บทที่ 3

การเปรียบเทียบประสิทธิภาพของรหัส

3.1 การสร้างแบบจำลองการเข้ารหัสและถอดรหัส

หลังจากการศึกษาทฤษฎีต่างๆ ของรหัสควบคุมความผิดพลาดของข้อมูลแล้ว ผู้วิจัยได้เลือกรหัสตัวอย่าง เพื่อทดสอบ และเปรียบเทียบประสิทธิภาพในการสืบหาและแก้ความผิดพลาดของข้อมูล โดยสร้างแบบจำลองการเข้ารหัสและถอดรหัส เมื่อเกิดความผิดพลาดของข้อมูลขึ้น ซึ่งจากการศึกษา อัตราการเกิดความผิดพลาดของข้อมูลนั้น พบว่า การประมาณค่าโอกาสการเกิดความผิดพลาดของข้อมูล เป็นสิ่งสลับซับซ้อนและยุ่งยากมาก เพราะการเกิดความผิดพลาดของข้อมูลเป็นอิสระต่อกัน การคำนวณโอกาสเกิดความผิดพลาดของข้อมูลนั้น จึงจำเป็นต้องรู้ค่าเฉลี่ยของโอกาสที่บิตของข้อมูลจะเกิดความผิดพลาดขึ้น ซึ่งทำได้โดยการทดลองส่งผ่านข้อมูลเป็นจำนวนมาก จึงจะสามารถคำนวณค่าที่ต้องการได้ การประมาณค่าอัตราการเกิดความผิดพลาดของข้อมูล หรือ BER จึงเป็นวิธีหนึ่งที่ดีที่สุดในการตรวจสอบคุณภาพของการสื่อสารข้อมูล และระบบการเก็บข้อมูล

การพิจารณาประสิทธิภาพ ในการสืบหา และแก้ความผิดพลาดของรหัสแบบต่างๆ ในการวิจัยครั้งนี้ จึงใช้การสร้างแบบจำลองการเข้ารหัสและถอดรหัส ที่มีอัตราการเกิดความผิดพลาดของข้อมูลอยู่ระหว่าง 10^{-3} บิต ถึง 10^{-5} บิต โดยใช้แรนดอมฟังก์ชั่น (RANDOM Function) ในภาษาซี (Schildt 1988: 245) เพื่อสุ่มตำแหน่งและจำนวนบิตที่เกิดความผิดพลาด ซึ่งรหัสที่เลือกมาพิจารณาคือ

3.1.1 รหัสแบบแฮมมิ่ง

การใช้รหัสแบบแฮมมิ่ง กับข้อมูลที่จะนำไปเข้ารหัส ซึ่งเป็น 7 บิต สำหรับรหัสแอสกี (ASCII) ของตัวอักษรภาษาอังกฤษ และ 8 บิต สำหรับตัวอักษรภาษาไทย ดังนั้นจึงสร้างรหัสแบบแฮมมิ่ง ในรูปของ $(2^4-1, 2^4-1-4)$ ซึ่งคือ รหัสแฮมมิ่ง (15,11) แต่เนื่องจากบิตข้อมูลที่จะใช้ต่อ 1 ตัวอักษรเป็น 8 บิต จึงใช้หลักการจัดการรหัสเชิงเส้น กับรหัสซึ่งมีความยาวบลอกเป็น n ให้เปลี่ยนแปลงจำนวนบิตข้อมูล k ให้เพิ่มขึ้นหรือลดลงได้ ซึ่งในที่นี้ใช้การทำรหัสให้สั้นลง หรือชอทเทนนิ่ง โดยลดจำนวนบิตที่เป็นข้อมูลลง 3 บิต ซึ่งจะต้องลดมิติของเจเนเรเตอเมตริกซ์ลง ด้วยจำนวน

เดียวกันคือ 3

ดังนั้นจึงได้รหัสแฮมมิ่งขนาด (12,8) จากรูปแบบตามทฤษฎีคือ (15,11) โดยเมตริกซ์ H, G และ C แสดงได้ดังต่อไปนี้

เมื่อ n = 5, k = 11 , r = n-k = 4

ดังนั้น $H_{4x11} = [\ -P^{T}_{4x11} : I_{4x4}\ ]$

$$H = \left[\begin{array}{cccccccccccc:cccc} 1&1&1&1&1&1&1&0&0&0&0&:&1&0&0&0 \\ 1&1&1&1&0&0&0&1&1&1&0&:&0&1&0&0 \\ 1&1&0&0&1&1&0&1&1&0&1&:&0&0&1&0 \\ 1&0&1&0&1&0&1&1&0&1&1&:&0&0&0&1 \end{array}\right]$$

และ $G_{11x15} = [\ I_{11x11} : P_{11x4}\ ]$

$$G = \left[\begin{array}{cccccccccccc:cccc} 1&0&0&0&0&0&0&0&0&0&0&:&1&1&1&1 \\ 0&1&0&0&0&0&0&0&0&0&0&:&1&1&1&0 \\ 0&0&1&0&0&0&0&0&0&0&0&:&1&1&0&1 \\ 0&0&0&1&0&0&0&0&0&0&0&:&1&1&0&0 \\ 0&0&0&0&1&0&0&0&0&0&0&:&1&0&1&1 \\ 0&0&0&0&0&1&0&0&0&0&0&:&1&0&1&0 \\ 0&0&0&0&0&0&1&0&0&0&0&:&1&0&0&1 \\ 0&0&0&0&0&0&0&1&0&0&0&:&0&1&1&1 \\ 0&0&0&0&0&0&0&0&1&0&0&:&0&1&1&0 \\ 0&0&0&0&0&0&0&0&0&1&0&:&0&1&0&1 \\ 0&0&0&0&0&0&0&0&0&0&1&:&0&0&1&1 \end{array}\right]$$

เพราะ $C_{1x15} = I_{1x11}\ G_{11x15}$

ดังนั้น $C_{1x12} = I_{1x8}\ G_{8x12}$

เช่น $I_{1x11} = [\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ ]$

จะได้ $C_{1x15} = [\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 1 : 1\ 0\ 0\ 1\ ]$

หรือ $C_{1x12} = [\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 1 : 1\ 0\ 0\ 1\ ]$

โดย C เป็นโคดเวิร์ดที่ส่งไปยังตัวรับ และเกิดความผิดพลาดของข้อมูล โดยการสุ่มตำแหน่งและจำนวนบิตที่ผิดพลาดได้เป็นโคดเวิร์ด v เมื่อ v หรือโคดเวิร์ดที่ตัวรับได้รับ ถูกตรวจสอบความผิดพลาด ซึ่งถ้า vH^T มีค่าเป็น 0 ก็แสดงว่าข้อมูลที่ได้รับนั้นถูกต้อง แต่ถ้าไม่เป็น 0 ก็จะตรวจสอบหาตำแหน่งบิตที่เกิดความผิดพลาดได้จากบิตเพิ่ม ซึ่งจากการทดลองสลับที่คอลัมน์ของเมตริกซ์ H และ G พบว่า เมื่อ

$$H_{4x12} = \begin{bmatrix} 1\ 1\ 1\ 1\ 0\ 0\ 0\ 0 : 1\ 0\ 0\ 0 \\ 1\ 0\ 0\ 0\ 1\ 1\ 1\ 0 : 0\ 1\ 0\ 0 \\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 1 : 0\ 0\ 1\ 0 \\ 0\ 1\ 0\ 1\ 1\ 0\ 1\ 1 : 0\ 0\ 0\ 1 \end{bmatrix}$$

$$G_{8x12} = \begin{bmatrix} 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0 : 1\ 1\ 0\ 0 \\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0 : 1\ 0\ 1\ 1 \\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0 : 1\ 0\ 1\ 0 \\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0 : 1\ 0\ 0\ 1 \\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0 : 0\ 1\ 1\ 1 \\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0 : 0\ 1\ 1\ 0 \\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0 : 0\ 1\ 0\ 1 \\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1 : 0\ 0\ 1\ 1 \end{bmatrix}$$

