Software and Digital Systems Program - Data Integrity Techniques

FAA Contract DTFACT-11-C-00005

Project Final Presentation
1 October 2013

Philip Koopman
Carnegie Mellon University
koopman@cmu.edu

Co-PIs: Kevin Driscoll, Brendan Hall
Honeywell Laboratories

This work was supported by the Federal Aviation Administration, Aircraft Certification Service, and Assistant Administration for NextGen, William J. Hughes Technical Center, Aviation Research Division, Atlantic City International Airport, New Jersey 08405.

The findings and conclusions in this presentation are those of the author(s) and do not necessarily represent the views of the funding agency. This presentation does not constitute FAA policy.

Agenda

- Background
- Introduction and results summary
- Error code high level comparison
  - Additive, Fletcher, and ATN-32 checksums
  - CRCs
- Summary of industry survey results
- System-level effects and interaction
  - Multiple error protection (parity, dual CRC)
  - Bit encoding interaction
- Mapping to criticality levels
- Discussion
  - Future research suggestions
  - Q&A

Project Overview

- Overall Goals:
  Comparative summary of CRC/Checksum performance
  - Which algorithm, how many error coding bits, what is achieved error detection
  - How do they map to functional integrity levels?

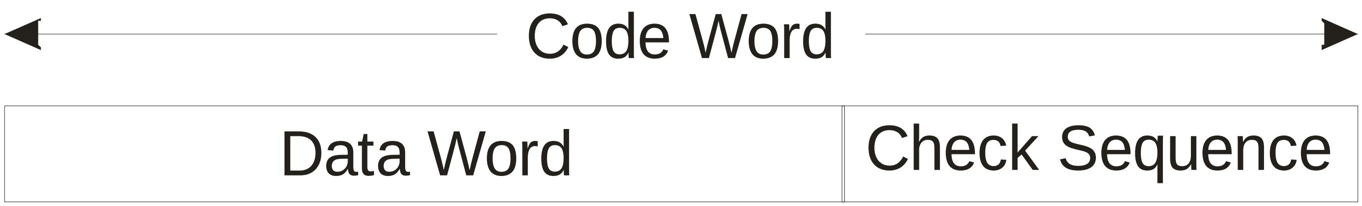
- Specific SOW Goals:
  1. Literature review of criteria, parameters, and tradeoffs
  2. Study CRC parameters and functional integrity levels
  3. Study checksum parameters and functional integrity levels
  4. Recommendations and suggest future work directions

Investigators

- Philip Koopman – CRC/Checksum Evaluation
  - Assoc. Prof of ECE, Carnegie Mellon University (PA)
  - Embedded systems research, emphasizing dependability & safety
  - Industry experience with transportation applications
- Kevin Driscoll – Industry Surveys & Criticality Mapping
  - Engineering Fellow, Honeywell Laboratories (MN)
  - Ultra-dependable systems research & security
  - Extensive data communications experience for aviation
- Brendan Hall – Industry Surveys & Criticality Mapping
  - Engineering Fellow, Honeywell Laboratories (MN)
  - Fault tolerant system architectures & devlopment process
  - Extensive experience with aviation computing systems

BACKGROUND

Error Detection Overview



- Error Detection: pick a “good” mathematical hash function
  - Check sequence is hash(data word), e.g., sent after data word in a message
  - Receiver re-computes the hash and checks for code word validity:
    - If hash(data word) == Check Sequence then assume data word is error free
- Common hash function approaches:
  - Checksum: various schemes to “add up” chunks of data word
  - CRC: use a linear feedback shift register arrangement to mix bits
- Want good “bit mixing” to detect combinations of bit errors
  - Undetected error fraction is ratio of bit errors that are undetectable
  - Usually want good error detection properties for *small numbers of bit errors*
  - Hamming Distance (HD): min # bit errors that might be undetected
    - E.g., HD=4 means 100% of 1-, 2-, 3-bit errors are detected

Terminology– Error Codes

- Error Coding
  - Generic term for parity, checksum, or CRC calculation
- Checksum
  - Error code based primarily on addition of data chunks
  - E.g., one’s complement sum, Adler, Fletcher, ATN-32
- CRC: Cyclic Redundancy Code
  - Error code based upon polynomial division over GF(2)
- Polynomial: CRC feedback polynomial
  - Primitive polynomial generates maximal length LFSR sequence
    - (These slides use ‘implicit +1’ notation for hexadecimal values)
  - Polynomial may have prime factors (e.g., divisible by (x+1))
    - Factors of polynomial influence error detection effectiveness
- Code Size
  - Number of bits in the error code result
  - By construction, this is same as check sequence size

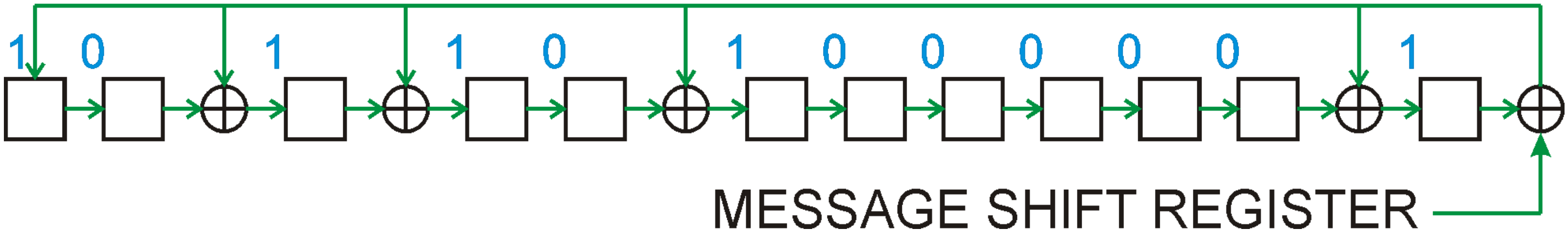
Terminology – Error Detection Effectiveness

- Hamming Distance:
  - Minimum number of bit errors that is undetectable
  - Hamming Weight is a number of undetectable errors for a given number of bit errors (e.g., 4161 undetectable 4-bit errors)
- BER: Bit Error Ratio
  - Probability per bit of bisymmetric inversion error (e.g., 10^{-6})
    - Usually assumes independent random inversions
  - P_{ud}: Probability of undetected error, assuming some BER
    - NOTE: even if very small, there are many messages across system life
- Burst Error
  - Arbitrary corruption of k consecutive bits
- Error Check
  - Determining whether a check sequence matches the data word
  - Compute code on data word and match result to check sequence

Classical CRC Overview

- Cyclic Redundancy Code operation
  - Computes a (non-secure) message digest using shift and XOR
  - This is a hardware implementation of polynomial division

POLYNOMIAL: 1011 0100 0001 = 0xB41



(the “+1” is implicit in the hex value)

$0xB41 = x^{12}+x^{10}+x^{9}+x^{7}+x \; +1$

$= (x+1)(x^{3}+x^{2}+1)\,(x^{8}+x^{4}+x^{3}+x^{2}+1)$

(presentation uses implicit +1 notation for hex values)

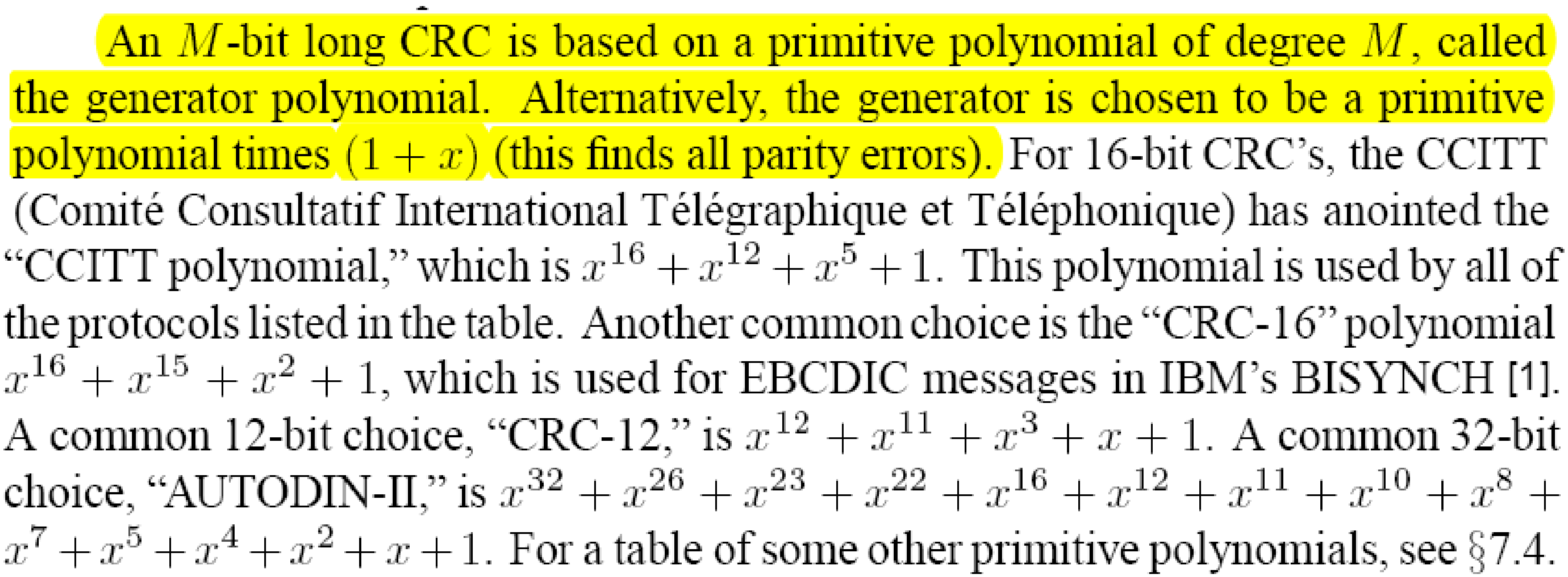
MESSAGE PAYLOAD	CRC

- Detected error if received digest doesn't match CRC of payload

Isn’t Optimality A Long-Solved Problem? (No)

- Compute power wasn’t available to evaluate error code performance when they were invented
  - So originators relied on mathematical analysis…
    … which gives bounds and heuristics, but not exact answers
  - Exact enumeration of error detection performance is recent
- Literature is poorly accessible to computer engineers
  - Much of it is written in dense math that is hard to apply
    - Many papers concentrate on interesting math rather than applicable results
    - Sources that are readily accessible may have incorrect folklore
  - Practitioners mostly use what someone else previously used
    - Assume it must be good (often this is incorrect)
    - Unaware of limitations or mistakes in previous work
  - Practical application knowledge poorly documented
    - There are a handful of protocol experts who know this stuff, mostly originating in the European rail domain

A Flawed, But Typical, CRC Selection Method

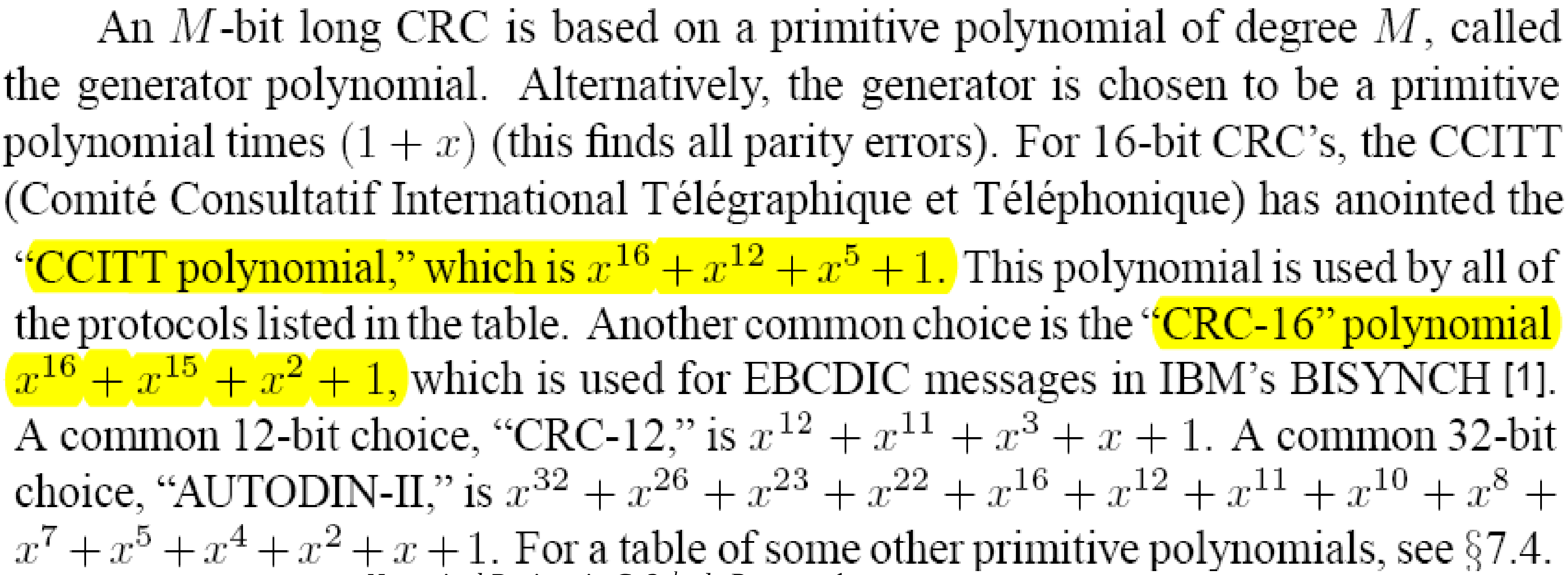


An M-bit long CRC is based on a primitive polynomial of degree M, called the generator polynomial. Alternatively, the generator is chosen to be a primitive polynomial times $(1+x)$ (this finds all parity errors). For 16-bit CRC's, the CCITT (Comité Consultatif International Télégraphique et Téléphonique) has anointed the "CCITT polynomial," which is $x^{16}+x^{12}+x^5+1$. This polynomial is used by all of the protocols listed in the table. Another common choice is the "CRC-16" polynomial $x^{16}+x^{15}+x^2+1$, which is used for EBCDIC messages in IBM's BISYNCH [1]. A common 12-bit choice, "CRC-12," is $x^{12}+x^{11}+x^3+x+1$. A common 32-bit choice, "AUTODIN-II," is $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$. For a table of some other primitive polynomials, see §7.4.

» ***Numerical Recipes in C, 2nd ed.*, Press et al.**

- But, there are some problems:
  - Many good polynomials are not primitive nor divisible by (x+1)
  - Divisibility by (x+1) doubles undetected error rate for even # of bit errors

A Typical Polynomial Selection Method

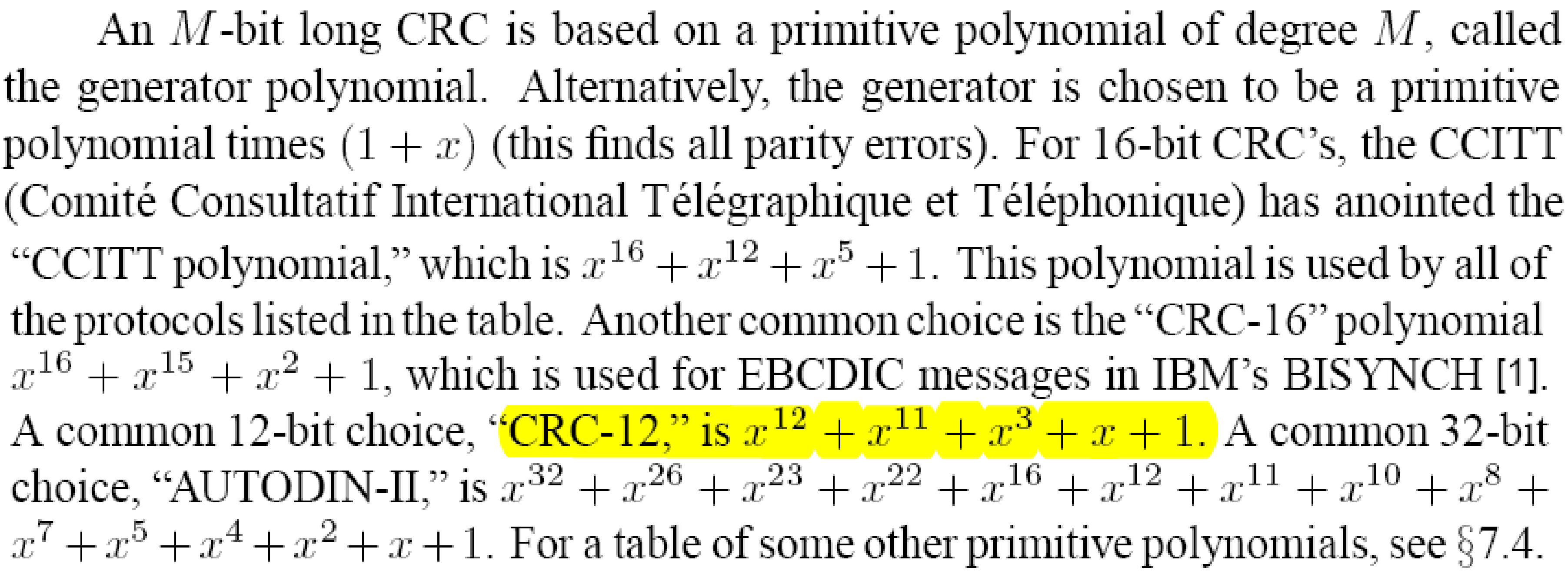


An M-bit long CRC is based on a primitive polynomial of degree M, called the generator polynomial. Alternatively, the generator is chosen to be a primitive polynomial times $(1 + x)$ (this finds all parity errors). For 16-bit CRC's, the CCITT (Comité Consultatif International Télégraphique et Téléphonique) has anointed the "CCITT polynomial," which is $x^{16} + x^{12} + x^5 + 1$. This polynomial is used by all of the protocols listed in the table. Another common choice is the "CRC-16" polynomial $x^{16} + x^{15} + x^2 + 1$, which is used for EBCDIC messages in IBM's BISYNCH [1]. A common 12-bit choice, "CRC-12," is $x^{12} + x^{11} + x^3 + x + 1$. A common 32-bit choice, "AUTODIN-II," is $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$. For a table of some other primitive polynomials, see §7.4.

