatta düzeltme için yapılacak işlemler dolayısı ile daha fazla hata yapılır. Örneğin, 0000000 gönderildiğinde alıcı 0001100 almış olsun. Alınan kod kelimesi geçerli olmadığından hatalı olduğunu biliyoruz. Ancak, sendrom vektörüne göre ya da Şekil 9.10'daki listeden en yakın kelime aranarak yapılan düzeltme ile 0011100 elde edilir, ki bu sonuç 3 bitin birden hatalı olmasına yol açar.

Burada Hamming kodlarının ve dolayısı ile üretici matrislerinin tekil olmadığını hatırlatalım. Yani, örneğin (15,11) kodu için çok sayıda çözüm bulunabilir. Denklem (9.25)'in öncesinde, parite değerlerinin hata durumlarına atanması işleminde aslında yaptığımız, parite bitlerinin döngüsel permütasyonlarını çok sayıda 1 içerenlerden başlayarak mesaj bitlerine atamak idi. Burada (15,11) kodu için bir örnek yapalım. (15,11) kodunda 4 parite biti var. Parite vektörü 1111'den başlayarak sırasıyla x_i bitlerindeki hataları göstermek üzere atayalım. Örneğin 1111 değeri x_0'daki hatayı göstersin. 1111'in başka permütasyonu olmadığı için sıradaki 3 biti 1 olan parite permütasyonlarına, örneğin 0111'e geçelim (tabi ki 1110'dan da başlayabilirdik ve farklı ama eşdeğer kod çıkardı). Permütasyonlar sırasıyla 0111, 1011, 1101 ve 1110 olur. Ardından iki biti 1 olan permütasyonları bulalım. Sistematik ilerlemek adına 0011'in döngüsel permütasyonları olarak sırasıyla 0011, 1001, 1100, 0110 bulunur ve ardından 0101 ve 1010 bulunur. Şu ana kadar toplam 11 permütasyon, yani 11 parite vektörü elde ettik (ki bunlar $2^4 = 16$ dört bitlik sayının 0 ve 1 biti 1 olanların dışında kalanlardır, yani en az 2 biti 1 olanlardır). Şimdi de bu 11 parite vektörünü istediğimiz sırayla parite matrisine atayalım (denklem (9.28)'de son satırdan ilk satıra doğru atanmış). Üretici matrisini

$$P = \begin{bmatrix} 1\,0\,1\,0 \\ 0\,1\,0\,1 \\ 0\,1\,1\,0 \\ 1\,1\,0\,0 \\ 1\,0\,0\,1 \\ 0\,0\,1\,1 \\ 1\,1\,1\,0 \\ 1\,1\,0\,1 \\ 1\,0\,1\,1 \\ 0\,1\,1\,1 \\ 1\,1\,1\,1 \end{bmatrix} \tag{9.28}$$

olmak üzere $G = [I \mid P]$ ile

$$G = \begin{bmatrix} 1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,1\,0 \\ 0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,1 \\ 0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,0 \\ 0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,1\,1\,0\,0 \\ 0\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,1\,0\,0\,1 \\ 0\,0\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,1\,1 \\ 0\,0\,0\,0\,0\,0\,1\,0\,0\,0\,0\,1\,1\,1\,0 \\ 0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0\,1\,1\,0\,1 \\ 0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,1\,0\,1\,1 \\ 0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,1\,1\,1 \\ 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,1\,1\,1 \end{bmatrix} \tag{9.29}$$

şeklinde yazabiliriz. 11 mesaj biti içeren herhangi bir 15 bitlik kod vektörünü $(xG)^T$ ile bulabiliriz. Örneğin x=10011000000 mesajını temsil edecek kod $g_{10} + g_7 + g_6$ modulo toplamından c_x=100110000001111 elde edilir. Yani, doğrusal kod olduğundan, G matrisinin istenilen sayıdaki satırının toplamı yine bir kod vektörüdür. G matrisi üzerinde yapılacak elemanter satır/sütun işlemleri ile elde edilecek bir G_2 matrisi de doğrusal koddur.

Denklem (9.29) ile verilen kodun parite denklemleri, P matrisinden

$$\begin{aligned} p_3 &= x_{10} \oplus x_7 \oplus x_6 \oplus x_4 \oplus x_3 \oplus x_2 \oplus x_0 \\ p_2 &= x_9 \oplus x_8 \oplus x_7 \oplus x_4 \oplus x_3 \oplus x_1 \oplus x_0 \\ p_1 &= x_{10} \oplus x_8 \oplus x_5 \oplus x_4 \oplus x_2 \oplus x_1 \oplus x_0 \\ p_0 &= x_9 \oplus x_6 \oplus x_5 \oplus x_3 \oplus x_2 \oplus x_1 \oplus x_0 \end{aligned} \tag{9.30}$$

