Simulation of Error Detection Algorithms and Comparison of Efficacy in Wireless and Wired Networks

Daelon Shockley

Missouri University of Science and Technology

CpE 5430

October 7th, 2024

Abstract

This paper conducts a programmatic simulation of single-parity, 2D parity, checksum, and CRC error detection methods in network links with experimentally accurate BER values for common wired and wireless networks. The results of the simulation found 2D parity and checksum to be 100% effective against errors introduced in the simulation, with single parity being fully effective in detecting odd-numbered errors and fully ineffective in detecting even numbered errors. CRC was fully effective against single bit errors, with declining efficacy against higher number errors. The results of this paper recommend single parity for efficiency prioritized networks which are tolerant to rare undetected errors, and 2D parity for networks which are not tolerant to those errors in networks with BERs similar to the realistic rates tested in this paper. Networks with highly inflated rates approaching 1% BER or above are recommended to implement checksum. This paper found no practical difference in wireless and wired networks when it comes to selecting an error detection scheme.

Introduction

The purpose of this simulation is to test the implementation of different error detection techniques in wireless and wired networks. Four common error detection methods: single parity check, two-dimensional parity check, checksum, and cyclical redundancy check (CRC) will be tested against a simulated transmission of binary data with different bit error rates (BER). BER values used are based on the real-world rates provided to us in literature by Okpeki et al [1], utilizing these values will allow for accurate comparison of detection methods for realistic BER values in both wireless and wired networks. To measure the efficacy of these error detection methods for each BER simulated, the accuracy of error detection and bit overhead required for implementation will be compared at the end of the simulation. This work is intended to better inform the selection and application of common error detection methods in networks based on the type of network used (wired vs wireless) and the transmission protocol chosen (TCP, IPv4, IPv6).

Background

The most common error detection methods employed in networks are single parity check, two-dimensional parity check, checksum, and cyclical redundancy check [2]. For this reason, this simulation explores these four methods. Implementation of single parity is extremely simple. Typically, for every seven bits transmitted an eighth bit is added to ensure the number of 1s in the byte is even (for even parity), or odd (for odd parity). For example, the letter P in 7-bit ASCII is 1010000, if this was transmitted using odd parity an additional one would be added to ensure the number of ones in the byte is odd. For even parity, a zero would be added as the number of 1s in the data is already even. Once these bits are applied, the receiver of the transmission knows that if the number of 1s in a received byte aren't even (for even parity), then an error has occurred [3]. Similarly, in two dimensional parity an extra bit is added to each byte transmitted; however, after a number of bytes have been sent, an additional parity byte is sent to check the parity of a given position of the previous bytes. If we imagine sent bytes as a table of data, two-dimensional parity can not only check the rows of the table like single parity, it can also check the columns to provide an additional layer of detection [4]. Equally simple, a checksum detection method simply adds up the content of the data being transmitted, and transmits the result. The receiver can then perform the same calculation as the sender, and if the computed checksum doesn't match the received checksum, an error has occurred [5]. Finally, a little more complex, a cyclical redundancy check algorithm treats the transmitted data as a binary polynomial and divides it by a predefined polynomial known as the generator polynomial. This creates a checksum, which is then translated along with the data. Similar to a checksum algorithm, the receiver then repeats the calculation and compares it to the transmitted value; differences in these values means an error has occurred [6]. Existing literature on the topics of these algorithms for error detection in networks generally fail to meet the specifications this simulation hopes to fulfill. Most of the existing literature either only provides analysis of theory [7], focuses only on a specific network [8], or focuses on a specific algorithm [9]. This study will provide a realistic simulation in which multiple algorithms can be tested on different network types, wired and wireless, and using different transmission protocols. No similar studies, simulations, or surveys matching these specifications were found during research for this project.

