Forward Error Correction in Sensor Networks

Jaein Jeong, *Cheng-Tien Ee*

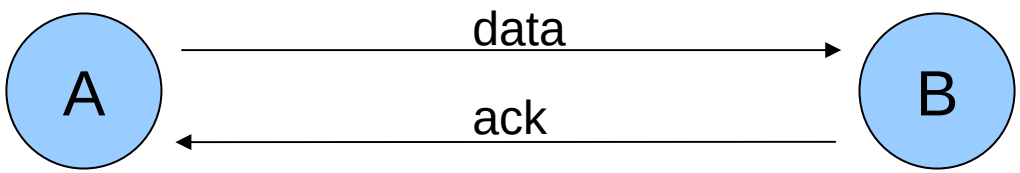
University of California, Berkeley

Motivation

- **Packet errors occur in WSN.**
  - Error recovery is required for correct delivery.
- **Questions.**
  - What kinds of error recovery method?
  - What level of error recovery capability?

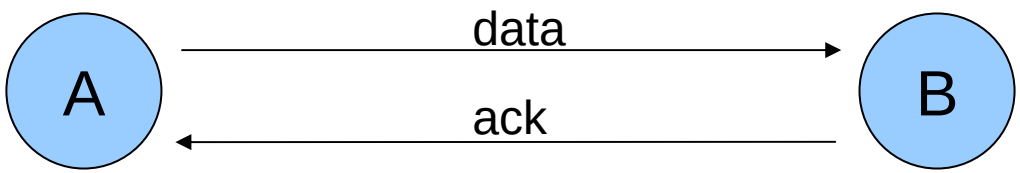
Two methods for error recovery

- **ARQ (Automatic Repeat reQuest)**
  - A sends, and B acks.

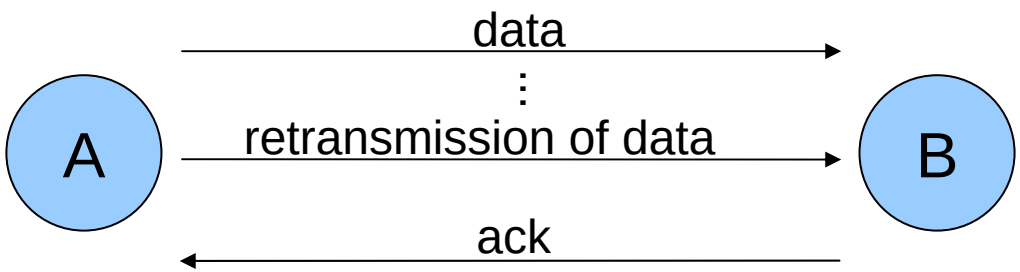


Two methods for error recovery

- **ARQ (Automatic Repeat reQuest)**
  - A sends, and B acks.

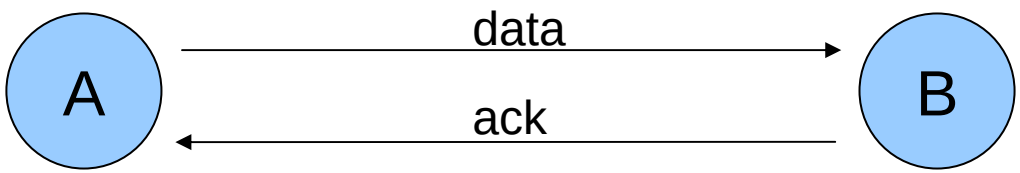


  - A sends, B misses, and A resends.

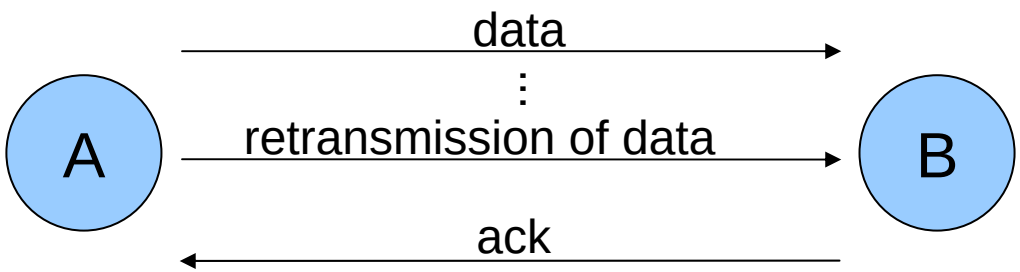


Two methods for error recovery

- **ARQ (Automatic Repeat reQuest)**
  - A sends, and B acks.



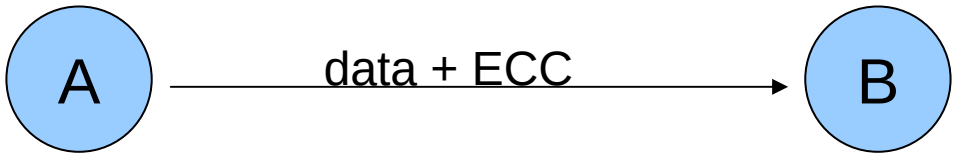
  - A sends, B misses, and A resends.



  - TX cost increases with (#-nodes, #-TX).

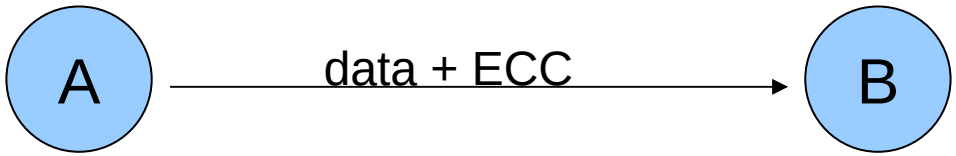
Two methods for error recovery

- **FEC (Forward Error Correction)**
  - A sends data with error correction code (ECC).



Two methods for error recovery

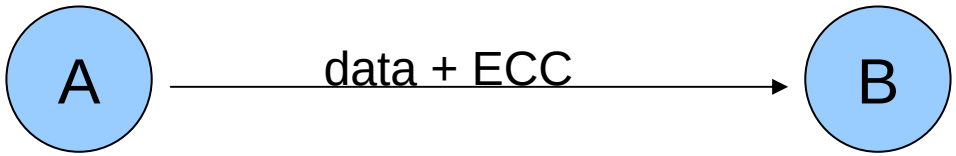
- **FEC (Forward Error Correction)**
  - A sends data with error correction code (ECC).



