บทที่

9

การตรวจจับและแก้ไขข้อผิดพลาด (Error Detection and Correction)

Error Detection and Correction is the ability to detect and to correct errors caused by noise or other impairments during data transmission.

การเชื่อมต่อข้อมูล Data Link Layer (DLL) ทำหน้าที่เชื่อมต่อระหว่าง Network และ Physical Layer โดยเฉพาะอย่างยิ่ง รับผิดชอบการส่งข้อมูลระหว่างจุดต่อจุด (Hop-to-Hop) อย่างสมบูรณ์

หน้าที่ของชั้นเชื่อมต่อข้อมูล
เพื่อให้บรรลุวัตถุประสงค์ในการสื่อสาร หน้าที่รับผิดชอบของชั้นเชื่อมต่อข้อมูลมี 5 ประการดังนี้

Packetizing นิยามการจำกัดระดับการเข้าถึง (Encapsulate) ข้อมูลในรูปแบบของ Packet หรือ Cell ตามข้อกำหนดของ Protocol

Addressing นิยามกลไกการกำหนดตำแหน่งของอุปกรณ์ใน Local Network

Error Control ควบคุม (ตรวจจับ และแก้ไข) ข้อผิดพลาดของข้อมูล

Flow Control กำหนดปริมาณข้อมูลในการส่งให้สอดคล้องกับสมรรถนะในการประมวลผลของอุปกรณ์ด้านรับ เพื่อป้องกันการไหลท่วมท้นของข้อมูล

Medium Access Control (MAC) ควบคุม หรือจัดสรร การเข้าถึง (เข้าใช้) ตัวกลางของอุปกรณ์สื่อสารข้อมูล

IEEE จำแนก Protocol ชั้นการสื่อสาร Data Link ออกเป็น 2 ชั้นย่อย ได้แก่

Logical Link Control (LLC) กล่าวถึงการเชื่อมโยงข้อมูลระหว่างอุปกรณ์ภายในเครือข่าย

Media Access Control (MAC) กล่าวถึงการเข้าถึงตัวกลางการสื่อสาร

ในบทนี้จะกล่าวถึงหน้าที่ประการที่ 3 ของ DLL ได้แก่ การตรวจจับและแก้ไขข้อผิดพลาดของข้อมูล (Error Detection and Correction)

ชนิดของข้อผิดพลาด

การรบกวนที่เกิดขึ้นภายในตัวกลางระหว่างการรับส่งข้อมูล เช่น จากสัญญาณรบกวน (Noise) หรือความผิดเพี้ยน (Distortion) เป็นต้น อาจก่อให้เกิดการเปลี่ยนรูปร่างของสัญญาณ ซึ่งอาจทำให้การตีความหมายของข้อมูลที่ด้านรับผิดพลาด ซึ่งในกรณี การสื่อสารข้อมูลดิจิตอล รหัส 0 จะเปลี่ยนเป็น 1 และรหัส 1 จะเปลี่ยนเป็น 0

ข้อผิดพลาดชนิดบิตเดียว

ข้อผิดพลาดชนิดบิตเดียว (Single Bit Error) หมายถึง ความผิดพลาดที่เกิดขึ้นเพียง 1 บิต ต่อหน่วยข้อมูล 1 หน่วย (เช่น Byte Character Data Unit หรือ Packet) ดังรูป

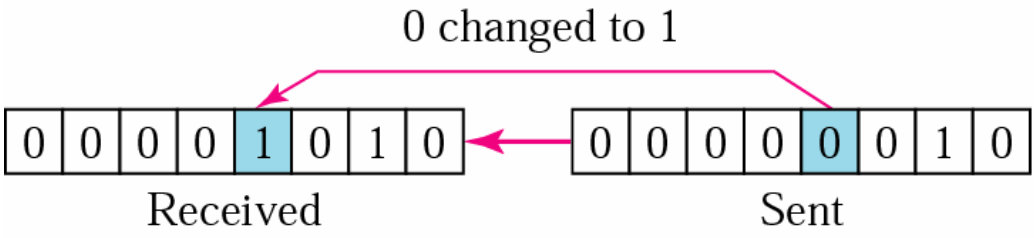


FIGURE 9.1 Single Bit Error ของข้อมูลขนาด 1 Byte โดยบิตที่ 5 เปลี่ยนจาก 0 เป็น 1

การสื่อสารข้อมูลทั่วไป มีโอกาสเกิด Error ชนิดนี้น้อยมาก เนื่องจากช่วงเวลา 1 บิตมีค่าสั้นกว่าสัญญาณรบกวนปกติ (แต่มักเกิดขึ้นกับการส่งข้อมูลแบบขนาน)

ข้อผิดพลาดชนิดหลายบิต

ข้อผิดพลาดชนิดหลายบิต (Burst Error) หมายถึง ความผิดพลาดที่เกิดขึ้นตั้งแต่ 2 บิตขึ้นไป ต่อหน่วยข้อมูล 1 หน่วย (เช่น Byte หรือ Packet) จากตัวอย่างในรูปที่ 9.2 สังเกตว่า Error ชนิดนี้ ไม่จำเป็นต้องเกิดกับบิตที่ติดกัน ความยาวของ Error วัดได้จาก บิตแรกจนถึงบิตสุดท้าย ที่เกิด Error โดยที่ข้อมูลบางบิตในช่วงนั้นอาจถูกต้องก็ได้ จำนวนบิตที่อาจเกิด Error ขึ้นกับช่วงเวลาของ Noise