Methodology

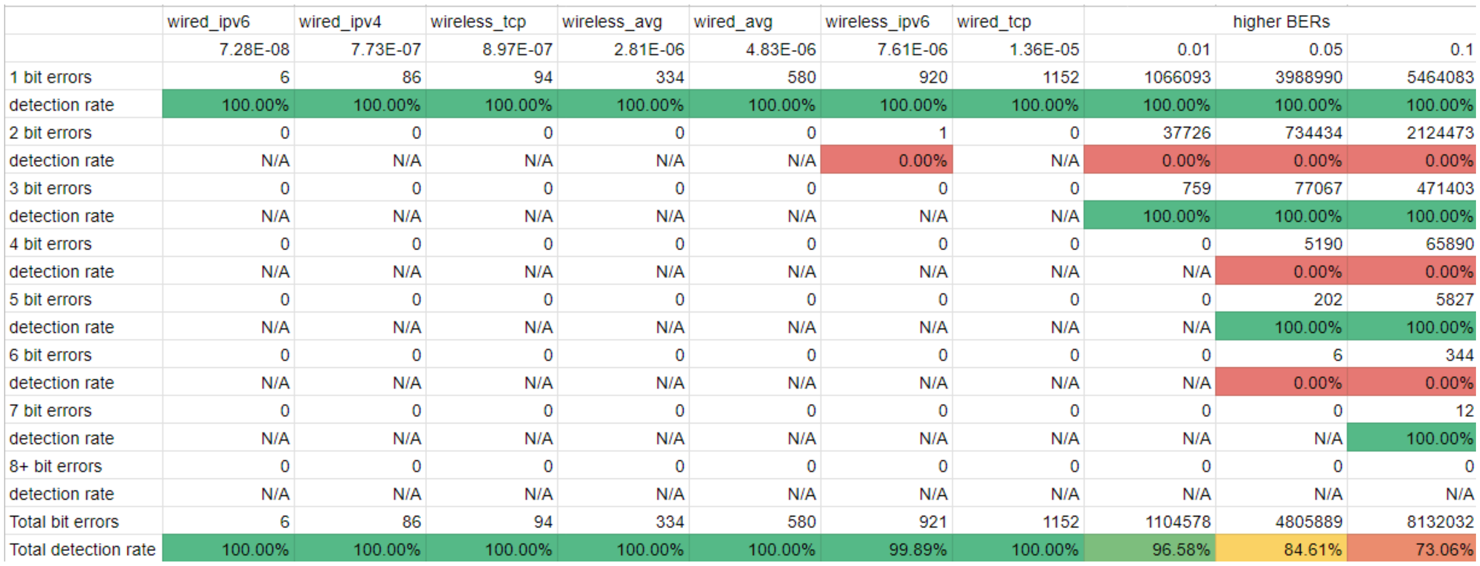
This simulation was performed fully programmatically. Prior to running the body of the simulation, the python script generated a file with 100 million randomly generated bits, which would serve as the dataset for all upcoming simulations. First, bits were read from the file and packaged according to the error detection algorithm being performed. Then, bit errors would be randomly introduced at rates consistent with those defined by Okpeki et al [1] (as well as some higher error rates to test the limits of the algorithm). Finally, the data would be passed to a corresponding function which would attempt to detect whether an error was present, with the results being tabulated.

As previously discussed, four error detection algorithms were implemented for use in this simulation. Single parity was implemented using odd parity, with one parity bit for each seven bits of data being sent. 2D parity was implemented similarly, with one parity bit for each seven data bits, and one parity byte for each seven data bytes. Meaning that for each 7x7 matrix of data prior to processing, an 8x8 matrix was sent with 2D parity implemented. In order to implement checksum, two bytes of data were appended with an additional checksum byte, derived from adding the two data bytes together. Finally, CRC was implemented by adding a simple three bit remainder to each six data bits. This remainder was found by dividing the data bits with a key; in this simulation a key of “1101” was used, corresponding to a generator polynomial of $x^3 + x^2 + 1$. The results of each were tracked, including their total error detection rate, as well as their detection rate of each type of bit error, from a one bit error to an eight or greater bit error.

Results and Discussion

As this simulation is really four different simulations wrapped up in one, we’ll discuss each algorithm being simulated in the order they were introduced in this paper. Starting with single parity, followed by 2D parity, checksum, and CRC. This section will be wrapped up by comparing the results of each.