  - Preferable in broadcast and multi-hop network.

Two methods for error recovery

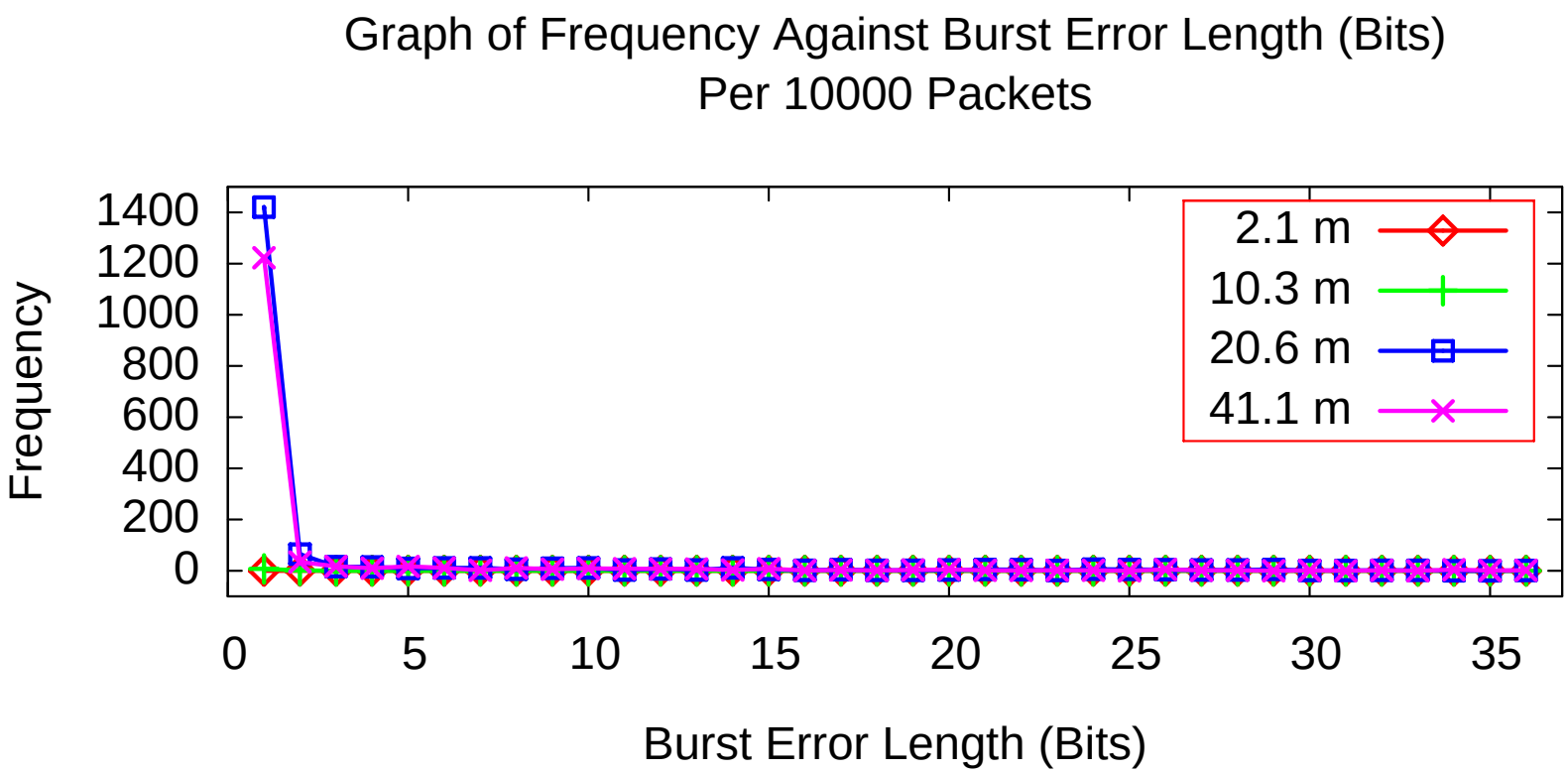
- **FEC (Forward Error Correction)**
  - A sends data with error correction code (ECC).



  - Preferable in broadcast and multi-hop network.
  - We focus on FEC for WSN.