ค่าของรหัสแบบแฮมมิ่งที่คำนวณได้ จะมีบิตในตำแหน่งที่ 1, 2, 4 และ 8 เป็นบิตเพิ่ม ส่วนบิตในตำแหน่งที่ 3, 5, 6, 7, 9, 10, 11 และ 12 เป็นบิตข้อมูล โดยบิตที่เป็นบิตเพิ่มนี้ จะทำให้ค่าดิสแทนส์ที่น้อยที่สุดของรหัสมีค่าเป็น 3 และเมื่อเกิดความผิดพลาดขึ้น ตำแหน่งของบิตที่ผิดพลาด จะได้จากการคำนวณค่า VH^T หรือแสดงได้ดังสมการ

ตำแหน่งบิตที่ผิดพลาด = $(v_3p_3 \oplus v_5p_5 \oplus v_6p_6 \oplus v_7p_7 \oplus v_9p_9$

$\oplus v_{10}p_{10} \oplus v_{11}p_{11} \oplus v_{12}p_{12})$

$\oplus (v_8 2^3 \oplus v_4 2^2 \oplus v_2 2^1 \oplus v_1 2^0)$

เมื่อ v_i = ค่าของบิตตำแหน่งที่ i ของโคดเวิร์ดที่ได้รับ

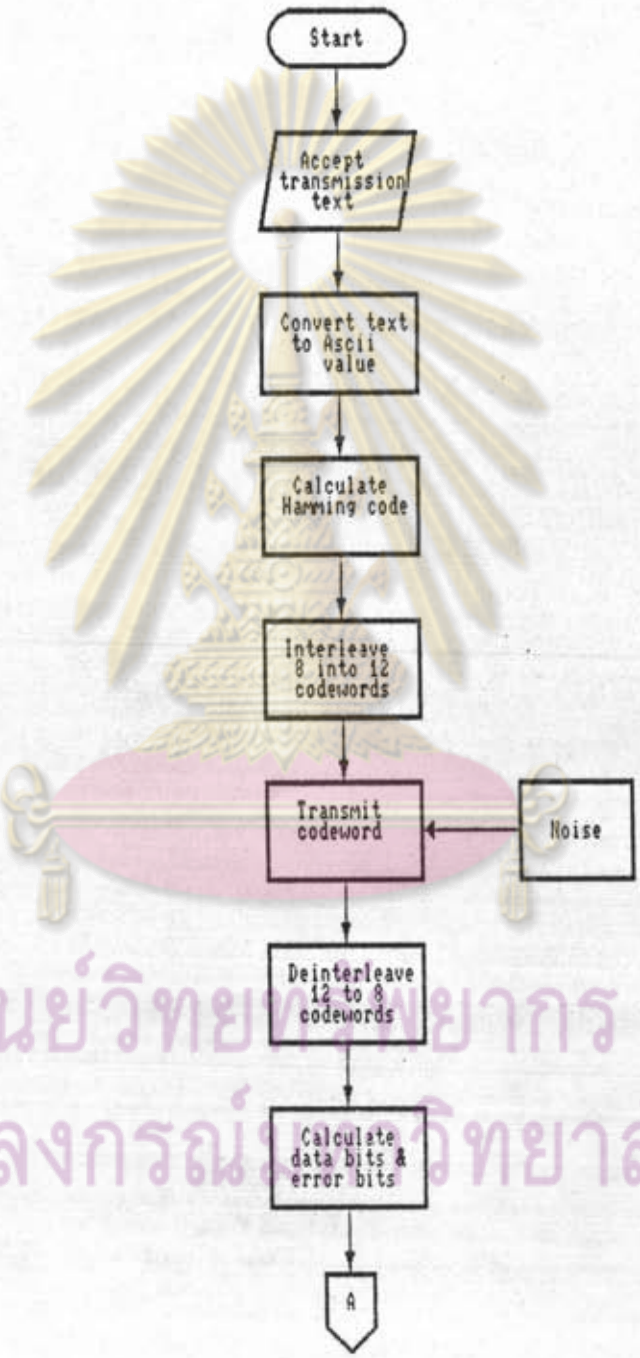
p_i = ค่าของ i ในรูปเลขฐานสอง โดย

$i = 1, 2, \ldots, 12$

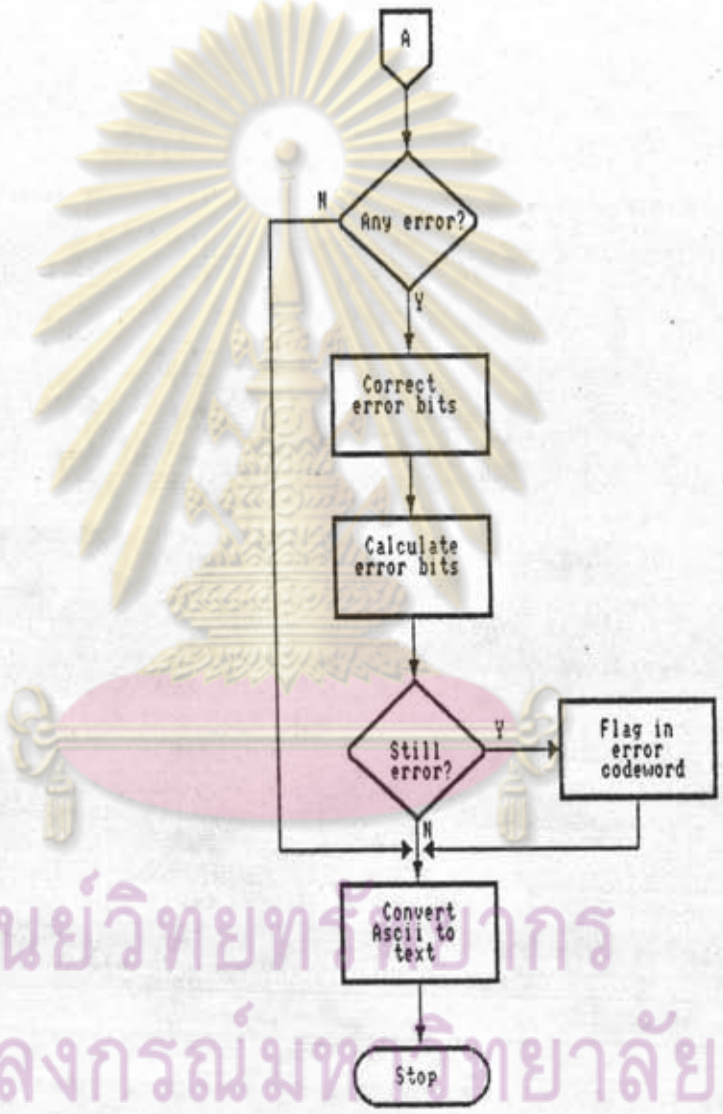
ซึ่งตำแหน่งความผิดพลาดที่ได้จาก สมการดังกล่าว จะเป็นตำแหน่งบิตที่ผิดพลาดในรูปของเลขฐานสอง และเมื่อหลังจากรีเวอร์ส (Reverse) บิตข้อมูลนั้นแล้ว ก็จะได้บิตข้อมูลที่ถูกต้อง จากตำแหน่งที่ 3, 5, 6, 7, 9, 10 และ 11 ของโคดเวิร์ด