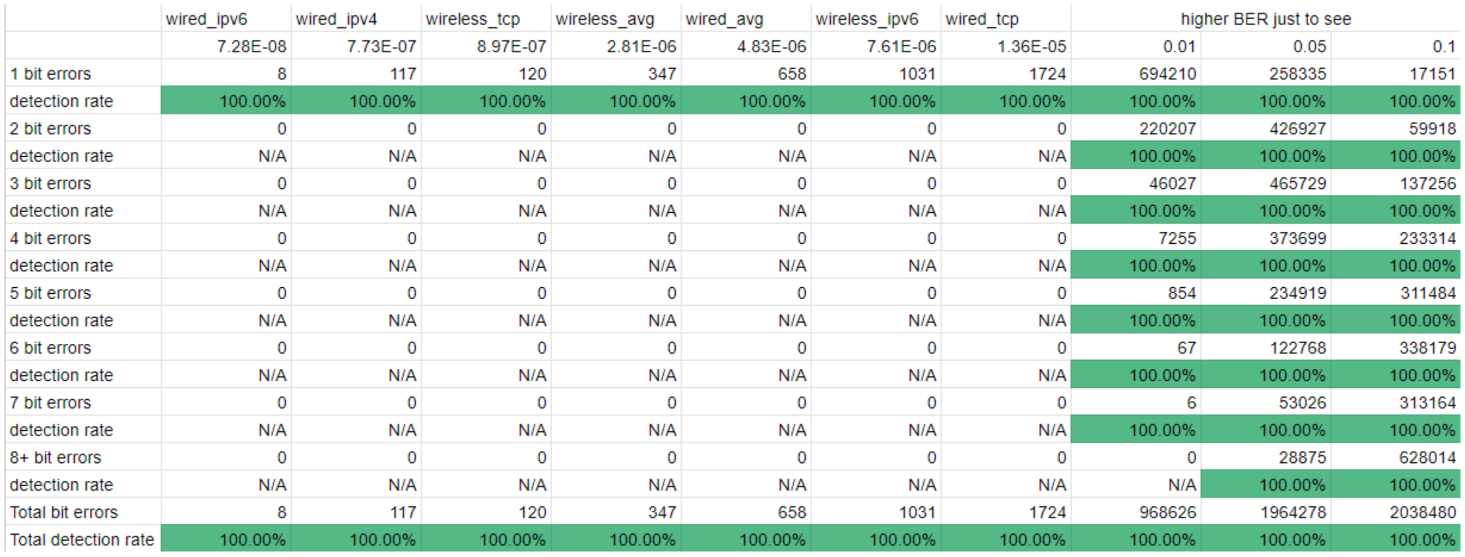
The results of the single parity simulation are displayed in the table below. Fields labeled with “0” and “N/A” are the result of no errors of that order being introduced, despite the high number of bits passing through. This was relatively common among the lower error rates explored throughout this simulation.



	wired_ipv6	wired_ipv4	wireless_tcp	wireless_avg	wired_avg	wireless_ipv6	wired_tcp		higher BERs	
	7.28E-08	7.73E-07	8.97E-07	2.81E-06	4.83E-06	7.61E-06	1.36E-05	0.01	0.05	0.1
1 bit errors	6	86	94	334	580	920	1152	1066093	3988990	5464083
detection rate	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
2 bit errors	0	0	0	0	0	1	0	37726	734434	2124473
detection rate	N/A	N/A	N/A	N/A	N/A	0.00%	N/A	0.00%	0.00%	0.00%
3 bit errors	0	0	0	0	0	0	0	759	77067	471403
detection rate	N/A	N/A	N/A	N/A	N/A	N/A	N/A	100.00%	100.00%	100.00%
4 bit errors	0	0	0	0	0	0	0	0	5190	65890
detection rate	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	0.00%	0.00%
5 bit errors	0	0	0	0	0	0	0	0	202	5827
detection rate	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	100.00%	100.00%
6 bit errors	0	0	0	0	0	0	0	0	6	344
detection rate	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	0.00%	0.00%
7 bit errors	0	0	0	0	0	0	0	0	0	12
detection rate	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	100.00%
8+ bit errors	0	0	0	0	0	0	0	0	0	0
detection rate	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Total bit errors	6	86	94	334	580	921	1152	1104578	4805889	8132032
Total detection rate	100.00%	100.00%	100.00%	100.00%	100.00%	99.89%	100.00%	96.58%	84.61%	73.06%