şeklinde yazılır. Alıcı tarafında sendrom vektörünün bitleri de

$$\begin{aligned} s_3 &= p_3 \oplus x_{10} \oplus x_7 \oplus x_6 \oplus x_4 \oplus x_3 \oplus x_2 \oplus x_0 \\ s_2 &= p_2 \oplus x_9 \oplus x_8 \oplus x_7 \oplus x_4 \oplus x_3 \oplus x_1 \oplus x_0 \\ s_1 &= p_1 \oplus x_{10} \oplus x_8 \oplus x_5 \oplus x_4 \oplus x_2 \oplus x_1 \oplus x_0 \\ s_0 &= p_0 \oplus x_9 \oplus x_6 \oplus x_5 \oplus x_3 \oplus x_2 \oplus x_1 \oplus x_0 \end{aligned} \tag{9.31}$$

ile hesaplanır. Herhangi bir hata yoksa $S = 0$ bulunur, aksi halde her bir tek bitlik hata için S olası diğer 15 değerden birini alır. Bir bitlik hata sonucu alabileceği değerleri aslında denklem (9.31)'den ve P matrisinden görebiliyoruz. i bitindeki hata sonucunda $S = P_i$ (P'nin i'inci satırı) olur.

$H = [P|I]$ şeklinde tanımlanan parite kontrol matrisi n bitlik kod kelimesinin hangi bitlerinin birbiriyle ilişkili olduğunu söyler, ki bu sendrom denklemlerinden farklı değildir. Örnek kodumuz için

$$H = \begin{bmatrix} 1\,0\,0\,1\,1\,0\,1\,1\,1\,0\,1\,1\,0\,0\,0 \\ 0\,1\,1\,1\,0\,0\,1\,1\,0\,1\,1\,0\,1\,0\,0 \\ 1\,0\,1\,0\,0\,1\,1\,0\,1\,1\,1\,0\,0\,1\,0 \\ 0\,1\,0\,0\,1\,1\,0\,1\,1\,1\,1\,0\,0\,0\,1 \end{bmatrix} \tag{9.32}$$

şeklinde yazılan parite kontrol matrisinin en alt satırından $x_9 \oplus x_6 \oplus x_5 \oplus x_3 \oplus x_2 \oplus x_1 \oplus x_0 \oplus p_0 = 0$ olması gerektiğini anlıyoruz. Buradan $p_0 = x_9 \oplus x_6 \oplus x_5 \oplus x_3 \oplus x_2 \oplus x_1 \oplus x_0$ yazılabilir di ki zaten öyle yaptık.

Döndürerek elde ettiğimiz sıralı permütasyonlar yerine $2^4 = 16$ seçeneğin en az iki adet 1 içerenlerini parite kelimeleri olarak istediğimiz satırlara atasaydık ne olurdu? Tabi ki bu şekilde elde edilen üretici matrisleri de geçerli olurdu. Üretici matrisinin satırlarının, ya da mesaj kelimesi bitlerinin yerleri değiştirilmiş gibi düşünün. Aynı şeyi zaten satır/sütun işlemleri ile de yapabilirdik.

9.3 Döngüsel Kodlar

Hamming kodları doğrusal blok kodlar ailesindendir. Yani, aynı kod içindeki iki kod kelimesi toplandığında da (modulo toplam) yine geçerli bir kod kelimesi elde edilir. Buna ilave olarak, bir kod içindeki herhangi bir kod kelimesi sağa ya da sola döndürüldüğünde yine geçerli bir kod kelimesi elde ediliyorsa, o zaman bu koda döngüsel kod (cyclic code) ismi verilir. Örnek olarak $C = \{000,110,101,011\}$ kodunu ele alalım. C döngüseldir, çünkü 4 kod kelimesinden hangisi sağa ya da sola döndürülürse yine C içinden bir kod kelimesi bulunur (000'a aşikar (trivial) kod kelimesi ismi verilir).

(Bu noktadan itibaren okunaklılığı arttırmak için, modulo toplamı belirtmek üzere, $\oplus$ yerine $+$ kullandık)

Döngüsel kodlardaki $c = c_{n-1}c_{n-2} \dots c_0$ kod kelimeleri

$$c(x) = \sum_{i=0}^{n-1} c_i x^i = c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + \cdots + c_0 \tag{9.33}$$

şeklinde $n-1$'inci dereceden bir polinom olarak düşünülür. $c(x)$'i x ile çarpalım ve

$xc(x) = c_{n-1}x^n + c_{n-2}x^{n-1} + c_{n-3}x^{n-2} + \cdots + c_0 x$

$xc(x) = c_{n-2}x^{n-1} + c_{n-3}x^{n-2} + \cdots + c_0 x + c_{n-1}x^n$ (x^n terimini sona aldık)

elde edelim. Ardından $c_{n-1} + c_{n-1}$ ekleyelim ($c_{n-1} + c_{n-1} = 0$).

$xc(x) = (c_{n-2}x^{n-1} + c_{n-3}x^{n-2} + \cdots + c_0 x + c_{n-1}) + c_{n-1} + c_{n-1}x^n$.

