An Efficient Lossless Data Compression Algorithm for Fluctuating Environment Variables in WSN

Pawan Acharya, Uma Pun, and Chen Ming

***Abstract*—A major performance bottleneck in sensor networks is energy since it is impractical to replace the batteries in embedded sensor nodes post-deployment so (Wireless Sensor Networks) WSNs are energy constrained. Radio transmission is most energy draining task that a node performs. Data aggregation and data compression are the widely discussed cases to save a single nodes' lifetime by reducing data to be transmitted but as the memory and computational resources are very limited, these tasks must be performed efficaciously. Here we present a modified version of data compression algorithm presented by Francesco Marcelloni in IEEE Communications Letters 2008. The algorithm Marcelloni presented performs best on variables having a trend of reading implying having least fluctuation with previous reading but as the data fluctuates, the trend vanishes and algorithm becomes inefficacious. We add a pseudo-trend to those fluctuating data on referred algorithm and make it more efficacious on such scenarios. Our experimental results are shown in this article explaining how the existing algorithm, that is referred to, cannot perform its best at those conditions and how our modified algorithm outperforms.**

***Index Terms*—Data compression, wireless sensor network, in-network data aggregation.**

I. Introduction

Wireless sensor networking (WSN) makes use of miniaturization made possible by advanced IC design to couple full wireless subsystems to sophisticated sensors, enabling people and companies to measure a myriad of things in the physical world and act on this information through IT monitoring and control systems. WSN based solutions have been designed and implemented in diverse areas, including environment and habitat monitoring, building automation, disaster and waste management, infrastructure monitoring, etc. [1]. In particular, WSNs deployed for remote area monitoring usually comprise a large number of tiny static sensing devices. Over a geographically wide area, sensor nodes are deployed to sense parameters of interest and forward data to a sink. As large numbers of sensor-nodes are densely deployed, neighbor nodes may be very close to each other. Since the transmission power of a wireless radio is proportional to distance squared or even higher order in the presence of obstacles, data is forwarded in multi-hop routing manner and this routing will consume less energy than direct communication. Radio transmission or reception is the most power expensive task that a node does thus these operations has more significant impact on node power consumption and lifetime than other task that a node does. As sensors nodes are constrained in energy supply and bandwidth, there are many challenges to the design and manage of sensor networks. Mainly there are two measures taken for power saving achieved by reducing radio communication: duty cycling "unpublished" [2] and in-network processing [3]. Unlike many high performance data networks, wireless sensor networks do not require high bit rates. 10-100 Kbps of raw network bandwidth is sufficient for many applications. As bit rates increase, transmission times decrease. By increasing the bit rate without increasing the amount of data being transmitted, the radio duty cycle is decreased. Similarly on a standard bit-rate if the data is compressed or aggregated, the radio duty cycle can be decreased.

To utilize energy resources in a greedy manner, many researches [4]-[7] have been focused on aggregating sensed data also known as in-network data processing, using different mathematical functions: such as MAX, MIN, SUM, AVG, etc. Therefore, only useful aggregation results are sent instead of sending many raw sensed data. Data aggregation is an essential paradigm for prolonging the lifetime of WSN. Indeed, data aggregation reduces the number of broadcasts and hence collisions and energy consumption. However, data aggregation is potentially vulnerable to attackers who may inject bogus information or forge aggregated values without being detected. At the aggregation module, many security services can be provided. But the in-network data compression requires certain compression codes to be executed in node which drains energy. The compression algorithms are indeed helpful only if the energy consumed on executing compression algorithm is less than the targeted energy saving from data transmission. Since transmission of a bit needs energy comparable to the execution of a thousand instructions, just saving only a bit by compressing original data corresponds to reduce power consumption [8]. In case of multi-hop routing, on each hop, transmission energy is saved due to reduced radio cycle, even if the sensing node has expended its energy on data compression.