» *Numerical Recipes in C, 2nd ed.*, Press et al.

- But, there are some problems:
  - Many good polynomials are not primitive nor divisible by (x+1)
  - Divisibility by (x+1) doubles undetected error rate for even # of bit errors
  - How do you know which competing polynomial to pick?

A Typical Polynomial Selection Method

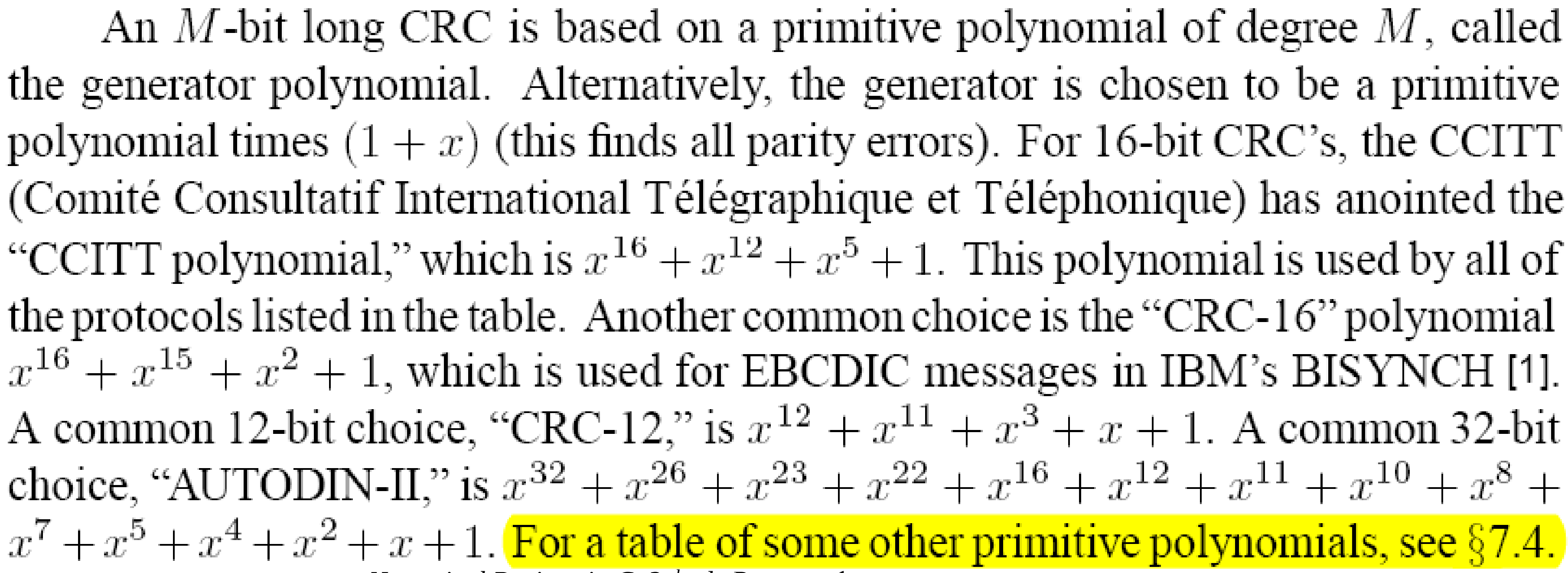


An M-bit long CRC is based on a primitive polynomial of degree M, called the generator polynomial. Alternatively, the generator is chosen to be a primitive polynomial times $(1 + x)$ (this finds all parity errors). For 16-bit CRC's, the CCITT (Comité Consultatif International Télégraphique et Téléphonique) has anointed the "CCITT polynomial," which is $x^{16} + x^{12} + x^5 + 1$. This polynomial is used by all of the protocols listed in the table. Another common choice is the "CRC-16" polynomial $x^{16} + x^{15} + x^2 + 1$, which is used for EBCDIC messages in IBM's BISYNCH [1]. A common 12-bit choice, "CRC-12," is $x^{12} + x^{11} + x^3 + x + 1$. A common 32-bit choice, "AUTODIN-II," is $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$. For a table of some other primitive polynomials, see §7.4.

» *Numerical Recipes in C, 2nd ed*, Press et al.

- But, there are some problems:
  - Many good polynomials are not primitive nor divisible by (x+1)
  - Divisibility by (x+1) doubles undetected error rate for even # of bit errors
  - How do you know which competing polynomial to pick?
  - This CRC-12 polynomial is incorrect (there is a missing +x^2)

A Typical Polynomial Selection Method



An M-bit long CRC is based on a primitive polynomial of degree M, called the generator polynomial. Alternatively, the generator is chosen to be a primitive polynomial times $(1 + x)$ (this finds all parity errors). For 16-bit CRC's, the CCITT (Comité Consultatif International Télégraphique et Téléphonique) has anointed the "CCITT polynomial," which is $x^{16} + x^{12} + x^5 + 1$. This polynomial is used by all of the protocols listed in the table. Another common choice is the "CRC-16" polynomial $x^{16} + x^{15} + x^2 + 1$, which is used for EBCDIC messages in IBM's BISYNCH [1]. A common 12-bit choice, "CRC-12," is $x^{12} + x^{11} + x^3 + x + 1$. A common 32-bit choice, "AUTODIN-II," is $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$. For a table of some other primitive polynomials, see §7.4.

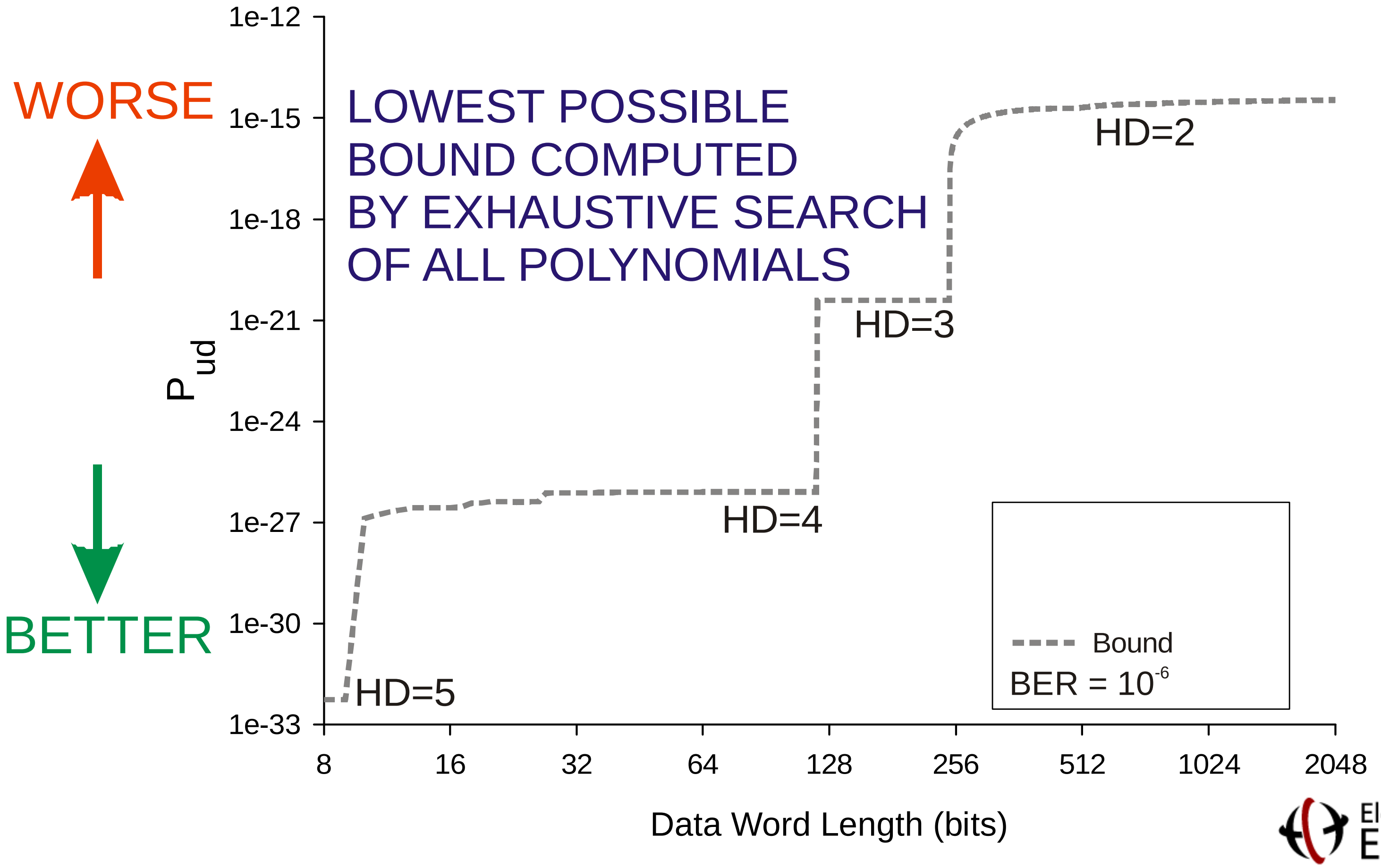
» *Numerical Recipes in C, 2nd ed.*, Press et al.

- But, there are some problems:
  - Many good polynomials are not primitive nor divisible by (x+1)
  - Divisibility by (x+1) doubles undetected error rate for even # of bit errors
  - How do you know which competing polynomial to pick?
  - This CRC-12 polynomial is incorrect (there is a missing $+x^2$)
  - You can't pick at random from a list!

NOTE: 3rd Edition fixed these problems based on our feedback

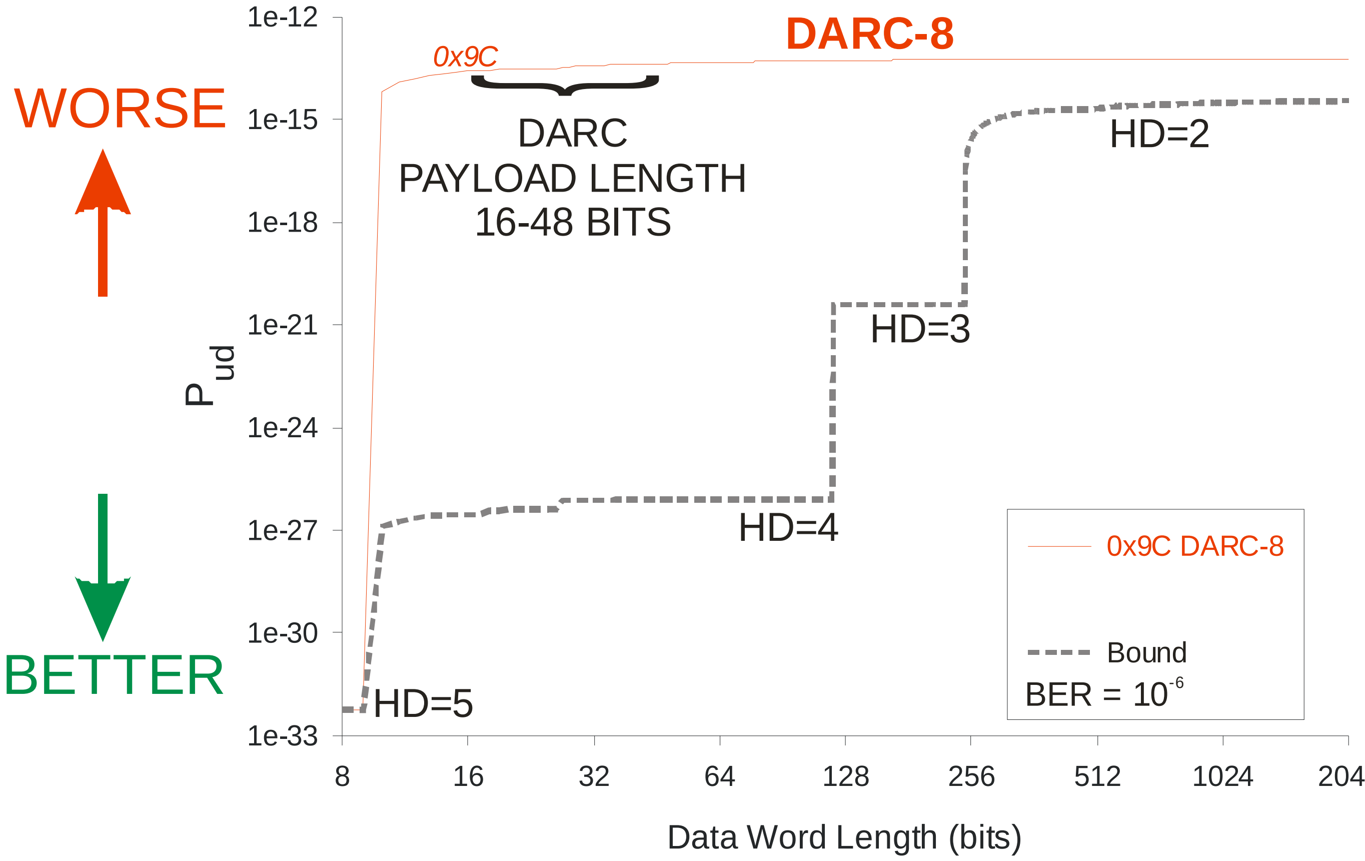
Example – 8 Bit Polynomial Choices

- P_{ud} (undetected error rate) is one way to evaluate CRC effectiveness
  - Uses Hamming weights of polynomials
  - Uses assumed random independent Bit Error Rate (BER)



What Happens When You Get It Wrong?