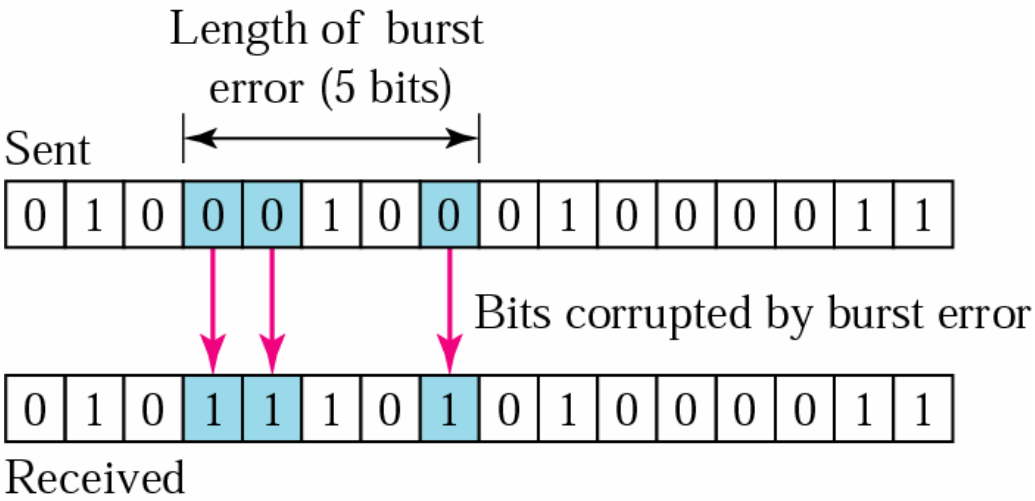


FIGURE 9.2 Burst Error ซึ่งมีความยาว 5 บิต โดยมีบิตที่ผิดพลาดเพียง 3 บิต

การตรวจจับข้อผิดพลาด

จุดประสงค์ของการศึกษา Error คือเพื่อต้องการแก้ไข ซึ่งจำเป็นต้องมีการตรวจจับข้อผิดพลาด (Error Detection) ก่อน ซึ่งวิธีการดังกล่าวมักอาศัยเทคนิคที่เกี่ยวข้องกับข้อมูลซ้ำซ้อน (Redundancy)

แนวคิดพื้นฐานของข้อมูลซ้ำซ้อน

วิธีตรวจจับ Error ที่ง่ายที่สุด คือการส่งข้อมูล ซ้ำซ้อน 2 ชุด แล้วอุปกรณ์ ทางด้านรับจะทำการเปรียบเทียบหาความแตกต่าง ซึ่งถ้ามี แสดงว่าได้มี Error เกิดขึ้น จุดเด่น ของวิธีนี้ คือเป็นวิธีที่สะดวก โอกาสตรวจไม่พบต่ำ เนื่องจากโอกาสที่ข้อมูลทั้ง 2 ชุดที่ซ้ำกัน ผิดตำแหน่งเดียวกัน เกือบเป็นไปไม่ได้ ส่วน จุดด้อย คือ มีความเร็วในการประมวลผลต่ำ และสิ้นเปลืองทรัพยากรช่องสัญญาณ เช่นต้องเพิ่ม Band Width เป็น N เท่า ถ้าส่งข้อมูลซ้ำซ้อนจำนวน N ชุด

อย่างไรก็ดี ด้วยหลักการพื้นฐานของการใช้ข้อมูลซ้ำซ้อน เราอาจเพิ่มจำนวนบิตของข้อมูลที่มีขนาดเล็กกว่าหน่วยของข้อมูล (Redundancy) เพื่อใช้ในการตรวจจับข้อผิดพลาดโดยเฉพาะ ในส่วนนี้จะกล่าวถึง *วิธีการสร้าง Redundancy ของข้อมูลเพื่อจุดประสงค์ในการตรวจจับข้อผิดพลาด* ดังแผนผัง

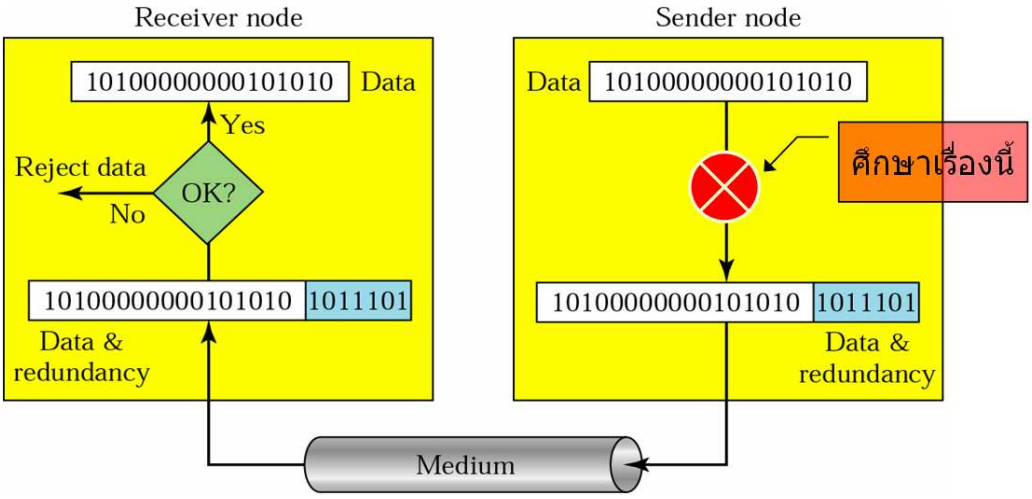


FIGURE 9.3 แผนผังแสดงกระบวนการตรวจจับข้อผิดพลาดของข้อมูล โดยด้านส่ง (Sender Node) จะทำการสร้าง Redundancy จากตัวข้อมูล (กากบาท) แล้วส่งต่อท้ายไปกับข้อมูลในตัวกลาง ทางฝั่งด้านรับ (Receiver Node) จะตรวจสอบข้อมูล กับ Redundancy ด้วยกฎที่ตกลงไว้เพื่อพิจารณาว่าเกิดข้อผิดพลาดหรือไม่

ขั้นตอนวิธีการตรวจสอบข้อผิดพลาดเบื้องต้น ที่นิยมใช้ในการสื่อสารข้อมูลมี 3 วิธี ได้แก่ Parity Check CRC Check และ Checksum นอกจากนี้ยังมีวิธีการเข้ารหัสขั้นสูง ที่มีประสิทธิภาพมากกว่า ได้แก่ Hadamard Quadratic Residue Golay (มีการใช้จริงในโครงการอวกาศ Voyager I, II 1979 – 81) และ Reed Codes เป็นต้น

Parity Check

Parity Check เป็นขั้นตอนวิธีแพร่หลายที่สุด และใช้ทรัพยากรน้อยที่สุด มี 2 ประเภท ได้แก่ อย่างง่าย หรือแบบเชิงเส้น (Linear) และแบบ 2 มิติ (2 Dimensional – 2D)

หลักการทำงานของการตรวจจับข้อผิดพลาดด้วยวิธี Parity Check อิงกับแนวคิด Redundancy ดังกล่าวข้างต้น กล่าวคือ Sender Node จำคำนวณ Parity Bit ซึ่งเป็น Redundancy ขนาด 1 บิต

แล้วส่งต่อท้ายไปกับข้อมูล ส่วน Receiver Node จะทำการนับบิตที่มีค่าเป็น 1 (รวม Parity Bit) ว่าเป็นจำนวนคู่ (หรือจำนวนคี่ ขึ้นกับชนิดของ Parity) ซึ่งถ้าตรงกับเงื่อนไขที่กำหนดจะยอมรับ (Accept) มิฉะนั้นจะปฏิเสธ (Reject) ข้อมูลนั้น ดังแผนผังในรูปที่ 9.4