Consistent with expectations, single parity is 100% effective at detecting any odd-numbered errors, while entirely ineffective at detecting even-numbered errors. Interestingly though, despite the simplicity of single parity, it was sufficient to detect all of the errors introduced by the realistic BER's, with the exception of a single 2-bit error introduced using an error rate consistent with Wireless IPv6. This means that in certain networks for which an occasional undetected bit error might be acceptable, single parity may actually be a viable choice. With its low overhead, it offers increased efficiency in exchange for occasional (one in every 100 million or less) undetected bit errors (at realistic BERs). However, it must be noted that this simulation did not artificially introduce burst errors, which may reduce its viability in real-life conditions.

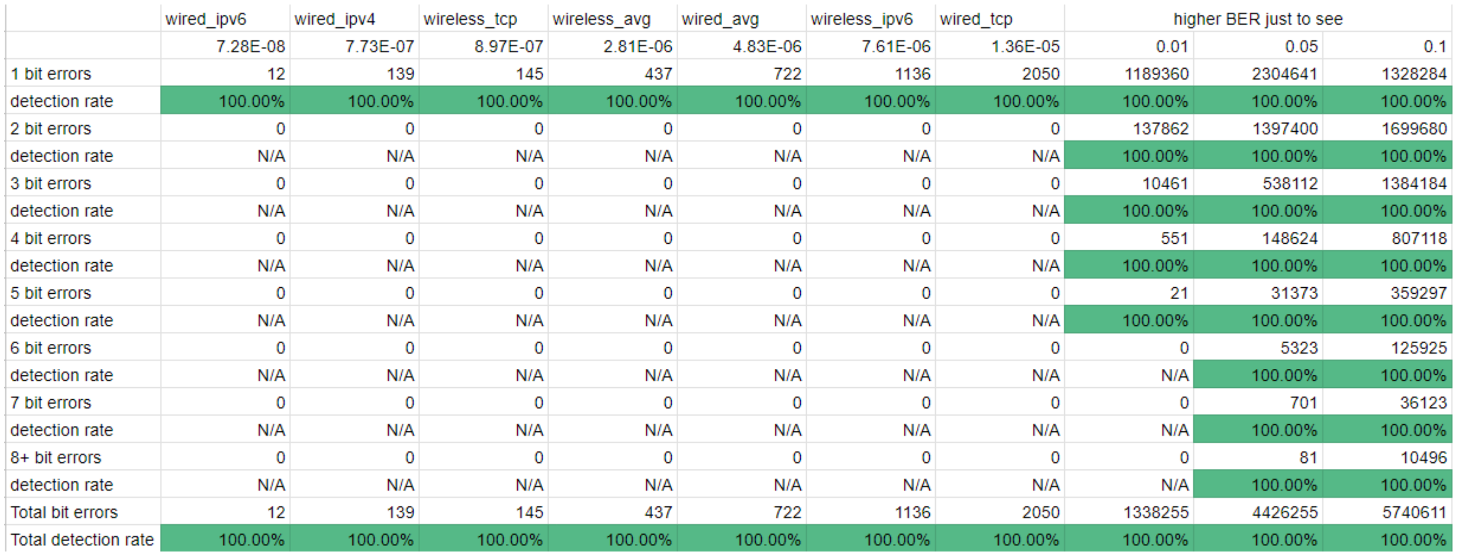
The results of 2D parity proved to be a bit more effective. The results of that simulation can be seen below. It is worth noting here that errors were tabulated based on the entire matrix rather than a given byte. This means that for a 4 bit error in the table, four bits in the generated 8x8 matrix were flipped, rather than four bits being flipped in a given byte.



