Physical Layer (Part I) Fundamentals

Based on:

- IEEE Std 802.11-2012 (Clause 18, on Moodle)
- Wikipedia
- C. Beard & W. Stallings (2016), Wireless Communication Networks and Systems, Chapter 10 – Coding and Error Control

Learning Outcomes

- Understand the sources and causes of transmission errors
- Describe error detection techniques in MAC layer
- Understand the processing of bits in Physical layer and use of scrambling
- Discuss error correction technique via convolution coding, puncturing and interleaving

Contents

- Errors In Wireless Transmission, BER and PER
- Error Detection: FCS/CRC in the MAC layer
- Processing of bits by the PHY layer
- Scrambling
- Error Correction techniques:
  - convolutional codes
- Puncturing
- Interleaving

Errors In Wireless Transmission

- Recall from week 3 that wireless transmission is characterised by a very high **Bit Error Rate** (BER) due to all types of noise, atmospheric absorption, reflection, diffraction, scattering, and multipath propagation of radio waves.
- The bit error rate, **BER**, is measured as a ratio of incorrect bits to the total number of transmitting bits.
- BER is estimated by the related probability, p_b
- It is easier to measure the related **Packet Error Rate** (PER) and related probability p_p.
- The above two rates, are linked by the following relationship:

$$p_p = 1 - (1 - p_b)^N \approx p_b N$$

where N is the size of the packet

Bit Error Rate and Packet Error rate

- For example if **BER** is: $p_b = 10^{-6}$ (one in million) and if we use the 802.11g MAC frame where

  $N = 2300 \times 8 = 18400$ bits

  then **PER** is: $p_p = 0.0182 \approx 18400 \times 10^{-6} = 0.0184 \approx 2\%$ (two packets in hundred erroneous)
- In reality the PER in 802.11 networks can be as high as 10% with the receiving power of –68dBm
- Therefore, error detection and error correction are fundamental features ensuring a satisfactory performance of the wireless link.
- **Error detection** is performed by the MAC layer and is based on FCS – CRC (Frame Check Sequence/Cyclic Redundancy Check)
- **Error correction** is performed by the PHY layer and is based on the redundant coding of information.

Error Detection: Cyclic Redundancy Check (CRC)

- Cyclic Redundancy Check (CRC) is one of the most powerful and commonly used **error-detecting** codes
- Assume that we have a k-bit binary data string D
- Given is a $(r+1)$-bit **CRC generator polynomial** (or an equivalent binary string) G
- The CRC calculation is based on a **binary division** (all additions and subtractions are **XOR operations**) of the data string D by the CRC polynomial G to arrive at the r-bit **remainder** R which is used as the CRC bits.
- The transmitted frame $\boxed{T = [D \ \ R]}$ consists of the original data D with the CRC R appended, and has $n = k + r$ bits:

Transmitted frame:

k-bit Data	*r*-bit CRC

Mathematics of the Cyclic Redundancy Check (CRC)

- The transmitted frame $\boxed{T = [D\ R]}$ has $n = k + r$ bits
- Mathematically we divide shifted D by G to get R

$$\frac{2^r D}{G} = \mathrm{Q} + \frac{R}{G}$$

$2^r D$ represents data D with r zero bits appended.

After division by G we get:

Q $-$ the quotient, which is ignored

R – the remainder, which is the CRC appended to the data

- The resulting frame to be transmitted is:

$$\boxed{T = 2^r D + R}$$

Remember that all additions are **modulo 2 additions** (XORs)

CRC at the receiver

- At the receiver the error detection process is similar:
- The received frame $T = [D\ R]$ has $n = k + r$ bits
- It is divided by the same $(r + 1)$-bit CRC polynomial G
- If the remainder from this division is zero, **no error** is detected.
- Mathematically it can be described as:

$$\frac{T}{G} = \frac{2^r D + R}{G} = \frac{2^r D}{G} + \frac{R}{G} = \mathrm{Q} + \frac{R}{G} + \frac{R}{G} = \mathrm{Q}$$

we used the fact that $R \oplus R = 0$ (XORs)

- Hence, if the received frame

$T = [D\ R]$ has **no errors**

the remainder R is zero.

a	b	$a \oplus b$
0	0	0
0	1	1
1	0	1
1	1	0

CRC example ("pencil and paper" method)

6-bit CRC string $r = 5$

to be ignored

```
                              1 1 0 1 0 1 0 1 1 0    Q
G = 1 1 0 1 0 1 /  [1 0 1 0 0 0 1 1 0 1|0 0 0 0 0]  2^r D
                 ⊕  1 1 0 1 0 1
                    -----------
                      1 1 1 0 1 1 . . . . . . . .
                      1 1 0 1 0 1
                      -----------
                          1 1 1 0 1 0 . . . . . .
                          1 1 0 1 0 1
                          -----------
                              1 1 1 1 1 0 . . . .
                              1 1 0 1 0 1
                              -----------
                                  1 0 1 1 0 0 . .
                                  1 1 0 1 0 1
                                  -----------
                                    1 1 0 0 1 0 .
                                    1 1 0 1 0 1
                                    -----------
                                      [0 1 1 1 0]    R

                   [1 0 1 0 0 0 1 1 0 1|0 1 1 1 0]   T
```

$r = 5$

5-bit CRC-check

remainder

10+5 bit transmitted frame T = [D R]

