Constant Time, Fixed Memory, Zero False Negative Error Logging for Low Power Wearable Devices

Saffiyah Doolan*, Nicholas Hosein†, Patrick Hosein* and Devika Bhagwandin*
*Department of Computer Science, The University of the West Indies, St. Augustine, Trinidad
†
Email: saffiyah@lab.tt, nhosein@ucdavis.edu, patrick.hosein@sta.uwi.edu, devika@lab.tt

***Abstract*—Wireless wearable embedded devices dominate the Internet of Things (IoT) due to their ability to provide useful information about the body and its local environment. The constrained resources of low power processors, however, pose a significant challenge to run-time error logging and hence, product reliability. Error logs classify error type and often system state following the occurrence of an error. Traditional error logging algorithms attempt to balance storage and accuracy by selectively overwriting past log entries. Since a specific combination of firmware faults may result in system instability, preserving all error occurrences becomes increasingly beneficial as IOT systems become more complex. In this paper, a novel hash-based error logging algorithm is presented which has both constant insertion time and constant memory while also exhibiting no false negatives and an acceptable false positive error rate. Both theoretical analysis and simulations are used to compare the performance of the hash-based and traditional approaches.**

***Index Terms*—Error Logging; Hashing; Wearable Devices, Embedded Devices; Body Sensors; Caching Algorithm**

I. Introduction

Wearable embedded devices are growing in popularity due to their ability to connect users to their internal physiology and external environments [1].These devices, also referred to as smart wearables, utilize System on a Chip (SoC) technology to reduce device size, extend battery life and consolidate many typical IOT hardwares, such as wireless communication and external flash, into a single chip. Wireless-enabled smart wearables are becoming increasingly common in the health, sport and entertainment industries as manufacturing cost go down due to increasing consumer demands. Wearable devices are required to collect and transmit valuable data without user intervention, independently for extended periods while handling internal errors caused by physical abuse, electromagnetic influence and firmware bugs. As such, error detection becomes a critical aspect of wearable systems, especially given the long runtimes and high reliability expected of consumer and medical devices.

For these high stress embedded systems, error handling poses a significant challenge [2]. When a fatal error occurs, its properties must be stored for analysis despite restrictions in computational resources. The small flash storage of the SoC is almost entirely used for program files, program execution, data constants and sometimes for offline sensor data storage. Some standard microcontrollers such as [3] comprise as little as 128KB flash memory. For sophisticated systems with many sensors and algorithms running simultaneously, storage of potentially large numbers of errors becomes critical as missing fatal error information may result in delayed firmware patches or inability to isolate the source of the fault altogether.

Current solutions perform error logging by overwriting old log entries when log memory runs out or by logging errors selectively at the risk of excluding potentially useful minor errors. In addition to saving all errors, real-time logging capability is a necessary feature as periodic scheduled error assessments could be preempted by a critical failure resulting in an error not being logged at all. In order to log system errors efficiently and reliably where memory and processing power are both extremely limited, an error logging algorithm that operates in constant time, fixed memory and which exhibits zero false negatives is proposed. In this novel approach, log memory size is defined based on the acceptable false positives rate and given a maximum expected error rate. In the proposed approach only false positives are possible while in the more common rotating log method only false negatives are possible. False positives result in checking for an error that may not exist (extra work) while false negatives result in missing potentially critical errors (low reliability). Both analytic and simulation results provided comparisons for both the proposed and traditional approach.

The structure of the paper is as follows. Firstly, previous works are covered. Secondly, the proposed constant time, constant memory error logging algorithm is outlined. Next, mathematical model and analytic results presented for both proposed and traditional methods followed by simulation results. Final thoughts are then provided in the conclusion.

II. Related Work

In this study we propose the use of hash functions to store errors in wireless sensor embedded devices. This is an effort to address the issue of fixed memory storage. A hash function is a function that maps data of an arbitrary size to a fixed size. They may be distribution-dependent, distribution-independent, cluster-separating, multiplicative or index function-based [4].