	wired_ipv6	wired_ipv4	wireless_tcp	wireless_avg	wired_avg	wireless_ipv6	wired_tcp	higher BER just to see		
	7.28E-08	7.73E-07	8.97E-07	2.81E-06	4.83E-06	7.61E-06	1.36E-05	0.01	0.05	0.1
1 bit errors	8	117	120	347	658	1031	1724	694210	258335	17151
detection rate	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
2 bit errors	0	0	0	0	0	0	0	220207	426927	59918
detection rate	N/A	N/A	N/A	N/A	N/A	N/A	N/A	100.00%	100.00%	100.00%
3 bit errors	0	0	0	0	0	0	0	46027	465729	137256
detection rate	N/A	N/A	N/A	N/A	N/A	N/A	N/A	100.00%	100.00%	100.00%
4 bit errors	0	0	0	0	0	0	0	7255	373699	233314
detection rate	N/A	N/A	N/A	N/A	N/A	N/A	N/A	100.00%	100.00%	100.00%
5 bit errors	0	0	0	0	0	0	0	854	234919	311484
detection rate	N/A	N/A	N/A	N/A	N/A	N/A	N/A	100.00%	100.00%	100.00%
6 bit errors	0	0	0	0	0	0	0	67	122768	338179
detection rate	N/A	N/A	N/A	N/A	N/A	N/A	N/A	100.00%	100.00%	100.00%
7 bit errors	0	0	0	0	0	0	0	6	53026	313164
detection rate	N/A	N/A	N/A	N/A	N/A	N/A	N/A	100.00%	100.00%	100.00%
8+ bit errors	0	0	0	0	0	0	0	0	28875	628014
detection rate	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	100.00%	100.00%
Total bit errors	8	117	120	347	658	1031	1724	968626	1964278	2038480
Total detection rate	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%

As can be seen in the table, 2D parity was 100% effective at detecting errors of all types in this simulation. Of course, error patterns do exist which can evade the detection of 2D parity, but these are so rare that not a single one appeared with 100 million bits being sent at each error rate. This means that 2D parity is extremely effective; however, this error detection capability comes at a cost, with 23.4% of each bit being transmitted as overhead.

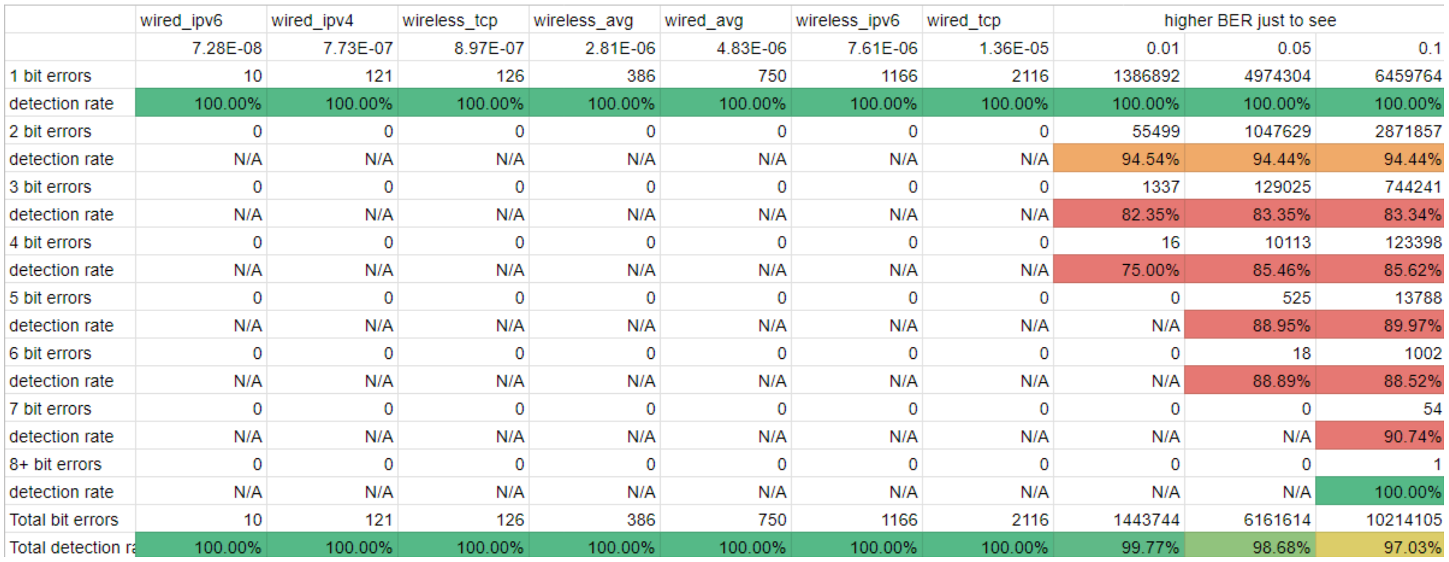
Checksum proved to be equally as effective, as can be seen in the following table. It is worth noting that bit errors are once again referring to errors in the entire data transferred, including both data bytes and the checksum byte, rather than any individual byte.