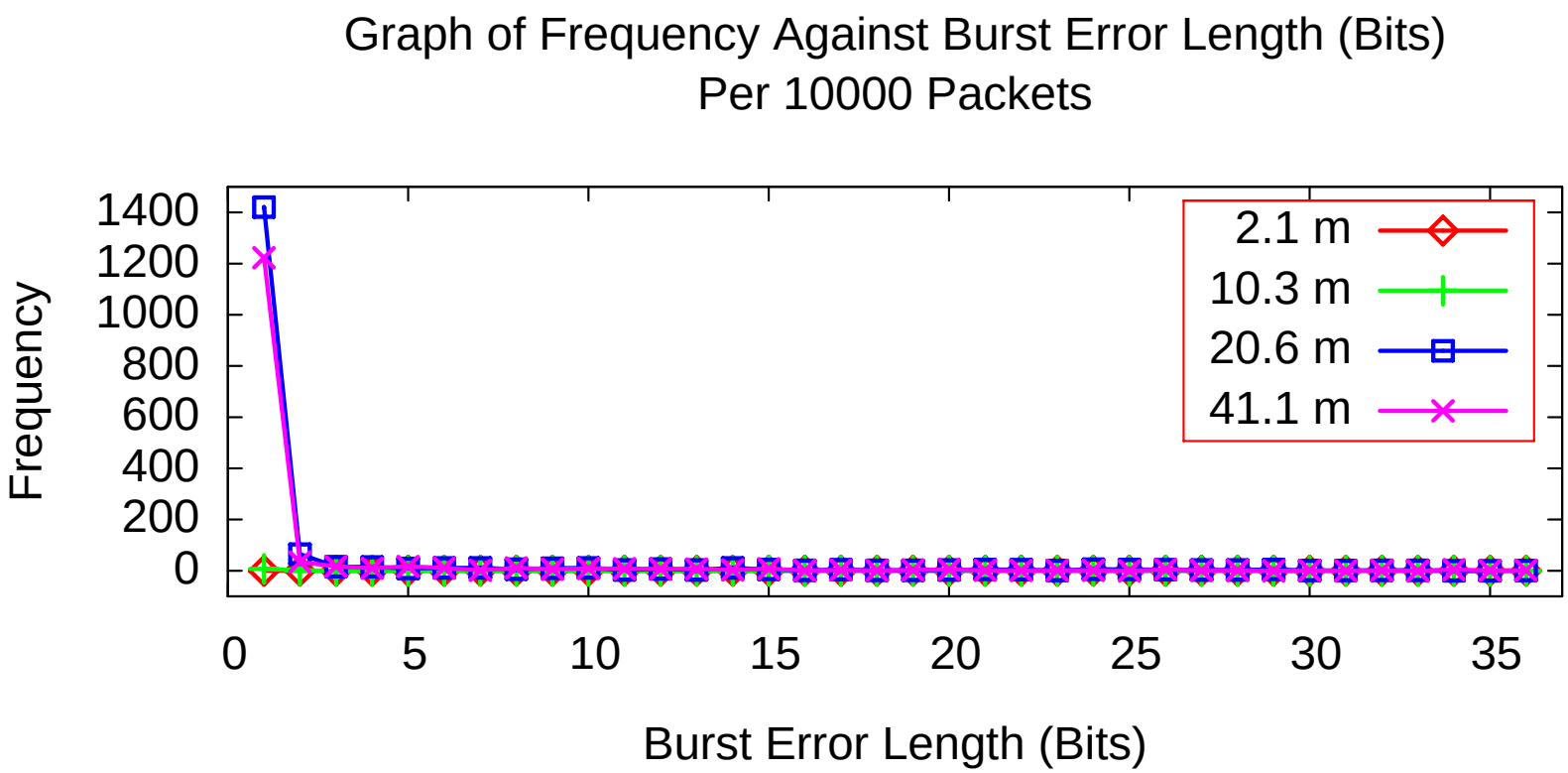
Choosing Right ECC for WSN

- **Preliminary Experiment**
  - Most packet errors are 1-bit or 2-bit.



Choosing Right ECC for WSN

- **Preliminary Experiment**
  - Most packet errors are 1-bit or 2-bit.



- **Our approach: 1-bit & 2-bit ECC for WSN.**

Organization

- **Background**
- **Theory**
- **Implementation**
- **Experiment**
- **Conclusion**

Organization

- **Background**
  - Reed-Solomon, LT-code, 1-bit ECC.
- **Theory**
- **Implementation**
- **Experiment**
- **Conclusion**

Organization

- **Background**
  - Reed-Solomon, LT-code, 1-bit ECC.
- **Theory**
  - Linear block code, 1-bit & 2-bit ECC.
- **Implementation**
- **Experiment**
- **Conclusion**

Organization

- **Background**
  - Reed-Solomon, LT-code, 1-bit ECC.
- **Theory**
  - Linear block code, 1-bit & 2-bit ECC.
- **Implementation**
  - ECC implementation for Mica2dot w. CC1000.
- **Experiment**

- **Conclusion**

Organization

- **Background**
  - Reed-Solomon, LT-code, 1-bit ECC.
- **Theory**
  - Linear block code, 1-bit & 2-bit ECC.
- **Implementation**
  - ECC implementation for Mica2dot w. CC1000.
- **Experiment**
  - Outdoor & indoor tests for several ECC.
- **Conclusion**

Background

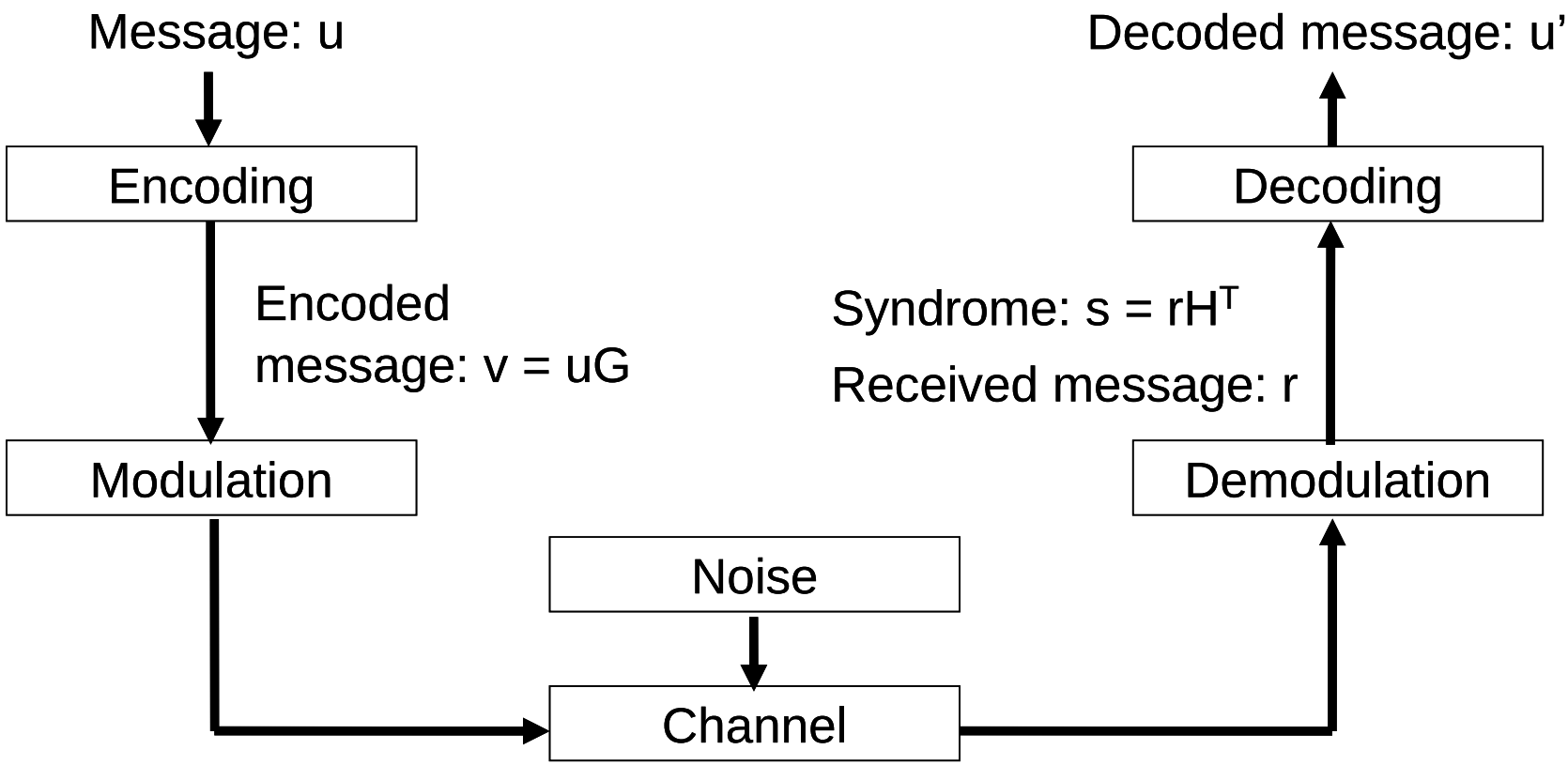
- **Reed-Solomon code, LT code**
  - Better error-correction capability.
  - Complex computation, larger memory space.