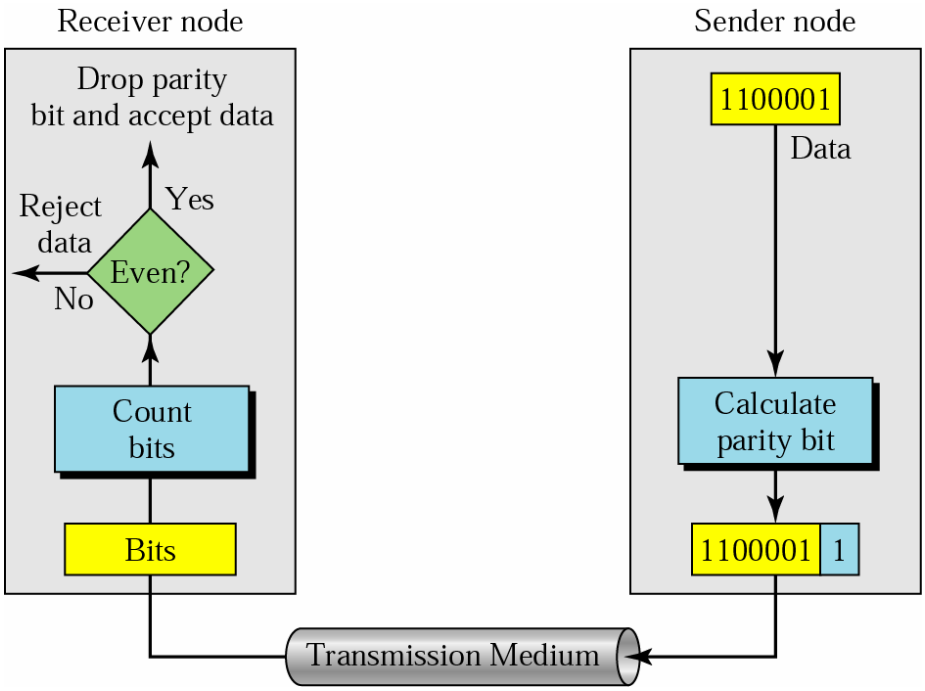


FIGURE 9.4 แผนผังแสดงการตรวจจับข้อผิดพลาดด้วยวิธี Parity Check โดยฝั่งด้านส่งจะคำนวณ Parity Bit ต่อท้ายไปกับข้อมูล ส่วนฝั่งด้านรับจะตรวจสอบข้อผิดพลาดของข้อมูลโดยการนับจำนวนบิตตามเงื่อนไข Parity

Linear Parity เงื่อนไขของการคำนวณ และตรวจสอบ Parity Bit ขึ้นกับชนิดของ Parity กล่าวคือ

Even Parity เพิ่มบิต 1 หรือ 0 ท้ายหน่วยข้อมูล เพื่อให้จำนวน 1 ทั้งหมดรวม Parity เป็น **จำนวนคู่**

Odd Parity เพิ่มบิต 1 หรือ 0 ท้ายหน่วยข้อมูล เพื่อให้จำนวน 1 ทั้งหมดรวม Parity เป็น **จำนวนคี่**

Sending Node/Tx Medium		Receving Node
1110 111 1100 100	⇒	1110 111 0 1100 100 1
Send W and D ASCII Codes		Parity 0 for W and 1 for D
1110 1**0**1 0 11**11** 100 1	⇒	*1110 101* 0 1111 100 1
Errors 1 Bit at W and 2 Bits at D		Error found in W but not D

หมายเหตุ บิตที่ขีดเส้นใต้ ได้แก่ Parity Bit บิตที่เน้นตัวหนา ได้แก่บิตที่เกิดข้อผิดพลาด และข้อมูลที่เป็นตัวเอน ได้แก่ข้อมูลที่ตรวจพบข้อผิดพลาดด้วยวิธี Parity Check

Parity Check สามารถตรวจจับ Single-Bit Error เท่านั้น ไม่เหมาะกับกรณี Burst Error นอกจากเราจะทราบล่วงหน้าว่าจะเกิด Burst Error เป็นจำนวนคี่

2–Dimensional Parity อ้างอิงหลักการเดียวกันกับ Linear Parity กล่าวคือ จะจัดข้อมูลออกเป็นกลุ่ม กลุ่มละ *N* หน่วยข้อมูล ดังตัวอย่างในรูปที่ 9.5 ข้อมูลถูกแบ่งออกเป็นกลุ่ม กลุ่มละ 4 หน่วยข้อมูล โดยที่ 1 หน่วยมีขนาด 7 บิต (Original Data) ข้อมูลแต่ละหน่วยในกลุ่ม จะถูกนำมาคำนวณบิต Parity อิสระจากกัน แล้วจึงนำมาเรียงกันเป็น Matrix แถวละ 1 หน่วยข้อมูล Column ด้านซ้ายมือสุด ซึ่งเป็น Parity Bit รวมของแต่ละหน่วยข้อมูล เรียกว่า Row Parities หลังจากนั้นจะทำการคำนวณบิต Parity ของข้อมูลในแต่ละ Column (บิตที่ตรงกันของทุกหน่วยข้อมูล) อิสระจากกัน ผลลัพธ์ที่ได้จะเป็นข้อมูล Row ด้านล่างสุด ซึ่งเป็น Parity Bit รวมของแต่ละบิตที่ตรงกันของทุกหน่วยข้อมูล เรียกว่า Column Parities และสุดท้าย ข้อมูลทั้งชุด (รวม Linear Parity ของแต่ละหน่วย) และ Column Parities รวมทั้งหมด *N + 1* หน่วย (5 หน่วยตามตัวอย่าง) จะถูกส่งไปในตัวกลาง