Bit Pattern versus Polynomial Notation

- The 802.11 standard uses the polynomial notation to represent the binary strings involved in CRC calculations.
- Powers of the polynomial variable, x, indicate positions of ones in the string, e.g.
- The 6-bit CRC binary string from the previous example $G = [\,1\ 1\ 0\ 1\ 0\ 1]$ can be written in polynomial notation as

$$G = x^5 + x^4 + x^2 + 1$$

- If the string represents the CRC generator, the most significant position is always one.
- In hex notation the **most significant one is assumed** and we typically write: $G = \text{0x15}$ instead of $G = \text{0x35}$

CRC — More practical algorithm

d_0

1 0 1 0 0 0 1 1 0 1	D
1 1 0 1 0 1	G
0 1 1 1 0 1	$D \oplus Gd_0$
1 1 1 0 1 1 1 0 1 | 0	$D = shl(D \oplus G)$
1 0 1 0 1	
0 1 1 1 0	
1 1 1 0 1 0 1 | 0 0 0	$shl2$
1 0 1 0 1	
0 1 1 1 1	
1 1 1 1 1 | 0 0 0 0 0	$shl2$
1 0 1 0 1	
0 1 0 1 1	
1 0 1 1 0 0 0 0 0 0	$shl2$
1 0 1 0 1	
1 1 0 0 1	
1 1 0 0 1 0 0 0 0 0	shl
1 0 1 0 1	
0 0 1 1 1	
0 1 1 1 0	R

D — data, a binary string, the **least significant** bit d_0 first

G — the generator string/polynomial.

The most significant bit is 1 hence can be assumed.

The algorithm:

If $d_0 = 1$, $D = shl(D \oplus G)$

If $d_0 = 0$, $D = shl(D)$

where shl denotes a shift-left operation

802.11 CRC/FCS

- The 802.11 standard uses the 32-bit CRC generated by the following generator polynomial $G(x) =$

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

- This can be represented as: 0x**04C11DB7** where x^{32} is assumed
- Read: http://en.wikipedia.org/wiki/Cyclic_redundancy_check
- In addition to **division** of the "calculation field" (data) by the generator polynomial, the 802.11 standard requires **complementation** of the reminder. A quote from the standard:

  As a typical implementation, at the transmitter, the initial remainder of the division is preset to all ones and is then modified by division of the calculation fields by the generator polynomial *G*(*x*).

  The ones complement of this remainder is transmitted, with the highest-order bit first, as the FCS field.

- Details will be discussed in the practical exercise with MATLAB

Implementation hints for 802.11 CRC

- Recall from the previous slides that the transmitted frame
  $T = [D \quad R]$ has $n = k + r$ bits
- We have defined:

$$\frac{2^r D}{G} = \mathrm{Q} + \frac{R}{G} \quad \text{or} \quad 2^r D = QG + R$$

$2^r D$ represents data D with $\boldsymbol{r}$ **zero bits appended.**

- In order to complement the remainder R we add a vector of r ones, $\mathbf{1}_{1:r}$

$$2^r D + \mathbf{1}_{1:r} = QG + R + \mathbf{1}_{1:r}$$
$$2^r D + \mathbf{1}_{1:r} = QG + \overline{R}$$

- The equation says that in order to get a complemented remainder $\overline{R}$ we divide $2^r D + \mathbf{1}_{1:r}$ by G, where
- $2^r D + \mathbf{1}_{1:r}$ is data **appended with** $\boldsymbol{r}$ **ones**.

Testing the correctness of the transmission

- Due to the additional inversion/complementation steps in calculating the FCS/CRC, the remainder for the **error-free** message in **not zero**.
- The quote from the standard:

  > At the receiver, the initial remainder is preset to all ones and the serial incoming bits of the calculation fields and FCS, when divided by *G*(*x*), results (in the absence of transmission errors) in a unique nonzero remainder value.

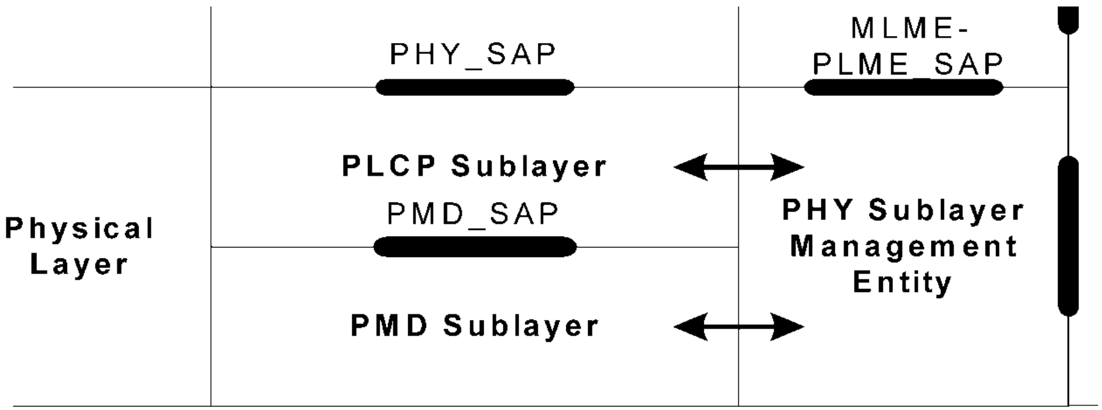
➢ The unique **correct remainder** value is the polynomial:

$$R(x) = x^{31} + x^{30} + x^{26} + x^{25} + x^{24} + x^{18} + x^{15} + x^{14} + x^{12} + x^{11} + x^{10} + x^{8} + x^{6} + x^{5} + x^{4} + x^{3} + x + 1$$