Burada $c^{(1)}(x) = c_{n-2}x^{n-1} + c_{n-3}x^{n-2} + \cdots + c_0 x + c_{n-1}$ teriminin c'nin 1 kere sola dondürülmüş halini temsil eden polinom olduğunu görüyoruz. Yani, $c^{(1)}(x) = xc(x) + c_{n-1}$. Ancak ilave olarak $p(x) = c_{n-1} + c_{n-1}x^n = c_{n-1}(x^n + 1)$ terimi var. Yani, $xc(x) = c^{(1)}(x) + c_{n-1}(x^n + 1)$, ya da

$$\frac{xc(x)}{(x^n+1)} = \frac{c^{(1)}(x)}{(x^n+1)} + c_{n-1} \tag{9.34}$$

yazılabilir. $\frac{xc(x)}{(x^n+1)}$ bölmesinin sonucu c_{n-1}, bölünemeyen kısmı (kalan) ise $c^{(1)}$'dir. Aynı işlemi, $c^{(1)}$'yi bir kez daha sola döndürmek için tekrarlamak isteyince

$\frac{x^2c(x)}{(x^n+1)} = \frac{c^{(2)}}{(x^n+1)} + c_{n-2}$ olduğunu, ve, i döndürme sayısı olmak üzere, genel olarak

$$\frac{x^i c(x)}{(x^n+1)} = \frac{c^{(i)}(x)}{(x^n+1)} + c_{n-i} \tag{9.35}$$

olduğunu görürüz. Kod kelimesi $i = n$ defa döndürülürse $c^{(n)}(x) = c(x)$ olduğu bulunur.

Döngüsel kodlarda kod kelimesi polinomlarını mesaj kelimesi polinomlarından üretmek için, $g(x)$'e üretici polinom ismi verilerek, $c(x) = m(x)g(x)$ ile üretmek isteriz. Ancak $g(x)$ polinomu $(x^n + 1)$'i tam olarak bölmelidir ki kalan olmasın. $(x^n + 1)/g(x)$ bölmesinden kalan olmaması için $g(x)$'in $(x^n + 1)$'in faktörlerinden (çarpan polinomlarından) birisi olması gerekir. Ayrıca, bir (n, k) kod için $m(x)$ $k-1$'inci dereceden olduğuna göre, $g(x)$'in de $n-k$'inci dereceden olması gerekir, ki kod polinomları $n-1$'inci dereceden olsun.

Örnek: (n, k)=(7,4) döngüsel kodu için $m(x)$ k-1=3'üncü derece, $g(x)$ ise n-k=3'üncü derece, yani $c(x) = m(x)g(x)$ polinomu da 3x3=6'ıncı derece olur. $(x^n + 1) = (x^7 + 1)$ polinomunu tam olarak bölen polinomlar $(x^7 + 1)$ polinomunu çarpanlarına ayırarak bulunabilir.

$$(x^7+1)=(x+1)(x^3+x^2+1)(x^3+x+1)$$

Burada 3'üncü dereceden 2 adet ilkel (primitive) polinom var. İlkel kelimesi bu polinomların faktörlerine ayrılamayacağı anlamına geliyor. Öncelikle doğru ayırdığımızı teyid edelim;

$(x^3+x^2+1)(x^3+x+1)=x^6+x^4+x^3+x^5+x^3+x^2+x^3+x+1=x^6+x^5+x^4+x^3+x^2+x+1$

$(x+1)(x^6+x^5+x^4+x^3+x^2+x+1)=x^7+x^6+x^5+x^4+x^3+x^2+x+x^6+x^5+x^4+x^3+x^2+x+1=x^7+1.$

Modulo toplamın 1+1=0, 1+1+1=1, $x^a+x^a=0$ ve $x^a+x^a+x^a=x^a$ özelliklerini kullandık.

Şimdi de üretici polinom olmak üzere 3'üncü dereceden ilkel polinomlardan birini seçelim, örneğin $g(x)=x^3+x^2+1$ olsun. Herhangi bir mesaj kelimesinin 3'üncü dereceden polinom ifadesini $g(x)$ ile çarparak kod polinomunu elde ederiz. Örneğin 1100'ı temsil eden x^3+x^2 polinomu x^3+x^2+1 ile çarpılınca kod kelimesi olarak $(x^3+x^2)(x^3+x^2+1)=x^6+x^5+x^3+x^5+x^4+x^2=x^6+x^4++x^3+x^2$'nin temsil ettiği 1011100'ı buluruz. Tabi ki olası tüm mesaj kelimeleri için bu işlemleri tekrar ederek tüm kod kelimelerini bulabiliriz. Ancak sistem doğrusal olduğu için sadece taban mesaj kelimeleri (1000, 0100, 0010, 0001) için kod kelimelerini bularak üretici matrisini yazabiliriz;

1000 için $x^3(x^3+x^2+1)=x^6+x^5+x^3$ ile 1101000'ı

0100 için $x^2(x^3+x^2+1)=x^5+x^4+x^2$ ile 0110100'ı

0010 için $x(x^3+x^2+1)=x^4+x^3+x$ ile 0011010'ı ve

0001 için $1(x^3+x^2+1)=x^3+x^2+1$ ile 0001101'ı buluruz (kod kelimelerinin birbirlerinin kaymışı olduğuna dikkat ediniz). Üretici matrisi de

$$G=\begin{bmatrix}1&1&0&1&0&0&0\\0&1&1&0&1&0&0\\0&0&1&1&0&1&0\\0&0&0&1&1&0&1\end{bmatrix} \tag{9.36}$$