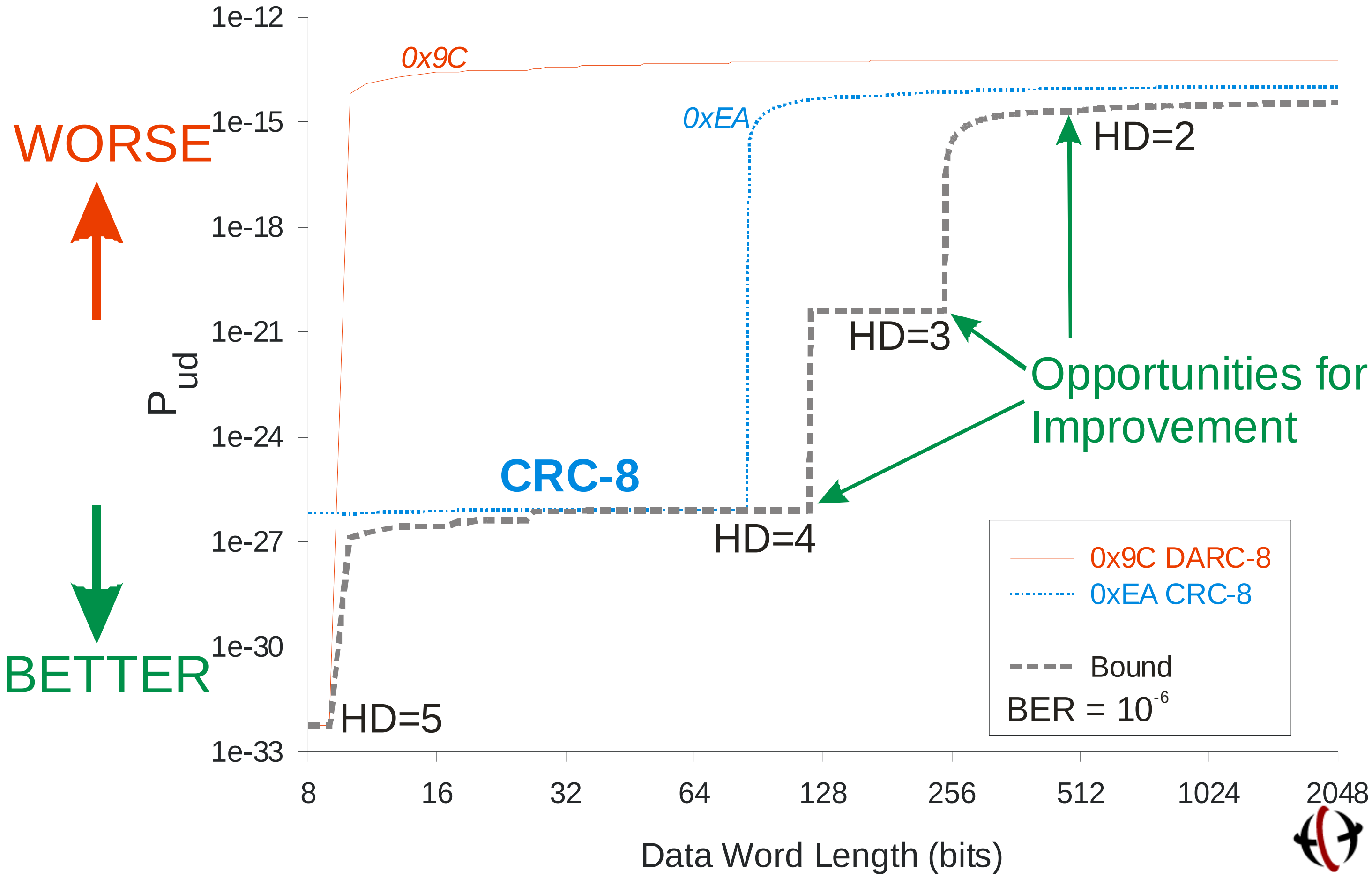
- DARC (Data Radio Channel), ETSI, October 2002
  - DARC-8 polynomial is optimal for 8-bit payloads
  - BUT, DARC uses 16-48 bit payloads, and misses some 2-bit errors
  - Could have detected all 2-bit and 3-bit errors with same size CRC!



Source:
Koopman, P. & Chakravarty, T., "Cyclic Redundancy Code (CRC) Polynomial Selection for Embedded Networks,“ DSN04, June 2004

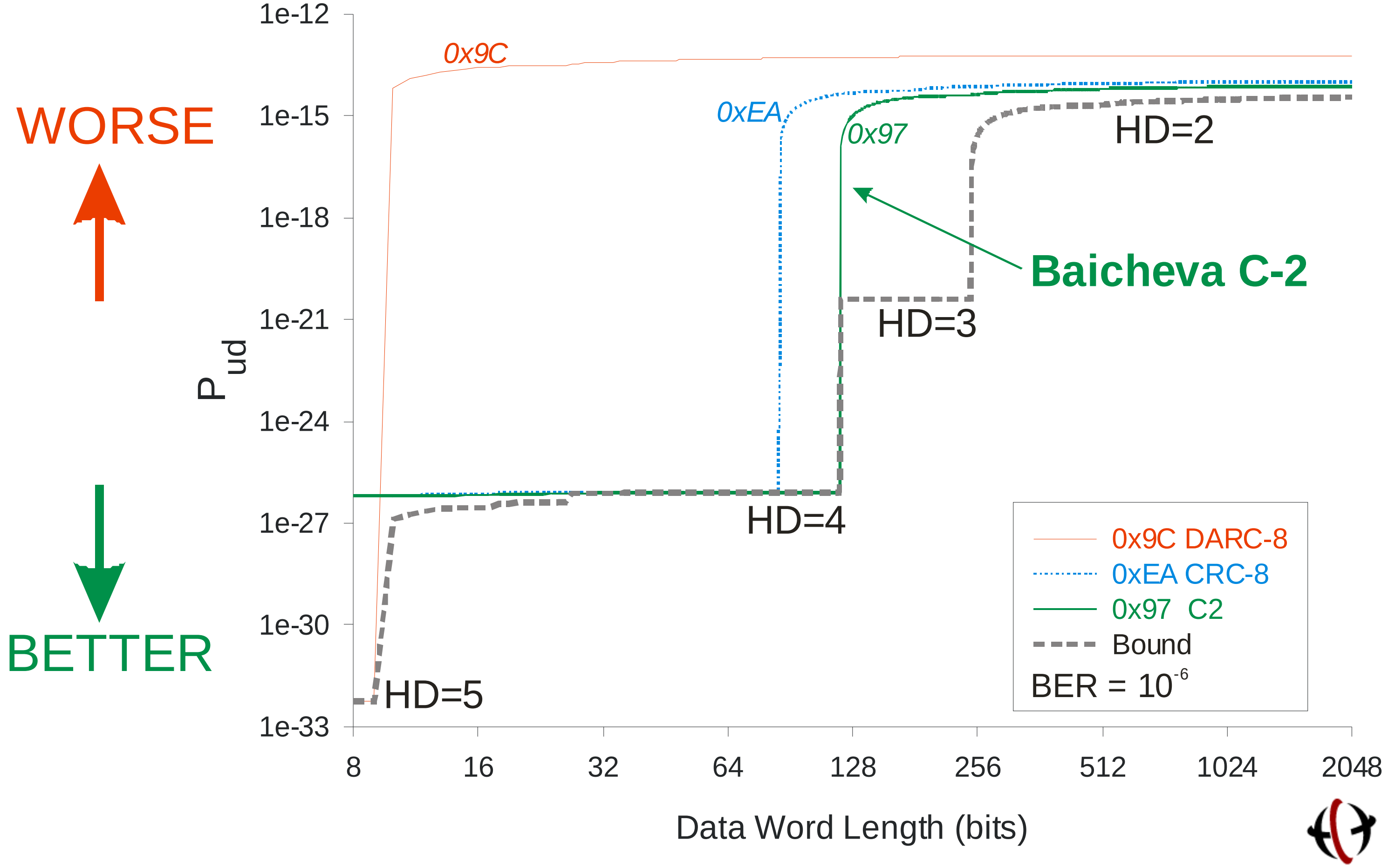
CRC-8 Is Better

- CRC-8, 0xEA, is in common use
  - Good for messages up to size 85
  - But, room for improvement at longer lengths. Can we do better?



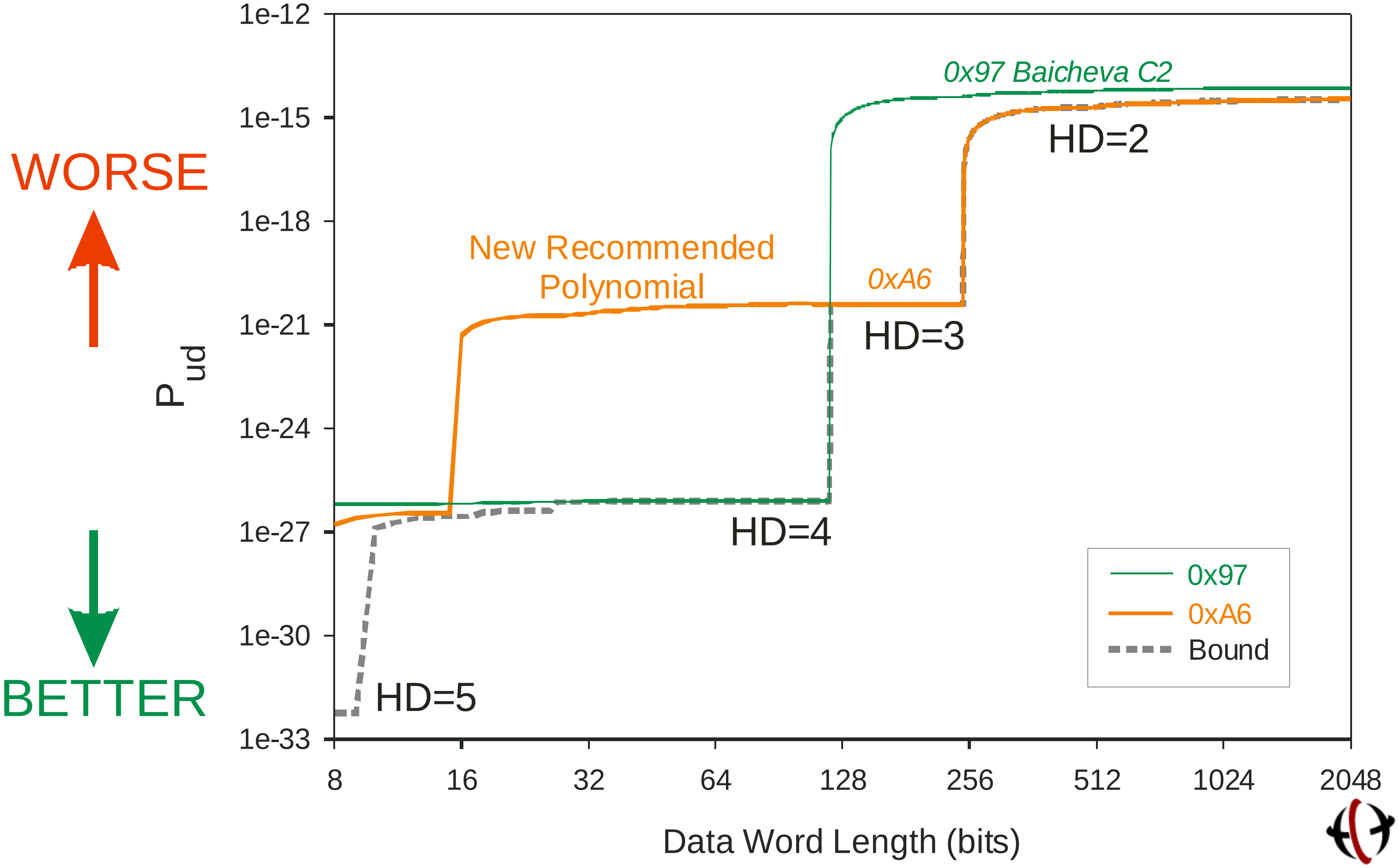
Baicheva’s Polynomial C2 Is Yet Better

- [Baicheva98] proposed polynomial C2, 0x97
  - Recommended as good polynomial to length 119
  - Dominates 0xEA (better P_{ud} at every length)



But What If You Want the HD=3 Region?

- No published polynomials proposed for HD=3 region
  - We found that 0xA6 has good performance
  - Better than C2 and near optimal at all lengths of 120 and above



Project Technical Approach

- Literature Survey
  - Identify candidate codes and candidate evaluation criteria
- Survey aviation industry
  - Data characteristics, usage scenarios, fault exposure
- Analytically evaluate error code effectiveness
  - In light of codes, data, usage, fault exposure, evaluation criteria
  - Using a mixture of analytic and fault simulation approaches
  - Document weak practices; propose better alternatives
- Map effectiveness to DO-178B software levels
  - Taking into account usage scenarios
- Identify future work topics

PROJECT RESULTS

High Level Results

- It's all about the Hamming Distance (HD)
  - For BER model, number of 100% detected error bits dominates error detection ratio for higher numbers of bit errors
    - Bit Error Ratio (BER) model assumes random independent bit errors
  - CRCs can have *much* better HD than checksums
- What we saw in industry surveys
  - Mostly use of standard CRCs
    - Significantly better HD would be available if using a better CRC
  - A few near-optimal and a few problematic examples
- Mapping to criticality levels:
  - There is no one-size-fits all answer per criticality level
    - (DO-178b criticality level terminology fine point is discussed later)
  - Based on system's message length, BER, fault containment…
    - … and secondary effects that may come into play …
    - Select error code based on HD of required coverage

Literature Survey Example Result

- Starting point: P. Koopman, T. Chakravathy, “Cyclic Redundancy Code (CRC) Polynomial Selection for Embedded Networks”, The International Conference on Dependable Systems and Networks, DSN-2004.

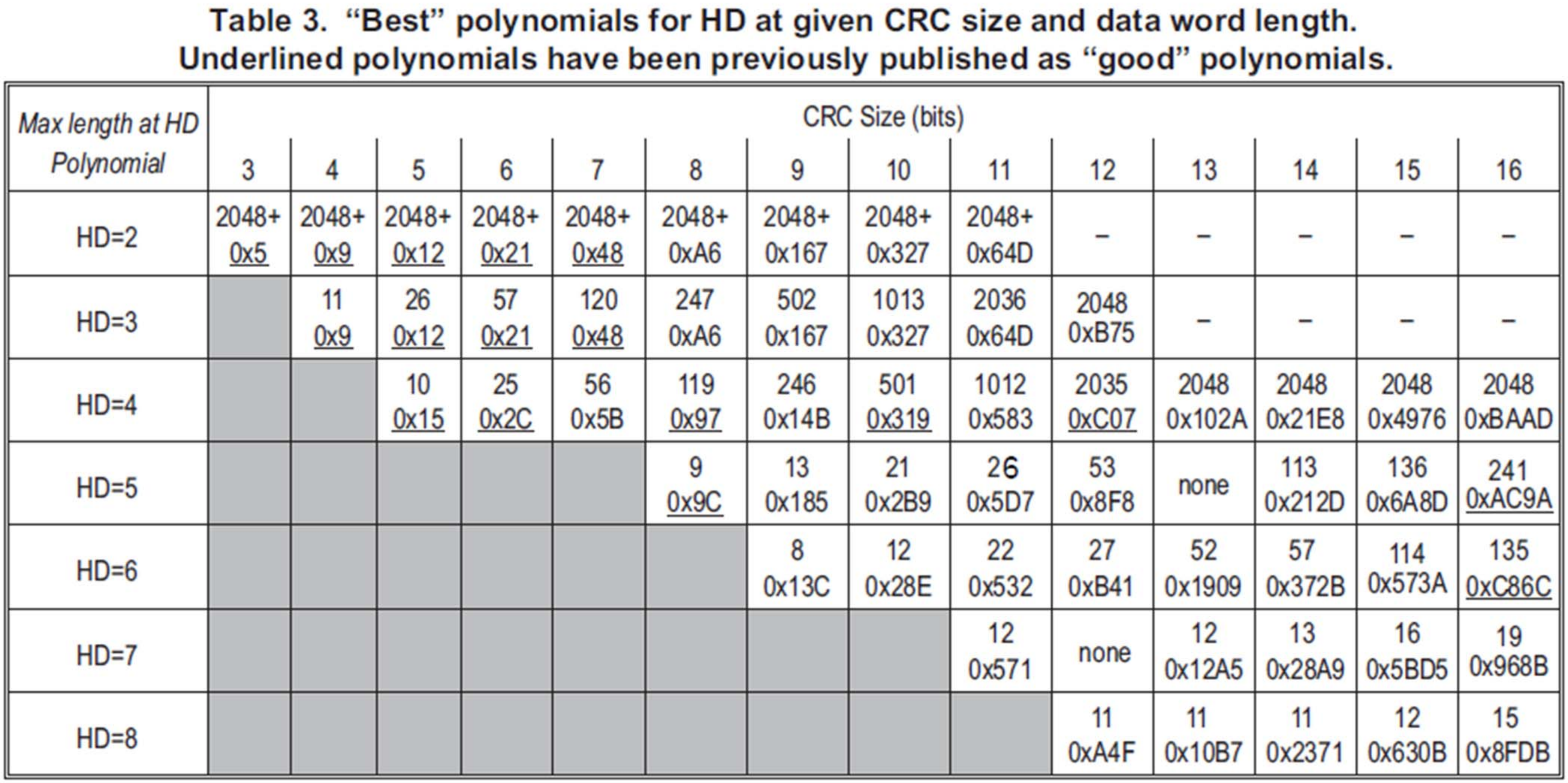


Table 3. “Best” polynomials for HD at given CRC size and data word length.
Underlined polynomials have been previously published as “good” polynomials.