	wired_ipv6	wired_ipv4	wireless_tcp	wireless_avg	wired_avg	wireless_ipv6	wired_tcp	higher BER just to see		
	7.28E-08	7.73E-07	8.97E-07	2.81E-06	4.83E-06	7.61E-06	1.36E-05	0.01	0.05	0.1
1 bit errors	12	139	145	437	722	1136	2050	1189360	2304641	1328284
detection rate	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
2 bit errors	0	0	0	0	0	0	0	137862	1397400	1699680
detection rate	N/A	N/A	N/A	N/A	N/A	N/A	N/A	100.00%	100.00%	100.00%
3 bit errors	0	0	0	0	0	0	0	10461	538112	1384184
detection rate	N/A	N/A	N/A	N/A	N/A	N/A	N/A	100.00%	100.00%	100.00%
4 bit errors	0	0	0	0	0	0	0	551	148624	807118
detection rate	N/A	N/A	N/A	N/A	N/A	N/A	N/A	100.00%	100.00%	100.00%
5 bit errors	0	0	0	0	0	0	0	21	31373	359297
detection rate	N/A	N/A	N/A	N/A	N/A	N/A	N/A	100.00%	100.00%	100.00%
6 bit errors	0	0	0	0	0	0	0	0	5323	125925
detection rate	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	100.00%	100.00%
7 bit errors	0	0	0	0	0	0	0	0	701	36123
detection rate	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	100.00%	100.00%
8+ bit errors	0	0	0	0	0	0	0	0	81	10496
detection rate	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	100.00%	100.00%
Total bit errors	12	139	145	437	722	1136	2050	1338255	4426255	5740611
Total detection rate	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%

Similar to 2D parity, checksum was 100% effective. However, checksum requires 33.3% overhead, whereas 2D parity only requires 23.4%. As checksum is based on the addition of two bytes, it is also susceptible to offsetting errors which wouldn’t affect the resulting checksum. However, these errors are rare enough that they did not appear a single time for any simulated error rate.

Finally, we have the implementation of CRC, the data for which is seen below. These results are not what were expected. CRC struggled with higher order errors more than the other detection methods.



	wired_ipv6	wired_ipv4	wireless_tcp	wireless_avg	wired_avg	wireless_ipv6	wired_tcp	higher BER just to see		
	7.28E-08	7.73E-07	8.97E-07	2.81E-06	4.83E-06	7.61E-06	1.36E-05	0.01	0.05	0.1
1 bit errors	10	121	126	386	750	1166	2116	1386892	4974304	6459764
detection rate	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
2 bit errors	0	0	0	0	0	0	0	55499	1047629	2871857
detection rate	N/A	N/A	N/A	N/A	N/A	N/A	N/A	94.54%	94.44%	94.44%
3 bit errors	0	0	0	0	0	0	0	1337	129025	744241
detection rate	N/A	N/A	N/A	N/A	N/A	N/A	N/A	82.35%	83.35%	83.34%
4 bit errors	0	0	0	0	0	0	0	16	10113	123398
detection rate	N/A	N/A	N/A	N/A	N/A	N/A	N/A	75.00%	85.46%	85.62%
5 bit errors	0	0	0	0	0	0	0	0	525	13788
detection rate	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	88.95%	89.97%
6 bit errors	0	0	0	0	0	0	0	0	18	1002
detection rate	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	88.89%	88.52%
7 bit errors	0	0	0	0	0	0	0	0	0	54
detection rate	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	90.74%
8+ bit errors	0	0	0	0	0	0	0	0	0	1
detection rate	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	100.00%
Total bit errors	10	121	126	386	750	1166	2116	1443744	6161614	10214105
Total detection ra	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	99.77%	98.68%	97.03%