şeklinde yazılabilir. Matrisin sistematik olmadığını görüyoruz. Ancak, elementer satır/sütun işlemleriyle sistematik hale getirilebilir;

$$G=\begin{bmatrix}1&1&0&1&0&0&0\\0&1&1&0&1&0&0\\0&0&1&1&0&1&0\\0&0&0&1&1&0&1\end{bmatrix}=\begin{bmatrix}1&0&1&1&1&0&0\\0&1&1&0&1&0&0\\0&0&1&1&0&1&0\\0&0&0&1&1&0&1\end{bmatrix}=\begin{bmatrix}1&0&0&0&1&1&0\\0&1&0&1&1&1&0\\0&0&1&1&0&1&0\\0&0&0&1&1&0&1\end{bmatrix}=\begin{bmatrix}1&0&0&0&1&1&0\\0&1&0&0&0&1&1\\0&0&1&0&1&1&1\\0&0&0&1&1&0&1\end{bmatrix}. \tag{9.37}$$

Sistematik üretici matrisinin parite bitleri kısmının (sağdaki 4x3'lük kısım) daha önce denklem (9.26)'bulduğumuz üretici matrisindekilere ne kadar benzediğini kontrol edebilirsiniz. Aynısı da çıkabilirdi. Üretici polinomu verilen bir kodun üretici matrisini hiç işlem yapmadan da yazabiliriz. Örneğin $g(x)=(x^3+x+1)$ (diğer ilkel polinom) olsun. Bu durumda G matrisinin ilk satırı 1011000, ikinci satırı ise bunun sağa kaymışı olan 0101100, üçüncüsü 0010110, ve dördüncü satır 0001011 olarak ezbere yazılabilir. Sistematik matris isteniyorsa da elementer satır/sütun işlemleri uygulanabilir

Ancak elementer satır/sütun işlemlerine girmeden döngüsel sistematik matris üretilebilir. Bunun için, $m(x)$ mesaj polinomu olmak üzere, $\frac{x^{n-k}m(x)}{g(x)}$ bölmesinden kalan $\rho(x)$ olsun. $c(x)=x^{n-k}m(x)+\rho(x)$'dir. (n,k) için üretici polinom olarak $g(x)=x^3+x^2+1$ verilsin. $m_3m_2m_1m_0$ mesaj kelimesini temsil eden $m(x)=m_3x^3+m_2x^2+m_1x+m_0$ polinomunun $\frac{x^{n-k}m(x)}{g(x)}$ bölmesinden kalanlar bizim parite kelimelerimizi temsil eden polinomlar olacaktır. x^{n-k} ile çarpımın $g(x)$'e bölümü sonucu

$$\frac{x^{n-k}m(x)}{g(x)}=m_3x^3+(m_3+m_2)x^2+(m_3+m_2+m_1)x+m_3+m_2+m_1+m_0+\frac{\rho(x)}{g(x)}$$

bulunur. $\rho(x)$ ise $(m_3+m_1+m_0)x^2+(m_3+m_2+m_1)x+(m_2+m_1+m_0)$ olarak şekillenir. Yani sistematik kod polinomları

$$c(x)=m_3x^6+m_2x^5+m_1x^4+m_0x^3+(m_3+m_1+m_0)x^2+(m_3+m_2+m_1)x+(m_2+m_1+m_0)$$

şeklinde, buradan da sistematik üretici matrisi

$$G=\left[\begin{array}{cccc|ccc}1&0&0&0&1&1&0\\0&1&0&0&0&1&1\\0&0&1&0&1&1&1\\0&0&0&1&1&0&1\end{array}\right] \tag{9.38}$$

olarak yazılır, ki satır/sütun işlemleriyle de bunu bulmuştuk (denklem (9.37)). $(m_3+m_1+m_0)x^2$'nin $[1\ 0\ 1\ 1]^T$ kolon matrisi (diğerleri de) şeklinde yazıldığına dikkat edelim. G matrisi sistematik doğrusal ve döngüsel bir kodlamayı temsil ediyor. Yani olası kod kelimelerinin (16 adet) tümü üretilirse her birinin bir başka kod kelimesinin döndürülmüş hali olduğu görülür (Şekil 9.11). Örneğin şekilde yukarıdan 2, 4, 7 ve 14'üncü kod kelimeleri birbirlerinin döndürülmüş halleridir.

```
0 0 0 0 0 0 0
0 0 0 1 1 0 1
0 0 1 0 1 1 1
0 0 1 1 0 1 0
0 1 0 0 0 1 1
0 1 0 1 1 1 0
0 1 1 0 1 0 0
0 1 1 1 0 0 1
1 0 0 0 1 1 0
1 0 0 1 0 1 1
1 0 1 0 0 0 1
1 0 1 1 1 0 0
1 1 0 0 1 0 1
1 1 0 1 0 0 0
1 1 1 0 0 1 0
1 1 1 1 1 1 1
```

Şekil 9.11. $g(x)=x^3+x^2+1$ döngüsel kodunda (sistematik) geçerli tüm kod kelimeleri.