Max length at HD *Polynomial*	CRC Size (bits)													
	3	4	5	6	7	8	9	10	11	12	13	14	15	16
HD=2	2048+ 0x5	2048+ 0x9	2048+ 0x12	2048+ 0x21	2048+ 0x48	2048+ 0xA6	2048+ 0x167	2048+ 0x327	2048+ 0x64D	–	–	–	–	–
HD=3		11 0x9	26 0x12	57 0x21	120 0x48	247 0xA6	502 0x167	1013 0x327	2036 0x64D	2048 0xB75	–	–	–	–
HD=4			10 0x15	25 0x2C	56 0x5B	119 0x97	246 0x14B	501 0x319	1012 0x583	2035 0xC07	2048 0x102A	2048 0x21E8	2048 0x4976	2048 0xBAAD
HD=5						9 0x9C	13 0x185	21 0x2B9	26 0x5D7	53 0x8F8	none	113 0x212D	136 0x6A8D	241 0xAC9A
HD=6							8 0x13C	12 0x28E	22 0x532	27 0xB41	52 0x1909	57 0x372B	114 0x573A	135 0xC86C
HD=7									12 0x571	none	12 0x12A5	13 0x28A9	16 0x5BD5	19 0x968B
HD=8										11 0xA4F	11 0x10B7	11 0x2371	12 0x630B	15 0x8FDB

Further Evaluation Methodology

- Checksums evaluated via Monte Carlo simulation
  - Mersenne Twister random number generator
  - 1e9 (10^9) to 1e12 total trials for checksums
    - More trials if required to get at least 1000 undetected error events at that data word length for that checksum type
    - At least 1000 potential data points worth of trials at length N+1 words to conclude that length N probably is longest length at a particular HD.
- CRCs evaluated using analytically exact computations
  - All error bit patterns exhaustively evaluated for undetected errors
- Results compared against published results
  - Checksums (except ATN-32) compared to journal paper
  - CRCs previously compared against all known published results

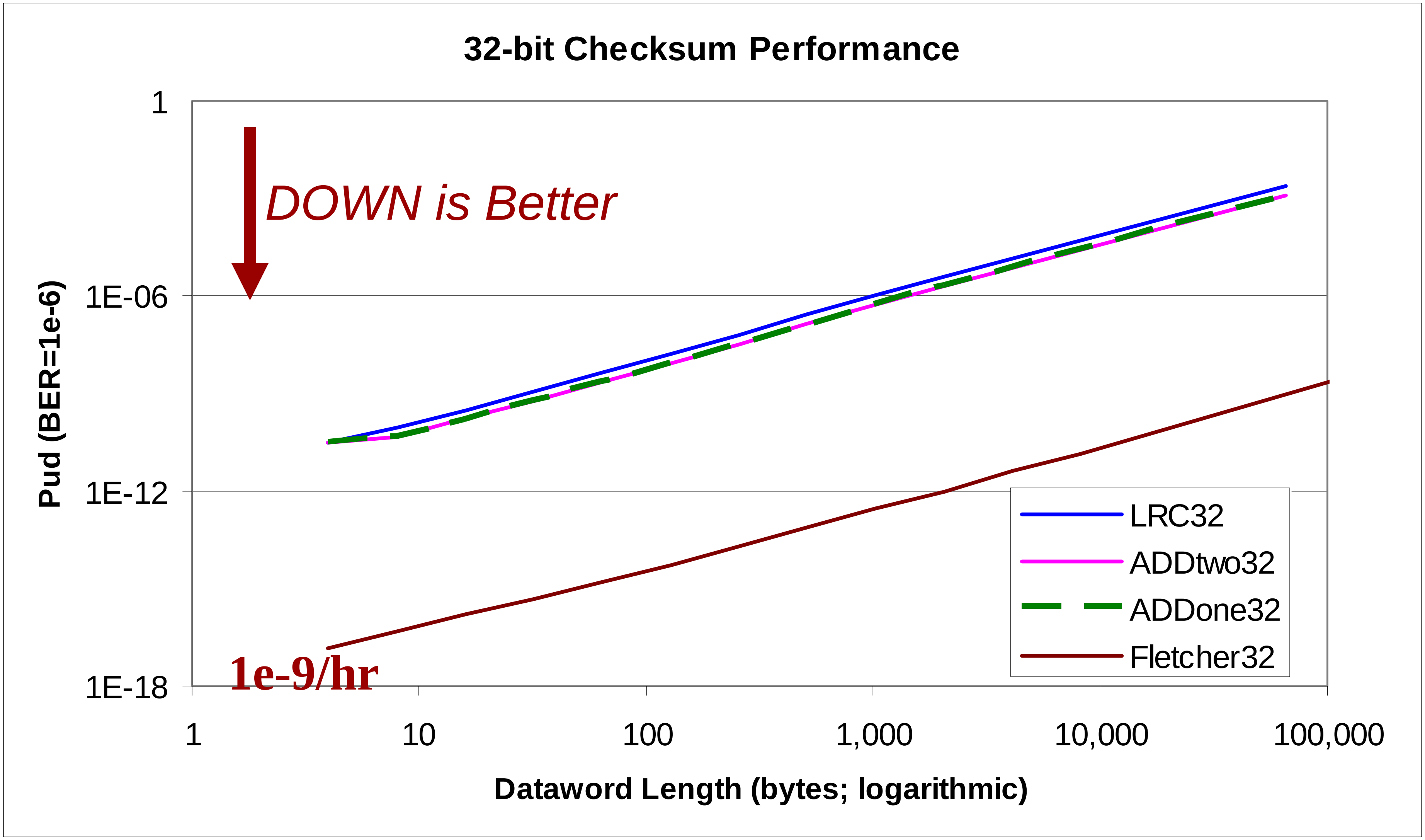
Checksum Results

- Checksum performance is data dependent
  - Worst case is 50% one bits and 50% zero bits
  - (Previously published; reconfirmed via simulation)

- Traditional checksum types across all "data chunks":
  - LRC: Longitudinal Redundancy Check – XOR all data chunks
  - ADDtwo: Integer two's complement addition
  - ADDone: One's complement addition ("carry bit wrap-around")
  - Fletcher: two half-length running ADDone sums:
    - SUMA = SUMA $+_{\text{one}}$ next data chunk
    - SUMB = SUMB $+_{\text{one}}$ SUMA

- Results graphed based on Bit Error Ratio (BER)
  - Probability of Undetected Error (Pud)
  - Pud takes into account how likely an m-bit error is
  - Small numbers of bit errors are more likely

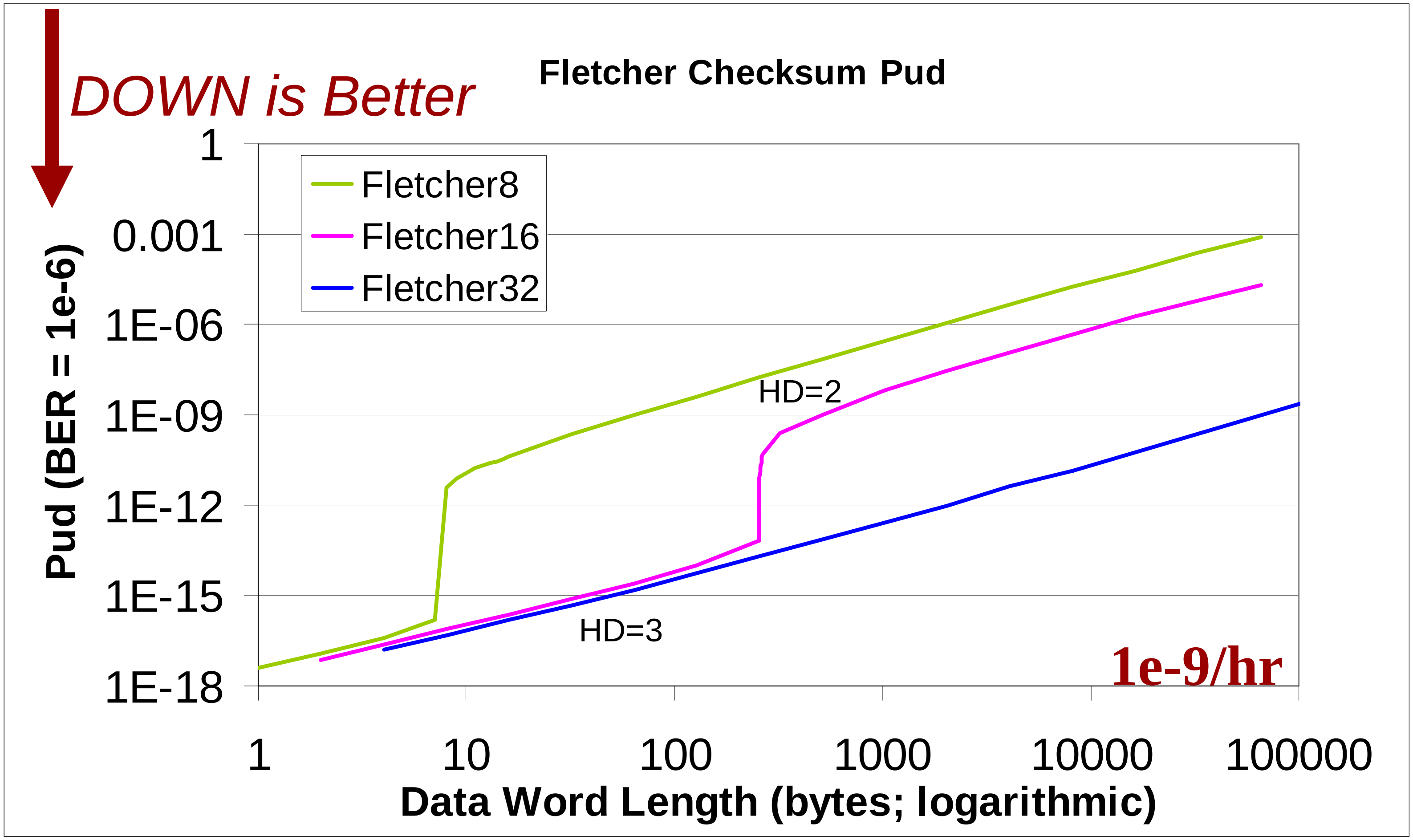
Fletcher Is A Better Checksum



Note: Pud is per message; perhaps up to 10^9 msgs/hr

Fletcher Achieves HD=2 or HD=3

- Break point depends upon size
  - E.g., Fletcher 8 is two 4-bit running sums

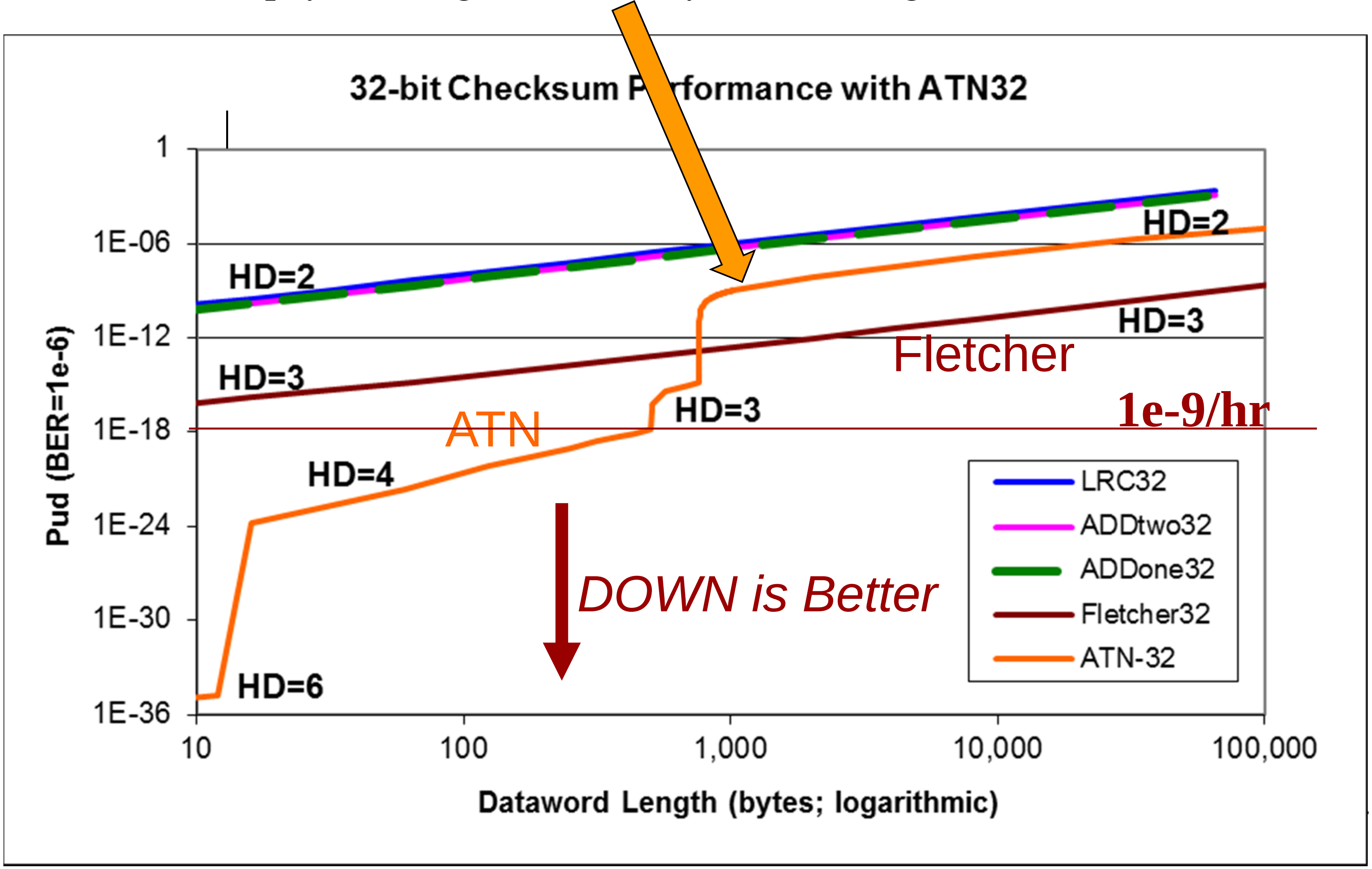


ATN-32 Checksum

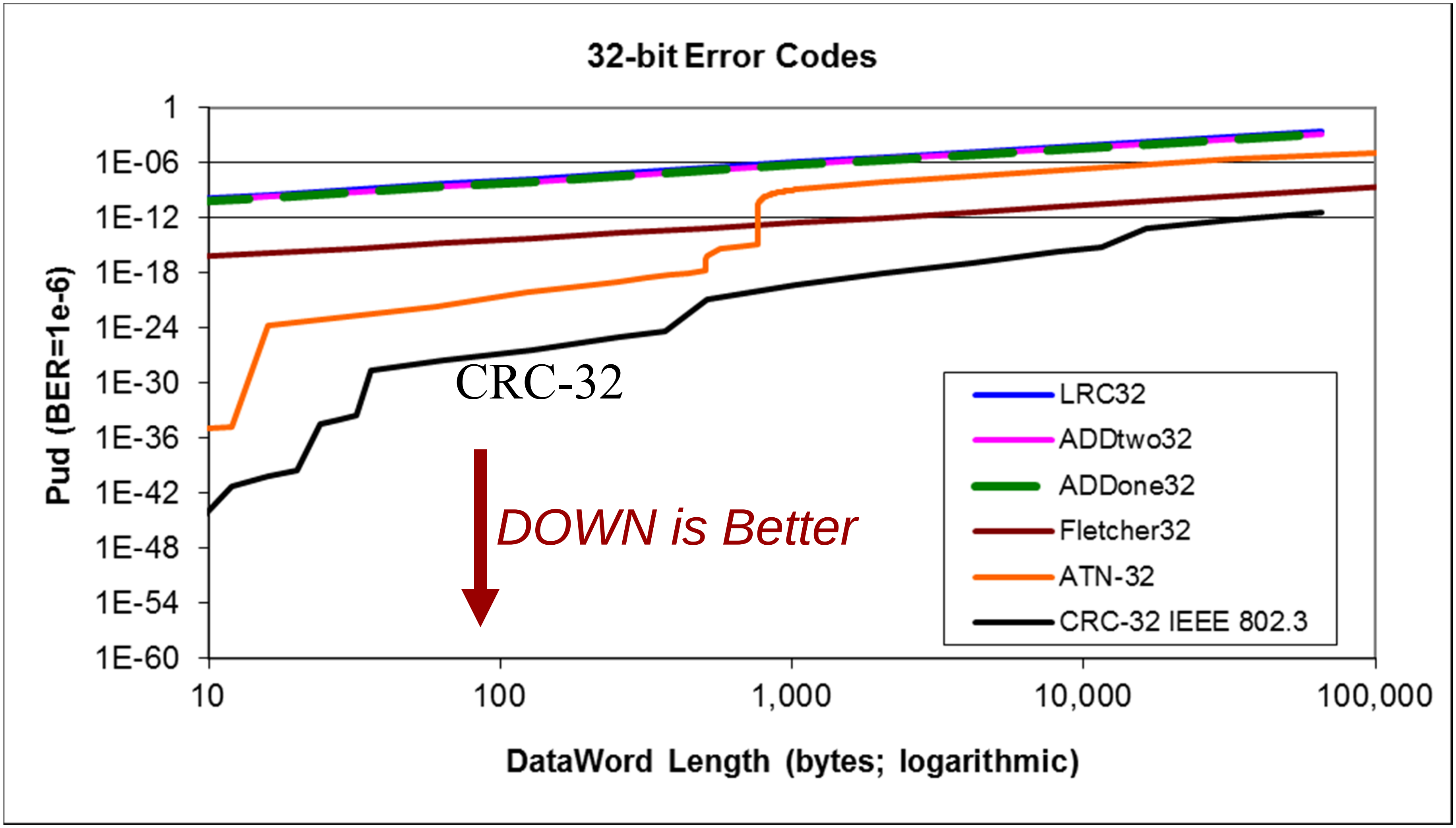
- ATN-32 is a modified Fletcher Checksum
  - [AN/466] "Manual on Detailed Technical Specifications for the Aeronautical Telecommunication Network (ATN) using ISO/OSI Standards and Protocols, Part 1 – Air-Ground Applications." Doc 9880, International Civil Aviation Organization, Montreal, First Edition, 2010.
- Uses four running 8-bit sums:
  - SUMA = SUMA $+_{\text{one}}$ next data chunk
  - SUMB = SUMB $+_{\text{one}}$ SUMA
  - SUMC = SUMC $+_{\text{one}}$ SUMB
  - SUMD = SUMD $+_{\text{one}}$ SUMC
  - Final bytes are a defined mixture of above four sums
- Performance is better than Fletcher32 for short lengths
  - Appears to be optimized for short messages
  - But significantly worse than Fletcher32 for long lengths!