Our research focuses on in-sensor-node data aggregation before transmission. We use the concept of data compression, which would compress the volume of data to be transmitted. If considered the same bit rate as of un-compressed data, the transmission time decreases for the compressed data hence making the model more energy efficient. Our research is based on compression algorithm

Manuscript received August 13, 2013; revised October 20, 2013.

The authors are with Shanghai Ocean University, Shanghai, China (e-mail: linkpawan@gmail.com, pun_uma@yahoo.com, mchen@shou.edu.cn).

proposed by F. Marcelloni [8], where author presents data aggregation algorithm by differentiation of current ADC output with previous one. This algorithm performs very efficiently on the environmental variables with a trend. For example, the temperature or humidity variables have a pattern or a curve of their value for a day and consecutive samples taken by a sensor node on short interval time do not fluctuate much. But for variables like sound the consecutive reading of a sensor may fluctuate. We add a pseudo-trend which we call pseudo-mean, to those fluctuating data on referred algorithm and make it more efficacious on such scenarios.

II. Related Work

F. Marcelloni on [8] proposes a simple compression algorithm particularly suited to the reduced memory and computational resources of a WSN node. The compression scheme exploits the high correlation that typically exists between consecutive samples collected by a sensor onboard a node. In a sensor node, each measure m_i from a sensor is converted by an ADC to a binary representation r_i on R bits (R being resolution of the ADC).

Marcelloni's compression algorithm computes the difference $d_i = r_i – r_{i–1}$, for each new reading m_i, which is input to an entropy encoder (in order to compute d_0 it assume that $r_{–1}$ is equal to the central value among the 2R possible discrete values). But our algorithm, computes the d_i for current reading with a pseudo-mean, which is explained on following section.

Remaining work is almost identical for both, ours and Marcelloni's algorithm; the entropy encoder then performs compression by encoding differences d_i. Each d_i is represented as a bit sequence bs_i composed of two parts $s_i|a_i$, where s_i codifies the number n_i of bits needed to represent d_i and a_i is the representation of d_i. Code s_i is a variable-length symbol generated from n_i by using Huffman coding. Algorithm uses principles of entropy compression [9] and the algorithm presented is able to compute a compressed version of each value acquired from a sensor, using a very small dictionary whose size is determined by the resolution of the analog-to-digital converter (ADC). Compression algorithm is lossless as it follows a scheme similar to the one used by the baseline JPEG algorithm for compressing the DC coefficients of a digital image [10], as, such coefficients are characterized by a high correlation, very similar to that characterizing data collected by WSNs.

The compression algorithm performs splendidly with less fluctuating environment variables such as temperature and relative humidity if the samples are taken every 2 minutes, but for the highly fluctuating variables like sound, the algorithm does not shows same compression performance which will be elaborated on the next section.

III. Modified Compression Algorithm

We have modified the algorithm on the aspect of differentiation. Marcelloni's algorithm differentiates ADC output (r_i) of the current measurement (m_i) with previous reading (r_{i-1}) but we differentiate with the pseudo-mean of previous state (pm_{i-1}) making the algorithm compatible with fluctuating data.

We studied that differentiating between current state and previous state on the fluctuating data yielded larger differences. The Marcelloni's algorithm gives higher output as the n_i becomes smaller implying that as little magnitude in difference becomes smaller compression is higher. In order to minimize those differences we wanted to differentiate the current data with a value closer to the current reading. As we noticed that there can be established pseudo-trend on fluctuating data while evaluating large numbers of reading of randomly generated sound data. We tried to evaluate the mean of the all data but as the node is memory limited device and we wanted to make the new algorithm as good as that of Marcelloni, we came with the concept of pseudo-mean. As mean would imply the average of data and it is explained that we cannot evaluate mean with all the readings, we decided to generate a pseudo-mean of our current reading with previous mean or pseudo-mean. Pseudo-mean (pm_i) is the average of the previous mean with current reading (r_i). We add a pseudo-trend to data to make it more efficacious on fluctuating scenarios. In our case:

$$d_i = r_i - pm_{i-1}, \tag{1}$$

And

$$pm_i = \lfloor((pm_{i-1} + r_i)/2)\rfloor. \tag{2}$$

The inconsistency of the formula for the first reading is surpassed as following to make the pseudo-mean variable more efficacious:

$$d_1 = r_1 \tag{3}$$

$$pm_1 = r_1 \tag{4}$$

As it can be seen from the (3) and (4) equation, at the first reading no compression and encoding is performed.