กรณีที่เกิดความผิดพลาดมากกว่า 1 บิต ตำแหน่งบิตที่ผิดพลาดที่ได้จากการคำนวณ อาจจะมากกว่า 12 ซึ่งเป็นตำแหน่งบิตสูงสุดในโคดเวิร์ด ตัวรับก็จะทราบได้ว่า เกิดความผิดพลาดในข้อมูลขึ้น และไม่สามารถแก้ความผิดพลาดนั้นได้ แต่ถ้าตำแหน่งบิตที่ผิดพลาดที่คำนวณได้ เป็นบิตใดบิตหนึ่งในโคดเวิร์ด ซึ่งกรณีนี้ตัวรับจะรีเวอร์สบิตที่ได้จากการคำนวณว่าผิดพลาด แล้วทำการคำนวณหาตำแหน่งบิตที่ผิดพลาดใหม่ ถ้าได้ค่าที่ไม่เป็น 0 อีก ก็จะทราบได้ว่า เกิดความผิดพลาดที่ไม่สามารถแก้ให้ถูกต้องได้ ซึ่งอาจจะแฟลกไว้ที่โคดเวิร์ดนั้น ว่าเป็นโคดเวิร์ดที่ผิดพลาด แต่ถ้าการคำนวณหาตำแหน่งบิตที่ผิดพลาดใหม่นี้ ได้ค่าเป็น 0 ตัวรับก็จะได้รับข้อมูลที่ผิดพลาด โดยที่ไม่ทราบเลยว่าข้อมูลที่ได้รับมานั้นผิด

เนื่องจาก รหัสแบบแฮมมิ่งสามารถแก้ความผิดพลาดของข้อมูลได้เพียง 1 บิต ที่เกิดความผิดพลาดเท่านั้น แต่จะไม่สามารถแก้ความผิดพลาดที่เกิดขึ้นเป็นช่วงได้ ดังนั้นจึงได้นำวิธีอินเทอลีฟวิ่ง มาใช้กับรหัสแบบแฮมมิ่งด้วย ซึ่งรหัส (12,8) ที่ทำอินเทอลีฟ จะได้เป็นรหัส (96,64) โดยการนำ 8 โคดเวิร์ดมารวมกันแล้วจัดเรียงตำแหน่งของแต่ละโคดเวิร์ดเสียใหม่ ทำให้ได้รหัสที่สามารถแก้ความผิดพลาดที่ต่อเนื่องได้ถึง 8 บิต เนื่องจากเมื่อตัวรับ นำรหัสที่ได้มาทำดีอินเทอลีฟแล้ว ความผิดพลาดเป็นช่วงนี้ ก็จะถูกแบ่งออกเป็นช่วงสั้นๆ เพื่อให้ตัวรับสามารถจัดการกับโคดเวิร์ดที่ผิดพลาดนั้นได้ โดยรหัสแบบแฮมมิ่งนี้ สามารถสืบหาความผิดลาดของข้อมูลได้ 2 บิตใน 12 บิต และแก้ความผิดพลาดของข้อมูลได้ 1 บิต ใน 12 บิต แสดงได้ดังผังงานที่ 3.1



รูปที่ 3.1 แสดงผังงานแบบจำลองการเข้ารหัสและถอดรหัสแบบแฮมมิ่ง



รูปที่ 3.1 แสดงผังงานแบบจำลองการเข้ารหัสและถอดรหัสแบบแฮมมิ่ง (ต่อ)

3.1.2 รหัสแบบไซคลิก

รหัสแบบไซคลิกที่ได้นำมาทดสอบนี้ เป็นรหัส (16,9) โดยมีเจเนเรเตอไพลีโนเมียล คือ $x^6+x^5+x^4+x^3+1$ ซึ่งรหัสที่ได้จะมีบิตในตำแหน่งที่ 8 ถึง 16 เป็นบิตข้อมูล และบิตในตำแหน่งที่ 1 ถึง 7 เป็นบิตเพิ่ม โดยบิตข้อมูลจำนวน 8 บิต หรือ $M(x)$ จะถูกคูณด้วย x^7 เพื่อให้ตำแหน่งของบิตข้อมูล ไปอยู่ในตำแหน่งที่สูงสุดของรหัสเสียก่อน ในที่นี้คือบิตตำแหน่งที่ 15 แล้วเจเนเรเตอไพลีโนเมียล จึงจะนำไปทำการคำนวณเพื่อหาเศษเหลือที่ต้องการ โดยหาร $M(x)x^7$ ด้วย $g(x)$ ซึ่งจะได้ $Q(x)$ และเศษเหลือ คือ $R(x)$ แสดงได้ในรูปของสมการ

$$M(x)x^7/g(x) = Q(x) + R(x)/g(x)$$

การส่งผ่านข้อมูล จะส่งไปในรูปของ

$$T(x) = M(x)x^7 + R(x)$$

ซึ่งก็คือ การส่งเศษเหลือ หรือบิตเพิ่มที่ได้จากการคำนวณไปกับข้อมูลในบิตตำแหน่งที่ 1 ถึง 7 ของโคดเวิร์ด $M(x)x^7$ นั่นเอง

เช่น ข้อมูลเป็น $M(x) =$ 01100001

คูณด้วย x^7 จะได้ $M(x)x^7 =$ 0011000010000000

หารด้วย $g(x)$ จะได้ $R(x) =$ 11010

ดังนั้น $T(x) =$ 0011000010011010

จะเป็นโคดเวิร์ดที่ส่งไปยังตัวรับ และเมื่อตัวรับหารโคดเวิร์ดนี้ด้วยเจเนเรเตอไพลีโนเมียลเช่นเดียวกัน ซึ่งหากไม่เกิดความผิดพลาดของข้อมูลที่ตัวรับก็จะได้ว่า $R(x) = 0$

แต่หากไม่เป็น 0 ก็แสดงว่าข้อมูลที่ตัวรับได้รับมานั้น เกิดความผิดพลาด ซึ่งจะคำนวณหาโคดเวิร์ดที่ถูกต้องได้ โดยนำค่า $R(x)$ ไปหาตำแหน่งและจำนวนบิตที่ผิดพลาด จากตารางการคำนวณซินโดรมของรหัส