ATN-32 Is Only Good At Short Lengths

- HD=3 up to 504 bytes; HD=2 at and above 760 bytes
- Max ATN payload might be 1471 bytes including checksum

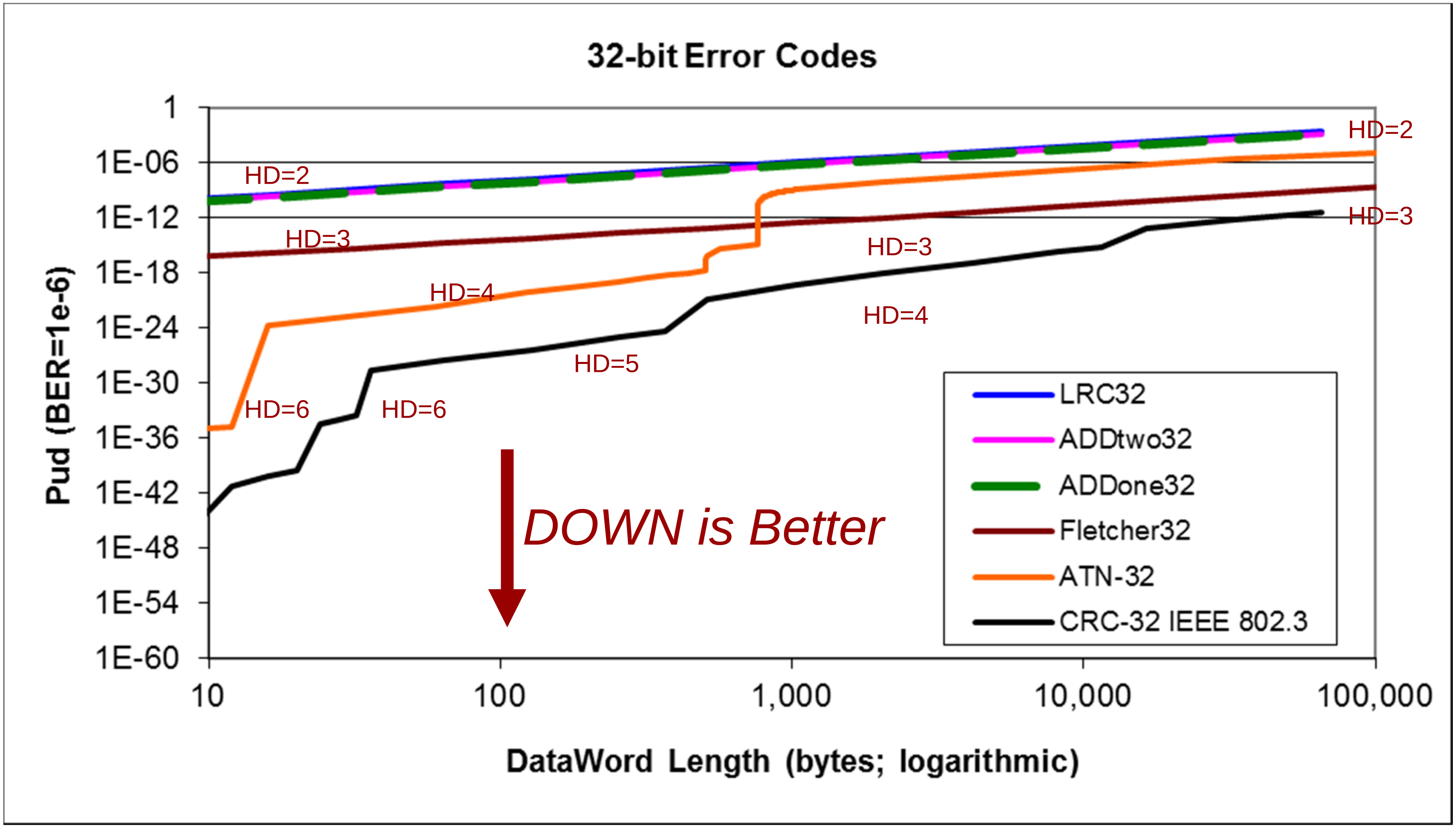


CRC-32 Beats All Checksums



It’s All About The HD

- Better bit mixing of Fletcher & CRC helps at any particular HD



HD Is the Dominant Factor

- BER fault model is random independent bit errors
  - Each increasing number of bit errors is significantly less likely
  - E.g. each bit error in message might be ~1e4 less likely

- HD is what matters the most
  - If 4 bit errors 1e4 less likely than 3 bit errors, then:
    - Detecting all 2 & 3 bit errors helps a lot!
    - Detecting all 4 bit errors helps even more, but is 1e4 less useful than detecting all 3 bit errors
  - Undetected error fraction at the HD is what matters next
    - But all things being equal, HD is more important!

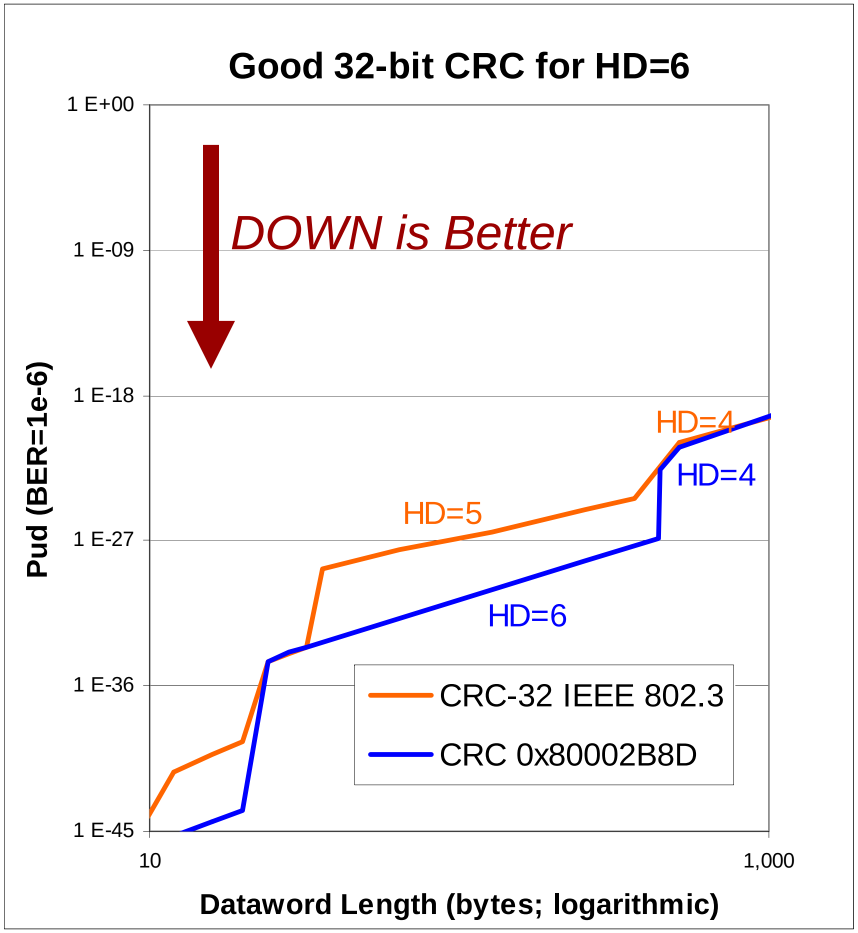
Error Codes Dataword Length (bytes) At HD

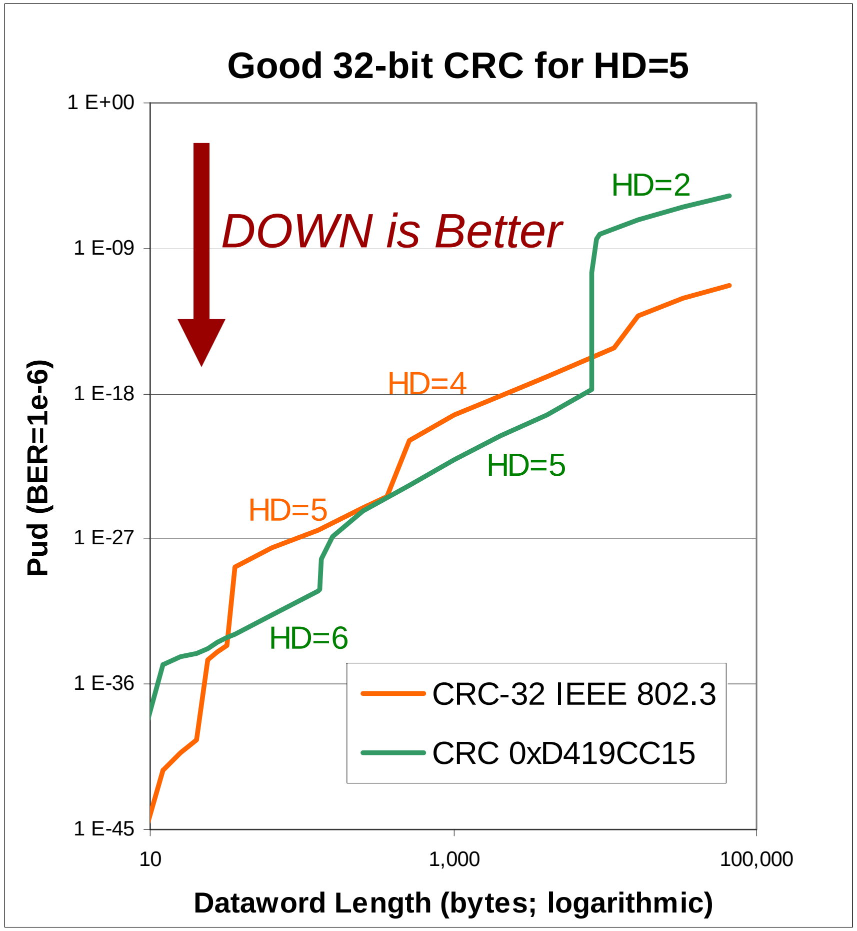
Code	HD=2	HD=3	HD=4	HD=5	HD=6
LRC	All	/ / / / / / / /	/ / / / / / / /	/ / / / / / / /	/ / / / / / / /
ADDtwo32	All	/ / / / / / / /	/ / / / / / / /	/ / / / / / / /	/ / / / / / / /
ADDone32	All	/ / / / / / / /	/ / / / / / / /	/ / / / / / / /	/ / / / / / / /
Fletcher32	Longer	8191	/ / / / / / / /	/ / / / / / / /	/ / / / / / / /
ATN32	Longer	760	504	--	12
CRC-32	Longer	8 Mbyte+	11450	371	33

- *Higher at longer dataword lengths is better*
- Given same HD, better bit mixing gives lower undetected fraction
  - Fletcher better mixing than ADD/LRC checksums
  - ATN-32 better mixing than Fletcher32
  - CRC better mixing than Fletcher

All CRCs Are Not The Same

- No single CRC is “best” everywhere
  - CRC-32 is OK for general purpose use ... but gives up HD





Good CRC Choices – 8 & 16 Bits

Polynomial	HD=2	HD=3	HD=4	HD=5	HD=6
0xA6	Good	30	1	/ / / / / / / / /	/ / / / / / / / /
0x97	Longer	--	14	/ / / / / / / / /	/ / / / / / / / /
0x9C	Longer	--	--	1	/ / / / / / / / /
0x8D95	Good	8189	143	7	2
0xD175	Longer	--	4093	--	6
0xBAAD	Longer	--	998	13	2
0xAC9A	Longer	--	--	30	4
0xC86C	Longer	--	--	--	16
0xC002 (std)	Longer (*)	--	4093	/ / / / / / / / /	/ / / / / / / / /

(Table shows maximum dataword length in bytes at that HD)

* Lower performance than 0xD175 at shorter HD=4 lengths

Good CRC Choices – 24 & 32 Bits

Polynomial	HD=2	HD=3	HD=4	HD=5	HD=6
0x80000D	Good	2,097,151	726	63	/ / / / / / / /
0x9945B1	Longer	--	1,048,575	--	102
0x98FF8C	Longer	--	--	509	28
0xBD80DE	Longer	--	509	--	253
0x80000057	Good	536,870,911	--	346	40
0x80002B8D	Longer	--	268,435,455	--	440
0xD419CC15	Longer	--	--	8188	132
0x90022004	Longer	--	8188	--	4092
0x8F6E37A0 (iSCSI)	Longer	--	268,435,455	--	655
0x82608EDB (CRC-32)		8 Mbyte+	11450	371	33

(Table shows maximum dataword length in bytes at that HD)

INDUSTRY SURVEY RESULTS

Industry Survey Responses

- 15 responses: 9 memory applications; 6 communications
- 9 Memory applications
  - All are CRCs
  - Some have additional error detection (discussed later)
- 6 Communication applications
  - One checksum
  - Four CRCs
  - One dual CRC (discussed later)

Industry Memory Responses

- CRC-16-IBM for 64K bytes (0xC002)
  - Achieves same HD=2 as optimal CRC
    - A better CRC polynomial 0x8D95 is 1.87x better undetected error fraction

- Other CRCs use IEEE 802.3 polynomial
  - 6 responses could get 1 bit better HD by using 0x80002B8D
  - 1 response could get 2 bits better HD by using 0xD419CC15
  - 2 response could get 2 bits better HD by using 0x90022004
  - 1 response could get 1 bits better HD by using 0x8F6E37A0
  - (9 responses total; one response tallied twice in this list)

Industry Communication Responses

- Addone8 Checksum for mostly 64 bit messages
  - Fletcher8 gives 38x better undetected fraction at HD=2
  - CRC 0x97 gives HD=4 at 64 bits and good for longer messages
- CRC-24 for 2040-bit message at HD=4
  - Could get 1 bit better HD by using 0x98FF8C
  - Could get 2 bits better HD for if max message length isn't critical; TTP polynomial 0xE2D2F1 gives HD=6 to 2024 bits
- Dual 16-bit CRC response discussed later
- Other CRCs use IEEE 802.3 polynomial
  - 1 response could get 1 bit better HD by using 0x90022004
  - 2 responses could get 2 bits better HD by using 0x90022004
- One response indicated incorrect IEEE 802.3 polynomial
  - Suggests need for validation that CRC performs as expected in final implementation (bit error injection testing)

Multiple Error Detection Techniques

- Some industry survey responses & standards use multiple error protection
  - SECMED code + CRC on memory
  - Small CRC on memory blocks + large CRC on all memory
  - Parity per word + CRC or checksum on entire message
  - Two CRCs on a message

- Can only take credit for better of the two techniques except if analysis guarantees a better outcome
  - We did analysis for parity and one instance of two CRCs
  - (SECMED: Single Error Correction Multiple Error Detection is similar to parity but with better properties)

Parity + Checksum or CRC

- ARINC-629 uses parity per word + checksum
  - Parity for each 16-bit data chunk
  - (a) 16-bit CRC across 256 x 16-bit dataword size = 4096 bits
    OR
  - (b) 16-bit ADDtwo16 across that same 4096 bit dataword

- Analysis:
  - CRC gives HD=4
    - Parity doesn't preclude chance of two words, each with a two-bit error, evading CRC detection
  - ADDtwo16 requires 2 errors to be in different data chunks
    - But, each data chunk has parity, and needs 2 errors in that data chunk to fool parity
    - So, net result is HD=4
  - Conclusion: CRC and ADDtwo16 both give HD=4 for this example when parity is present.
    - CRC gives better undetected error fraction however

Multiple CRCs

- Common to assert that multiple CRCs provide extra benefit
  - For example, two 16-bit CRCs said to be as good as one 32-bit CRC
  - Arguments are, at best, based on factorization math
- One industry response uses two 16-bit CRCs
  - One CRC has HD=4
    - Other has worse HD
    - Designed to avoid common factors … but …
  - Searching found a great many 4-bit errors that get past both
  - Result was still HD=4, but with lower undetected error fraction
- Recommendations:
  - Unless it can be proven otherwise via exhaustive analysis, HD not improved by adding a second CRC
    - (In our experience, mathematical arguments based on factorization are flawed)
  - Second CRC without common factors improves undetected error fraction, but usually gives up a lot of HD compared to a single bigger CRC