TABLE I: Encryption and Decryption

i	Encoder Side			Tx-Rx	Decoder Side		
	Reading	*pm_i*	*d_i*		*d_i*	Reading	pm_i
1	10	10	10	10	10	10	10
2	5	7	-5	-5	-5	5	7
3	4	5	-3	-3	-3	4	5
4	11	8	6	6	6	11	8
5	12	10	4	4	4	12	10

TABLE II: The Huffman Variable Length Codes Used in the Experiments

n_i	s_i	d_i
0	00	0
1	010	-1,+1
2	011	-3,-2,+2,+3
3	100	-7,. . . ,-4,+4,. . . ,+7
4	101	-15,. . . ,-8,+8,. . . ,+15
5	110	-31,. . . ,-16,+16,. . . ,+31
6	1110	-63,. . . ,-32,+32,. . . ,+63
7	11110	-127,. . . ,-64,+64,. . . ,+127
8	111110	-255,. . . ,-128,+128,. . . ,+255
9	1111110	-511,. . . ,-256,+256,. . . ,+511
10	11111110	-1023,. . . ,-512,+512,. . . ,+1023
11	111111110	-2047,. . . ,-1024,+1024,. . . ,+2047
12	1111111110	-4095,. . . ,-2048,+2048,. . . ,+4095
13	11111111110	-8191,. . . ,-4096,+4096,. . . ,+8191
14	111111111110	-16383,. . . ,-8192,+8192,. . . ,+16383

A. Decoding Successfully

The pm_i is a parameter to which next reading will be compared. If all values of pm_i can be generated on decoding side, real values can be regenerated in lossless fashion. Let's say current reading is 273 and the pm_{i-1} is 250 then from our algorithm the pm_i would be 261.5 and after flooring that would be 261 and the transmitted value would be 23. In the decoder side we have the pm_{i-1} already calculated i.e. 250 and from transmitted difference 23, current reading 273 by adding 250 and 23 can be calculated. If we have pm_{i-1} and regenerated current reading r_i the pm_i which will be used to decode next reading can be calculated by using same formula $\lfloor((250+273)/2)\rfloor$ which would be 261. As the algorithm includes the precise extracting calculation on the receiver side, the r_i can be generated with pm_{i-1} and d_i (which is transmitted) and new pm_i can also be seamlessly generated for next r_i calculation making algorithm as lossless as of Marcelloni's algorithm. Table I show the successive regeneration of reading on decoder side which is encoded by encoding side.

```
Encode (d_i, Table)
  IF d_i = 0 THEN
        SET n_i TO 0
  ELSE
        SET n_i TO (⌈log_2(|d_i|)⌉+1)          //compute category
  ENDIF
  SET s_i TO Table[n_i]                         //extract s_i from Table
  IF n_i = 0 THEN                               //build bsi
        SET bs_i TO s_i                         //a_i is not needed
  ELSE
        IF d_i > 0 THEN                         //build a_i
              SET a_i TO (d_i)|n_i
        ELSE
              SET a_i TO (d_i – 1)|n_i
        ENDIF
        SET bs_i TO << s_i, a_i >>              // build bsi
  ENDIF
  RETURN bsi
```

Fig. 1. Pseudo-code.