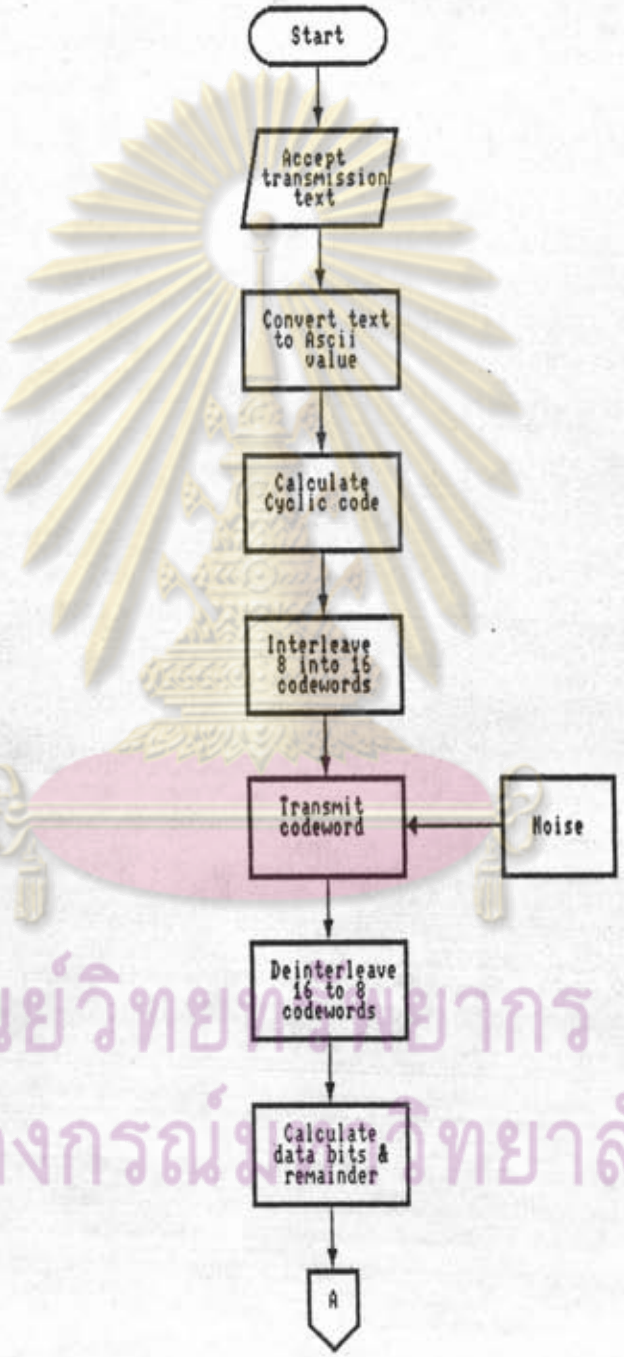
เนื่องจากความผิดพลาดที่เกิดขึ้นกับโคดเวิร์ดจะขึ้นอยู่กับรูปแบบของความผิดพลาดเท่านั้น โดยไม่ขึ้นกับบิตข้อมูล แสดงได้ดังสมการ

$$\begin{aligned} s(x) &= R_{g(x)}[v(x)] \\ &= R_{g(x)}[T(x) + e(x)] \\ &= R_{g(x)}[e(x)] \end{aligned}$$

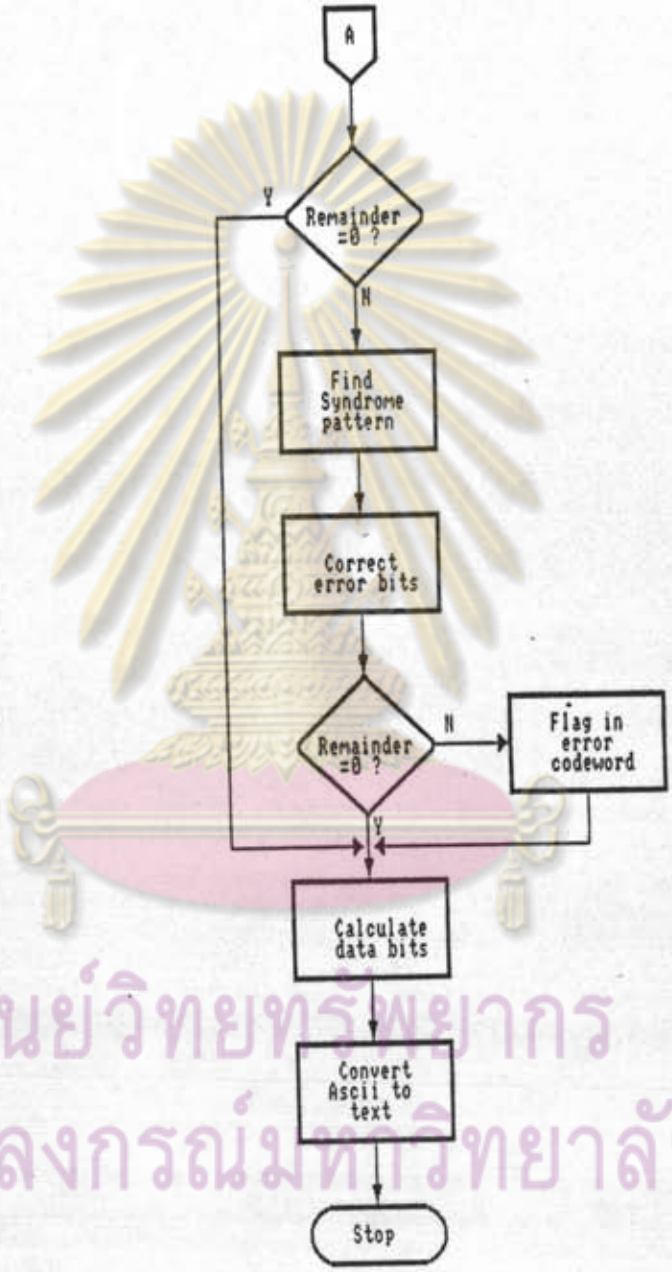
ดังนั้น ซินโดรมของแต่ละรูปแบบความผิดพลาด จึงมีได้เพียงค่าเดียว และเมื่อตัวรับรีเวิร์สบิตที่ผิดพลาดแล้ว ก็จะได้บิตข้อมูลที่ถูกต้องในตำแหน่งที่ 8 ถึง 16 ของโคดเวิร์ดที่ได้รับ

ยกเว้น กรณีรูปแบบความผิดพลาดที่เกิดขึ้น เกินความสามารถของรหัสที่จะแก้ให้ถูกต้องได้ ก็อาจจะหารูปแบบความผิดพลาดนั้นไม่พบ ในตารางการคำนวณซินโดรม หรืออาจจะพบ ซึ่งหลังจากรีเวอร์สบิตตามตำแหน่ง และจำนวนบิต ที่ได้จาก ตารางการคำนวณซินโดรมแล้ว เมื่อนำโคดเวิร์ดนั้น มาหารด้วยเจเนเรเตอโพลีโนเมียลอีกครั้งหนึ่ง จะพบว่า R(x) ไม่เท่ากับ 0 แสดงว่าเกิดความผิดพลาดที่ไม่สามารถแก้ให้ถูกต้องได้ในโคดเวิร์ดนั้น ซึ่งอาจจะแฟลกไว้ ว่าเป็นโคดเวิร์ดที่ไม่ถูกต้อง ซึ่งก็คือไม่สามารถแก้ความผิดพลาดในโคดเวิร์ดนั้นได้ แต่ถ้าคำนวณหาเศษเหลือแล้ว ได้ค่าเป็น 0 ในกรณีนี้ ตัวรับจะได้รับข้อมูลที่ผิดพลาด โดยไม่สามารถทราบได้เลยว่า ข้อมูลที่ได้รับมานั้นผิด