Alıcı tarafında (ya da kod çözme tarafında) ise 1 bitlik hataların düzeltilmesi hedeflenir. Bunun için yine, kod kelimeleri, arasındaki en küçük uzaklık d_{min}=3 olduğundan, en yakın geçerli kod kelimesi bulunarak (d=1) hata düzeltilmiş olur. Tabi ki yine, büyük tablolarda en yakın olan kod kelimesini bulmak algoritmik olarak zorlayıcı olduğundan bir sendrom vektörü/polinomu hesaplayıp hatalı olduğuna karar verilen bitin düzeltilmesi daha çok tercih edilen yoldur. Sendrom polinomu gönderilen c kod kelimesine karşılık alınan r kod kelimesini temsil eden $r(x)$ polinomunun üretici polinomuna bölünmesinden kalandır;

$$s(x)=Rem\left\{\frac{r(x)}{g(x)}\right\} \tag{9.39}$$

Örneğini yaptığımız $g(x)=x^3+x^2+1$ üretici polinomu için

$$s(x)=(r_6+r_4+r_3+r_2)x^2+(r_6+r_5+r_4+r_1)x+(r_5+r_4+r_3+r_0) \tag{9.40}$$

olarak bulunur. Normalde, hiç hata yoksa, sendrom polinomu 0 bulunur. i'inci bitteki hata sendromunu bulmak için r_i=1 diğer bitler $r_{i\neq0}$=0 yapılarak denklem (9.40)'tan hesaplanır. Tabi ki k=4 için sadece

$r_{i=6,5,4,3}$=1 testlerini yapmak ve sendromları bulmak yeterlidir. Parite bitlerindeki hataları düzeltmeye gerek yoktur. Yani ilgilenilen sendrom vektörleri sırasıyla $r_{i=6,5,4,3}$ bitlerindeki hataları gösteren 110, 011, 111 ve 101'dir.

(n, k)=(7,4) döngüsel kodlarının bir diğeri de $g(x) = x^3 + x + 1$ ilkel polinomu ile tanımlanandır. Sistematik üretici matrisi de, ister doğrudan sistematik olmayan matrisi yazıp satır/sütun işlemleriyle sistematik hale getirerek, isterse yukarıdaki örnekteki gibi polinom hesaplarıyla,

$$G = \left[\begin{array}{cccc|ccc} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{array}\right] \tag{9.41}$$

şeklinde bulunur. Peki $(x^7 + 1)$'i tam olarak bölen 2 adet 3'üncü dereceden polinom olduğuna göre bunlardan başka bir (7,4) döngüsel kod yok mu? Tabi ki var, ama hepsi bu ikisinde uygulanan elementer satır/sütun işlemleriyle üretilenler, yani eşdeğerleridir. Ayrıca ilkel polinomların çarpımları sonucu elde edilecek polinomlar da döngüsel kod verecektir. Ama bu kodlar en az parite biti kullanan optimum kodlar olmayacaktır. Döngüsel kodlardaki bu 2 örneğimiz ile Hamming kodları arasındaki benzerlik ilgimizi çekmiştir. Haberleşme sistemlerindeki kullanımları açısından bakıldığında aslında bir fark yok, sadece P matrisinin satırları yer değiştirmekte, ki bu da mesaj bitlerinin yerlerinin değiştirilmesiyle eşdeğer. Bu durumda, 1 bitlik hata düzelten (n, k) kod için $k!$ farklı bit yerleşimi ya da P satırlarının $k!$ farklı yerleşimi sözkonusu olup hata düzeltme konusunda hepsi eşdeğerdir. Ayrıca $k!$ seçenekte yapılacak satır/sütun işlemleriyle de yine eşdeğer ama sistematik olmayan kodlar üretilebilir.

Benzeri şekilde (15,11) döngüsel kodu için $(x^{15} + 1)$'i faktörlerine ayırırsak

$$(x^{15} + 1) = (x + 1)(x^2 + x + 1)(x^4 + x + 1)(x^4 + x^3 + x^2 + x + 1)(x^4 + x^3 + 1)$$

çarpımında 3 adet 15-11=4'üncü dereceden polinom olduğunu görürüz. Yani temel olarak 3 adet sistematik döngüsel kod üretebiliriz. Ama sistematik olma şartı yoksa, bunlar üzerindeki elementer işlemlerle istediğimiz kadar kod üretebiliriz.