While effective at detecting 1-bit errors, CRC was less effective at detecting higher order errors. This is because certain error patterns, particularly those related multiples of the generator polynomial can cause results at the receiver that appear as if there are no errors. Whatever the reason, at 33.3% overhead the results produced by CRC are outclassed by 2D parity and checksum (and arguably single parity for realistic conditions!). Although there are of course nuances to the validity of each method. One positive of CRC however, is that despite its high overhead, it operates on smaller packets of data. This means that if an error is detected, a sender using CRC will only have to resend nine bits, compared to sixty-four and twenty-four for 2D parity and checksum respectively. This may result in CRC having a lower effective overhead depending on network conditions.

All four error detection schemes were 100% effective at detecting single bit errors, as expected. However, their effectiveness varied when facing higher order errors. Single parity was fully effective in detecting all odd-numbered errors, but was entirely ineffective with even-ordered errors. CRC was fully effective for single bit errors, with declining efficacy for higher bit errors. Both 2D and checksum were fully effective for all errors introduced in this simulation. It’s worth noting that bit errors larger than a single flipped bit were extremely rare for all realistic BERs. Across four error detection algorithms, 100 million bits were simulated for seven realistic error rates. Out of all of these (approximately 2.8 billion bits!), only a single two-bit error was observed. This means that for systems that value efficiency and are tolerant to extremely rare undetected errors, the error detection algorithms which failed to detect higher-number errors may still be viable options.

Conclusions

This simulation tested single parity, 2D parity, checksum, and CRC algorithms with a variety of realistic and inflated error rates to determine their efficacy in realistic simulations of wireless and wired networks. The 2D parity and checksum algorithms were 100% effective at detecting all errors. Single parity was able to detect all odd-number errors, and no even number errors. CRC was fully effective at detecting one-bit errors, which reduced efficacy for higher number errors. Based on the result of these simulations, it is recommended that networks which prioritize efficiency and are tolerant to occasional missed errors implement single-parity error detection for realistic error rates similar to those used in this simulation. For networks in which undetected errors are unacceptable, 2D parity is most efficient while detecting nearly all errors at normal BER values. For networks with unusually high error rates, closer to the tested 1%, 5% and 10% error rates, checksum will be most effective as it detects nearly all errors, and will reduce the overhead resulting from resent packets. The results of this study have determined that there is no practical difference between wireless and wired networks when it comes to picking an error detection scheme, at least for the experimental BER values in the scope of this paper.

References

[1] Okpeki, U. K., Egwaile, J. O., & Edeko, F. (2019). Performance and Comparative Analysis of Wired and wireless communication systems using Local Area Network based on IEEE 802.3 and IEEE 802.11. Journal of Applied Sciences and Environmental Management, 22(11), 1727. https://doi.org/10.4314/jasem.v22i11.3

[2] Computer Network: Error Detection - javatpoint. www.javatpoint.com. (n.d.). https://www.javatpoint.com/computer-network-error-detection

[3] Ferguson, S., & Hebels, R. (2003). CHAPTER 7 - Communications and networking. In Topics in Australasian Library and Information Studies. essay. Retrieved July 10, 2024, from https://www.sciencedirect.com/science/article/abs/pii/B9781876938604500136.

[4] Laghane, P. (n.d.). Two dimensional parity check - piyu's CS. piyuscs.com. https://piyuscs.com/two-dimensional-parity-check/

[5] Peterson, L., & Davie, B. (2023). Chapter 2: Direct Links. In Computer Networks: A Systems Approach. Retrieved October 7, 2024,.