รหัสแบบไซคลิกที่ได้นำมาทดสอบนี้ สามารถแก้ความผิดพลาดในโคดเวิร์ดได้ 1 ถึง 3 บิต ที่ผิดพลาดใน 16 บิต ดังนั้นจึงได้นำวิธีอินเทอลีฟวิ่งมาใช้กับรหัสแบบไซคลิก ซึ่งรหัส (16,9) ที่ทำอินเทอลีฟ จะได้เป็นรหัส (128,72) โดยการนำ 8 โคดเวิร์ดมารวมกัน แล้วจัดเรียงตำแหน่งของแต่ละโคดเวิร์ดเสียใหม่ ทำให้ได้รหัสที่สามารถแก้ความผิดพลาดที่ต่อเนื่องได้ 8 ถึง 24 บิต ใน 128 บิต แสดงได้ดังผังงานที่ 3.2



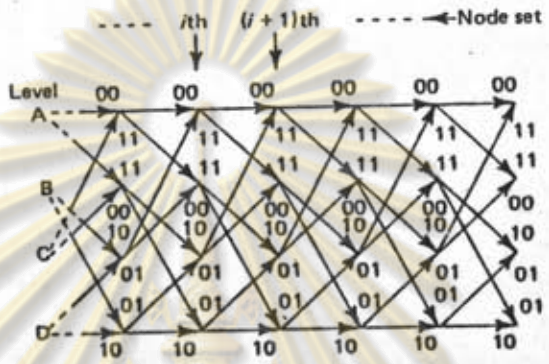
รูปที่ 3.2 แสดงผังงานแบบจำลองการเข้ารหัสและถอดรหัสแบบไซคลิก



รูปที่ 3.2 แสดงผังงานแบบจำลองการเข้ารหัสและถอดรหัสแบบไซคลิก (ต่อ)

3.1.3 รหัสแบบคอนโวลูชันนัล

รหัสแบบคอนโวลูชันนัลที่ได้นำมาทดสอบนี้ เป็นรหัส (16,8) โดยมีความยาวเฟรมของบิตข้อมูลเป็น 8 และความยาวควบคุมเป็น 3 ซึ่งจะได้โคดเวิร์ดที่มีความยาว 16 บิต โดยการเข้ารหัสและถอดรหัส แสดงได้ดังกราฟทรีลิสที่ 3.3



รูปที่ 3.3 แสดงกราฟทรีลิสของรหัสคอนโวลูชันนัล

เช่น บิตข้อมูลเป็น 01100001...

ดังนั้น บิตข้อมูลแรกคือ 0

จะเข้ารหัสเริ่มแรกที่สถานะ 1 ได้เป็นสถานะ 1 และจะได้ลำดับของ 00 เป็นบิตลำดับแรกที่ได้จากการเข้ารหัส และถูกชิฟไปยังโคดเวิร์ดเฟรม

บิตข้อมูลถัดไปคือ 1 จะถูกชิฟเข้ามาในเฟรมการเข้ารหัส

โดย จะเข้ารหัสที่สถานะ 1 ได้เป็นสถานะ 2 และจะได้ 11 เป็นลำดับของ 2 บิตถัดไปที่ได้จากการเข้ารหัส

บิตข้อมูลถัดไปคือ 1 จะถูกชิฟเข้ามาในเฟรมการเข้ารหัส

โดย จะเข้ารหัสที่สถานะ 2 ได้เป็นสถานะ 4 และจะได้ 01 เป็นลำดับของ 2 บิตถัดไปที่ได้จากการเข้ารหัส

บิตข้อมูลถัดไปคือ 0 จะถูกชิฟเข้ามาในเฟรมการเข้ารหัส

โดย จะเข้ารหัสที่สถานะ 4 ได้เป็นสถานะ 3 และจะได้ 01 เป็นลำดับของ 2 บิตถัดไปที่ได้จากการเข้ารหัส

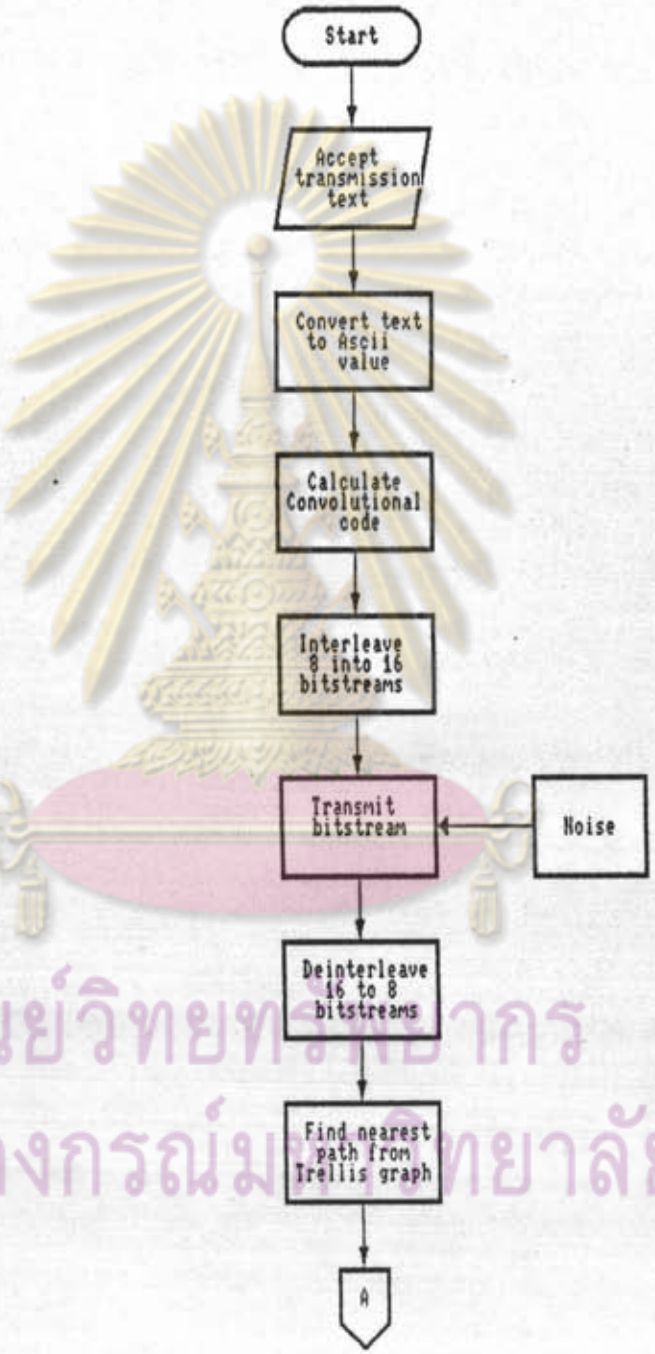
โดยการเข้ารหัสในทำนองเดียวกันนี้ ไปจนหมดลำดับของบิตในเฟรมบิตข้อมูล ก็จะได้รหัสแบบคอนโวลูชันนัลคือ 0011010111000011...