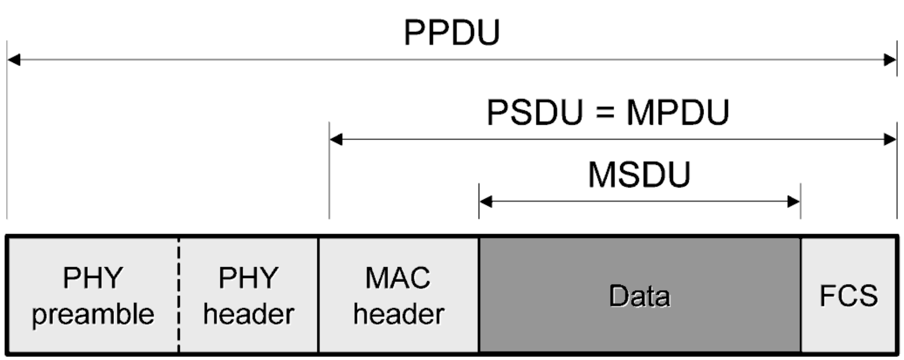
- The equivalent hex string is R = 0x**C704DD7B**

  (can you prove it?)

PHY – Physical Layer



- Physical layer (PHY) consists of two sublayers
  - PLCP (Convergence Procedure) -- processing bits of PSDU
  - PMD (Medium Dependent) -- converting bits into the OFDM (Orthogonal Frequency Division Multiplexing) symbols



Key:

PPDU = PLCP protocol data unit
PSDU = PLCP service data unit
MPDU = MAC protocol data unit
MSDU = MAC service data unit

PLCP = physical (PHY) layer convergence procedure
MAC = medium access control

PHY Frame

- The MAC frame (MPDU) with the MAC header and FCS is the PHY convergence layer data unit PSDU.
- PSDU is then encapsulated in the PPDU frame:

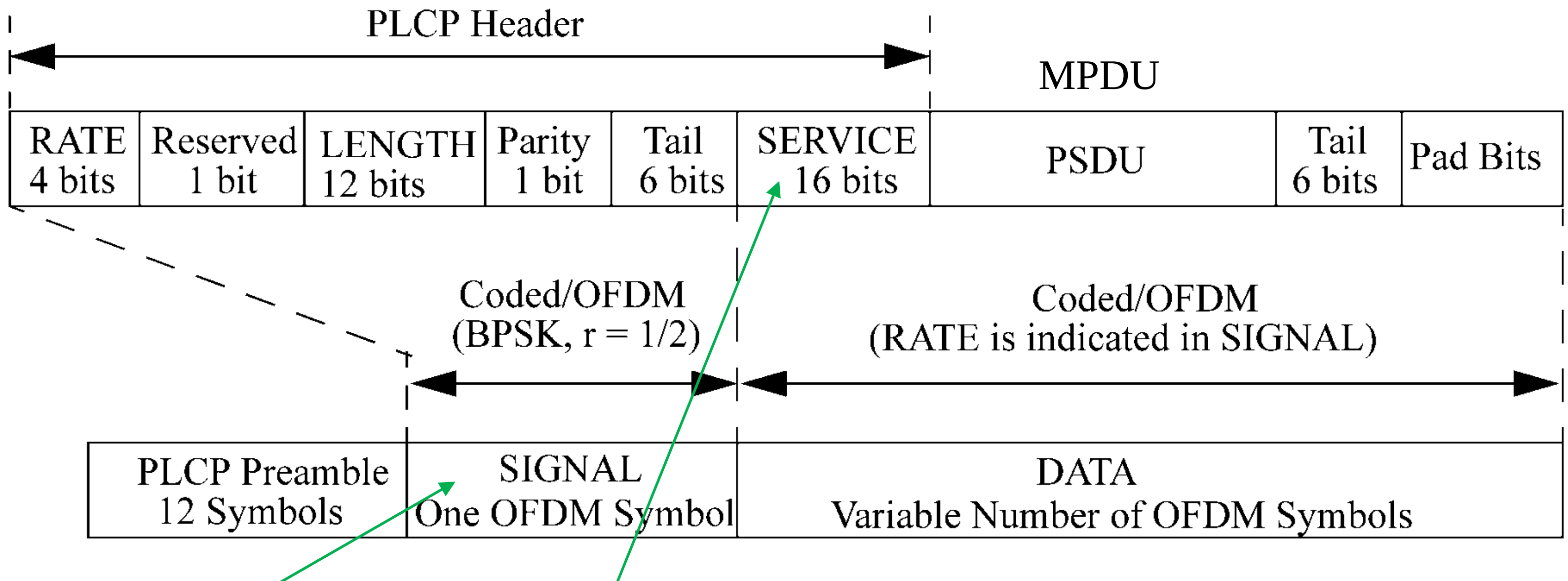


Figure 18-1—PPDU frame format

- SIGNAL (24 bits) + SERVICE (16 bits) forms the PLCP header
- PLCP Preamble of 12 **OFDM** symbols