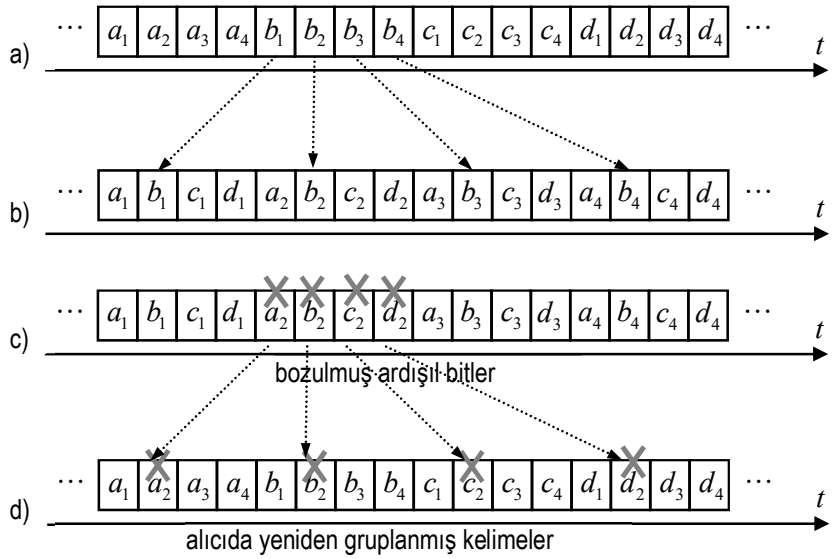
d_{min}=3 alarak 1 bitlik hataları düzeltelim ama kod kelimelerinde olası 2 hatanın da farkına varalım istiyorsak bölüm başında anlattığımız ilave bir düz parite biti ekleyebiliriz. Hata düzeltme kodlarının tümünü böyle bir haberleşme kitabında hakkıyla içermek mümkün değil. Reed-Solomon, Turbo code ve diğer 2 düzine hata düzeltme kodlarının işleyişini anlamak için ilgili makalelere/yayınlara yönelmek gerekir. Hata düzeltme ihtiyaçları uygulamaya göre değişiklik göstermekle beraber 1 bitlik hata düzeltmenin pratikte kullanımının azaldığı gözardı edilmemelidir.

9.4 Serpiştiriciler

Haberleşme ve veri saklama sistemlerinde kelime başına en fazla 1 bitlik hata öngörülmesi kabul edilemez gibidir. Genellikle karşılaşılan çoğuşma (burst) yani çoğunlukla hatasız ilerleyen bir kanalda birdenbire çok sayıda hatanın arka arkaya oluşmasıdır. Yıldırım düşmesi anında kısa süreliğine iletişim kalitesinin düşmesi ve ardışıl bir ya da daha çok kelimenin bozulması, ya da üzerine veri kaydı yapılan CD'nin çizilmesi dolayısıyla verinin büyük çoğunluğunun sağlam kalmasına rağmen bazı ardışıl bitlerin/kelimelerin bozulması, bunlara örnektir. Veri neredeyse tamamen doğru olmasına rağmen, aynı kelimede çok sayıda hata olması verinin kullanılamaz olmasına yol açabilir. Halbuki tek kelimede çok sayıda hata yerine bu hatalar diğer kelimelere dağılsaydı düzeltilebilecekti. Bu durumda akla gelen bir çözüm, aslında bir kelimenin içinde olan bitlerin önceki ve sonraki birçok kelimeye dağıtılması, böylelikle bir kelime içindeki bitlerin farklı kelimelerin bitlerinden oluşmasını sağlamaktır. Yani bir

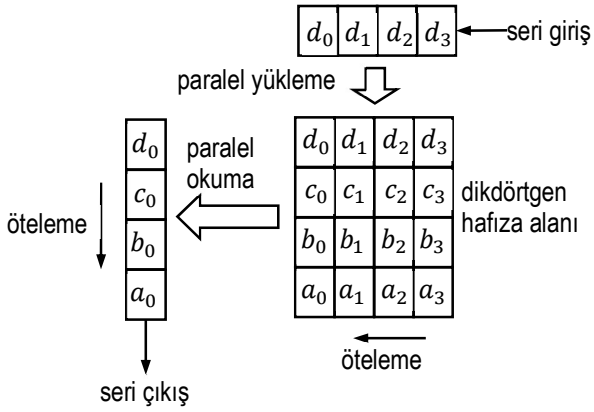
kelimede çok sayıda hata olması, bitler doğru yerlere dağıtıldığında, çok sayıda kelimede tek bitlik hata oluşması demektir, ki bunlar da düzeltilebilir. Bu işlemin adı serpiştirmedir (interleaving).

Şekil 9.12 serpiştirmedeki ana fikri açıklıyor. Çok bilinen ve anlaşılması/tasarımı kolay iki serpiştiriciyi burada açıklayalım.



Şekil 9.12. Serpiştirmenin çalışma prensibi. a) Bitleri ardı ardına gönderilen kelimeler. b) Her biti farklı kelimeye serpiştirilen veri. c) İletişim kanalında komşu birçok bitin aynı zamanda bozulması. d) Asıl yerlerine yerleştirilen bitlerin her kelimede düzeltilebilir 1 bit hataya dönüşmesi.