Background

- **Reed-Solomon code, LT code**
  - Better error-correction capability.
  - Complex computation, larger memory space.
- **1-bit ECC code for Mica (RFM TR1000)**
  - Handles both 1-bit ECC & DC-balancing.
  - Not efficient for radio that already supports DC-balancing (e.g. CC1000).

Theory

- **Based on linear block code over GF(2).**



Theory

- **Encoding:**
  - Encodes k-bit msg **u** to (k+r)-bit codeword **v**.
  - **v** = **u**G (**u**: msg, G: generator)

Theory

- **Encoding:**
  - Encodes k-bit msg **u** to (k+r)-bit codeword **v**.
  - **v** = **u**G (**u**: msg, G: generator)
- **Decoding:**
  - Decodes (k+r)-bit received data r into k-bit data u'.
  - Calculates syndrome **s** = **r**H^T (r: received msg, H: parity) for locating bit errors.

Theory

- **Locating bit errors:**
  - $s = rH^T = (v+e)H^T = uGH^T + eH^T = eH^T$
  - $GH^T = [I_k : C][C^T : I_r]^T = C + C = 0$
  - Any non-zero syndrome **s** implies an error.

Theory

- **Locating bit errors:**
  - $s = rH^T = (v+e)H^T = uGH^T + eH^T = eH^T$
  - $GH^T = [I_k : C][C^T : I_r]^T = C + C = 0$
  - Any non-zero syndrome **s** implies an error.

- **Correcting bit errors:**
  - If **s** matches *i*-th column of H, invert *i*-th bit of **r**.
  - Otherwise, bit error is not correctable.

Odd-weight-column code

- **Odd-weight-column code is SECDED.**
  - Single-Error-Correction & Double-Error-Detection.
- **Ex: odd-weight-column w. k = 8, r = 5.**

$$G = [I_8 : C] = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$H = [C^T : I_5] = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Odd-weight-column code

- **Encoding:** let message $u = [0100 \quad 0010]$
  - Then, codeword $v = uG = [0100 \quad 0010 \quad 10111]$

Odd-weight-column code

- **Encoding:** let message $u = [0100 \quad 0010]$
  - Then, codeword $v = uG = [0100 \quad 0010 \quad 10111]$
- **TX error:** suppose 2nd bit of **v** is inverted.
  - Received bits $v' = [0000 \quad 0010 \quad 10111]$

$$s = v' H^T = [01011]$$

Odd-weight-column code

- **Encoding:** let message $u = [0100 \quad 0010]$
  - Then, codeword $v = uG = [0100 \quad 0010 \quad 10111]$
- **TX error:** suppose 2nd bit of **v** is inverted.
  - Received bits $v' = [0000 \quad 0010 \quad 10111]$
- **Detecting error:** $s = v'H^T = [01011]$
  - **s** matches 2nd column of H ➔ 2nd bit of **v** inverted.

$$H = [C^T : I_5] = \begin{bmatrix} & & 1 & 1 & 1 & 1 & 1 & 1 & 1 & & & & \\ & 1 & & & 1 & 1 & 1 & 1 & & 1 & & & \\ 1 & & 1 & 1 & & & 1 & 1 & & & 1 & & \\ 1 & 1 & & 1 & & 1 & & 1 & & & & 1 & \\ 1 & 1 & 1 & & 1 & & & 1 & & & & & 1 \end{bmatrix}$$

Odd-weight-column code

- **Error correction:**
  - Calculating correct codeword.

$$v = v' + [0100 \quad 0000 \quad 00000]$$

$$= [0000 \quad 0010 \quad 10111] + [0100 \quad 0000 \quad 00000]$$

$$= [0100 \quad 0010 \quad 10111]$$

Odd-weight-column code

- **Error correction:**
  - Calculating correct codeword.

$$v = v' + [0100 \quad 0000 \quad 00000]$$

$$= [0000 \quad 0010 \quad 10111] + [0100 \quad 0000 \quad 00000]$$

$$= [0100 \quad 0010 \quad 10111]$$

  - Since first k-columns of G is identity matrix,

$$uG = [0100 \quad 0010 \quad 10111]$$

$$u = [0100 \quad 0010]$$

Double-bit error correction code

- **Used (16,8) systematic, quasi-cyclic code.**
  - Can correct 2-bit error and detect 3-bit error (DECTED).

Double-bit error correction code

- **Used (16,8) systematic, quasi-cyclic code.**
  - Can correct 2-bit error and detect 3-bit error (DECTED).
  - Similar to SECDED except decoding.
    - If syndrome **s** matches i^{th} column of H, invert i^{th} bit of r.
    - If **s** matches sum of ith column of H and j^{th} column of H, invert i^{th} and j^{th} bits of **r**.
    - Otherwise, bit error is not correctable.