OFDM fundamentals

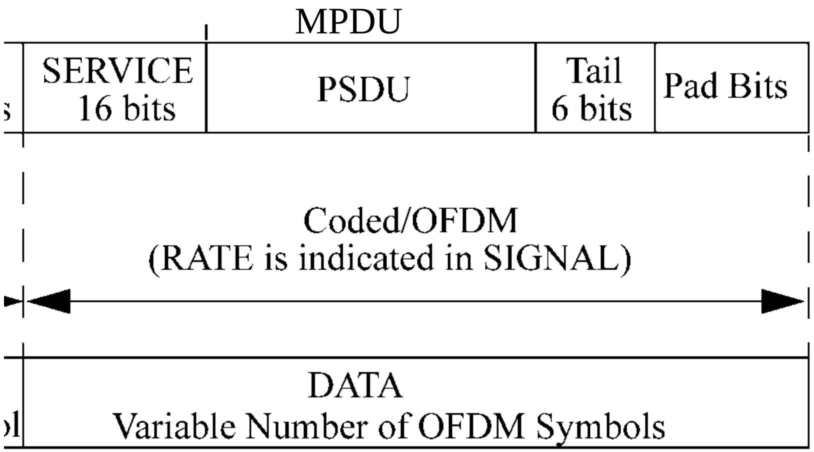
- OFDM (Orthogonal Frequency Division Multiplexing) is the method of converting a **group of bits** into sinusoidal signals called **OFDM symbols**.
- In OFDM the 20MHz frequency channel is divided into 64 frequency **subcarriers**.
- The subcarrier spacing: $\Delta_F = 20/64 = 315.5\text{kHz}$
- The **duration** of one OFDM symbol t_s (symbol interval) is the inverse of the subcarrier spacing plus time guard $t_g = 0.8\mu s$:

$$t_s = \frac{1}{\Delta_F} + t_g = \frac{64}{20} + t_g = 3.2\mu s + 0.8\mu s = 4\mu s$$

- The **number of bits per OFDM symbol** N depends on the **bit rate** R and is equal to $N = R \cdot t_s$
- For example if $R = 36\text{Mb/s}$, then

$$N = 36 \cdot 4 = 144 \textbf{ bits per OFDM symbol}$$

The PPDU encoding process



The process of encoding bits from MPDU=PSDU into PHY DATA involves a number of steps to ensure high level of transmission reliability over the un-reliable wireless medium

The encoding steps are divided in three groups:

1. Creation of the PCLP preamble and header
2. Creation the final bit stream by applying the
   - **Scrambling**,
   - **Forward correction encoding**,
   - **Puncturing** and **interleaving** steps.
3. **Generation** of OFMD signals and **modulation** to be ready for wireless transmission

Scrambling

- Scrambling is a process of encoding a binary data by XOR-ing it with a **scrambling sequence**
- Scrambling **eliminates long sequences** consisting of only '0s' or '1s’
- As a result, scrambling:
  - facilitates the correct working of the receiver, specifically its timing recovery circuit and its other adaptive circuits
  - shapes the signal's power spectrum making it more dispersed over the frequency range.

DATA scrambler and descrambler

- The PHY DATA field is scrambled with a length-127 frame-synchronous scrambler described by the following polynomial:

$$S(x) = x^7 + x^4 + 1$$

- The scrambler is implemented as a 7-bit **shift register** with feedback connections as indicated by $S(x)$

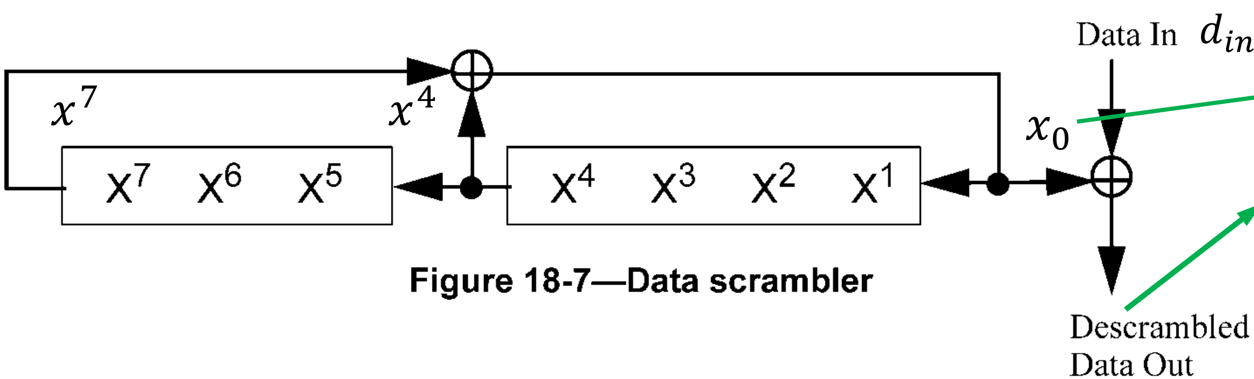


Figure 18-7—Data scrambler

$x_0 = x^4 \oplus x^7$

$d_{out}(t) = d_{in}(t) \oplus x_0(t)$

$X = shL(X, x_0)$

- The scrambler register output is calculated as: $x_0 = x^4 \oplus x^7$
- For each input bit d_{in} the output bit d_{out} is calculated as XOR with the x_0 bit from the scrambler shift register.
- Then the register is shifted left by one position

A scrambler example

- Assume that you would like to transfer a message consisting of one word: txt = **song** In hex: 73 6F 6E 67
- In a binary form the text is (least significant bit of each character first):

  txt_b = 11001110 11110110 01110110 11100110
- The **scrambler** produces the following **sequence** (if initialized with all ones):

  scr = 00001110 11110010 11001001 00000010
- The **scrambled text**: txt_s = txt_b $\oplus$ scr is:

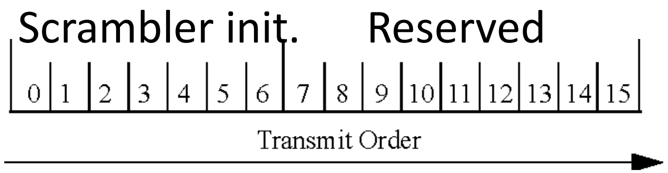
  txt_s = 11000000 00000100 10111111 11100100
- The **descrambled text**: txt_d = txt_s $\oplus$ scr is:

  txt_d = 11001110 11110110 01110110 11100110

  Is identical to the original text txt_b

More on scrambling

- The scrambler is designed in such a way that if initialised with a non-zero contents, it will produce repeatedly a **127-bit sequence** (independent of data)
- In the example we have initialised the scrambler with an all-ones state and produced a 32-bit sequence to match the length of data.
- To descramble the data we need to know the scrambling sequence, specified by the **initial contents** of the scrambler.
- In 802.11 the scrambler is **initialised to a random value** and this value is transmitted in the SERVICE field of the PLCP header.



Forward Error Correction

- CRC has an excellent ability to detect errors (99.99…%).
- Once an error is detected, typically the ARQ (Automatic Repeat reQuest), that is, re-transmission of the frame is invoked.
- ARQ is time consuming and inefficient, but unavoidable.
- **Forward Error Correction** (FEC) is based on adding redundancy to the transmitting data, hence creating a possibility of correcting erroneous data without retransmission.
- A simplistic example of FEC is to transmit each data bit 3 times, that is, 0 as 000 and 1 as 111.
- Through a noisy channel, a receiver might see 8 versions of each bit and is able to select the most likely input bit.
- Note that the **bit rate has been reduced** by the factor 1/3.
- List of available error-correcting codes
- 802.11 PHY uses two FEC encoding methods: **convolutional codes** and **LDPC** (Low-Density Parity-Check) code (in 802.11n)

Convolutional Codes

- Convolutional codes are very popular.
- Typically, each input bit, say, $u(t)$, is replaced by n output bits, $y_1(t) \dots y_n(t)$, where t is an (integer) time step number.
- Each output bit $y_i(t)$ is a sum (XOR) of the current input bit $u(t)$ and the $K-1$ past bits $u(t-1), u(t-2), \dots$
- Such a code is classified as $(1, n, K)$ code (a single bit is replaced by n bits based on K input bits)
- Note that since a single bit is replaced by n bits, the **bit rate** is reduced by the factor $R = 1/n$

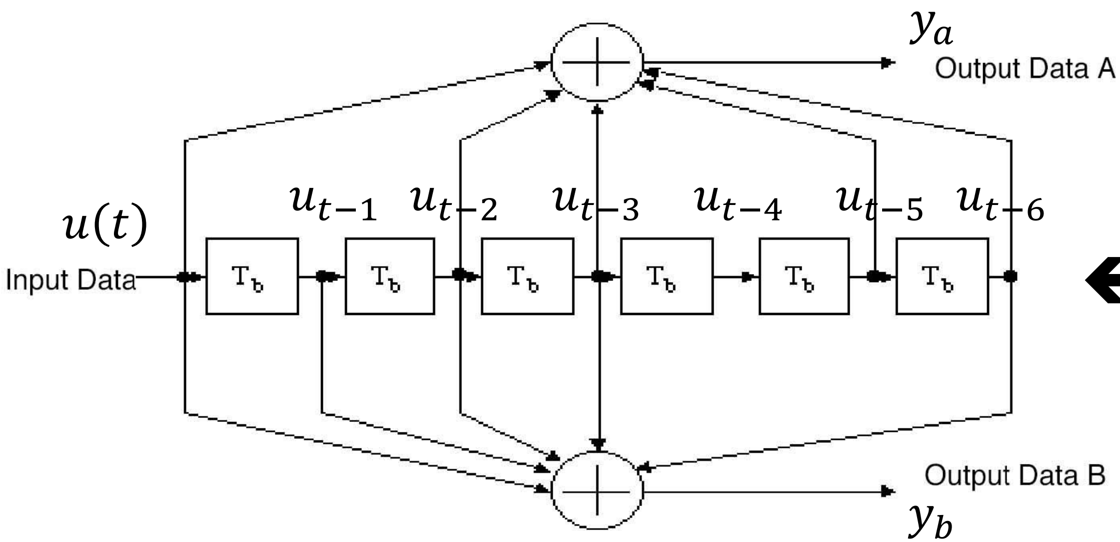
802.11g,n PHY convolutional encoder

- The convolutional code used in 802.11 PHY is (1, 2, 7), that is, a single bit is replaced by $\mathbf{2}$ bits based on $\mathbf{7}$ input bits)
- The bit rate will be reduced by the factor of 1/2
- Seven recent bits $u(t)$, … , $u(t-6)$ are used to calculate two output bits $y_a(t)$ and $y_b(t)$ that replace the current bit $u(t)$
- The input bits that are used to calculate the output bits are specified by two generator polynomials