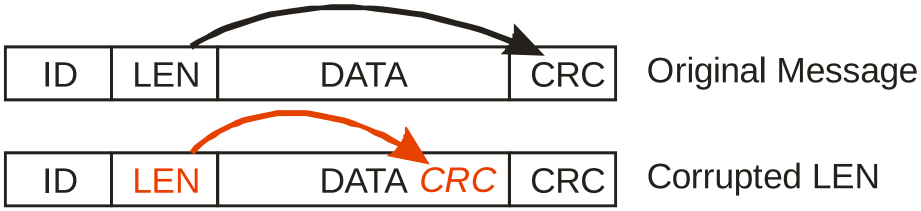
Memory-Specific CRC Properties

- Multi-bit errors can cause errors one row apart
  - BUT, which bit they hit depends upon interleaving strategy
  - So, need detailed physical info to give a specific result
- However, there is a CRC property that helps
  - HD for a burst error of size m is the same as a dataword size m
  - If all errors in dataword are within m bits of each other, HD is at least as good as it would be for entire dataword message size m
    - (Almost always better undetected error fraction, but often same HD)
- Example of use:
  - CRC 0x90022004 used on a 4 Gbyte memory image
    - HD=2 for entire memory image (~1/4G undetected fraction for errors)
    - HD=4 for all multi-bit errors within span of 65506 bits
    - HD=6 for all multi-bit errors within span of 32738 bits

SYSTEM-LEVEL EFFECTS AND CROSS-LAYER INTERACTIONS

Unprotected Length Field

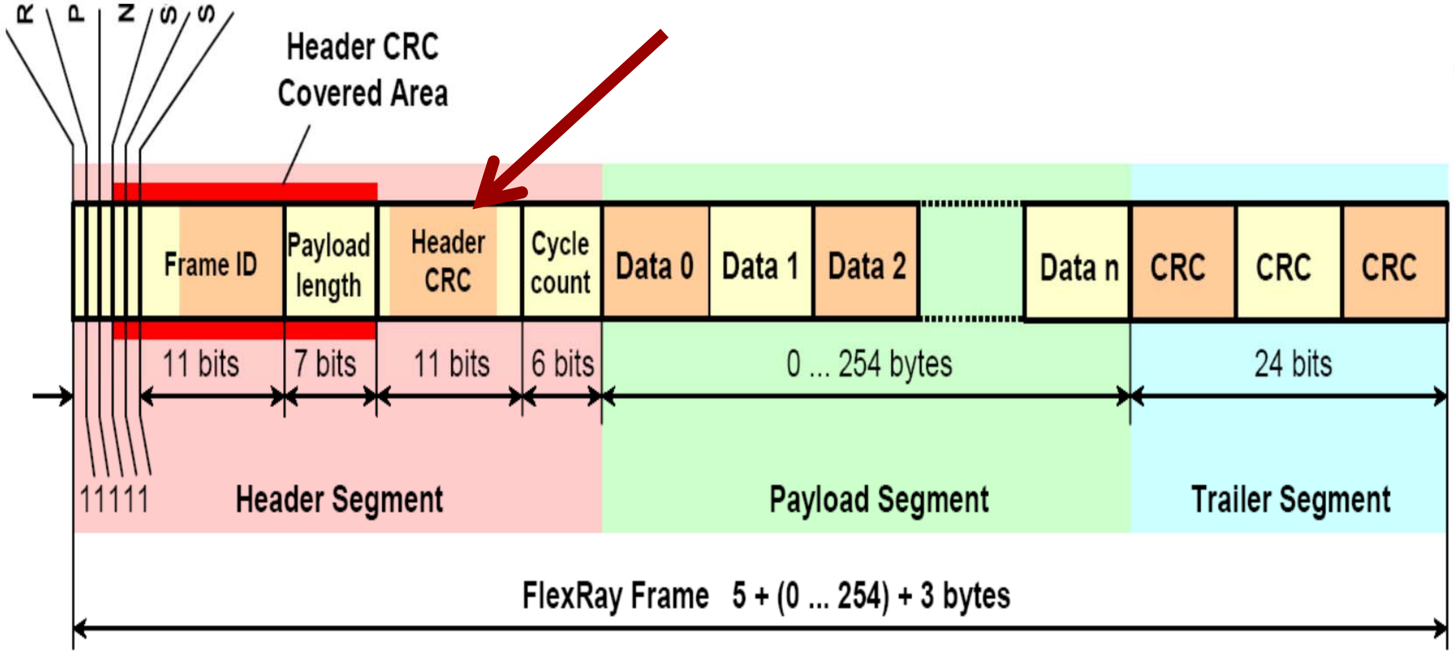
- Many protocols have an unprotected length field
  - Error in length field can point to wrong CRC location
  - Wrong CRC location means 1 bit error (in length field) can escape CRC detection (HD=1)



- Common mitigation methods
  - Parity on length field (ARINC-629)
  - Assert that incorrect length message always caught
    - Many, including CAN/ARINC-825
  - Independent CRC on length/header field (FlexRay)

CAN vs. FlexRay Length Field Corruptions

- FlexRay solves this with a header CRC



Ray Standard, 2004

Figure 4-1: FlexRay frame format.

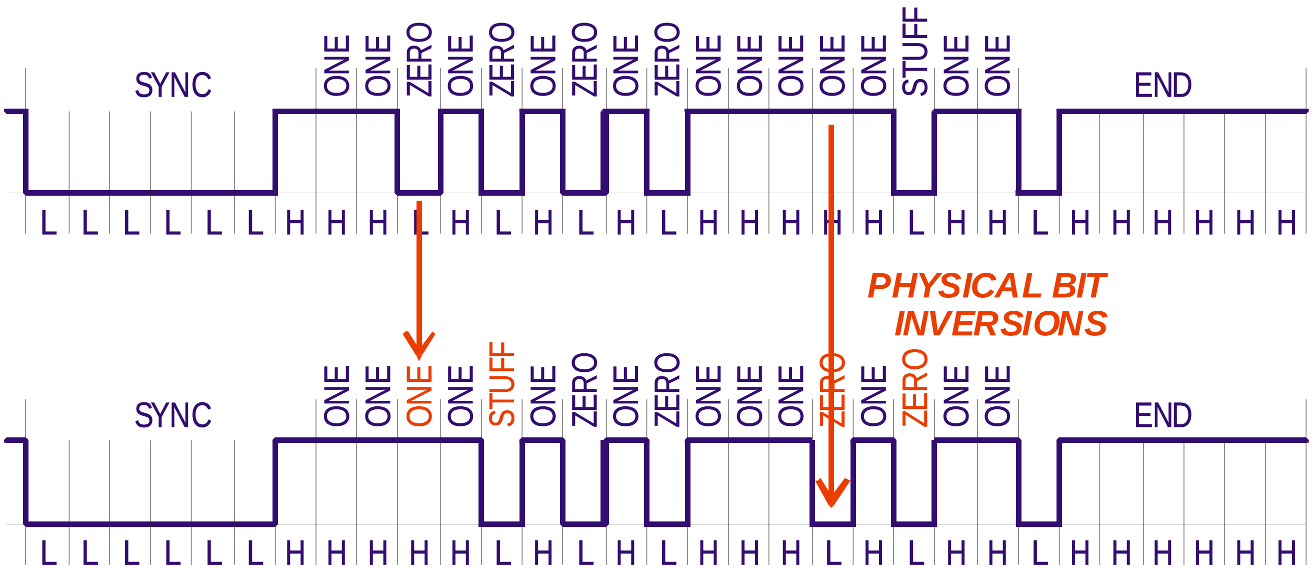
- Recommendation:
  - It is *essential* that length field corruptions be caught, or it invalidates the HD-based safety arguments
  - That means there must be validation of length field corruption being caught

CAN Bit Stuffing Vulnerability (ARINC-825)

- CAN bit stuffing: add stuff bit after 5-in-a-row
- Two bit errors in just the wrong place cause a problem
  - Cascades to an essentially arbitrary number of bit errors
  - Thus, CAN fails to detect some 2-bit errors in data payload
  - FlexRay uses bytes with start/stop bits to avoid this problem

Cascaded bit stuff error example:

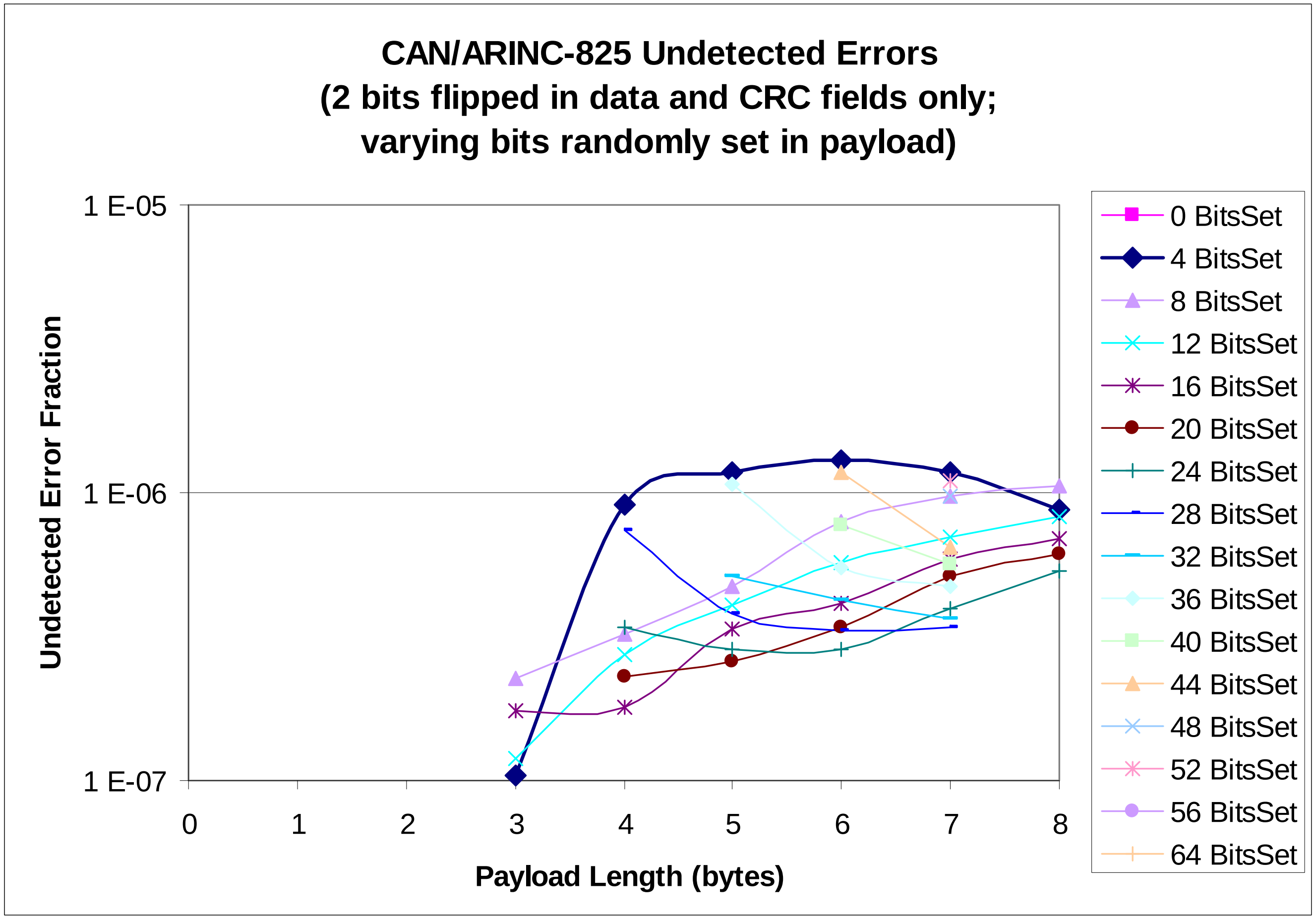
TRANSMITTED LOGICAL BITS: 1101 0101 0111 1111


SYNC
ONE ONE ZERO ONE ZERO ONE ZERO ONE ZERO ONE ONE ONE ONE ONE STUFF ONE ONE
END
L L L L L L H H H L H L H L H L H H H H H L H H L H H H H H H
PHYSICAL BIT INVERSIONS
SYNC
ONE ONE ONE ONE STUFF ONE ZERO ONE ZERO ONE ONE ONE ZERO ONE ZERO ONE ONE
END
L L L L L L H H H H H L H L H L H H H L H L H H L H H H H H H

RECEIVED LOGICAL BITS: 1111 1010 1110 1011

Effect of CAN Bit Stuffing Vulnerability

- Gives effective HD=2 instead of CRC HD=6 for ARINC-825
  - Worst case undetected error fraction at HD=2 is about 1.3E-6
    - (About 3.3% of errors that would be undetected by a good HD=2 code)



Electrical & Computer ENGINEERING

ARINC-825 MIC Excerpts (High Integrity)

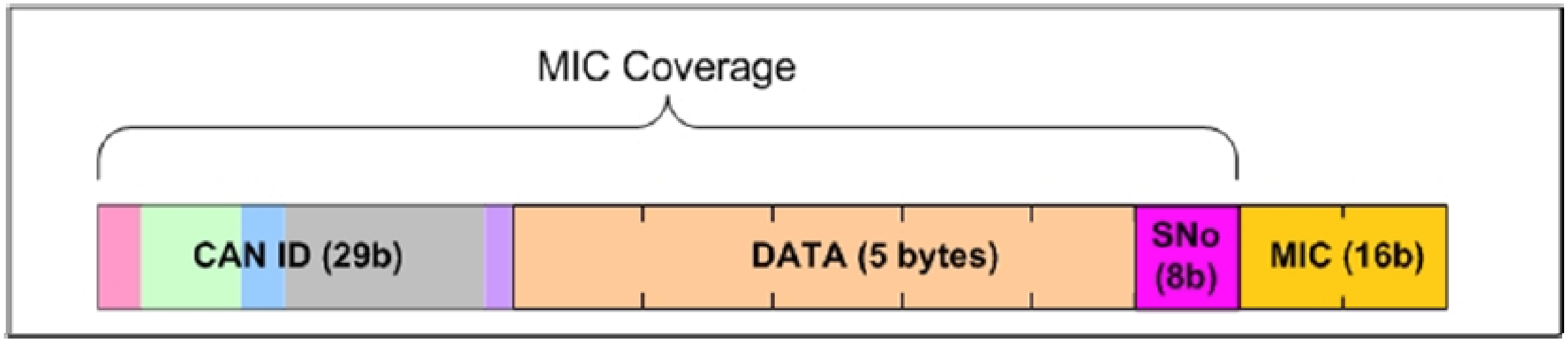


Figure 5-15 – High Integrity Message Protocol Format

5.7.4 Message Integrity Check (MIC)

High integrity messages shall use a 16-bit cyclic redundancy code for the Message Integrity Check (MIC), defined by the following polynomial,

$x^{16}+ x^{15}+ x^{14}+ x^{11}+ x^{6}+ x^{5}+ x^{3}+ x^{2}$ (also known as 0xC86C).

Missing +1 (?)

The MIC shall include the CAN message ID, the data payload, and the SNo.

The following shall be used to compute the CRC:

1. The initial CRC value shall be 0xFFF.
2. Each input byte shall be reflected about its center.
3. The final CRC shall be reflected about its center and XOR with 0xFFF.
4. The message ID is assumed to be 32 bits with the leading 3 bits set to zero.
5. An algorithm that properly computes the CRC shall produce 0xC405 when it computes the CRC on the following test string of characters, “0123456789”.

Why not 0xFFFF?