ทุกๆ 1 บิตข้อมูลที่เข้ารหัสจะได้เป็น 2 บิต โดยการเข้ารหัส

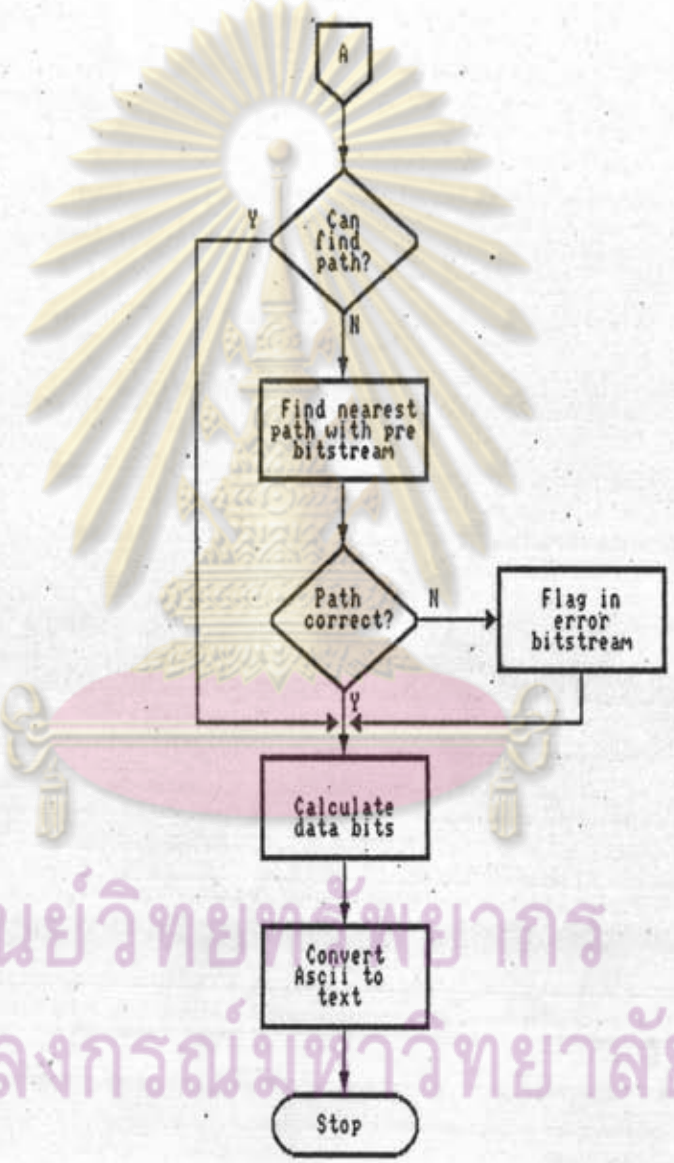
และถอดรหัส จะทำได้อย่างต่อเนื่อง ตราบเท่าที่มีข้อมูลเข้า การส่งผ่านข้อมูลจึงทำได้อย่างต่อเนื่องไปยังตัวรับ และเมื่อตัวรับได้รับข้อมูล ก็จะถอดรหัสได้ในทำนองกลับกันกับการเข้ารหัส โดยการหาเส้นทาง ที่ใกล้เคียงกันกับข้อมูลที่ได้รับมามากที่สุด คือมีค่าดิสแทนส์น้อยที่สุด เช่นเฟรมของโคดเวิร์ดที่ได้รับมาคือ 0011010111000011...
ดังนั้น 2 บิตแรกของโคดเวิร์ดเฟรมที่เริ่มการถอดรหัสคือ 00
ก็จะเริ่มการถอดรหัสที่สถานะ 1 ได้เส้นทางไปยังสถานะ 1 และจะได้บิตข้อมูลแรก ที่ถอดรหัสได้คือ 0 ซึ่งจะถูกชิฟไปยังเฟรมของบิตข้อมูลที่ได้จากการถอดรหัส
2 บิตถัดไป ของโคดเวิร์ดที่จะถูกชิฟเข้ามาในเฟรมการถอดรหัสคือ 11
ก็จะถอดรหัสที่สถานะ 1 ได้เส้นทางไปยังสถานะ 2 และได้บิตข้อมูลคือ 1
2 บิตถัดไป ของโคดเวิร์ดที่จะถูกชิฟเข้ามาในเฟรมการถอดรหัสคือ 01
ก็จะถอดรหัสที่สถานะ 2 ได้เส้นทางไปยังสถานะ 4 และได้บิตข้อมูลคือ 1
2 บิตถัดไป ของโคดเวิร์ดที่จะถูกชิฟเข้ามาในเฟรมการถอดรหัสคือ 01
ก็จะถอดรหัสที่สถานะ 4 ได้เส้นทางไปยังสถานะ 3 และได้บิตข้อมูลคือ 0
โดยการถอดรหัสในทำนองเดียวกันนี้ ไปจนหมดเฟรมของโคดเวิร์ด จะได้เฟรมของบิตข้อมูลคือ 01100001...

ในกรณีที่ไม่สามารถตัดสินได้ว่า เส้นทางใดเป็นเส้นทาง ที่ถูกต้อง ตัวถอดรหัสก็จะนำลำดับของบิตข้อมูลที่อยู่ถัดไปอีก 2 บิต มาพิจารณาร่วมด้วยว่าเส้นทางที่ถูกต้อง ควรเป็นเส้นทางใด โดยการคำนวณค่าดิสแทนส์ของทั้ง 4 บิต กับ 4 เส้นทางที่เป็นไปได้ แล้วเลือกเส้นทางที่ใกล้เคียงกับ 4 บิต ที่ตัวรับได้รับมามากที่สุด แต่หากมีความผิดพลาดเกินกว่า 1 บิต ก็อาจจะพบว่า เส้นทางที่ใกล้เคียงกันกับโคดเวิร์ดที่ได้รับมานั้นมีหลายเส้นทาง หรือเส้นทางที่ใกล้เคียงที่สุดนั้น มีค่าดิสแทนส์มากกว่า 2 ซึ่งอาจจะเกิดความผิดพลาดในการถอดรหัสได้ และหากในขั้นตอนใดของการถอดรหัสแบบคอนโวลูชันนัล เกิดความผิดพลาดที่ไม่สามารถแก้ให้ถูกต้องได้ที่ตัวรับ หรือการถอดรหัสแล้วได้ข้อมูลที่ผิด ก็จะทำให้ มีผลกระทบต่อการถอดรหัสในเฟรมถัดไปได้ด้วย

รหัสแบบคอนโวลูชันนัล ที่ได้นำมาทดสอบนี้ สามารถแก้ความผิดพลาด ในแต่ละเฟรมได้ 1 ถึง 2 บิต ที่ผิดพลาดใน 16 บิต ดังนั้น จึงได้นำวิธีอินเทอลีฟวิ่งมาใช้กับรหัสแบบคอนโวลูชันนัล ซึ่งรหัส (16,8) ที่ทำอินเทอลีฟ จะได้เป็นรหัส (128,64) โดยการนำ 8 เฟรมของโคดเวิร์ดมารวมกัน แล้วจัดเรียงตำแหน่งของแต่ละโคดเวิร์ดเสียใหม่ ทำให้ได้รหัสที่สามารถแก้ความผิดพลาดที่ต่อเนื่อง ได้ 8 ถึง 16 บิต ใน 128 บิต แสดงได้ดังผังงานที่ 3.4



รูปที่ 3.4 แสดงผังงานแบบจำลองการเข้ารหัสและถอดรหัสคอนโวลูชันนัล



รูปที่ 3.4 แสดงผังงานแบบจำลองการเข้ารหัสและถอดรหัสคอนโวลูชันนัล (ต่อ)