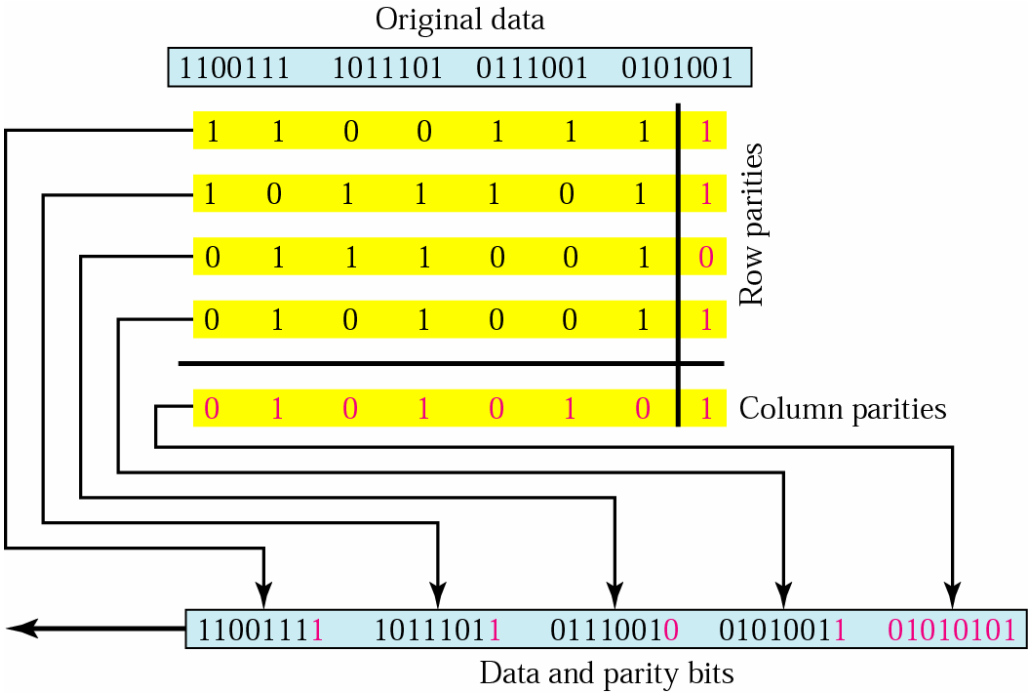


FIGURE 9.5 แผนผังแสดงการคำนวณบิต Parity สำหรับการตรวจสอบข้อผิดพลาดด้วยวิธี 2D Parity Check

สมมติว่าส่งข้อมูล ASCII 4 **ตัวอักษร พร้อม** Column Parities **ดังต่อไปนี้**

1010100 **1** 0011100 **1** 1101110 **1** 1110011 **1** 10101010

ภายในตัวกลางเกิดสัญญาณรบกวนขนาด 8 Bits **ทำให้มีข้อมูลบางส่วนผิดพลาด** (Burst Error)

1010001 1 1000100 1 1101110 1 1110011 1 10101010

เมื่อด้านอุปกรณ์ด้านรับตรวจสอบ Parity **ทั้ง** Row **และ** Column **พบว่าเกิดข้อผิดพลาดขึ้น (ต้องส่งใหม่)**

1010001 1 1000100 1 1101110 1 1110011 1 **1**0**101**0**1**0

เมื่อเทียบกับ Linear Parity วิธีการ 2–D Parity มีภูมิคุ้มกันต่อความผิดพลาดมากกว่ากล่าวคือสามารถตรวจจับ Burst Error ได้ **ยกเว้น** กรณีที่ หน่วยข้อมูลจำนวนคู่ (เช่น 2 Bytes) มีบิตผิดพลาด ณ ตำแหน่งตรงกัน เป็นจำนวนคู่

Cyclic Redundancy Check (CRC)

CRC ใช้หลักการหารเลขฐาน 2 กล่าวคือ Sender Node จะคำนวณ **กลุ่มของบิต** Redundancy ซึ่งเมื่อนำไปต่อท้ายหน่วยข้อมูลแล้ว จะทำให้อนุกรมของบิตผลลัพธ์สามารถหารด้วย จำนวนที่กำหนดไว้ล่วงหน้า (เรียกว่า ตัวหาร – Divisor หรือ **กุญแจรหัส**) ได้ลงตัว โดยที่ความยาวของบิต CRC น้อยกว่าของตัวหาร (Divisor) อยู่ 1 บิต

ทางด้าน Receiver Node ตรวจสอบข้อผิดพลาด โดยการ หารข้อมูลที่ได้รับมาด้วย **กุญแจรหัส** ค่าเดียวกันกับทาง Sender Node ซึ่งถ้าผลลัพธ์เป็นการหารลงตัวแสดงว่าข้อมูลถูกต้อง มิฉะนั้น แสดงว่าเกิดข้อผิดพลาดขึ้น แผนผังการทำ CRC แสดงดังรูปที่ 9.6

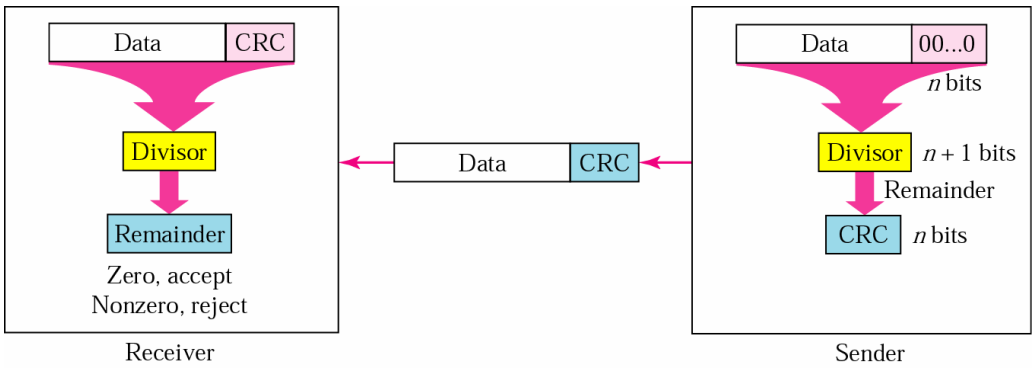


FIGURE 9.6 แผนผังแสดงการตรวจสอบข้อผิดพลาดด้วยวิธีการ (CRC) โดยการเพิ่ม CRC ขนาด n บิตต่อท้ายข้อมูล

หลักการคำนวณ Binary Division

ตัวตั้งที่สามารถหารด้วย Divisor ได้ต้องมีจำนวนบิต (ความยาว) เท่ากับ Divisor และผลหารมีค่าเท่ากับ 1

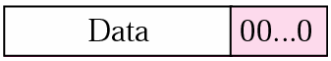
ถ้าตัวตั้งมีความยาวน้อยกว่า Divisor แล้ว ผลหารมีค่าเท่ากับ 0 และแทนค่าตัวหารในขั้นตอนนั้น ด้วยเลข 0 ที่มีความยาวเท่ากับ Divisor

การลบตัวตั้งด้วยตัวหารในแต่ละขั้น ใช้การลบเลขฐานสองแบบไม่มีตัวทด เศษเหลือในขั้นตอนสุดท้าย เมื่อใช้เลขตัวตั้งครบทุกบิต