In past studies, hash functions have been used to perform several tasks. Cryptographic hash function use (e.g., [5]) ranges from constructing message authentications [6], [7] to implementing key derivations [8]. In medical applications, hashing algorithms have been used to process k-mers in DNA sequencing [9] as well as produce unique identifications of patients while maintaining their privacy [10], [11]. Other uses

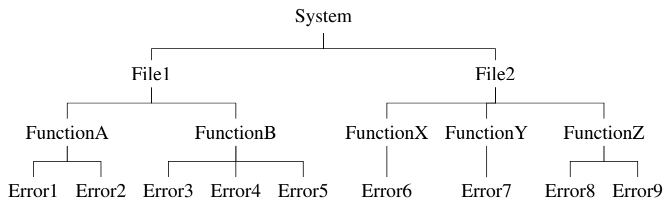

Fig. 1. Example of Error Generation

of hash functions include robust audio hashing to identify audio content [12] and image hashing for image authentication [13].

Studies [14] and [15] detail the use of hash functions for memory reduction. [14] proposed a hashing method for retrieval of lung module CT images. The dataset was partitioned into clusters, and the method was applied to compress high-dimensional image features into a compact hash code thereby reducing retrieval time and memory. [15] presented *Lighter*, a fast and memory efficient tool for sequencing error correction. It uses a pair of bloom filters and this avoids the counting of k-mers. Bloom filters have also been used in [16] to detect and correct errors in their associated datasets.

III. Proposed Error Logging Algorithm

A. Algorithm Description

In this section we provide a detailed description of the structure and operation of the algorithm. The algorithm provides all errors that have occurred along with details on the parameters of the errors. Program files comprise of functions each of which may generate errors. Figure 1 illustrates an example of the topology of a generic small wireless sensor technology system. It is a hierarchical structure consisting of three parameters belonging to each error: file ids, function ids and error ids. The size of the system depends on the application of the technology.

A simple one column hash table is used to store information relating to errors that have occurred. Slots in the table are initially filled with a bit 0 value. When an error x occurs, a hash function, $H(x)$ is performed on x which also acts as a key. The resulting hash value indicates the position of a slot in the hash table that must be flagged. The encoding process is the flagging of slots. The hashing may be a simple prime modulo function such that $H(x) = x \mod M$ where M is a prime number. This prevents bad behavior conflict misses from occurring within the hash table [17]. Thus, when an error occurs hashes are performed independently on the file, the function and the error and the associated slot for each is flagged by flipping the bit from 0 to 1.

Decoding the hash table is the end user activity which cross-references the entire error dataset with the completed hash table. Each parameter of an error is checked for its corresponding slot position as calculated by the same hash function. When all three parameters of an error appear to have been flagged, an error is said to have occurred. When two or more parameters are decoded with similarly flagged slots, a collision is said to have occurred. This is not concerning when the parameters have rightfully been flagged to those slots. However, when an unlogged error (never occurred) shows flagged bits due to other errors that did occur, then we would mistakenly believe that the unlogged error occurred (false positive). False positives arise when the slots of all parameters of an error are flagged, even though one or more of the error parameters were not logged and simply happen to collide with the slot of a logged parameter [18]. Therefore, the number of false positives is used as the performance metric for the proposed approach.

TABLE I
Sample Probability Computation

n	**False Positive Probability Vector**
1	[0, 1, 0, 0, 0, 0]
2	[0, 0.2, 0.8, 0, 0, 0]
3	[0, 0.04, 0.48, 0.48, 0, 0]

B. False Positive Probability Computation

For our proposed algorithm, we explore the options of inserting error data of different levels into the table. In general, we consider the case of κ insertions into the hash table per logged error. So if we log the file, function and error then $\kappa = 3$, if we log only the function and the error then $\kappa = 2$ and finally if we only log the error then $\kappa = 1$. We use the following notation in the analysis:

E	= Total number of logged errors
$N = \kappa E$	= Number of attempted insertions
T	= Total number of possible errors
M	= Number of hash table buckets (slots)
$p(n,k)$	= probability k entries after n insertions

We first compute $p(n,k)$ and then use this to compute the probability of a false positive. We do this recursively as follows. Consider the case of $n = 1$. Since a new insertion will be made then we have $p(1,1) = 1$ and $p(1,k) = 0$ for $k \neq 1$. Now let us assume that $p(n-1,k)$ is known for $0 \leq k \leq M$ and compute $p(n,k)$. Given that k hash entries have been made, there are two possibilities, either the new attempt results in a new entry for a total of $k+1$ bit flips or the hash falls on a previously flipped entry. Therefore

$$p(n,k) = p(n-1,k)\frac{k}{M} + p(n-1,k-1)\frac{M-k+1}{M} \quad (1)$$

and so we can recursively solve for $p(N,k)$. For example suppose that $M = 5$ then the probability vector for $k = 0, 1, \ldots, 5$ for $n = 1, \ldots, 3$ can be computed and is given in Table I.