Couldn’t reproduce this outcome

ARINC-825 MIC Issues

- 16-bit High Integrity Message ID specified by standard
  - 16-bit CRC added to end of message
  - CRC specification has at least typographical errors, maybe more issues depending upon error detection goals
- Apparent errors in ARINC-825 standard
  - Gives credit for HD=5 for base CAN system (pg. 22)
    - Does not take into account the CAN HD=2 problem
  - MIC polynomial is either incorrect or only gives 14-bit error detection effectiveness instead of 16-bit effectiveness (pg. 60)
    - Polynomial given doesn’t have a +1 … either typo or incorrect design
    - Seed values seem odd (0xFFF instead of 0xFFFF)
    - Can’t reproduce the stated test string
- ARINC-825 working group has been cordial and working with us to update this section of the standard

8b10b Encoding Effects for Gbit Ethernet

- Each 8 bits of data physically encoded as 10 bits
  - Maintains DC balance … but …
  - … not enough DC-balanced patterns in 10 bits … so …
  - … uses fairly complex "Running Disparity" imbalance tracking
    - [James05] shows that strict RD book-keeping and detection of encoding errors is required to avoid compromising error detection

- **Results:**
  - Concern that this could lead to cascaded bit errors
  - About a month of random simulations on 8 CPU cores did not find a problem
    - So it is a rare problem if one exists .. but hard to say how rare it is
  - Based on informal analysis looks like issue will be that one bit transmission errors will cause bursts of 5 bit data errors
    - Further analysis required to prove that no HD=2 or HD=3 vulnerability caused by this effect
    - Open research topic: multi-burst coverage of IEEE 802.3 polynomial

MAPPING TO DO-178 CRITICALITY LEVELS

Note: This is the contract wording.
It would be more correct to say
“functional criticality” levels

Criticality Mapping Overview

- No one-size-fits-all definitive rule
  - If we formulated one, it would be excessively conservative
- Need to formulate an error coverage argument based on:
  - Criticality of the function(s) using the data
  - How errors could affect those functions
    - Possibly mitigated by architectural features (e.g. redundancy with voting)
  - Probability of error occurrence given size of data to be protected
  - Possibly other subordinate factors
- In terms of error coding, the dominant factors are:
  - Hamming distance of the code
  - Error exposure (BER, message size, and message rate)
  - Consequence of undetected errors

Criticality Mapping Outline

- Some background information
  - Scope
    - Error detection usage in avionics
    - This program’s investigation area
  - Existing databus guidance (FAA Advisory Circular 20-156 paragraph 4)
- A flowchart of the process
- An explanation for each of the flowchart steps
  - Fictitious example used to illustrate the steps

Scope – Main Avionics Applications

- Error coding for data-transfer systems
  - Dynamic, data is moving
  - Includes on-board data networks and off-board communication
  - Some error detection coding guidance exists (FAA Advisory Circular 20-156 ¶ 4)
- Error coding for memory
  - Static, data is at rest; can be volatile (RAM) or non-volatile (flash, PROM)
  - Memory typically is hierarchical (Flash ➔ RAM ➔ cache levels)
  - No existing specific guidance for error detection coding requirements?*
- Error coding for transferable media
  - More like memory, even though data is moving
  - Magnetic disks, optical discs, and flash devices (USB, SD, etc,)
  - No existing specific guidance for error detection coding requirements?*
- We only look at "random" error coverage
  - Exclude complex IC failures
  - Exclude symbol encoding, data compression, or encryption effects.

* While not guidance, standards such as ARINC 665, RTCA DO-200A, and RTCA DO-201A describe some error detection methods.

Data-Transfer Requirements Background

- Data-transfer systems
  - The text of FAA Advisory Circular 20-156 “Aviation Databus Assurance” that pertains to databus error tolerance is the beginning of its paragraph 4, to wit:

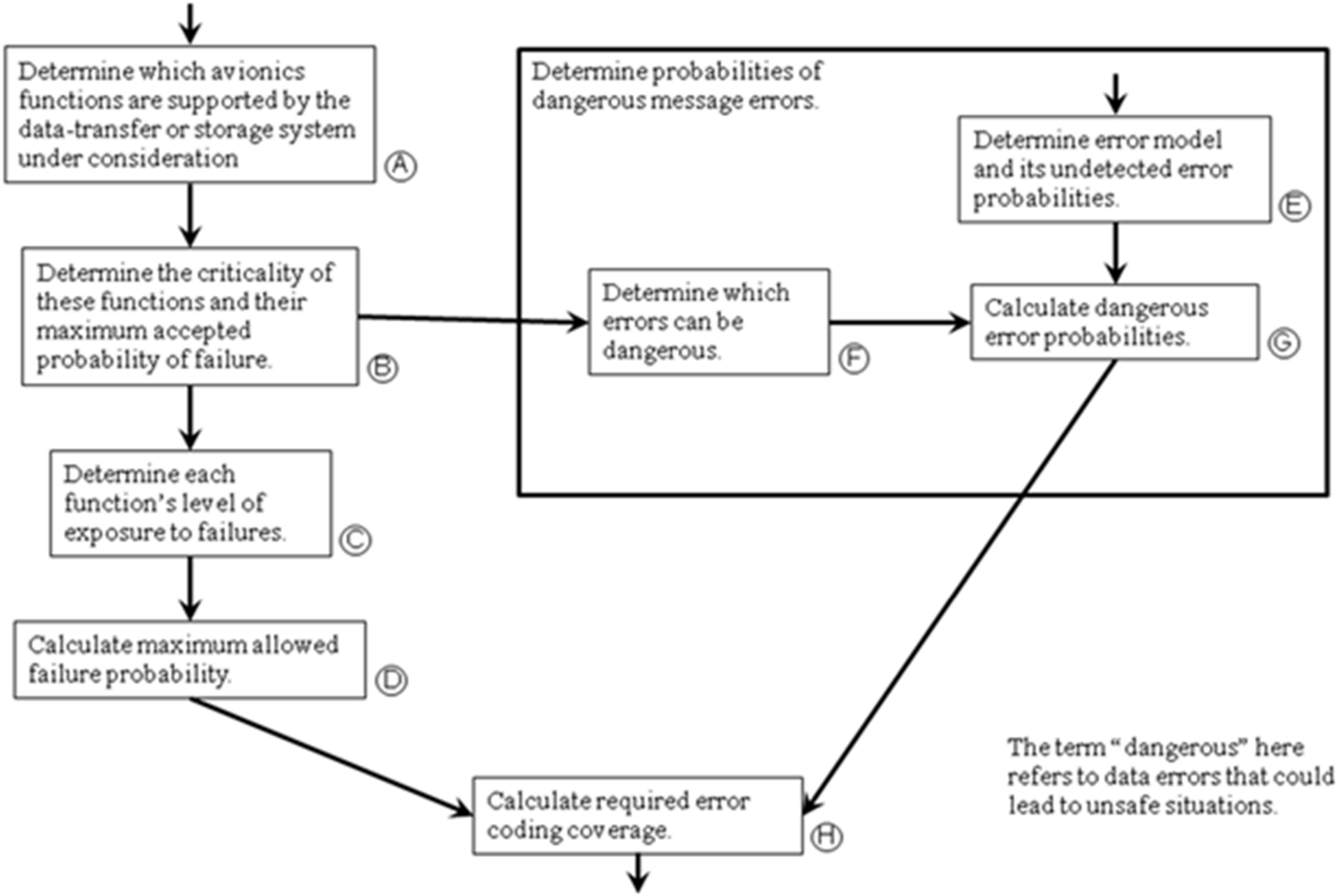
    4. DATA INTEGRITY REQUIREMENTS. Data that passes between LRUs, nodes, switches, modules, or other entities must remain accurate and meet the data integrity of the aircraft functions supported by the databus. To ensure data integrity through error detection and corrections resulting from databus architectures, evaluate databus systems for the following:

    a. The maximum error rate per byte expected for data transmission.

    b. Where applicable, databus-provided means to detect and recover from failures and errors to meet the required safety, availability and integrity requirements. Examples are cyclic redundancy checks, built-in detection, hardware mechanisms, and architecture provisions.

    c. […]
  - While specific to on-board databus systems, applicable to any data transfers
    - RF channels
    - “Sneaker net” transferable media
- We use “data-transfer system” to mean
  - One or more data networks within an aircraft and/or
  - Any other mechanisms that transfer data into an avionics system via messaging

Requirements Determination Flowchart



Determine which avionics functions are supported by the data-transfer or storage system under consideration Ⓐ

Determine the criticality of these functions and their maximum accepted probability of failure. Ⓑ

Determine each function’s level of exposure to failures. Ⓒ

Calculate maximum allowed failure probability. Ⓓ

Determine probabilities of dangerous message errors.

Determine error model and its undetected error probabilities. Ⓔ

Determine which errors can be dangerous. Ⓕ

Calculate dangerous error probabilities. Ⓖ

Calculate required error coding coverage. Ⓗ

The term “dangerous” here refers to data errors that could lead to unsafe situations.

Example for Coverage Determination

- Fictitious example avionics data-transfer system
  - Picked purposely to not represent any existing or planned system
  - To paraphrase Hollywood: All bits appearing in this example are fictitious. Any resemblance to real bits, living or dead, is purely coincidental.
- Example: autopilot control panel that uses a redundant data bus network to talk to a set of autopilot computers
  - This example (artificially)* increases the criticality of this function by ignoring the fact that pilots are required to know and maintain assigned altitudes, independent of the autopilot
- The steps in the flowchart are broken out in the following sides, with an "Application to example system" bullet that describes how that step would be applied to this example

* some may argue this increase is actually valid, because some pilots rely on the autopilot when they shouldn't

Requirements Determination Flowchart Steps Ⓐ and Ⓑ

- Ⓐ Determine which avionics functions are supported by the data-transfer or storage system under consideration
- Ⓑ Determine the criticality of these functions and their maximum accepted probability of failure.

- Application to example system:
  An FHA/SSA assessment of the system finds that messages from the control panel to the computer send altitude-hold changes <u>as increments or decrements</u> to the existing altitude-hold value and that an undetected error in these increments or decrements could cause an aircraft to try hold an incorrect altitude. This could lead to a mid-air collision, a condition that cannot be allowed with a probably more than 1e-9 per flight-hour (catastrophic failure probability limit per FAA AC 25.1309).

Requirements Determination Step ©

- Determine each function's level of exposure to failures.

- Application to example system:
  Because this system uses increments or decrements to the existing altitude-hold value rather than (re-)sending an absolute value for the altitude hold, any undetected transient error in a data bus message transfer can cause a permanent error in the autopilots' altitude hold value. The latter cannot be allowed to happen with a probability more than 1e-9 per flight-hour. Thus, the system must ensure that undetected transient errors in the data bus system do not exceed this probability. The system has redundancy mechanisms to deal with missing or rejected messages, but no integrity mechanisms above the network. Therefore, the network exposes the system to integrity failures.

Requirements Determination Step Ⓓ

- Calculate maximum allowed failure probability.

- Application to example system:
  Because this network transfers data that must not fail with a probability more than 1e-9 per flight-hour and there is no additional integrity mechanisms in the example system, messages must not have undetectable errors with the probability greater than this.

  - Max acceptable Pud is 1e-9/flight-hour

Requirements Determination Step Ⓔ

- Determine error model and its undetected error probabilities.

- Application to example system:
  The autopilot control panel is connected via data buses directly to the autopilot computers, with no intervening switches or other complex logic devices. So, we can use the results of Bit Error Ratio (BER) testing (see BER testing discussion in the following sides) to determine error probabilities. If there had been any complex devices in the data path, there would be no way to determine individual bit error probabilities nor the probabilities for specific bit error patterns.
- For this example, we will say this testing showed a BER of 1e-8 (meets ARINC 664 requirement).
  - Measured BER = 1e-8

Finding a Bit Error Ratio (BER)

- Need to confirm BER by testing
  - Just because standard specifies a BER doesn’t mean you meet it
  - Applies to COTS and custom components
- Aircraft aren’t the same as cars or offices
  - Connectors and cables typically not per COTS standard
  - Aircraft environment is not the same as home or office environment
  - Standards only require meeting particular test scenarios
  - COTS components only need to work for main customers (e.g. cars)
    - These parts do not have to meet BER requirements for all scenarios
    - In particular, they typically are not designed to meet the standard’s BER in an avionics environment with a confidence level commensurate with that required by avionics (e.g. AC 25.1309)
  - Can’t rely on test results from COTS part manufacturers/vendors
- Thus, all data-transfer systems need to be BER tested
  - At least during system testing
  - Arguably during system life to detect system degradation

Bit Error Ratio (BER) Testing

- Test BER in avionics worst-case environment (e.g. per DO-160)
- A worst-case bit pattern must be transmitted
  - Network standards typically define these patterns
    - If more than one worst-case bit pattern is defined, all such patterns must be tested
  - Certain pathological bit patterns (such as Ethernet IEEE 100BASE-TX “killer packets”) can be ignored if mechanisms prevent them from ever being transmitted
- Need worst-case eye pattern (most degraded signal) at receiver
  - Some test equipment can generate bit patterns with worst-case eyes
  - If this equipment is not available, build a physical analog of the worst-case cable
  - Note that error prediction calculations from eye measurements require a very accurate model of the actual receiver
    - Test equipment that calculate BER from eye measurements cannot be used
    - Need to test the receiver that will be used in the actual flight system
- BER = # bad bits output by receiver-under-test / total bits sent
  - A sufficient number of bits needs to be transmitted such that the confidence in the BER test results is commensurate with that needed by the avionics’ certification plan

BER Degradation

- During aircraft life, failures might increase the BER
  - This higher BER might overwhelm error detection mechanisms
  - A fraction (usually about 1/2^k for a k-bit error code) of errors are undetected
    - Higher BER ➔ more likely to be undetected errors

- Testing BER directly on the fly is typically infeasible
  - But, elevated number of detected errors implies elevated BER
  - Suggestion: compute expected detected errors / hour.
    - If this is significantly exceeded, you know BER specification has been violated
    - In response, system must stop trusting data source or network due to high BER

- Important point: detected errors only a symptom, not the problem
  - It is undetected errors that are the problem
  - If a bus or message source is throwing you a lot of erroneous messages, it is very likely a few messages that appear OK actually have undetected errors.

Requirements Determination Step Ⓕ

- Determine which errors can be dangerous.

- Application to example system:
  The messages between the autopilot control panel and the autopilot computer are 64 bytes long*, 4 bits of which are the critical altitude change data. If any of these four bits are corrupted and not detected, the aircraft will to try hold an incorrect altitude.

* 64 bytes is the minimum Ethernet frame (message) size --- including Ethernet addressing, length/type, and CRC fields; all of which are covered by the CRC

Requirements Determination Step ©

- Calculate dangerous error probabilities.

- Application to example system:
  The probability that a message has an attitude data error is the probability of any of the 4 altitude bits being erroneous. With a 1e-8 BER, this probability is:
  $1 - (1 - 1e\text{-}8)^4 \approx 3.9999999400000004e\text{-}8$
  If this system sends this message at 20 Hz, then 72,000 messages are sent per hour (20 msgs/sec * 60 sec/min * 60 min/hr). The probability that one or more of these messages has an altitude data error in an hour is:
  $1 - (1 - 3.9999999400000004e\text{-}8)^{72{,}000} \approx 2.8758567928056934555e\text{-}3$
  Because this is greater than 1e-9, we need to do error detection.

Note: The equation form "1 - (1 - *p*) ^ *n*" gives the probability that one or more of *n* events, each of independent probability *p*, occur. This can be approximated by *n* * *p* if *p* is small enough. However, this approximation can be used only if one is sure that the loss of accuracy is inconsequential. In this case, we can't be sure of that, a priori, since the example's failure probability limit is very small (1e-9 per flight-hour).

Requirements Determination Step Ⓗ

- Calculate required error coding coverage.

- The coverage of an error coding design has to be such that:
  - the conditional probabilities of particular error bit patterns (which may depend on BER, number of critical bits, and message length) occurring
  - times the probability of the error coding design failing to detect these patterns
  - is less than the maximum allowable failure probability for a message.

- Application to example system is given on next slide

Requirements Determination Step Ⓗ (cont.)