$$g_0 = 133_8 = 1011011\,,\ g_0\,(z) = 1 + z^{-2} + z^{-3} + z^{-5} + z^{-6}$$

$$g_1 = 171_8 = 1111001\,,\ g_1\,(z) = 1 + z^{-1} + z^{-2} + z^{-3} + z^{-6}$$

The block-diagram of the convolutional encoder



← A 6-bit shift register

$$y_a(t) = u(t) \oplus u(t-2) \oplus u(t-3) \oplus u(t-5) \oplus u(t-6)$$

$$y_a = (1 + z^{-2} + z^{-3} + z^{-5} + z^{-6})u \Rightarrow 1011011 = 133_8$$

$$y_b(t) = u(t) \oplus u(t-1) \oplus u(t-2) \oplus u(t-3) \oplus u(t-6)$$

$$y_b = (1 + z^{-1} + z^{-2} + z^{-3} + z^{-6})u \Rightarrow 1111001 = 171_8$$

Example of convolutional encoding

- As for the scrambler we will encode txt = **song**
- txt_b = 11001110 11110110 01110110 11100110
- In a practical situation we would use the scrambled string, to convolve, but for simplicity we use txt_b
- The 6-bit shift register is initialised with zeros
- At each step we calculate y_a and y_b counting the number of ones on a designated positions
- In the 1st step the register contents is 1⟶ 000000 hence $y = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$
- In the 2nd step the register contents is 1⟶ 100000 hence $y = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$

Finally we have

- txt_b = 11001110 11110110 01110110 11100110

$$y = [11100001\ 11011001\ \ 01001111\ \ 11101111]$$
$$[10000000\ 11010101\ \ 11000111\ \ 00101011]$$

Or simply txt_Conv =
11101000 00000010 1111001 11001001 10111000 01011111 11010110 011101111

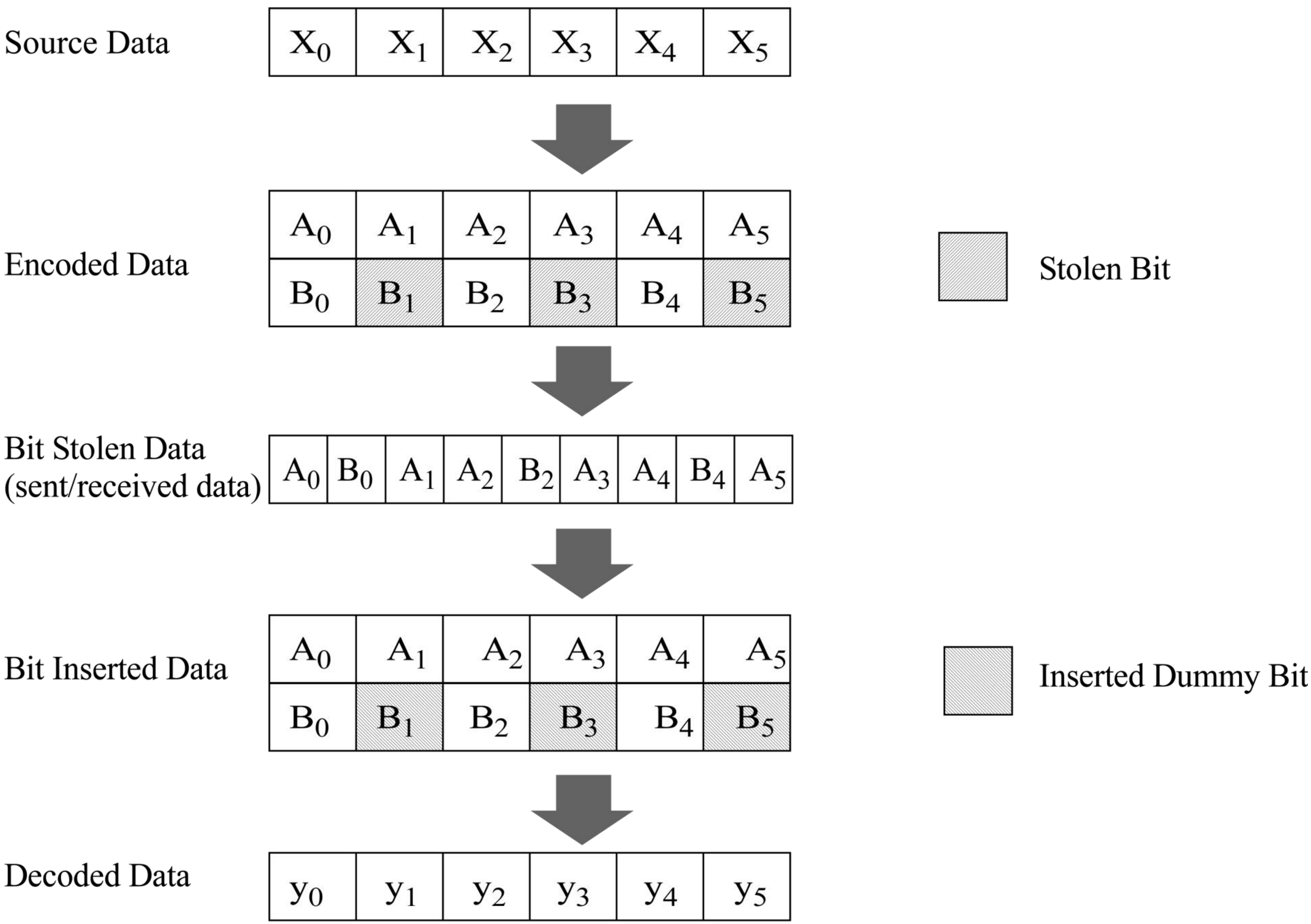
- Note that the number of bits doubled, hence the bit rate is halved.
- In order to recover the original sequence txt_b from the possibly erroneous sequence txt_cnv, we use the Viterbi algorithm.
- There is a relevant function in MATLAB that can be used.