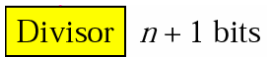
CRC Generator การสร้าง CRC Bits สำหรับข้อมูลที่กำหนดให้มีขั้นตอนดังนี้

1. กำหนดให้ความยาวของ CRC เท่ากับ n Bits
2. เลือกตัว Divisor ที่มีความยาว $n + 1$ Bits และมีบิตซ้ายมือเท่ากับ 1

3. นำเลข 0 จำนวน n Bits มาต่อท้ายข้อมูล



4. หารผลลัพธ์ที่ได้ในข้อ 3 ด้วย Divisor ด้วยวิธี Binary Division



5. เศษที่เหลือคือ CRC – ทำให้มีความยาว n Bits โดยเติม 0 ไปด้านซ้ายมือ

6. แทนที่เลข 0 ในข้อ 3 ด้วย CRC Bits ที่ได้ในข้อ 5

CRC Checker การตรวจสอบ CRC Bits สำหรับข้อมูลที่กำหนดให้มีขั้นตอนดังนี้

1. รับข้อมูลที่มี CRC ต่อท้าย n Bits

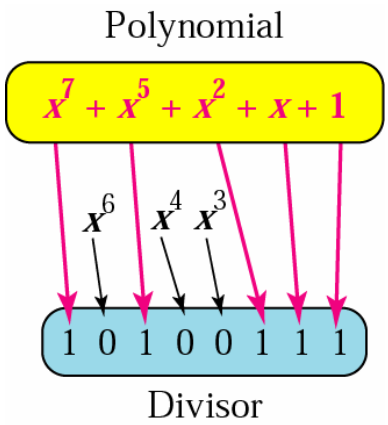
2. เลือกตัว Divisor ที่มีความยาว n + 1 Bits ตัวเดียวกับทางด้านรับ

3. หารข้อมูลที่ได้ในข้อ 1 ด้วย Divisor ด้วยวิธี Binary Division

4. พิจารณาเศษที่เหลือจากการหาร

5. ถ้าเศษที่เหลือมีค่าเท่ากับ 0 แสดงว่าข้อมูลที่รับเข้ามาในข้อ 1 ถูกต้อง ให้ตัดส่วนที่เป็น CRC ทิ้งเพื่อนำข้อมูลไปใช้งาน มิฉะนั้น แสดงว่าเกิดข้อผิดพลาดขึ้น ให้ส่งสัญญาณไปยังอุปกรณ์ส่งเพื่อให้ทำการส่งข้อมูลใหม่

Divisor สำหรับขั้นตอนวิธี CRC มักจะแสดงในรูปของฟังก์ชันพหุนาม (Polynomial) เนื่องจาก เป็นการนำเสนอ Divisor ที่กระชับสื่อความหมายได้ดี และเอื้อประโยชน์ต่อการวิเคราะห์ทางพีชคณิต (Arithmetic Modulo Two) ตัวอย่างเช่น พหุนามดีกรี 7 CRC $(x) = x^7 + x^5 + x^2 + x + 1$ หมายถึง Divisor = 10100111 (ดังรูป) ในทางทฤษฎีได้แนะนำให้ Divisor เป็น Irreducible Polynomial กล่าวคือ มีเฉพาะตัวมันเอง และ 1 เท่านั้นที่สามารถหารได้ลงตัว อย่างไรก็ดี สามารถอนุโลมข้อกำหนดดังกล่าวได้ เช่น CRC ที่ Divisor มีสัมประสิทธิ์ที่ไม่เป็น 0 อย่างน้อย 1 พจน์ จะสามารถตรวจจับ Single Bit Error ได้ CRC ที่ Divisor หารด้วย $x + 1$ ลงตัว จะสามารถตรวจสอบบิตที่ผิดพลาดเป็นจำนวนคี่ได้ และ CRC สามารถตรวจจับ Burst Error ที่มีความยาวน้อยกว่าหรือเท่ากับอันดับของพหุนามของ Divisor ได้เป็นต้น



ตัวอย่างของ Divisor Polynomial ที่นิยมใช้ในการสื่อสาร และการประยุกต์ใช้งาน แสดงดังตาราง

Name	Polynomial	Application
CRC-8	$x^8 + x^2 + x + 1$	ATM header
CRC-10	$x^{10} + x^9 + x^5 + x^4 + x^2 + 1$	ATM AAL
ITU-16	$x^{16} + x^{12} + x^5 + 1$	HDLC
ITU-32	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$	LANs

Checksum

Checksum มีหลักการคล้ายกับ Parity Check และ Cyclic Redundancy Check ในประเด็นของการใช้ Redundancy Bits ซึ่งสามารถอธิบายการทำงานได้ดังนี้

1. ข้อมูลต้นฉบับจะถูกแบ่งออกเป็น Segment ซึ่งมีความยาว n bits (เช่น $n = 16$)
2. ข้อมูลในแต่ละ Segment จะนำมาบวกกันด้วยวิธี 1's Complement ผลลัพธ์ที่ได้จะมีขนาด n bits
3. ผลลัพธ์ที่ได้ในข้อ 2 จะถูกทำ Complement ซ้ำ แล้วนำไปต่อท้ายข้อมูลต้นฉบับ กลายเป็น Redundant Bits
4. ข้อมูลต้นฉบับ พร้อมกับ Redundant Bits จะถูกส่งออกไปยังเครื่องรับปลายทาง ผ่านระบบเครือข่าย

ในการสื่อสารทั่วไปมักจะทำ Checksum กับข้อมูลหลาย Segments พร้อมๆ กัน ดังรูป แสดงการทำ Checksum กับข้อมูล k Segments แต่ละ Segment มีความยาว n Bits