[6] Lenovo. (n.d.). What is a cyclic redundancy check (CRC) and how does it work?. What is Cyclic Redundancy Check (CRC) and How Does it Work? | Lenovo US. https://www.lenovo.com/us/en/glossary/cyclic-redundancy-check/

[7] Panem, C., Gad, V., & Gad, R. (2019). Polynomials in Error Detection and Correction in Data Communication System. In Coding Theory.

[8] Kumar, N., Kedia, D., & Purohit, G. (2023). A review of Channel Coding Schemes in the 5G standard. Telecommunication Systems, 83(4), 423–448. https://doi.org/10.1007/s11235-023-01028-y

[9] Gong, C.-S. A., Chang, Y.-C., Huang, L.-R., Yang, C.-J., Ji, K.-M., Lu, K.-L., & Liou, J.-C. (2018). Two dimensional parity check with variable length error detection code for the non-volatile memory of Smart Data. Applied Sciences, 8(8), 1211. https://doi.org/10.3390/app8081211

[10] Cyclic redundancy check and modulo-2 division. GeeksforGeeks. (2024, May 8). https://www.geeksforgeeks.org/modulo-2-binary-division/

[11] Error detection code - checksum. GeeksforGeeks. (2024b, October 25). https://www.geeksforgeeks.org/error-detection-code-checksum/

[12] Eckhardt, D., & Steenkiste, P. (1996). Measurement and analysis of the error characteristics of an in-building wireless network. Conference Proceedings on Applications, Technologies, Architectures, and Protocols for Computer Communications, 243–254. https://doi.org/10.1145/248156.248178

[13] R. Khalili and K. Salamatian, "A new analytic approach to evaluation of packet error rate in wireless networks," 3rd Annual Communication Networks and Services Research Conference (CNSR'05), Halifax, NS, Canada, 2005, pp. 333-338, doi: 10.1109/CNSR.2005.14.

[14] Agrawal, D. P., Proakis, J. G., Takagi, H., Tobagi, F. A., Cali, F., Tay, Y. C., & Bianchi, G. (2005, August 31). Impact of bursty error rates on the performance of Wireless Local Area Network (WLAN). Ad Hoc Networks. https://www.sciencedirect.com/science/article/abs/pii/S1570870505000521

[15] Hari Balakrishnan, Srinivasan Seshan, Elan Amir, and Randy H. Katz. 1995. Improving TCP/IP performance over wireless networks. In Proceedings of the 1st annual international conference on Mobile computing and networking (MobiCom '95).

Association for Computing Machinery, New York, NY, USA, 2–11. https://doi.org/10.1145/215530.215544

[16] GeeksforGeeks. (2024c, July 12). Error detection in computer networks. GeeksforGeeks. https://www.geeksforgeeks.org/error-detection-in-computer-networks/

[17] Mesander, B. (2023, November 10). What is CRC networking? understanding the cyclic redundancy check. Cardinal Peak. https://www.cardinalpeak.com/blog/understanding-the-cyclic-redundancy-check

[18] What is a CRC (cyclic redundancy check)?. CBT Nuggets. (n.d.). https://www.cbtnuggets.com/blog/technology/networking/what-is-crc-cyclic-redundancy-check

[19] Yasar, K., & Fitzgibbons, L. (2022, December 12). What is a checksum?: A definition from techtarget.com. Search Security. https://www.techtarget.com/searchsecurity/definition/checksum

[20] Synopsys. (n.d.). https://www.synopsys.com/dw/dwtb/ethernet_mac/ethernet_mac.html

Appendices

Appendix I - Source Code, data used, and other resources

The source code for this simulation, the data used for the run presented in this paper, instructions for use, raw output, and other information can be found at https://github.com/DaelonShockley/Wired-Wireless-Network-Error-Detection-Simulation/tree/main

Appendix II - Results Tables

The table of results for this simulation can be found at https://docs.google.com/spreadsheets/d/1QxSQ2cAD0QKwCyvgvanRyame4YJOVAxU88V1dYupSiE/edit?usp=sharing