Given $p(N,k)$ for $k = 0, \ldots, M$ we now determine the false positive probability. An error is falsely determined to be positive if the error did not occur but the κ hashed positions for the error have all been previously flipped. Suppose that N/κ errors have been logged. For convenience consider the case of $\kappa = 3$ in which the file, function and error are hashed

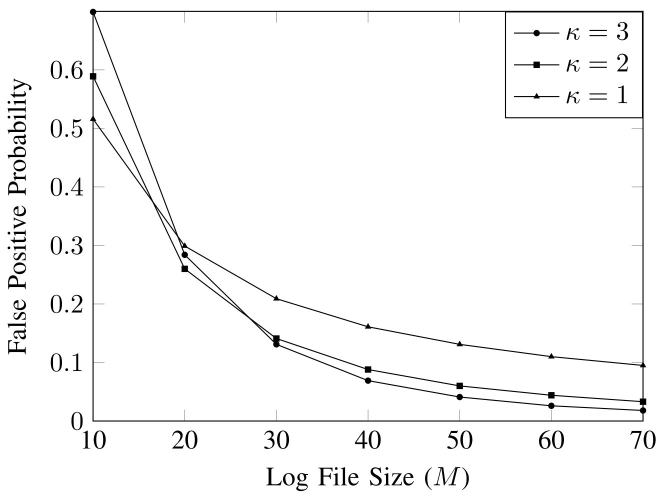

Fig. 2. Probability of False Positives showing the dependence on κ

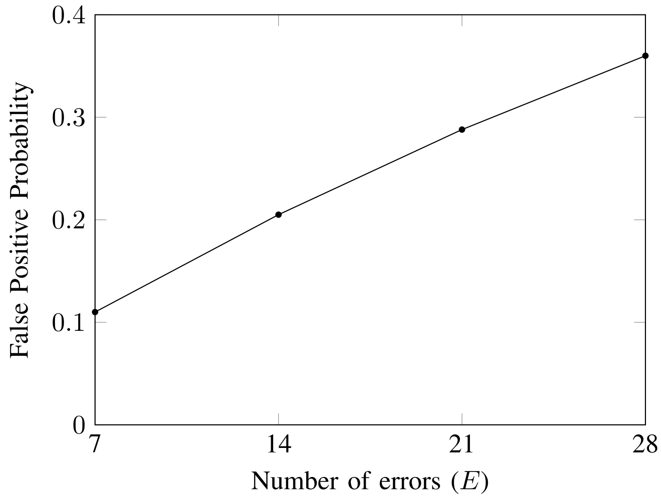

Fig. 3. Probability of False Positives as a function of number of errors E

and logged. The probability that the hashed location for the file is already flipped is the sum over k the probability that k bits have been flipped times the probability that the hashed location of the file is already flipped (k/M):

$$\sum_{k=1}^{M} p(N,k)\frac{k}{M}.$$

This is the same probability for the function insertion and the error insertion. Therefore, in general, if κ insertions are attempted per error then the probability that all κ insertions fall on already flipped bits is given by:

$$\left[\sum_{k=1}^{M} p(N,k)\frac{k}{M}\right]^{\kappa}$$

Since there are a total of T possible errors and E have been logged then the probability that a chosen error was not logged is $(T-E)/T$, Therefore the probability of a false positive, the probability that an error was not logged times the probability that it is indicated as being logged is given by:

$$P_{fp} = \frac{T-E}{T}\left[\sum_{k=1}^{M} p(\kappa E,k)\frac{k}{M}\right]^{\kappa} \tag{2}$$

C. Numerical Examples

In this section we will provide some illustrative examples. We first investigate which value of κ results in the lowest P_{fp}. We consider the case $T = 700$ total errors and $E = 7$. In Figure 2 we plot the probability of false positives as a function of hash table size for different values of κ. We find that for small hash table sizes, smaller κ values are better. However, in this region, the false positive probability is too large for practical purposes. For large hash tables, larger values of κ are better. Hence our proposed approach of hashing and inserting file, function and error ids is preferable over simply logging the error.