Dikdörtgen hafıza serpiştirici (rectangular interleaver) Şekil 9.13'teki gibi kelimelerin bir kenardan hafıza alanına kaydedilmesi ve ona dik kenardan okunması yöntemiyle çalışır. Şekilde paralel yükleme ve paralel okuma olarak işaretlenen veri yolları aslında seri de olabilir. Tabi ki okuma yapılabilmesi için yeterince verinin yüklenmesi gerekmektedir. Yükleme ile okuma arasındaki zaman farkına gecikme (latency) denir ve hafıza büyüklüğü, dolayısıyla serpme büyüklüğü ile orantılıdır. Yani çok serpmek istiyorsak çok hafıza gereklidir, dolayısı ile gecikme fazla olur.

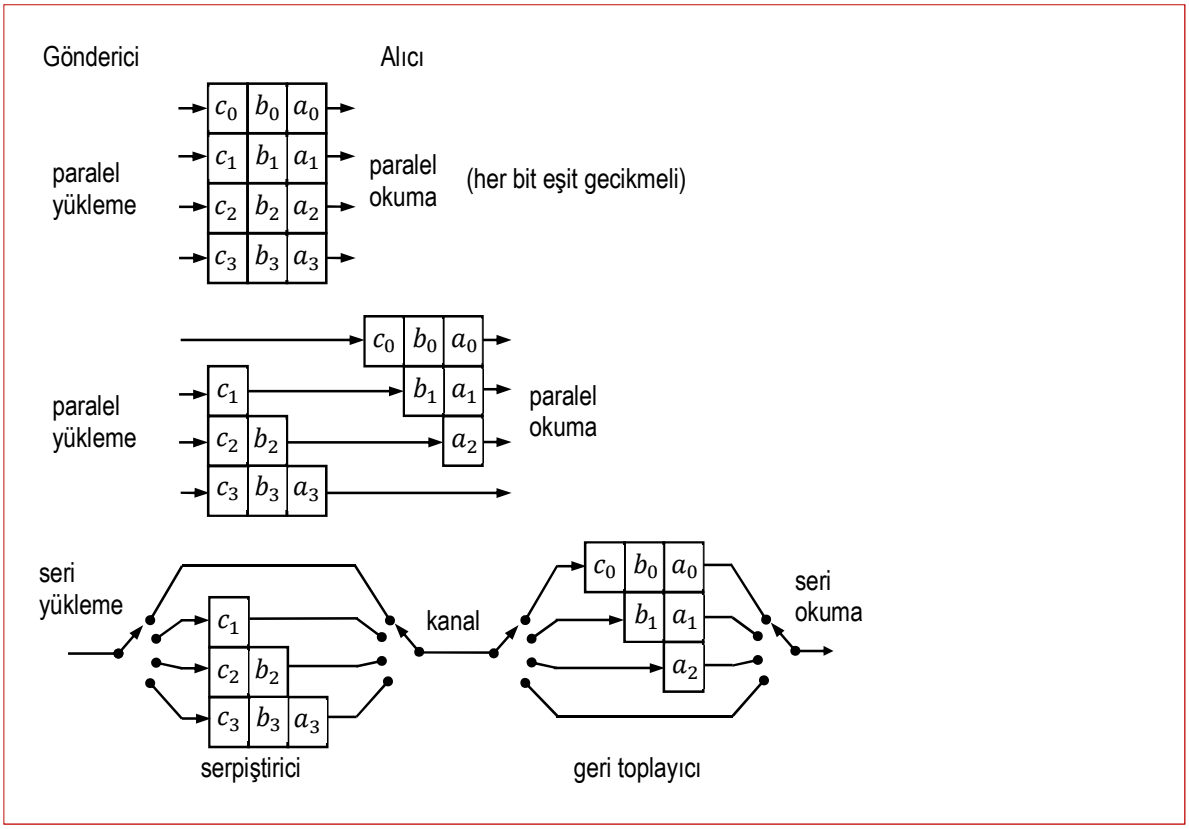


Şekil 9.13. Dikdörtgen serpiştirici örneği. 1) 1 kelime doluncaya kadar seri veri girişteki kaydırmalı yazmaca yazılır. 2) Kelimeler hafıza alanına kaydırılarak yazılır. Tüm hafıza doluncaya kadar 1 ve 2 tekrar edilir. 3) Hafıza alanı çıkış yazmacına doğru kaydırılır. 4) Çıkış yazmacındaki her çıkış kelimesi seri çıkışa kaydırılır. Hafıza alanı boşalıncaya kadar 3 ve 4 tekrar edilir.