Puncturing

- The convolutional encoding as described above results in the reduced data rate R = 1/2 (one bit replaced by two)
- It is possible to achieve higher data rates

  R = 2/3 and 3/4 using the **puncturing** process
- Puncturing is a procedure for omitting (removing) some of the encoded bits in the transmitter
- (thus reducing the number of transmitted bits and increasing the coding rate) and
- inserting a dummy "zero" bit into the convolutional decoder on the receiver side in place of the omitted bits.

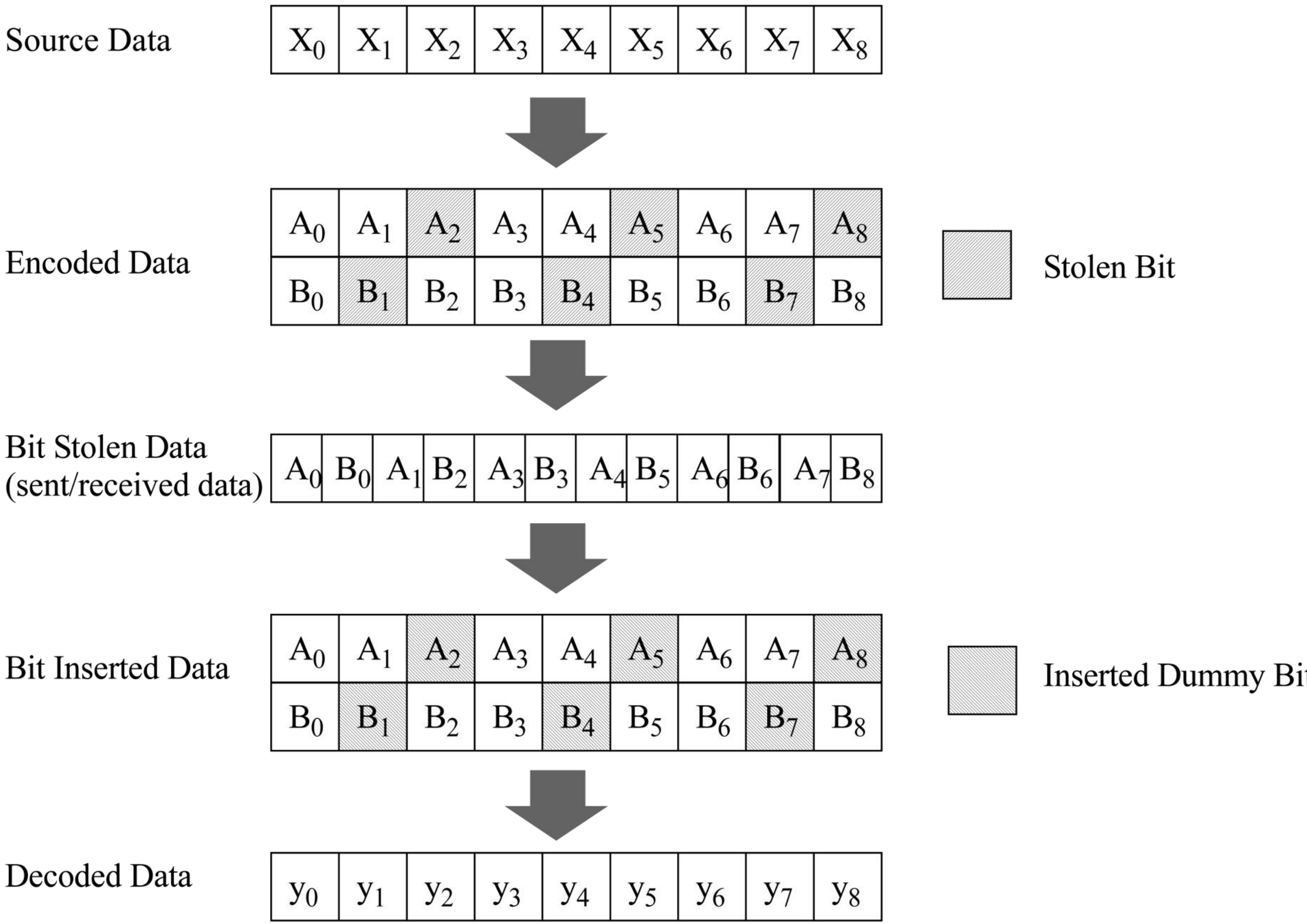
Punctured Coding (r = 2/3)



Puncturing to achieve the 2/3 bit rate

- Note from the previous drawing the we remove every forth bit (marked green)
- $A_0\ B_0\ A_1\ B_1\ A_2\ B_2\ A_3\ B_3\ A_4\ B_4\ A_5\ B_5\ A_6\ B_6 \ldots$
- If n is the original number of bits, then after convolution and puncturing as above we have
- $2n - 2n/4 = 3n/2$ bits and the bit rate is 2/3 of the original
- At the receiver, before decoding (de-convolution) dummy bits are inserted in place of the removed ones and the Viterbi algorithm is used to recover the original string

Punctured Coding (r = 3/4)



Puncturing to achieve the 3/4 bit rate

- Note from the previous drawing the we remove more bits: two out of group of six (marked green)
- A_0 B_0 A_1 B_1 A_2 B_2 A_3 B_3 A_4 B_4 A_5 B_5 A_6 B_6 ...
- If n is the original number of bits, then after convolution and puncturing as above we have
- $2n – 2n/3 = 4n/3$ bits and the bit rate is 3/4 of the original
- At the receiver, before decoding (de-convolution) dummy bits are inserted in place of the removed ones