Next we show how the false positive probability varies with the number of errors generated. We use the above example with a fixed $M = 60$. We then varied the number of errors generated from 7 (1%) to 28 (4%). The resulting plot is provided in Figure 3.

IV. Comparison with Traditional Error Logging

In the traditional error logging approach, errors are continuously logged and when the log file is full then previous entries are overwritten. Hence the log file will never have an entry that did not happen (a false positive) but may not have an entry for an error that did occur (a false negative). We use the same notation as in the previous section. If there are a total of T possible errors then to be able to uniquely encode these would require $\lceil \log_2(T) \rceil$ bits per error. Since there are a total of M bits available for logging errors then a maximum of $\lfloor M/\lceil \log_2(T) \rceil \rfloor$ errors can be stored. If fewer errors occur then the false negative probability is zero but if more errors occur then the false negative probability is given by:

$$P_{fn} = \frac{1}{T}\left\{E - \left\lfloor \frac{M}{\lceil \log_2(T) \rceil} \right\rfloor\right\} \tag{3}$$

Note that a false positive results in the checking of an error that did not occur. The cost of doing this is the additional time spent in troubleshooting but it is not catastrophic to the product. On the other hand the cost of a false negative can be significant since it results in an error that occurred but for which there is no way in determining the error and debugging the application. Let us denote the cost of a false positive by 1 and the cost of a false negative by $c > 1$. We compare the two approaches by comparing the total expected cost C which is given by:

$$C = P_{fp} + cP_{fn} \tag{4}$$

Let us consider a simple example with $T = 700$ possible errors and assume that 1% of these occur (i.e., $E = 7$). As mentioned before, false negative errors are not known and this

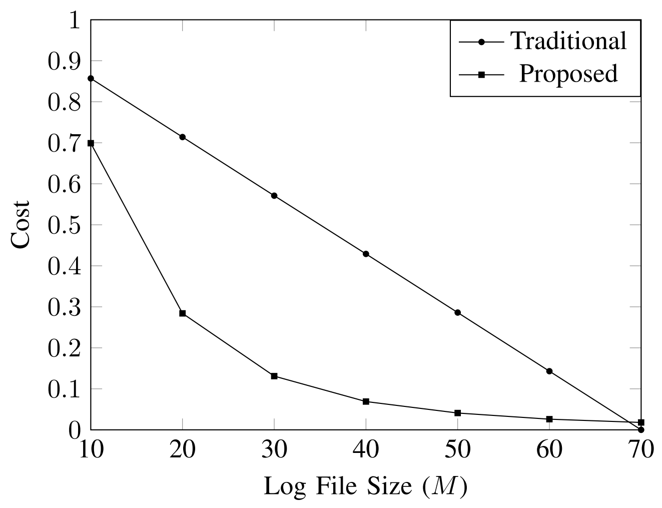

Fig. 4. Cost Comparison of Traditional and Proposed Approaches

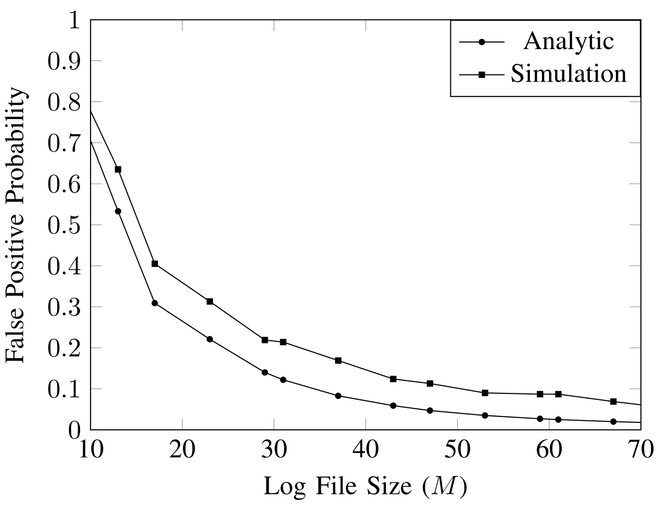

Fig. 5. Validation of Analytic Model

TABLE II
SIMULATION PARAMETERS

Number of files	10
Number of functions per file	10
Number of errors per function	7
Number of Errors	7
Range of M (Prime Numbers Used)	10 to 70

can lead to loss of reliability and reputation of the product (order of tens of millions in cost). A false positive error means additional debugging has to be performed (order of tens of thousands in cost). Therefore one would expect c to be in the 1000 value range. We use a conservative value of $c = 100$. In Figure 4 we plot the resulting cost for the two approaches as a function of the log file size. Note that for the region of interest to us (small memory size) there is a significant cost gain in using the proposed approach.