Double-bit error correction code

- **Encoding:** let message $u = [0100 \quad 0010]$
  - Then, codeword $v = uG = [0100 \quad 0010 \quad 1001 \quad 1100]$

Double-bit error correction code

- **Encoding:** let message $u = [0100 \quad 0010]$
  - Then, codeword $v = uG = [0100 \quad 0010 \quad 1001 \quad 1100]$
- **TX error:** 2nd & 3rd bits of **v** are inverted.
  - Received bits $v' = [0010 \quad 0010 \quad 1001 \quad 1100]$

Double-bit error correction code

- **Encoding:** let message $u = [0100 \quad 0010]$
  - Then, codeword $v = uG = [0100 \quad 0010 \quad 1001 \quad 1100]$
- **TX error:** 2nd & 3rd bits of **v** are inverted.
  - Received bits $v' = [0010 \quad 0010 \quad 1001 \quad 1100]$
- **Detecting error:** $s = v'H^T = [1010 \quad 1111]$

$$H = [C^T : I_8] = \begin{bmatrix} & & 1 & & 1 & & 1 & 1 & 1 & & & & & & & \\ 1 & & & 1 & & 1 & & 1 & & 1 & & & & & & \\ 1 & 1 & & & 1 & & 1 & & & & 1 & & & & & \\ & 1 & 1 & & & 1 & & 1 & & & & 1 & & & & \\ 1 & & 1 & 1 & & & 1 & & & & & & 1 & & & \\ & 1 & & 1 & 1 & & & 1 & & & & & & 1 & & \\ 1 & & 1 & & 1 & 1 & & & & & & & & & 1 & \\ & 1 & & 1 & & 1 & 1 & & & & & & & & & 1 \end{bmatrix}$$

Implementation

- **Platform: Mica2dot with CC1000 radio.**

Implementation

- **Platform: Mica2dot with CC1000 radio.**
- **Three versions of ECC (1-bit & 2-bit)**
  - SECDEC (13, 8) : 8-bit data, 13-bit codeword
  - SECDED (30, 24) : 24-bit data, 30-bit codeword
  - DECTED (16, 8) : 8-bit data, 16-bit codeword

Implementation

- **Platform: Mica2dot with CC1000 radio.**
- **Three versions of ECC (1-bit & 2-bit)**
  - SECDEC (13, 8) : 8-bit data, 13-bit codeword
  - SECDED (30, 24) : 24-bit data, 30-bit codeword
  - DECTED (16, 8) : 8-bit data, 16-bit codeword
- **Implemented within MAC layer providing transparent packet interface.**

Implementation

- **Platform: Mica2dot with CC1000 radio.**
- **Three versions of ECC (1-bit & 2-bit)**
  - SECDEC (13, 8) : 8-bit data, 13-bit codeword
  - SECDED (30, 24) : 24-bit data, 30-bit codeword
  - DECTED (16, 8) : 8-bit data, 16-bit codeword
- **Implemented within MAC layer providing transparent packet interface.**
- **Lookup table of H for faster decoding.**

Implementation

- **Overhead in bytes to transmit due to ECC.**
  - Assumes 20-byte preamble & 36-byte payload.
  - $r_{ecc} = \dfrac{\text{Bytes to be sent}}{\text{Bytes to be encoded}}$

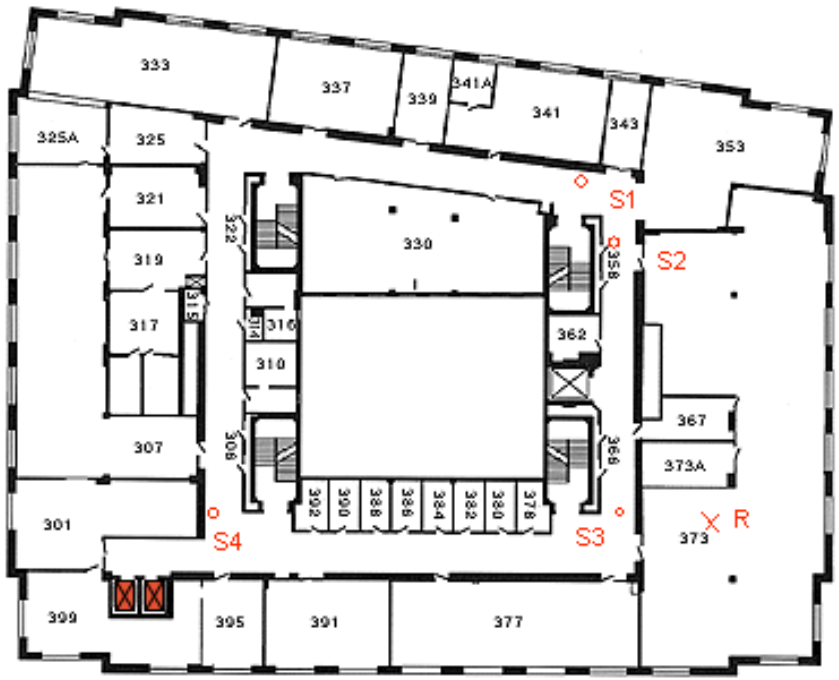
	SECDED (8,13)	SECDED (30,24)	DECTED (16,8)
r_{ecc}	2byte / 1byte	4byte / 3byte	2byte / 1byte
Overhead	64.3%	21.4%	64.3%

Experimental Setup

- **Four versions of ECC MAC were tested**
  - NO FEC
  - SECDED(13,8)
  - SECDED(30,24)
  - DECTED(16,8)
- **TX node sends a packet 5,000 times.**
- **Received data is logged for analysis.**

Experimental Setup

- **Outdoor test**
  - Sender / receiver were 183m apart L.O.S.
- **Indoor test**
  - Four different sender locations in Cory Hall.

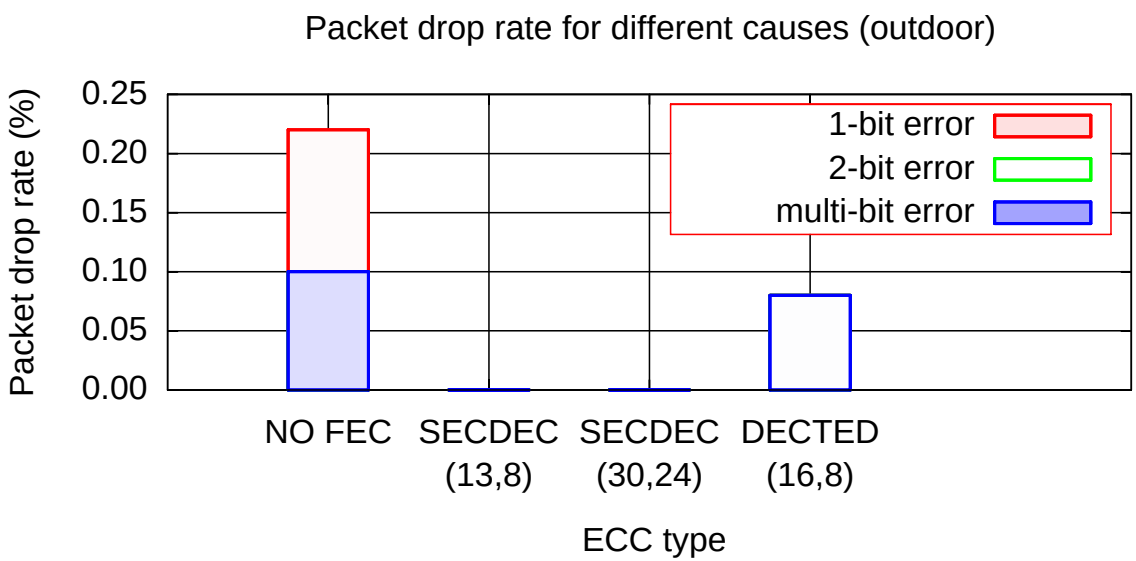


Result (Packet Drop)