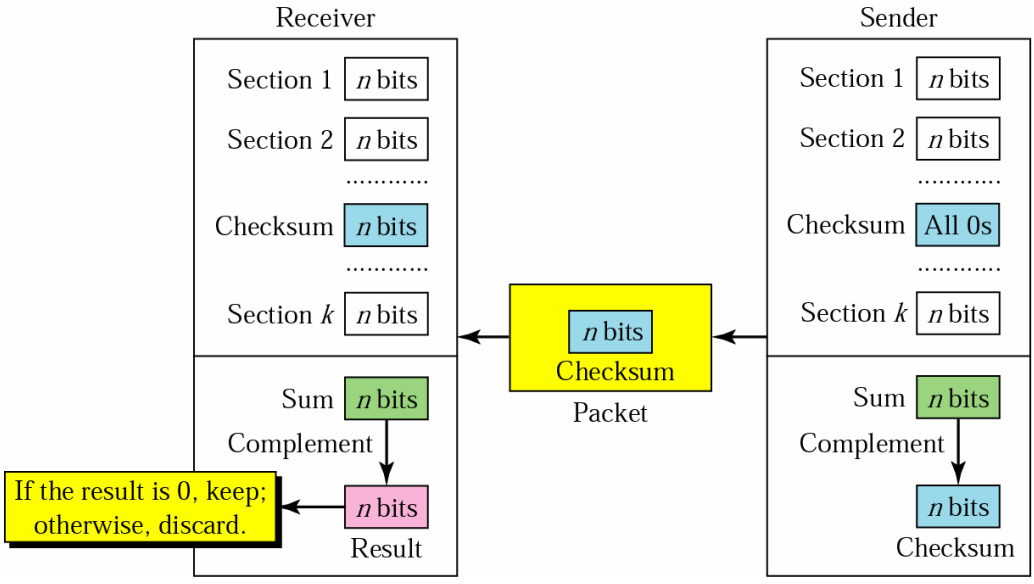
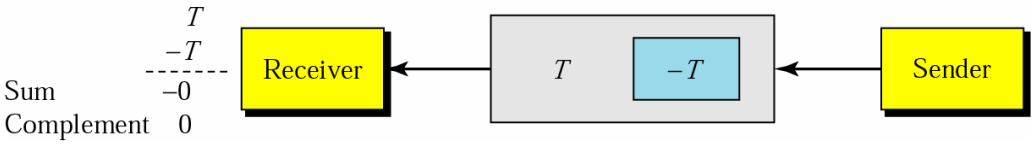


FIGURE 9.7 แผนผังการตรวจสอบข้อผิดพลาดด้วยวิธีการ Checksum กับข้อมูล k Segments แต่ละ Segment มีความยาว n Bits

Generating Checksum การตรวจสอบข้อผิดพลาดด้วย Checksum อาจแสดงได้ด้วยแผนผังดังนี้



สมมติให้ส่งข้อมูลจำนวน 16 bits โดยแบ่งออกเป็น 2 Segments และแต่ละ Segment มีความยาว 8 Bits ดังนี้ 10101001 001111001

นำจำนวนทั้งสองมาบวกกันด้วยวิธี 1's Complements ดังนี้

10101001

00111001

Sum 11100010

Checksum 00011101

ดังนั้นข้อมูลที่ส่งพร้อม Checksum ได้แก่ 10101001 00111001 00011101

Detecting Checksum สมมติให้รับข้อมูลที่ได้จากขั้นตอนข้างต้นโดยไม่มีข้อผิดพลาด (No Error Case) เมื่อบวกข้อมูลทั้ง 3 Segments เข้าด้วยกัน จะได้ผลลัพธ์เป็น 1 ทั้งหมด ซึ่งเมื่อทำ Complement แล้ว จะได้ผลลัพธ์เป็น 0 หมายถึง ไม่มีข้อผิดพลาดใดๆ

สมมติกรณีที่เกิด Burst Error ความยาว 5 บิต (ขีดเส้นใต้) ซึ่งทำให้ข้อมูลผิดไป 4 บิต ดังนี้

10101111 11111001 00011101 เมื่อนำข้อมูลทั้ง 3 segments มาบวกกันจะได้

10101111

11111001

00011101

Partial Sum **1** 11000101

Carry **1**

Sum 11000110

Complement 00111001 ผลลัพธ์ที่ได้มีค่าไม่เป็น 0 ซึ่งบ่งชี้ว่าเกิดข้อผิดพลาดขึ้น

หมายเหตุ Checksum ไม่สามารถตรวจพบข้อผิดพลาดได้ถ้า ความผิดพลาดใน Segment หนึ่ง สมดุลกับความผิดพลาดในอีก Segment หนึ่ง ซึ่งเกิดในตำแหน่งเดียวกัน (จำนวน 1 และ 0 ไม่เปลี่ยนแปลง)

การแก้ไขข้อผิดพลาด

เมื่อตรวจพบข้อผิดพลาด (Error) ด้วยวิธี Parity Check CRC หรือ Checksum ฯลฯ แล้ว กระบวนการถัดไป คือการแก้ไข Error นั้น (Error Correction) ซึ่งที่นิยมใช้มี 3 วิธี ดังนี้

1. **Retransmission** คือวิธีแก้ไข Error ที่ง่ายที่สุด คือเมื่อตรวจพบ อุปกรณ์ด้านส่งจะแจ้งไปยังอุปกรณ์ด้านส่ง เพื่อร้องขอ (REQ) ให้มีการส่งข้อมูลมาใหม่ทั้งชุด

2. **Forward Error Correction** (FEC) คือวิธีที่แก้ไข Error อัตโนมัติที่ด้านรับ ทั้งนี้โดยอาศัยกลไกของการเข้ารหัสแบบพิเศษ เรียกว่า Error Correcting Code ซึ่งมีความซับซ้อนมากกว่าการตรวจจับ Error ทั่วไป และต้องการ Redundant Bits มากกว่า

3. **Burst Error Correction** คือวิธีที่แก้ไข Burst Error โดยการจัดเรียงข้อมูลในรูปแบบใหม่

เนื่องจากวิธี Retransmission ค่อนข้างตรงไปตรงมา และลำดับขั้นการทำงานได้มีการนิยามรูปแบบเฉพาะ ซึ่งจะได้กล่าวโดยละเอียดในบทถัดไป ในที่นี้จึงอธิบายเฉพาะวิธี Forward Error Correction และ Burst Error Correction เท่านั้น

Forward Error Correction (FEC)

ในทางทฤษฎีรหัสสำหรับแก้ไขข้อผิดพลาด (Error Correcting Code) สามารถแก้ไขข้อผิดพลาดได้ทุกชนิด ทุกรูปแบบ เรียกการแก้ไขข้อผิดพลาด แบบนี้ว่า FEC

ตัวอย่างกรณี Single Bit Error การตรวจจับและแก้ไขข้อผิดพลาดชนิดนี้ ต้องสามารถระบุข่าวสาร 2 ประการ กล่าวคือ

1) มีข้อผิดพลาดเกิดขึ้นหรือไม่ (Error Detection)

2) ข้อผิดพลาดที่เกิดขึ้นนั้นอยู่ในตำแหน่งใดในอนุกรมข้อมูล (Error Identification)

หากต้องการเพียงตรวจจับข้อผิดพลาดชนิดนี้ ดังได้แสดงแล้วว่า ต้องการ Redundant Bit อีกเพียง 1 บิตเท่านั้น เพื่อระบุว่าชุดข้อมูลนั้นมีข้อผิดพลาดเกิดขึ้น (1) หรือไม่ (0) เช่นกรณี Parity Check