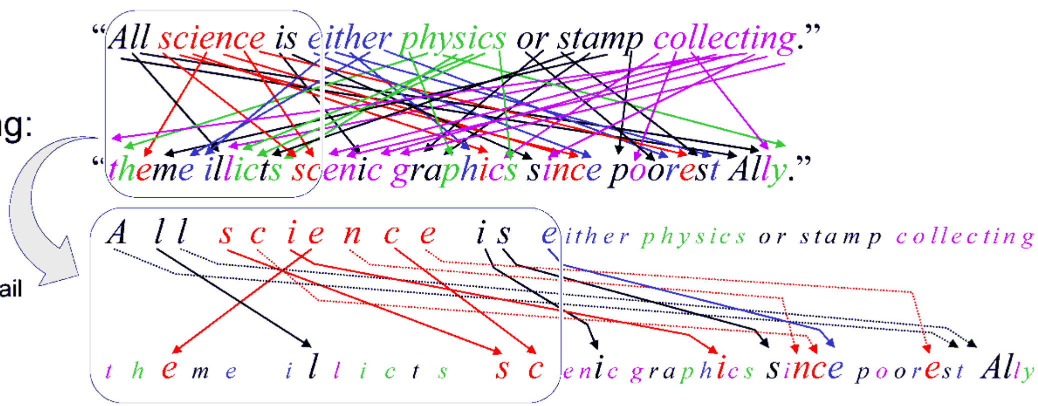
Decoding

- Decoding of convolutional encoding is typically performed by the Viterbi algorithm
- Trellis diagram
- The Viterbi algorithm

Interleaving

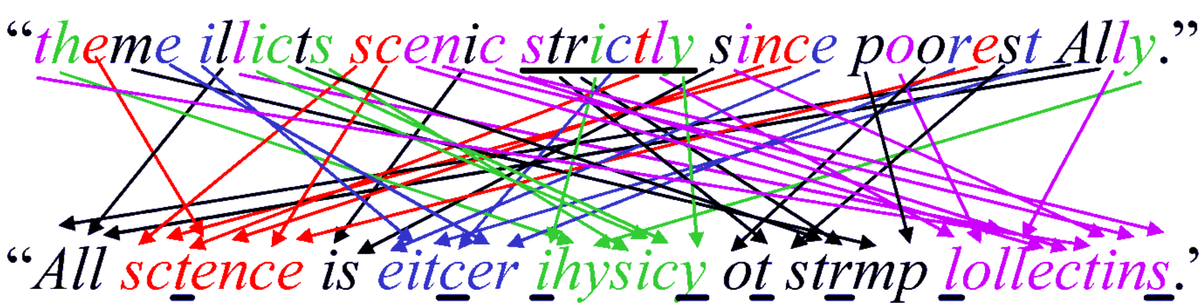
- Interleaving is the method of reducing the influence of the block errors on the final understanding/correction of the DATA by the Viterbi algorithm.
- Example:

Forward interleaving:



detail

Deinterleaving:



Assume that due to a group error, the word ”graphics” turns into “strictly” .

After deinterleaving we will get a text that we will be able to understand by applying a spell checker. The error is randomly distributed.

Data interleaving

- All encoded data bits are interleaved (permuted) by a block interleaver with a block size corresponding to the number of bits in a single OFDM symbol, N (e.g. 144 for 36Mb/s)
- The interleaver is defined by a **two-step permutation** of bits.
- The first permutation ensures that adjacent coded bits are mapped onto nonadjacent subcarriers.
- The second permutation ensures that the adjacent coded bits are mapped alternately onto less and more significant bits of the modulation constellation.
- Long runs of low reliability (LSB) bits are avoided.

Permutations (clause 18.3.5.7)

- k – the index of the coded bit before the first permutation
- i – the index after the first permutation; and
- j – the index after the second permutation, (prior to modulation mapping).
- N is the number of bits per OFDM symbol (144 for 36Mb/s)

1. The first permutation is defined by the rule

$$i = (N/16)\ (k \bmod 16) + \mathrm{floor}(k/16) \qquad k = 0,1,\ldots,N-1$$

The function floor (.) denotes the largest integer not exceeding the parameter.

2. The second permutation is defined by the rule

$$j = s \times \mathrm{floor}(i/s) + (i + N - \mathrm{floor}(16 \times i/N)) \bmod s$$

$$i = 0,1,\ldots N-1$$

The value of s is determined by the number of coded bits per subcarrier, N_B according to

$$s = \max(N_B/2, 1)$$

(e.g., 144 bits per 24 subcarriers, $N_B = 6$, $s = 3$)

The deinterleaver

- The deinterleaver, which performs the **inverse operation**, is also defined by two permutations.

1. The first permutation is defined by the inverse rule:

$$i = s \times \text{floor}(j/s) + (j + \text{floor}(16 \times j/N)) \bmod s$$

$$j = 0,1,\ldots N-1$$

2. The second permutation is defined by the inverse rule:

$$k = 16 \times i - (N-1)\,\text{floor}(16 \times i/N)$$

$$i = 0,1,\ldots N-1$$

Summary

Reflect on:

- Understand the sources and causes of transmission errors
- Describe error detection techniques in MAC layer
- Understand the processing of bits in Physical layer and use of scrambling
- Discuss error correction technique via convolution coding, puncturing and interleaving

Today's Tutorial 6 on processing bits in PHY and MAC layers in IEEE802.11 standards

Lecture 7 on IEEE802.11 PHY Part 2- from bits to signal