- **Our ECC implementation reduces packet error rate (PER), but it has imitations.**

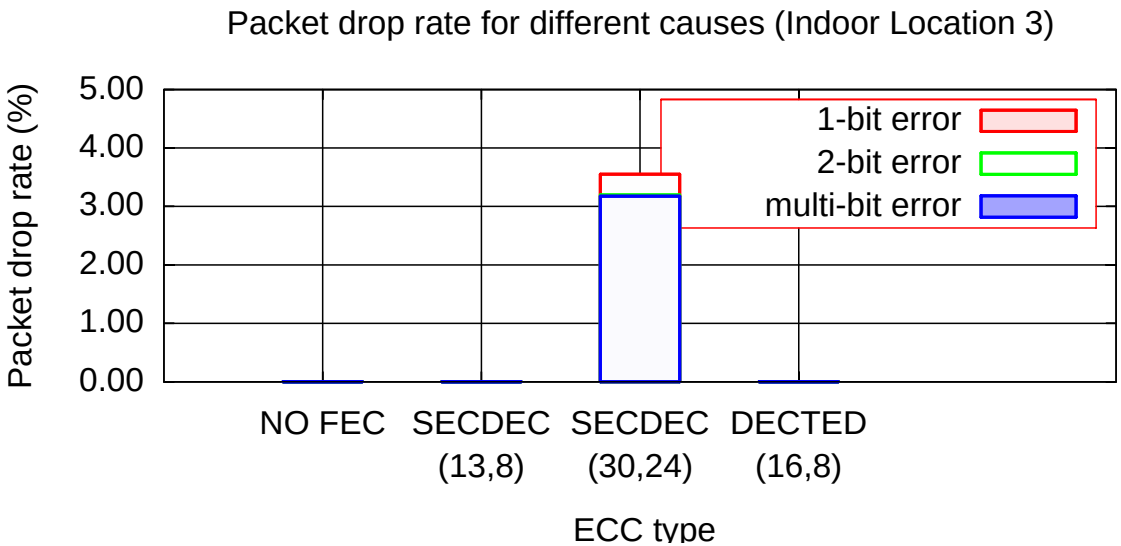
Result (Packet Drop)

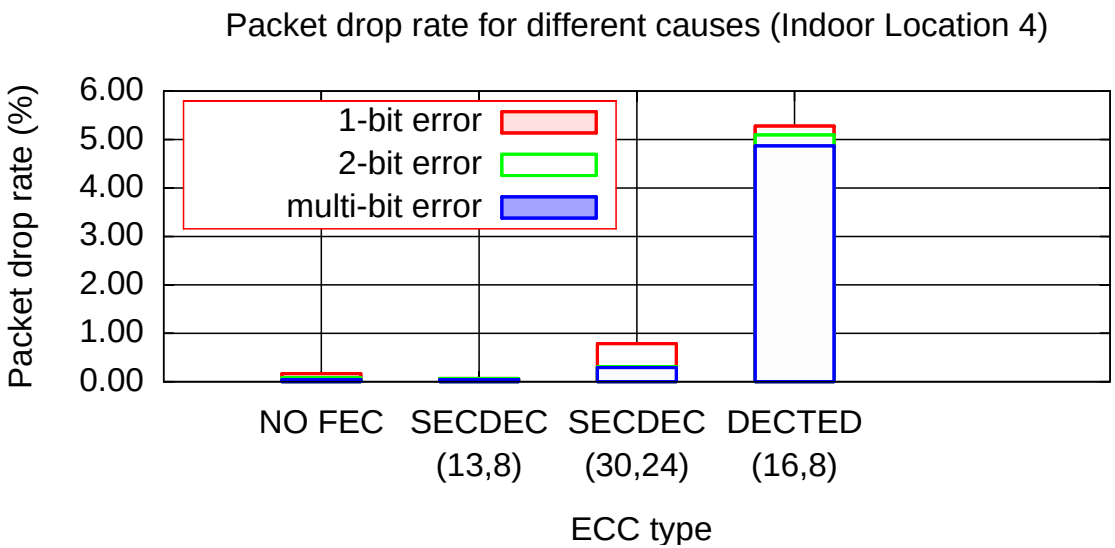
- **Our ECC implementation reduces packet error rate (PER), but it has imitations.**
- **Outdoor: ECC reduces PER to zero.**



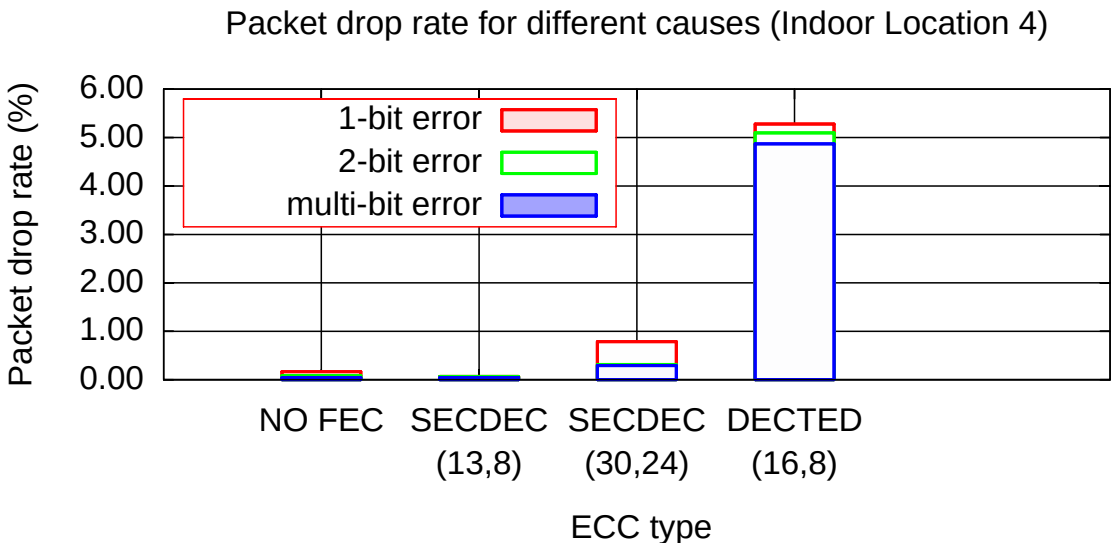
Result (Packet Drop)