รหัสต่างๆ ที่ได้นำมาทดสอบนี้ มีบางกรณีที่เมื่อเกิดความผิดพลาดขึ้นแล้ว ไม่สามารถแก้ความผิดพลาดที่เกิดขึ้นได้ หรือเกิดการถอดรหัสผิด โดยที่ตัวถอดรหัสไม่สามารถตรวจสอบได้ เนื่องจากความผิดพลาดที่เกิดขึ้น เกินกว่าความสามารถของรหัส ที่จะแก้ไขให้ถูกต้องได้ ในกรณีที่ ข้อมูลที่ตัวถอดรหัสได้รับ ไม่อยู่ในช่วงเสพียร์การถอดรหัสของโคดเวิร์ดใดๆ เลย ตัวถอดรหัสก็จะไม่สามารถถอดรหัสนี้ได้ เนื่องจากดิสแทนส์ที่น้อยที่สุด ระหว่างข้อมูลที่ตัวรับได้รับ กับทุกโคดเวิร์ดที่เป็นไปได้นั้น มากกว่าค่าดิสแทนส์ที่น้อยที่สุดของรหัส ซึ่งตัวรับก็จะทราบได้ว่า ข้อมูลที่ได้รับมานั้นไม่ถูกต้อง แต่ถ้าข้อมูลที่ถูกถอดรหัสนั้น ไปอยู่ในช่วงสเพียร์การถอดรหัสของโคดเวิร์ดอื่น ก็จะเกิดการถอดรหัสผิด ซึ่งกรณีนี้ตัวรับจะได้ข้อมูลที่ไม่ถูกต้อง โดยที่ไม่ทราบเลยว่าข้อมูลที่ได้รับนั้นผิด จึงเป็นข้อจำกัดหนึ่งของการใช้รหัส

จากการทดสอบประสิทธิภาพของรหัสแต่ละชนิด ในแบบจำลองการเข้ารหัสและถอดรหัสเป็นจำนวน 2000 ครั้งต่อรหัสแต่ละชนิด และสร้างรูปแบบความผิดพลาดของข้อมูล โดยการสุ่มตำแหน่งและจำนวนบิตที่เกิดความผิดพลาด ผลการทดสอบประสิทธิภาพของรหัส แยกตามอัตราการเกิดความผิดพลาด แสดงได้ดังตารางที่ 3.1

รหัส \ BER	10^{-3}	10^{-4}	10^{-5}	อัตราเฉลี่ย
แฮมมิ่ง	99.977	99.985	99.993	99.985
ไซคลิก	99.990	99.997	99.998	99.995
คอนโวลูชันนัล	99.978	99.988	99.995	99.987

ตารางที่ 3.1

แสดงการเปรียบเทียบความสามารถแก้ความผิดพลาดของรหัสแบบต่างๆ

3.2 การเปรียบเทียบประสิทธิภาพของรหัส

จากผลการวิจัย ซึ่งได้ศึกษา และเปรียบเทียบประสิทธิภาพของรหัส 3 แบบ โดยใช้หลักเกณฑ์ในการพิจารณาประสิทธิภาพของรหัสที่นิยมใช้โดยทั่วไป

3.2.1 หลักเกณฑ์ในการพิจารณาประสิทธิภาพของรหัส

3.2.1.1 ความสามารถในการสืบหา และแก้ความผิดพลาดของข้อมูล

ความสามารถของรหัสแต่ละชนิด จะแตกต่างกันไป และโดยทั่วไป จะขึ้นอยู่กับจำนวนของบิตเพิ่ม ซึ่งมีผลโดยตรงต่อความถูกต้อง และความเชื่อถือได้ของข้อมูล โดยรหัสแบบแฮมมิ่งจะมีขีดจำกัดของบิตเพิ่ม ซึ่งจะแปรตามจำนวนของบิตข้อมูล โดยสามารถสืบหาความผิดพลาดของข้อมูลได้ 1 ถึง 3 บิตที่ผิดพลาด และแก้ความผิดพลาดได้ 1 บิตเท่านั้น ส่วนรหัสแบบไซคลิก และรหัสคอนโวลูชันนัล มีจำนวนบิตเพิ่ม ขึ้นอยู่กับเจเนเรเตอเมตริกซ์ของรหัส ซึ่งเจเนเรเตอเมตริกซ์แต่ละค่า จะให้ค่าดิสแทนส์ที่น้อยที่สุดของรหัสที่แตกต่างกัน เนื่องจากบิตเพิ่มที่มีจำนวนแตกต่างกัน

ดังนั้น หากต้องการให้รหัส มีความสามารถสืบหาและแก้ความผิดพลาดของข้อมูลได้เท่าไร ก็เลือกใช้เจเนเรเตอเมตริกซ์ของรหัส ให้เหมาะสมได้

3.2.1.2 อัตราของรหัส

อัตราของรหัสจะได้จาก จำนวนบิตในโดดเวิร์ดต่อจำนวนบิตข้อมูลในโดดเวิร์ด หรือ k/n ซึ่งรหัสที่ดี ควรมีอัตรารหัสเป็นค่าสูงมากๆ แต่ก็ขึ้นอยู่กับความต้องการ ให้รหัสสามารถสืบหา และแก้ความผิดพลาดของข้อมูล ได้เป็นจำนวนเท่าใดด้วย เนื่องจากรหัสที่มีบิตเพิ่มเป็นจำนวนมากขึ้น จะทำให้อัตราของรหัสลดลง นั่นก็คือรหัสที่มีอัตราของรหัสสูงก็จะมีจำนวนบิตเพิ่มน้อย ซึ่งจะสามารถสืบหาและแก้ความผิดพลาดของข้อมูลได้น้อยด้วย และเมื่อเปรียบเทียบกับค่าใช้จ่าย ที่ต้องใช้ในการจัดการกับบิตเพิ่มเป็นจำนวนมาก กับความถูกต้องของข้อมูลแล้ว อัตราของรหัสจึงไม่ใช่ตัวแปรที่สำคัญนัก เนื่องจากอาจใช้ความเร็วในการส่งผ่านข้อมูลให้สูงขึ้นได้

3.2.1.3 เวลาที่ใช้ในการเข้ารหัสและถอดรหัส

รหัสแต่ละชนิดต่างก็มีอัลกอริทึมที่ใช้ในการเข้ารหัสและถอดรหัส ที่แตกต่างกัน ซึ่งรหัสที่มีความสลับซับซ้อนของอัลกอริทึมมาก จะใช้เวลาในการเข้ารหัสและถอดรหัสมาก โดยจะแปรตามจำนวนบิตข้อมูลและบิตเพิ่มด้วย รหัสที่ดี

ควรจะใช้เวลาในการเข้ารหัส และถอดรหัสไม่มากนัก เพื่อให้ตัวส่ง หรือตัวรับสามารถจัดการกับการส่ง หรือรับข้อมูลของโคดเวิร์ดถัดไปได้ทัน และจะทำให้การดำเนินการเข้ารหัสและถอดรหัสเป็นไปอย่างต่อเนื่อง ไม่หยุดชงัก ทั้งนี้รวมถึงหน่วยความจำที่จำเป็นต้องใช้ สำหรับการเข้ารหัสและถอดรหัสของรหัสแต่ละชนิดด้วย เนื่องจากตัวส่งหรือตัวรับอาจมีข้อจำกัดของหน่วยความจำที่มีใช้