อย่างไรก็ดี หากต้องการ **แก้ไข** ข้อผิดพลาดดังกล่าวด้วยนั้นต้องสามารถ **ระบุตำแหน่งที่เกิดข้อผิดพลาด** (Identification of Invalid Bit) ได้ ซึ่งเมื่อทราบข่าวสารดังกล่าว อุปกรณ์ด้านรับเพียง **กลับค่าบิต** ของข้อมูล ณ ตำแหน่งนั้น

Identifying Invalid Bit สำหรับข้อมูลชนิด ASCII ขนาด 7 บิต การแก้ไข Single Bit Error ต้องสามารถระบุ สถานะ ของข้อมูล ถึง 8 สถานะ ได้แก่

000 ไม่มี Error	**001** Error ณ บิต 1	**010** Error ณ บิต 2	**011** Error ณ บิต 3
100 Error ณ บิต 4	**101** Error ณ บิต 5	**110** Error ณ บิต 6	**111** Error ณ บิต 7

จากตารางดูเหมือนว่า Redundancy ต้องการเพียง 3 Bits ในการระบุตำแหน่งที่เกิด Error ได้ 7 ตำแหน่ง ($2^3 - 1$) แต่เงื่อนไขนี้ ไม่สามารถระบุตำแหน่งที่เกิด Error สำหรับ Redundant Bit เองได้

ดังนั้นถ้าต้องการ แก้ไข Error ได้ทุกกรณี ในที่นี้ หมายถึง Error ที่เกิดขึ้นใน Data Bit (7 สถานะ) ใน Redundant Bit (3 สถานะ) และไม่เกิด Error (1 สถานะ) ระบบ FEC จึงต้องการบิตเพิ่มเติม

Counting Redundant Bits ถ้าต้องการส่ง Data จำนวน m Bit และ Redundancy จำนวน r Bit แล้ว Redundant Bit จะต้องสามารถระบุตำแหน่ง Error ได้ถึง $m + r + 1$ สถานะ (สถานะสุดท้ายหมายถึง ไม่เกิด Error) ดังนั้น ความสัมพันธ์ระหว่างความยาว m และ r จึงเขียนได้เป็น

$$2^r \geq m + r + 1$$

ตัวอย่าง ถ้าต้องการส่งข้อมูลในรหัส ASCII ความยาว 7 บิต

กำหนดให้ $m = 7$

ดังนั้น $2^r \geq 7 + r + 1$

ย้ายข้างสมการ $2^r - 1 \geq 8$

จะได้ว่า $r = 4$ ต้องส่งข้อมูลทั้งสิ้น $m + r + 1 = 7 + 4 + 1 = 11$ บิต

Hamming Code เป็นเทคนิคหนึ่งของ FEC ซึ่งสามารถตรวจจับและแก้ไขข้อผิดพลาดของข้อมูลซึ่งมีความยาวใดๆ ได้ โดยที่ Redundant Bit จะปรากฏในตำแหน่งที่เป็นกำลังของ 2 ดังรูป

11	10	9	8	7	6	5	4	3	2	1
d	**d**	**d**	r_8	**d**	**d**	**d**	r_4	**d**	r_2	r_1

โดยที่ d คือ Data Bit และ r_i คือ Redundant Bit ตำแหน่งที่ i ซึ่งเป็น Parity ของกลุ่ม Data Bit ที่ได้กำหนดไว้ล่วงหน้า ดังนี้

r_1 เป็น Even Parity ของกลุ่มบิต 1 3 5 7 9 และ 11

r_2 เป็น Even Parity ของกลุ่มบิต 2 3 6 7 10 และ 11

r_3 เป็น Even Parity ของกลุ่มบิต 4 5 6 และ 7

r_4 เป็น Even Parity ของกลุ่มบิต 8 9 10 และ 11

สังเกตในกรณี Data Bit มีการคำนวณ Parity อย่างน้อย 2 ครั้ง ในขณะที่ Redundant Bit เองมีการคำนวณ Parity 1 ครั้ง

หมายเหตุ Even Parity เพิ่มบิตท้ายหน่วยข้อมูล เพื่อให้จำนวน 1 ทั้งหมดรวม Parity เป็นคู่

Hamming Calculation ตัวอย่างต่อไปนี้แสดงการคำนวณ r Bit จาก Data m = 100 1101

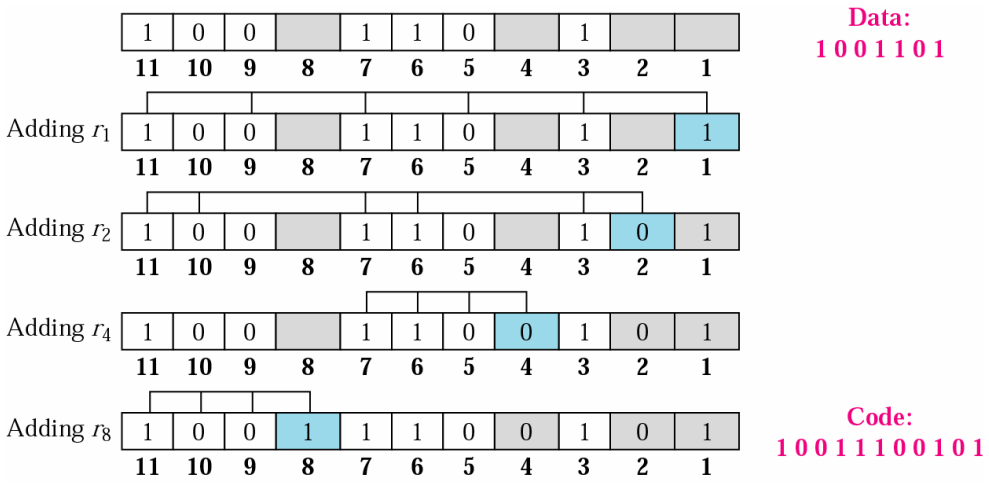


FIGURE 9.8 การคำนวณ r Bit จากข้อมูล 100 1101 โดยเส้นแขนงแสดงถึงตำแหน่งบิตที่เกี่ยวข้องในการคำนวณ Parity

Hamming Error Detection and Correction ในที่นี้สมมติให้ Data Bit ในตำแหน่งที่ 7 เกิดข้อผิดพลาด ทางด้านรับคำนวณ Parity Check สำหรับบิตข้อมูลแต่ละกลุ่ม เหมือนทางด้านส่ง โดยเขียนเรียงตัวเลขตามตำแหน่ง (r_8, r_4, r_2, r_1) ดังรูป ซึ่งจากตัวอย่างเห็นว่า Parity Check ที่ได้เป็นเลขฐาน 2 ของ 7 ซึ่งแสดงถึงตำแหน่งที่เกิดข้อผิดพลาด