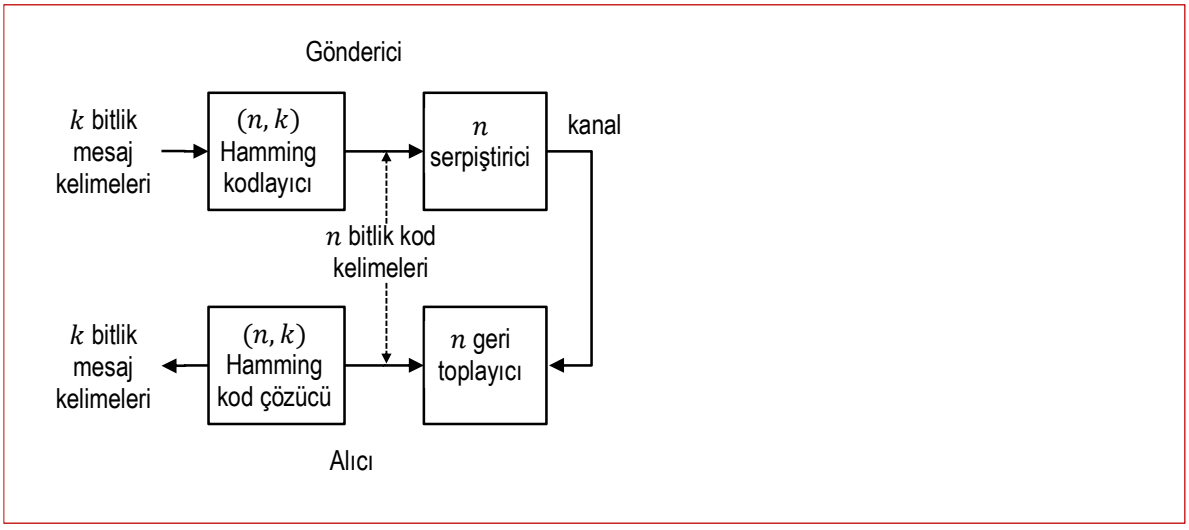
Serpiştirilmiş veriyi geri toplamak ve asıl sırasına koymak için alıcıda da bu işlemin tersi yapılır. Tabi ki hem serpiştiricide hem geri toplayıcıda, paralelleştirme ve serileleştirme için gerekli hafızayı saymazsak, kelime uzunluğu n ve serpiştirici derinliği M (kelimeye ait bitlerin dağılacağı kelime sayısı) olmak üzere, nM büyüklüğünde hafıza gereklidir. Hafızayı doldurduktan sonra göndericinin beklemesi gerekir. Alıcı tarafında da hafızanın boşaltılması sırasında iletişimin beklemesi (yeni veri gelmemesi) gereklidir. Bekleme önemli bir zorluk olduğundan, 2 adet nM hafıza kullanarak bunun üstesinden gelinebilir. Gönderici ilk hafızayı doldurduktan sonra o boşalıncaya kadar ikinci hafızayı doldurur. Yani hafızalar ping-pong misali dönüşümlü olarak serpiştirmede kullanılır. Aynı işlem alıcı tarafında da yapılır ve kesintisiz (ama gecikmeli) iletişim gerçekleşir. Böylece toplam $4nM$ hafıza ve $2nM$ gecikme ile iletişim sağlanmış olur.

Şekil 9.14 ise dikdörtgen hafıza alanını alıcı ve verici arasında bölüştürerek hem hafıza ihtiyacını azaltıyor hem de serpiştirici başarımını değiştirmiyor. Ayrıca, ping-ponk kullanımlı iki hafıza alanına ihtiyaç duymadan bekleme durumlarının önüne geçmiş oluyor. Karşılığında ise çoğullayıcılar var. Tabi ki çoğullayıcıları paralel-giriş/seri çıkış ötelemeli kaydediciler ile gerçekleştirmek mümkün. Amaç, veri kelimelerinin her bitine farklı gecikmeler uygulamak ve kanal zamanında birbirlerinden uzakta olmalarını sağlamak. Yarısı göndericide yarısı da alıcıda olmak üzere bu yaklaşım toplamda $n(n-1)$ hafıza gerektiriyor. Giriş ve çıkışında çoğullayıcı bulunan üçgen hafıza alanlı serpiştiriciye a, b, c ve d kelimelerinin $a_0a_1a_2a_3b_0b_1b_2b_3c_0c_1c_2c_3d_0d_1d_2d_3$ sırasıyla gelen bitler serpiştiriciden

$\circ\circ\circ\circ\ a_0\ \circ\circ\circ\ b_0a_1\ \circ\circ\ c_0b_1a_2\ \circ\ d_0c_1b_2a_3\ \circ\ d_1c_2b_3\ \circ\circ\ d_2c_3\ \circ\circ\circ\ d_3\ \circ\circ\circ\circ$ sırasıyla çıkarlar. Burada $\circ$ karakteri a, b, c ve d kelimelerinden önceki ve sonraki kelimelerin bitlerini temsil etmektedir. Böylece aynı kelimeye ait bitler $n+1$ aralıklarla çıkışta görülüyor. Yani kanalda ardışıl n bitin hepsinde hata olsa bile, alıcı bitleri doğru kelimelere yerleştirdiğinde aslında n kelimede 1'er bit hata oluştuğundan 1 bitlik hata düzeltme yöntemleriyle düzeltilebiliyor. Şekil 9.15 (n,k) Hamming kodlayıcı ile n serpiştiricinin beraberce kullanımını özetliyor (n serpiştirici $n \times (n-1)/2$ hafıza gerektirir, alıcıda da aynısı).



Şekil 9.14. Dikdörtgen hafıza alanının iki üçgen alan haline getirilerek verici ve alıcı arasında bölüştürülmesi.



Şekil 9.15. (n, k) Hamming kodlayıcı ve n serpiştiricinin ardışıl n bitlik hatalardan korunmak için kullanılışı.