- **Indoor: PER > 0 due to multiple-bit errors.**



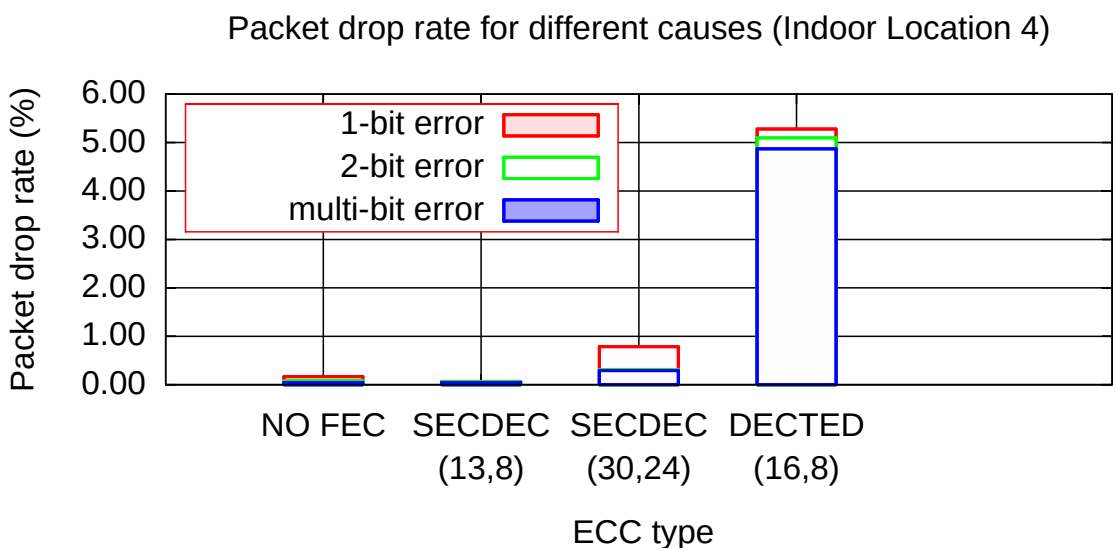


Comparison among ECC schemes



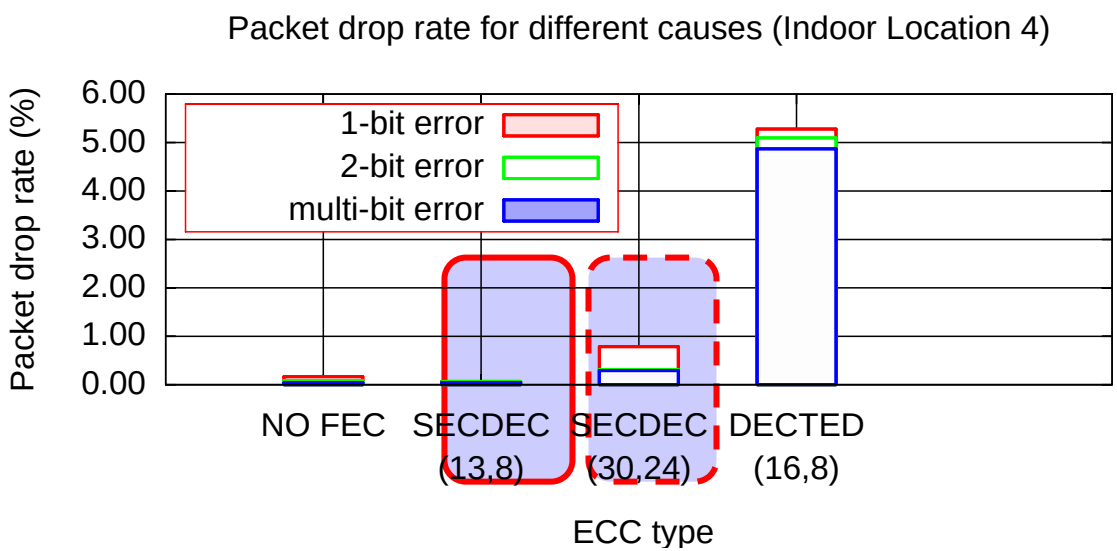
Comparison among ECC schemes

- **SECDED (13,8) has smallest packet drop.**
  - SECDED (30,24) is weaker than SECDED(13,8) although more space-saving.



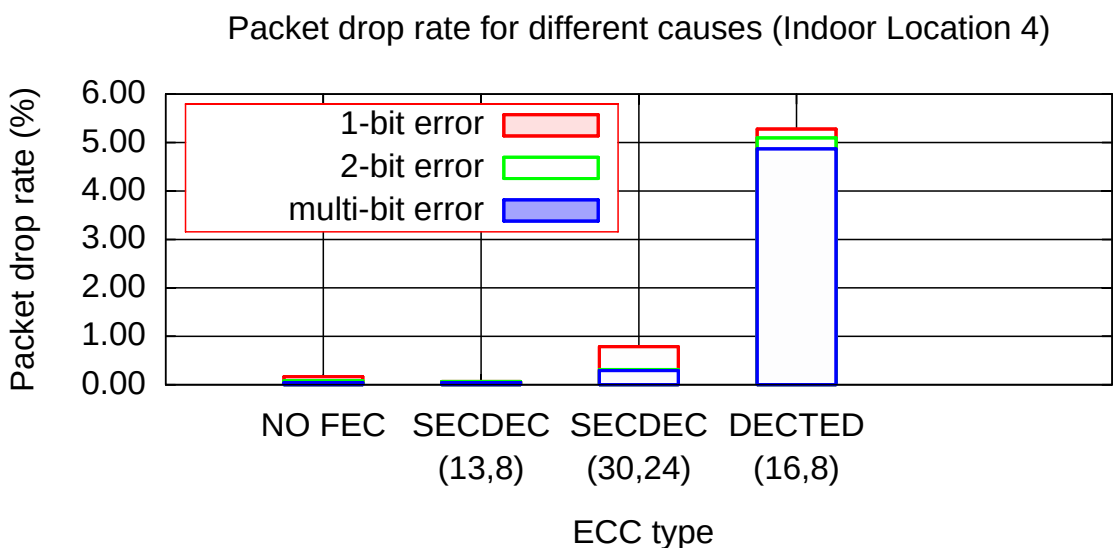
Comparison among ECC schemes

- **SECDED (13,8) has smallest packet drop.**
  - SECDED (30,24) is weaker than SECDED(13,8) although more space-saving.