V. Validation of Analytic Model with Simulations

In the previous analytical results, we assumed that the hashed result was equally likely to occur in any of the slots. In practice this may not be the case and so we performed simulations with the actual hashing function to validate the model and its assumptions. The simulation parameters are provided in Table II.

A large dataset with $\kappa = 3$ was built to include the id parameters of files, functions and errors. Unique ids up to 2^{32} bits in size were generated. The encoding process allowed respective id parameters of randomly selected errors to be encoded into the table. The hash function $H(x) = x \mod M$ was used to flag respective slots on the table.

The decoding process is a depth first search algorithm that utilizes the generated dataset. A node in the depth first search is defined as one of the κ parameters of an error. The hierarchical structure of the dataset is followed as each node, starting with the file ids, is checked against the hash table. The user moves adjacently until a node is flagged. A flagged node occurs when its hash value points to a flipped bit in the hash table. For instance, starting at a particular file id, the algorithm travels to the adjacent file id until it flags a node and travels to the first function id of the flagged file node. It moves to the next function id until a function id is flagged and it traverses to its error id node. The id parameters of flagged nodes are recorded until the top layer of file ids has been exhausted completely. When all κ parameters of an error provide bit 1 values, the error is decoded. The errors retrieved during decoding are compared to the true errors encoded into the hash table to count the total number of false positives present. Since the errors are randomly selected in the simulation, the encoding/decoding processes were performed for 100 trials and the arithmetic means for each M value were calculated. In Figure 5 we plot the analytic and simulation results. We find an acceptable match between the analytic and simulation results and hence the performance benefits of the proposed approach when compared with the traditional approach (as described above) are valid.

VI. Performance for Typical Applications

Embedded devices vary in levels of resource availability and error logging requirements. We can determine the probability of false positives using suitable parameters for three variables, the hash table size, the number of true errors and the total number of possible errors. Table III compares the probability of false positives for various embedded devices. The parameters chosen are reflective of those used for the specific devices. The results show the robustness of the error logging algorithm.

For example, a commercial heart rate monitor band or fitness band utilizing photopletysmography may incur significant errors [19]. This is possibly due to measurement and skin conditions, poor sensor placements, errors in syncing with a smart phone or updating. Although the overall reliability of the device is not severely affected as it is not intended

TABLE III
PERFORMANCE FOR VARIOUS EMBEDDED DEVICES

Embedded Device	T	M	E	P_{fp}
Anti-Lock Braking	2000	499	20	0.0218
Microwave Oven	2000	241	20	0.0529
Fitness Band	200	241	80	0.1668
Smart Energy Control Appliance	2000	1201	400	0.2650

for collecting granular data. With this application, assuming relatively medium sizes of the dataset and the hash table, such a fitness band experiences a low level of false positives in its error cache.

Other types of embedded devices included in the above table are described as follows. An anti-lock braking system requires real-time processing to keep wheels from skidding by a variety of functions. However, since safety is a high priority with this application, errors tend to be rare. An extremely low amount of false positives can occur with this larger system for a larger hash table. A microwave oven is another device with a critical response time and many potential problems that can occur with the micro-controller interface as well as temperature and timer control settings. Despite this, they are designed so that few errors occur. Again, an extremely low number of false positives is expected to occur with a large system and medium sized hash table. Finally a smart energy control device with the ability to turn on/off appliances may experience syncing problems and errors due to the various sensors it contains but in this case a sufficiently high false positive probability is acceptable considering the 1200 parameters (for 400 errors and $\kappa = 3$) are inserted into a table size of size 1201 bits only. This table is larger than the others as more information from many different sensors must be stored.