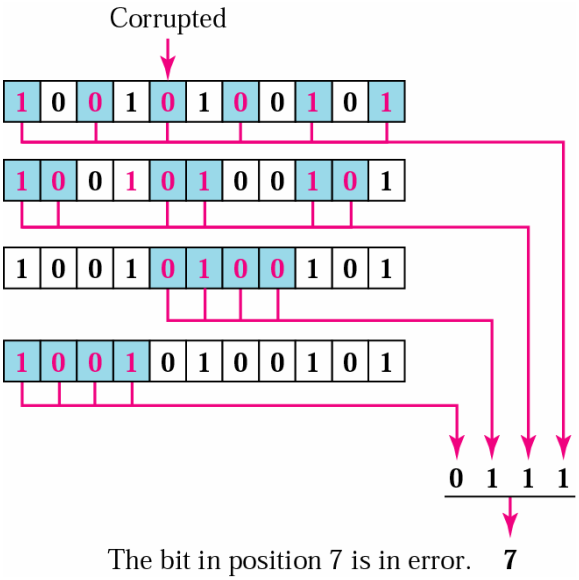


FIGURE 9.9 การตรวจสอบและแก้ไขข้อผิดพลาดด้วยวิธี Hamming Code ซึ่งเกิดข้อผิดพลาดในบิตข้อมูลตำแหน่งที่ 7

Burst Error Correction

วิธีนี้ไม่สามารถแก้ไข Burst Error ได้โดยตรง แต่ถ้าจัดข้อมูลเป็นกลุ่มกลุ่มละ N ชุด (แถวละ 1 ชุด) แล้วส่งข้อมูลนั้นเรียงไปตาม Column ถ้าเกิด Burst Error จำนวน M บิต และ ถ้า ($M < N$) แล้วจะเกิดข้อผิดพลาด ต่อข้อมูลชุดละ 1 บิตเท่านั้น ดังรูป

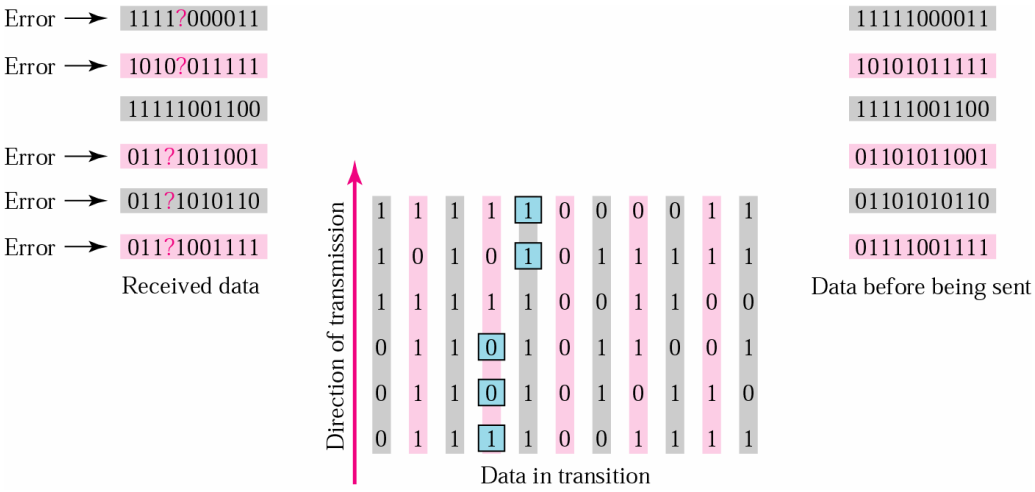


FIGURE 9.10 แผนผังแสดงการตรวจสอบและแก้ไขข้อผิดพลาดแบบ Burst Error

จากรูป ข้อมูลจำนวน $N = 6$ ชุด ชุดละ 11 บิต ข้อมูลจะถูกส่งเรียงลำดับตามบิตที่ตรงกันของแต่ละชุด เมื่อเกิดข้อผิดพลาดขึ้น ดังตัวอย่างเป็น Burst Error ขนาด $M = 5$ บิต (กรอบสี่เหลี่ยม) ที่ปลายทางเมื่อมีการจัดเรียงข้อมูลกลับเป็นเหมือนเดิมพบว่า ข้อมูลแต่ละชุดจะมีข้อผิดพลาดเพียงแค่ 1 บิตเท่านั้น

แบบฝึกหัด

1. อธิบายหน้าที่ของ Protocol ใน Data Link Layer ทั้ง 5 ประการ
2. อธิบายความแตกต่างระหว่าง Logical Link Control และ Media Access Control มาพอสังเขป
3. หากข้อมูลต้นทาง และปลายทางเป็น 1011 0110 และ 1001 0000 ตามลำดับ Burst Error ที่เกิดขึ้นมีความยาวกี่บิต
4. จากข้อ 3 หากใช้วิธี Parity Check จะตรวจพบข้อผิดพลาด หรือไม่ เพราะเหตุใด และหากใช้วิธี 2D Parity Check โดยแบ่งข้อมูลเป็น 2 แถว แถวละ 4 บิต อภิปรายเปรียบเทียบผลลัพธ์ที่ได้
5. สร้าง CRC ขนาด 3 บิต จากข้อมูล 1011 0110 โดยใช้ Divisor 1101 และสมมติให้ข้อมูลที่ปลายทางรับได้เป็น 1011 1010 สาธิตวิธีการตรวจจับข้อผิดพลาด
6. จากข้อ 3 ถ้าใช้ CRC ที่สร้างได้จาก Divisor 1101 จะสามารถตรวจจับข้อผิดพลาดที่เกิดขึ้นได้หรือไม่ เพราะเหตุใด อภิปราย
7. จากข้อ 3 ใช้วิธี Checksum ตรวจจับข้อผิดพลาด โดยแบ่งข้อมูลเป็น 2 ชุด ชุดละ 4 บิต
8. อธิบายหลักการของ Forward Error Correction มาพอสังเขป
9. กำหนดรหัส ASCII ต้นทาง และปลายทางเป็น 011 1011 และ 010 1011 ตามลำดับ สาธิตการใช้วิธี Hamming Code ในการตรวจจับและแก้ไขข้อผิดพลาดที่เกิดขึ้น
10. ศึกษา และอภิปรายเทคนิคการตรวจจับและแก้ไขข้อผิดพลาดที่แตกต่างจากบทเรียนมา 1 ตัวอย่าง