- Application to example system:
  - Adding **C** to represent simple coverage (the probability that an error in the four altitude bits will be detected) to the dangerous error probabilities equation from step Ⓖ and making it less than 1e-9 gives:

    $1 - (1 - (3.9999999400000004e\text{-}8 * \mathbf{(1 - C)}))^{72,000} \leq 1e\text{-}9$
  - Solving for **C**, gives **C** ≥ 0.99999965277777239583569847 6119
  - Thus, the error detection must have > 99.99997% coverage on each message
  - This would be sufficient coverage for an error detecting code only if all the following were true:
    1) The hardware/software that does the error detection never fails (cannot be guaranteed)
    2) The error code needs to protect only these 4 bits rather than the whole message (rarely done in practice)
    3) The error code itself never suffers any errors (cannot be guaranteed)
  - The first of these would be covered in the next slide with the other two combined and covered in slides after that

Error Detection Mechanism Failure

- For this example scenario, assume that the error detection mechanisms' failure rate is 1e-7 per hour
  - The probability for the error detection mechanisms being functional for t hours is: e ^ -(1e-7 * t)
- With "scrubbing" (testing that the error detection mechanisms are still functioning correctly) before each (3-hour max) flight, the probability of an error detection mechanism failing in flight is less than 3e-7.
  - The combined probability of an error detection mechanism failing and an error occurring in the four critical bits in an hour is: 3e-7 * 2.8758567928056934555e-3 ≈ 8.6e-10 ≤ 1e-9
  - Without "scrubbing", the probability of an error detection mechanism failing sometime during the life of aircraft would be too high
- There are two basic ways of doing scrubbing
  - "black box" (applying all possible error patterns and seeing that they all are detected)
  - "white box" (testing the components within the mechanism and their interconnects)
- Typically, "black box" testing is not feasible because of the huge number of possible error patterns
  - Most COTS devices don't have the ability to do "white box" testing
  - This means that the error detection mechanisms within COTS devices usually cannot be trusted and mechanisms that allow themselves to be "white box" tested need to be added to cover COTS devices

Other Errors in the Message

- What if, as is usual, our error detecting code covers whole 64-byte message?
  - HD=5 doesn't guarantee all 4 altitude bits are protected (i.e., not 100% coverage)
    - – Especially since errors could also be in error code too
    - – So this means we need to do probability analysis to see what is good enough
  - Other bits in message also might be corrupted, using up part of HD quota
  - These bad bits could be anywhere in the message, including in the error code itself
- A practical approach is to treat the whole message as critical
  - Thus: 72,000 messages/hr * 64 bytes/msg
  - $1 - (1 - 1e\text{-}8)^{(64*8)} = 5.119986918e\text{-}6$ errors/msg
  - $1 – (1 - 5.119986918e\text{-}6)^{(72000)} = 0.308326$ errors/hr, but want < 1e-9
- Example, "perfect" 16-bit random error detection HD=1 checksum; $1\text{-}C = 1/2^{16}$
  - $1 – (1 – (5.119986918e\text{-}6 * 2^{-16}))^{(72000)} = 5.62497e\text{-}6$ – not good enough!
- Example, "perfect" 32-bit random error detection HD=1 checksum; $1\text{-}C = 1/2^{32}$
  - $1 – (1 – (5.119986918e\text{-}6 * 2^{-32}))^{(72000)} = 8.79297e\text{-}11$
  - So, it is likely that a good 32-bit checksum will suffice
    - – HD=1 is enough in this case; not 100% coverage; but good enough for 1e-9

– If we use HD>1 how does it help?

Using HD To Exploit 16-bit CRC

- HD=n means all 1, 2, … n-1 bit errors are detected
- For example, any HD=4 code (e.g., 16-bit CRC) will do the job
  - Select 0xBAAD, which gives HD=4 $(x^{16}+x^{14}+x^{13}+x^{12}+x^{10}+x^{8}+x^{6}+x^{4}+x^{3}+x+1)$
  - HD=4 coverage (512+16) bits codeword is 64510 undetected / combin(528,4)
- Sum all probabilities that there are k bit errors
  - 1 bit, 2-bit, 3-bit errors – always caught; not relevant
  - 4 bit errors: (all combos * four bits error * rest of bits non-error)
    - combin(528,4) * (1e-8)$^{(4\text{ bad bits})}$ * (1-1e-8)$^{(64*8+16-4\text{ good bits})}$
= 3.201666e-23 errors/message
    - Zero HD=4 coverage: 1 – (1 – 3.201666e-23)$^{(72000\text{ msgs/hr})}$
= 2.3052e-18 undetected errors/hr (any HD=4 code)
    - 0xBAAD coverage: 1 – (1 – 3.201666e-23*(64510 undetected/combin(528,4)))$^{(72000)}$
= 4.6447e-23 undetected errors/hr (good 16-bit CRC)
  - 5 bit errors: contribution too small to really matter
    - combin(528,5)*(1e-8)$^{(5)}$*(1-1e-8)$^{(64*8+16-5)}$ = adds: 3.355346e-29 /message
    - Zero coverage: 1 – (1 – 3.355346e-29)$^{(72000)}$ = adds: 2.415849e-24 / hr
  - No point worrying about very unlikely 6+ bit errors

A Simplified Approach to Coverage

- Compute expected number of message errors per hour
  - For all 1-bit errors, 2-bit, 3-bit, etc.
  - Errors per hour is a function of BER, msgs/hr, and total message length
  - At some point the expected number is below your target value
    - For example, below 1e-9 … in other words you just won’t see that many bit errors
- Pick a code with a HD high enough
  - Assume “k” is the number of bit errors below your target
    - For example, 1-, 2-, 3-, 4- bit error probabilities are all above target rate
    - But 5-bit errors are less likely than 1e-9/hr … this means k=5 for you
  - Pick an error code that has HD=k
    - This gives perfect coverage above k, and conservatively assume zero coverage at k
    - If on the border, need detailed checksum coverage info
- Do you use a checksum or CRC?
  - For k=2 a checksum will suffice; sometimes for k=3
  - At k=4 or more, you usually need a CRC
  - CRC always provides better coverage

WRAP-UP & DISCUSSION

CRC/Checksum Seven Deadly Sins (Bad Ideas)

1. Picking a CRC based on a popularity contest instead of analysis
   - This includes using "standard" polynomials such as IEEE 802.3
2. Saying that a good checksum is as good as a bad CRC
   - Many "standard" polynomials have poor HD at long lengths
3. Evaluating with randomly corrupted data instead of BER fault model
   - Any useful error code looks good on random error patterns vs. BER random bit flips
4. Blindly using polynomial factorization to choose a CRC
   - It works for long dataword special cases, but not beyond that
   - Divisibility by (x+1) doubles undetected fraction on even # bit errors
5. Failing to protect message length field
   - Results in pointing to data as FCS, giving HD=1
6. Failing to pick an accurate fault model and apply it
   - "We added a checksum, so 'all' errors will be caught" (untrue!)
   - Assuming a particular standard BER without checking the actual system
7. Ignoring interaction with bit encoding
   - E.g., bit stuffing compromises HD to give HD=2
   - E.g., 8b10b encoding – seems to be OK, but depends on specific CRC polynomial

Possible Future Work Topics

- Other evaluation criteria
  - Other fault models (e.g., bit slip)
  - Other error detection effectiveness measures (e.g., data byte ordering errors)
- Better understanding of cross-layer interactions
  - Better understanding of bit encoding effects (some good/some bad)
    - Understanding 8b10b requires multi-burst error analysis techniques
  - Taking credit for message/data framing effects
    - E.g., per-block CRC in addition to entire memory CRC
- Alternative and hybrid approaches
  - Is there a pair of 16-bit CRCs that gives higher HD as a pair?
  - Are cryptographically secure hash functions suitable for HD=1 applications?
- Analytically exact evaluation of other codes
  - 64-bit CRC effectiveness
  - ATN-32 effectiveness
- High level system effects
  - Interaction of compression and encryption (put CRC outside encrypted data)
  - Are safety layer CRCs really as effective as people think?

Next Steps

- Final report submitted to FAA

Questions?

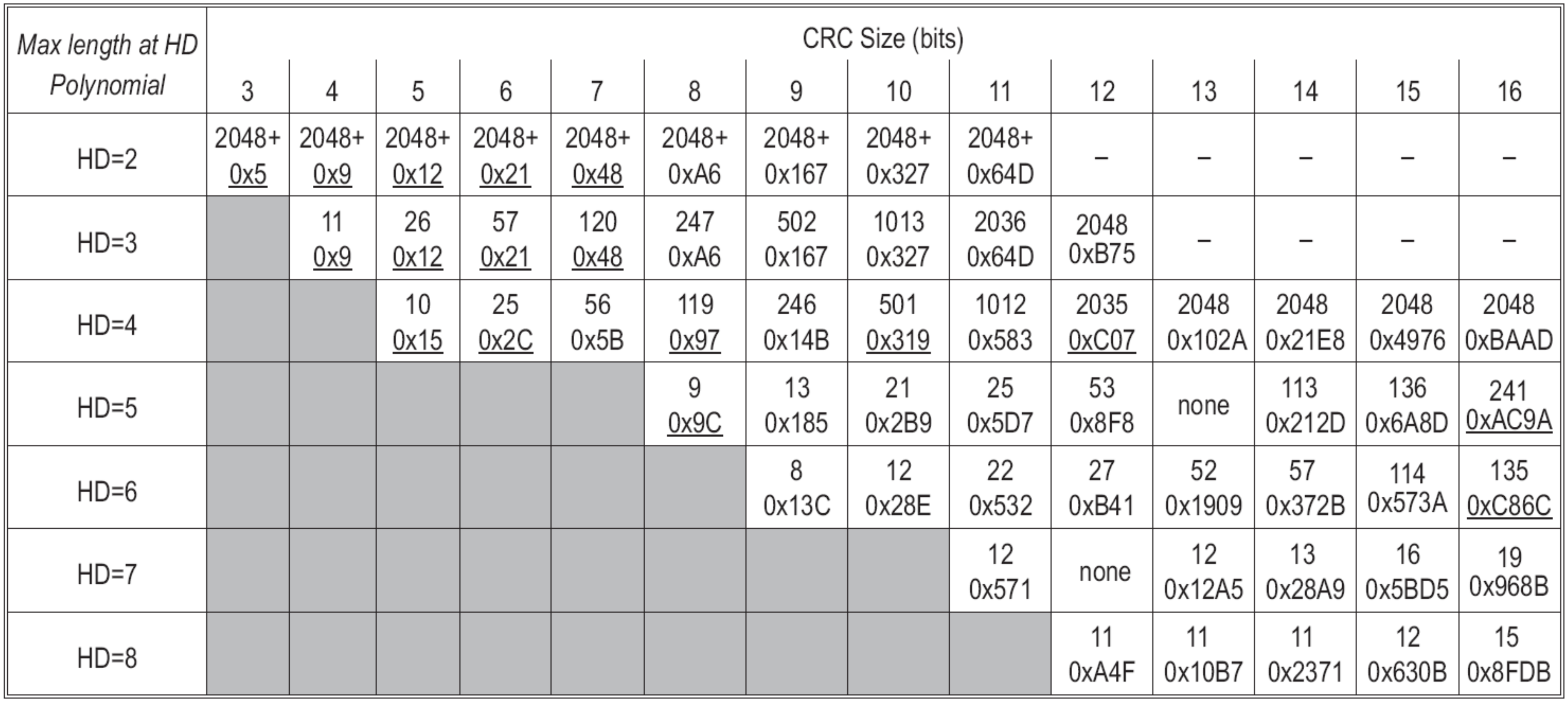
BACKUP SLIDES

Outline of Contract SubTasks
From Initial Project Briefing

SubTask #1 Literature Survey

- Starting point: P. Koopman, T. Chakravathy, “Cyclic Redundancy Code (CRC) Polynomial Selection for Embedded Networks”, The International Conference on Dependable Systems and Networks, DSN-2004.

Table 3. “Best” polynomials for HD at given CRC size and data word length. Underlined polynomials have been previously published as “good” polynomials.



Max length at HD *Polynomial*	CRC Size (bits)													
	3	4	5	6	7	8	9	10	11	12	13	14	15	16
HD=2	2048+ 0x5	2048+ 0x9	2048+ 0x12	2048+ 0x21	2048+ 0x48	2048+ 0xA6	2048+ 0x167	2048+ 0x327	2048+ 0x64D	–	–	–	–	–
HD=3		11 0x9	26 0x12	57 0x21	120 0x48	247 0xA6	502 0x167	1013 0x327	2036 0x64D	2048 0xB75	–	–	–	–
HD=4			10 0x15	25 0x2C	56 0x5B	119 0x97	246 0x14B	501 0x319	1012 0x583	2035 0xC07	2048 0x102A	2048 0x21E8	2048 0x4976	2048 0xBAAD
HD=5						9 0x9C	13 0x185	21 0x2B9	25 0x5D7	53 0x8F8	none	113 0x212D	136 0x6A8D	241 0xAC9A
HD=6							8 0x13C	12 0x28E	22 0x532	27 0xB41	52 0x1909	57 0x372B	114 0x573A	135 0xC86C
HD=7									12 0x571	none	12 0x12A5	13 0x28A9	16 0x5BD5	19 0x968B
HD=8										11 0xA4F	11 0x10B7	11 0x2371	12 0x630B	15 0x8FDB

SubTask #2: Survey of Data Characteristics

- Find out what aviation industry uses/needs
  - Use taxonomy from literature survey as starting point
  - What parts of the taxonomy for code design & usage matter?
  - Elicit operational characteristics as well as specific data error detection needs
- Example results might be:
  - Data network using ATN-32 on 1200 byte packets
  - In-memory 802.11 CRC-32 integrity checks on 24 MB program images
  - EEPROM data integrity check with 8-bit Fletcher checksum on 256 bits of data…
- Higher level expected results
  - Determine portions of taxonomy that are used by aviation
  - Determine practices not in use (some of which might be better)
  - Discover hybrid practices that might need to be addressed (e.g., composite CRC or non-standard checksums)

SubTask #3: Survey of Fault Exposure

- Find out what aviation industry faults matter
  - Use taxonomy from literature survey as starting point
  - What faults are likely to be seen in operation?
- Example results might be:
  - Random independent bit flips might (or might not) be relevant
  - Small burst errors might be relevant
  - BER might or might not fluctuate
  - Pay particular attention to cross-layer issues (e.g., burst error interaction with 8B/10B symbol encoding)
- Higher level expected results
  - Determine portions of taxonomy that are used by aviation
  - Determine which evaluation criteria make sense for aviation
  - Determine if hybrid or novel evaluation criteria are necessary

SubTasks #4,5: Evaluation of CRC Performance

- Task 4: speculate as to likely needs to get started
  - E.g., Evaluate CRC performance for 32-bit CRC size
  - E.g., Evaluate ATN-32 performance
- Task 5: Based on survey results, evaluate aviation-relevant operational scenarios
  - Specific error coding design parameters and usage scenarios
  - Specific fault models and evaluation criteria
  - Reasonable alternative situations to represent improved practices
- Expected results:
  - Probability of undetected errors for aviation-relevant scenarios
  - Identification of opportunities for improving aviation practices
  - Confirmation of good aviation practices

SubTask #5: Follow-Up Surveys

- Follow up to improve information collected
  - Based on looking for patterns and gaps in existing survey data
  - Based on results from analysis and simulation
    - E.g., results might be very sensitive to some parameter, and we didn't realize we needed to ask about that parameter

- Probably will select a handful of representative example scenarios
  - Will follow up with survey participants with the best opportunities for industry influence with our results

SubTask #6: Evaluation of Checksum Performance

- Similar to tasks 4 & 5, except perform with Checksums
  - Select representative algorithms, usage models, fault models, etc.

- Starts after CRCs because checksum performance depends heavily upon data patterns
  - (CRC is almost independent of actual data value being protected)
  - E.g., it may be common to have many zero fields
  - In many cases checksum performance must be evaluated via Monte Carlo simulation

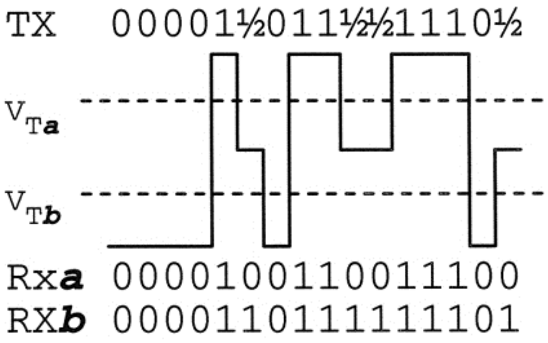
- Also will identify possible cross-layer interaction issues
  - For example: unprotected length field, bit encoding interactions with error coding, bit stuffing

SubTask #7: Mapping Error Performance to DO-178B

- Map practices to DO-178B software levels by combining:
  - Fault arrival rates and patterns
  - Error coding effectiveness
  - Probability of multiple fault accumulation (e.g., archival storage)
  - Probability of multiple faults in end-to-end scenario
  - Error codes are one factor in coverage
    - Criticality ➔ Probability of Failure/Error ➔ Coverage ➔ Error Coding
- Suggest good practices for relevant applications
  - E.g., CRC polynomial X is optimal for scenario Y
  - E.g., Fletcher checksum is virtually always better than Adler
- Note: Hardware integrity checks (DO-254) are possible
  - Out of scope for this study, but can use same coding approaches

SubTask #8: Recommendations for Future Work

- Accumulate potential future work opportunities; e.g., :
  - Cryptographic integrity checks and interactions with error coding
    - Hamming Distance of a good secure hash must be HD=2
  - Instances in which optimal code was too computationally difficult to find
    - Will give a known "pretty good" result and identify opportunity for search for optimal results
  - Byzantine failures for CRCs and Checksums



Example Schrodinger's CRC caused by non-saturated voltage values on a data bus. Two receivers (***a*** and ***b***) can see the same message as having two different values, and each view having a valid CRC

SubTask #9: CRC & Checksum Recommendations

- Create report with recommendations, addressing:
  - Assumptions that must be made to evaluate error codes
    - Deployment and usage scenarios
    - Data values
    - Fault scenarios
    - Technology used
  - Collected good practices
    - Focus on representative aviation usage scenarios
  - Collected known bad practices
  - Situations in which further evaluation is required
    - Out of scope, or
    - Instances in which our work demonstrates answer can’t be determined with current state of knowledge