ดังนั้นรหัสที่ดี จึงควรสามารถทำงานได้เร็ว และสอดคล้องกับระบบการส่งผ่านข้อมูลที่ใช้

3.2.2 การเปรียบเทียบรหัสแบบบลอกและรหัสคอนโวลูชันนัล

การใช้รหัสแบบบลอก โดยข้อมูลจะถูกแบ่งออกเป็นบลอก และแต่ละบลอก จะเป็นอิสระต่อกันในการเข้ารหัสและถอดรหัส โดยตัวถอดรหัสจำเป็นต้องรู้ขอบเขตของบลอก ซึ่งการใช้พีชคณิตจะช่วยให้การนำรหัสแบบบลอกไปใช้ ทำได้ง่ายขึ้น ส่วนการใช้รหัสแบบทรีที่นิยมกันคือ รหัสคอนโวลูชันนัล และการถอดรหัสคือ อัลกอริทึมของไวเทอบิ โดยตัวถอดรหัสจะต้องรู้ลำดับของ บิตของเฟรมก่อนหน้า และเฟรมถัดไปด้วย จึงจะสามารถทำการถอดรหัสได้ ซึ่งทำให้ตัวถอดรหัสต้องจัดการกับบิตจำนวนมาก

เมื่อเปรียบเทียบประสิทธิภาพของรหัสแบบบลอก คือรหัสแบบแฮมมิ่ง (12,8) และรหัสแบบไซคลิก (16,8) กับรหัสแบบทรี คือรหัสคอนโวลูชันนัล (16,8) เมื่อรหัสมีความสามารถในการสืบหาและแก้ความผิดพลาดของข้อมูล อัตราของรหัส และเวลาที่ใช้ในการเข้ารหัสและถอดรหัสที่ใกล้เคียงกันนั้น พบว่า

3.2.2.1 ความสามารถในการสืบหา และแก้ความผิดพลาดของข้อมูลของรหัสแบบบลอกจะดีกว่า คือความสามารถในการสืบหาความผิดพลาดของรหัสแบบไซคลิกได้ 1 ถึง 6 บิต ส่วนรหัสแบบคอนโวลูชันนัล มีความสามารถสืบหาความผิดพลาดของข้อมูลได้ 1 ถึง 4 บิต ความสามารถในการแก้ความผิดพลาดของข้อมูลของรหัสแบบไซคลิกเป็น 1 ถึง 3 บิต ส่วนรหัสแบบคอนโวลูชันนัล จะแก้ความผิดพลาดของข้อมูลได้เพียง 1 ถึง 2 บิต

3.2.2.2 ข้อมูลที่ได้จากรหัสคอนโวลูชันนัล จะมีความถูกต้องต่ำ โดยหลังจากการถอดรหัสแล้วจะยังคงมีความผิดพลาดของข้อมูล อยู่ประมาณ 10^{-3} ถึง 10^{-7}บิต เมื่ออัตราการเกิดความผิดพลาดหรือ BER เป็น 10^{-3} ในขณะที่รหัสแบบบลอกหลังจากถอดรหัสแล้ว จะยังคงมีความผิดพลาดอยู่ประมาณ 10^{-10} ถึง 10^{-11} บิตเท่านั้น เนื่องจากรหัสแบบบลอกจะมีการเข้ารหัสและถอดรหัสสิ้นสุดในแต่ละบลอก หากเกิดความผิดพลาดในการถอดรหัสก็จะมีผลในโคดเวิร์ดนั้นๆ เท่านั้น แต่รหัสคอนโวลูชันนัล

จะมีผลต่อเนื่องไปไม่สิ้นสุด ขึ้นอยู่กับความยาวของโคดเวิร์ดเฟรม ดังนั้นเมื่อเกิดความผิดพลาดที่ไม่สามารถแก้ให้ถูกต้องได้ ที่บิตใดบิตหนึ่งแล้ว ก็อาจจะมีผลให้ บิตถัดไปถูกถอดรหัสผิดไปได้ ซึ่งจะเกิดขึ้นในบิตต่อไปได้อย่างไม่สิ้นสุด

3.2.2.3 การใช้รหัสคอนโวลูชันนัลนั้น ความเร็ว ที่ใช้ในการส่งผ่านข้อมูล จะต้องต่ำพอที่ตัวถอดรหัส สามารถจัดการกับบิตจำนวนมากได้ทัน ซึ่งรหัสแบบบลอกจะใช้การเข้ารหัส และถอดรหัสที่จัดการกับบิตจำนวนน้อยกว่า โดยการทดสอบพบว่า รหัสแบบแฮมมิ่งจะมีการออ (OR) บิตข้อมูล 217 ครั้ง และมีการแอน (AND) บิต จำนวน 216 ครั้ง รหัสแบบไซคลิกจะมีการออบิตจำนวน 592 ครั้ง และการแอนบิตจำนวน 784 ครั้ง ส่วนรหัสคอนโวลูชันนัลมีการออบิตจำนวน 400 ครั้ง และการแอนบิตจำนวน 528 ครั้ง โดยถ้าเปรียบเทียบจำนวนครั้งการกระทำของการเข้ารหัสและถอดรหัสของรหัสแบบบลอก กับรหัสแบบคอนโวลูชันนัลแล้ว ที่ความสามารถในการสืบหาและแก้ความผิดพลาดของข้อมูลที่ใกล้เคียงกันนั้น รหัสแบบคอนโวลูชันนัลจะมีจำนวนครั้งมากกว่า และอัลกอริทึมของรหัสแบบคอนโวลูชันนัลจะซับซ้อนมากขึ้น เมื่อจำนวนบิตที่เป็นบิตเพิ่มมีจำนวนน้อยลง เนื่องจากต้องเก็บเส้นทางของโหนดก่อนหน้าไว้เป็นจำนวนมาก สำหรับการย้อนกลับไปหาเส้นทางใหม่ ซึ่งเมื่อต้องการความถูกต้องของข้อมูลมากก็ต้องเก็บเส้นทางเหล่านี้ไว้มาก และเสียเวลาย้อนกลับไปค้นหาเส้นทางมากขึ้น เมื่อเกิดความผิดพลาดของข้อมูลขึ้น ส่วนรหัสแบบบลอกนั้น ความยุ่งยากในการถอดรหัสจะลดลง เมื่อจำนวนบิตที่เป็นบิตเพิ่มลดลง ดังนั้นเมื่อต้องการรหัสที่มีอัตรารหัสสูงๆ เช่น 14/15 จึงเหมาะที่จะใช้รหัสแบบบลอก

นอกจากนี้ การสืบหาและแก้ความผิดพลาดของข้อมูลโดยใช้รหัสแบบบลอกจะเสียค่าใช้จ่ายน้อยกว่า เนื่องจากรหัสแบบคอนโวลูชันนัล จำเป็นต้องใช้บัฟเฟอร์จำนวนมากเพื่อเก็บเส้นทางของโหนดต่างๆ ไว้ การใช้รหัสคอนโวลูชันนัลจึงไม่เหมาะกับ ระบบที่ใช้การส่งผ่านข้อมูลที่มีความเร็วสูง และต้องการความถูกต้องของข้อมูลมาก ส่วนในกรณีของรหัสแบบบลอก จะเหมาะกับการส่งผ่านข้อมูลที่มีความเร็วสูงและมีความเชื่อถือได้ของข้อมูลสูง