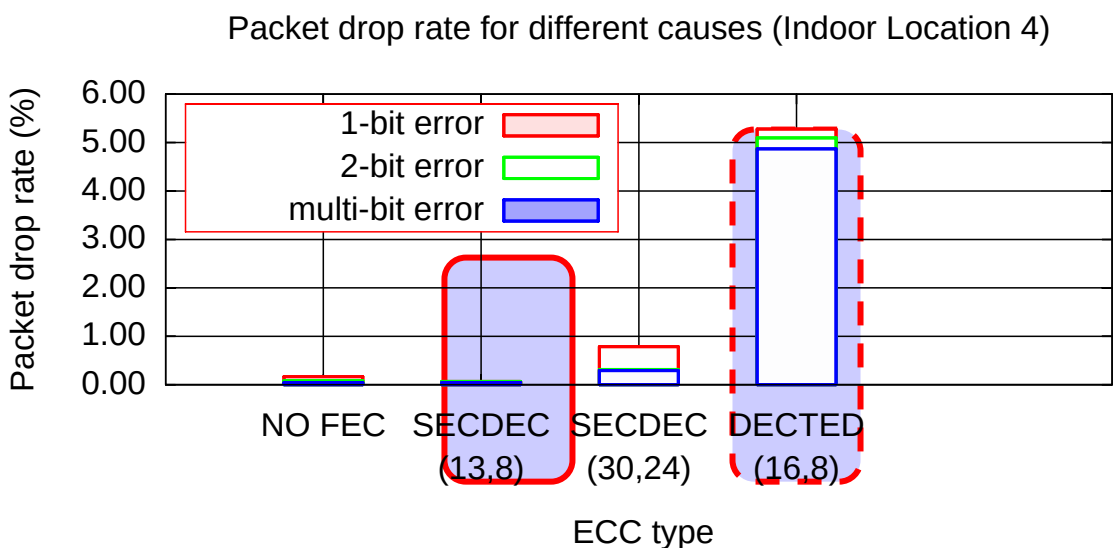
Comparison among ECC schemes

- **SECDED (13,8) has smallest packet drop.**
  - SECDED (30,24) is weaker than SECDED(13,8) although more space-saving.
- **DECTED(16,8) is no better than SECDEC (13,8).**
  - Most errors are single-bit or multiple-bit.



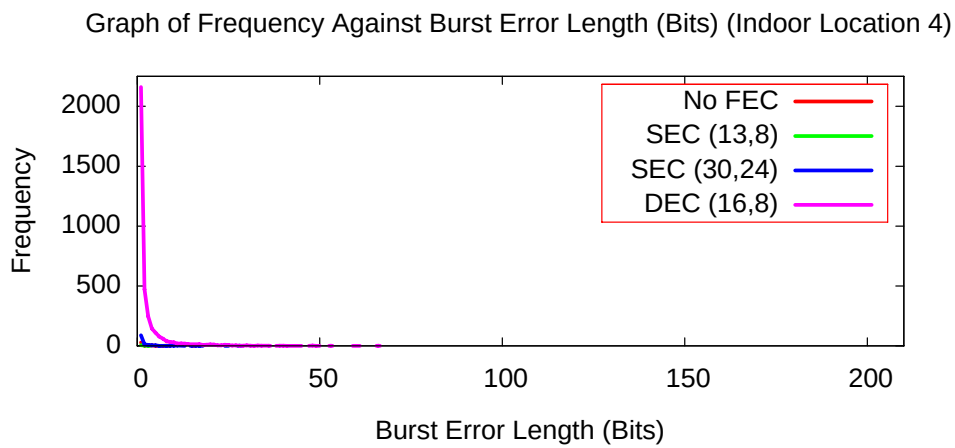
Comparison among ECC schemes

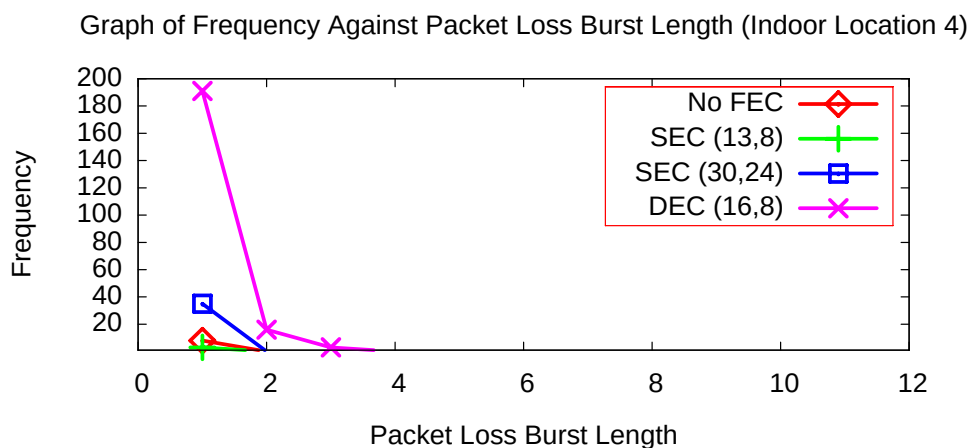
- **SECDED (13,8) has smallest packet drop.**
  - SECDED (30,24) is weaker than SECDED(13,8) although more space-saving.
- **DECTED(16,8) is no better than SECDEC (13,8).**
  - Most errors are single-bit or multiple-bit.



Burst bit errors & packet losses

- **Burst bit errors happen, but frequency of multiple packet drops is low.**
  - A few retransmissions would be enough.





Conclusion

- **A few versions of 1-bit & 2-bit ECC were implemented and tested on CC1000.**
- **ECC reduces packet drop rate, but not effective under burst bit errors.**
- **Under burst bit errors, a few re-TX can be used to further reduce packet drop rate.**