VII. Conclusion and Future Work

The proposed constant time and memory algorithm for logging errors in wireless wearable embedded devices is a simple solution to storing programming errors which can be later downloaded and analyzed. It enables error data retrieval to be a low maintenance, low cost operation. It outperforms traditional error logging methods when we compare error probabilities weighted by the impact of the associated errors. For most wearable technologies applications we believe that an acceptable false positive rate can be achieved. We plan to further investigate the various trade-offs and, in particular, be able to compute, for any given application, the optimal depth (i.e., κ) that should be used. This algorithm is being deployed in an actual wearable device and experimental results will be reported in future publications.

References

[1] F. Guo, Y. Li, M. S. Kankanhalli, and M. S. Brown, "An evaluation of wearable activity monitoring devices," in *Proceedings of the 1st ACM international workshop on Personal data meets distributed multimedia*. ACM, 2013, pp. 31–34.

[2] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer networks*, vol. 38, no. 4, pp. 393–422, 2002.

[3] T. Instruments, *MSP430FR5962*, 2017, http://www.ti.com/product/msp430fr5962.

[4] G. D. Knott, "Hashing functions," *The Computer Journal*, vol. 18, no. 3, pp. 265–278, 1975.

[5] B. Preneel, "The first 30 years of cryptographic hash functions and the nist sha-3 competition," in *Cryptographers' Track at the RSA Conference*. Springer, 2010, pp. 1–14.

[6] M. Bellare, R. Canetti, and H. Krawczyk, "Keying hash functions for message authentication," in *Annual International Cryptology Conference*. Springer, 1996, pp. 1–15.

[7] H. S. Kwok and W. K. Tang, "A chaos-based cryptographic hash function for message authentication," *International Journal of Bifurcation and Chaos*, vol. 15, no. 12, pp. 4043–4050, 2005.

[8] C. Percival, "Stronger key derivation via sequential memory-hard functions," *Self-published*, pp. 1–16, 2009.

[9] H. Mohamadi, J. Chu, B. P. Vandervalk, and I. Birol, "nthash: recursive nucleotide hashing," *Bioinformatics*, vol. 32, no. 22, pp. 3492–3494, 2016.

[10] C. Quantin, F.-A. Allaert, P. Avillach, M. Fassa, B. Riandey, G. Trouessin, and O. Cohen, "Building application-related patient identifiers: what solution for a european country?" *International journal of telemedicine and applications*, vol. 2008, p. 7, 2008.

[11] E. A. Mohammed, J. C. Slack, C. T. Naugler *et al.*, "Generating unique ids from patient identification data using security models," *Journal of Pathology Informatics*, vol. 7, no. 1, p. 55, 2016.

[12] J. Haitsma, T. Kalker, and J. Oostveen, "Robust audio hashing for content identification," in *International Workshop on Content-Based Multimedia Indexing*, vol. 4. Citeseer, 2001, pp. 117–124.

[13] R. K. Karsh, R. Laskar, and B. B. Richhariya, "Robust image hashing using ring partition-pgnmf and local features," *SpringerPlus*, vol. 5, no. 1, p. 1995, 2016.

[14] L. Pan, Y. Qiang, J. Yuan, and L. Wu, "Rapid retrieval of lung nodule ct images based on hashing and pruning methods," *BioMed Research International*, vol. 2016, 2016.

[15] L. Song, L. Florea, and B. Langmead, "Lighter: fast and memory-efficient sequencing error correction without counting," *Genome biology*, vol. 15, no. 11, p. 509, 2014.

[16] P. Reviriego, S. Pontarelli, J. A. Maestro, and M. Ottavi, "A synergetic use of bloom filters for error detection and correction," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 3, pp. 584–587, 2015.

[17] M. Kharbutli, K. Irwin, Y. Solihin, and J. Lee, "Using prime numbers for cache indexing to eliminate conflict misses," in *Software, IEE Proceedings-*. IEEE, 2004, pp. 288–299.

[18] P. Bose, H. Guo, E. Kranakis, A. Maheshwari, P. Morin, J. Morrison, M. Smid, and Y. Tang, "On the false-positive rate of bloom filters," *Information Processing Letters*, vol. 108, no. 4, pp. 210–213, 2008.

[19] J. Parak and I. Korhonen, "Evaluation of wearable consumer heart rate monitors based on photopletysmography," in *Engineering in Medicine and Biology Society (EMBC), 2014 36th Annual International Conference of the IEEE*. IEEE, 2014, pp. 3670–3673.