As the d_i is obtained, each d_i is represented as a bit sequence bs_i composed of two parts $s_i|a_i$. Each s_i and a_i is computed as per Marcelloni's coding using Huffman coding. The Huffman coding as on Table II maps an alphabet to a representation for that alphabet, composed of sequences of bits of variable sizes, so that symbols that occur frequently have a smaller representation than those that occur rarely.

After the d_i is obtained, the algorithm is basically same as that of Marcelloni's. The a_i part of the bit sequence bs_i is a variable-length integer code generated as follows:

1) If $d_i = 0$, s_i is coded as 00 and a_i is not represented.
2) Else
   if $d_i > 0$,
   a_i corresponds to the n_i low-order bits of the two's complement representation of d_i;
   Else
   a_i corresponds to the n_i low-order bits of the two's complement representation of $(d_i - 1)$;

The procedure used to generate a_i guarantees that all possible values have different codes. Once bs_i is generated, it is appended to the bit-stream which forms the compressed version of the sequence of measures mi. Fig. 1 summarizes the algorithm used to encode d_i. Here, $<< s_i, a_i >>$ denotes the concatenation of s_i and a_i, while $v|n_i$ denotes the n_i, low-order bits of v. Pseudo-code is presented on Fig. 1.

F. Marcelloni observed that the execution of the algorithm requires from 59 to 618 instructions with average of 355 instructions on average and the fluctuation is due to binary logarithmic operation, which is not generally available in the instruction set of microprocessors onboard sensor nodes and is, therefore, implemented by an algorithm that requires the execution of a number of instructions dependent on the value of the argument (the larger the value, the larger the number of instructions). Author tested algorithm on environment variables temperature and relative humidity and found that for environment variables the numbers of instructions to be 335. Algorithm is simple with fewer lines of code and easily implementable [8].

IV. Experimental Results

If CR is compression ratio expressed in percentage, CS being compressed size and OS being original size as on (5).

$$CR = 100.\left(1-\frac{cs}{os}\right) \tag{5}$$

F. Marcelloni's algorithm, with simulated results showed CR 66.99% and 67.33% for temperature and relative humidity respectively with R (resolution of ADC) being 16 bits, and samples acquired by a node every 2 minutes during 48 hours (in total, 1440 samples). This algorithm outperforms two well-known compression algorithms, namely gzip [11] and bzip2 [12], to the same datasets [8]. Supposing that each packet can contain at most 25 bytes of payload, the uncompressed versions of temperature and relative humidity data require 116 packets each to be forwarded to the data collector, while the compressed versions require only 38 packets each, thus allowing a considerable power saving.

As the Marcelloni's algorithm was experimented on less fluctuating environmental variables which means the consecutive samples magnitude didn't vary significantly. The range of highest to lowest magnitude of the reading was also low.

Simulation performed on the condition where consecutive samples' magnitude didn't vary significantly (non-fluctuating) and with the varied range (highest to lowest magnitude) of the environment variable is presented on Table III.

TABLE III: Compression Ratio for Non-Fluctuating Variable with Varying Range

Temperature (°C)	Compressed Size (bits)		Compression Ratio (%)	
Range (Max-Min)	Marcelloni's Algorithm	Our Algorithm	Marcelloni's Algorithm	Our Algorithm
5	6882	8080	70.13	64.93
10	7649	8972	66.8	61.06
20	9073	10414	60.62	54.8
40	10460	11838	54.6	48.62
50	10944	12442	52.5	46

Similarly, simulation performed on the condition where consecutive samples' magnitude vary (fluctuating) and with the varied range (highest to lowest magnitude) of the variable is presented on Table IV.

TABLE IV: COMPRESSION RATIO FOR FLUCTUATING VARIABLE WITH VARYING RANGE

Sound (dB)	Compressed Size (bits)		Compression Ratio (%)	
Range (Max-Min)	Marcelloni's Algorithm	Our Algorithm	Marcelloni's Algorithm	Our Algorithm
10	13686	11082	40.6	51.9
25	14193	12748	38.4	44.67
50	17050	15460	26	32.9
75	20897	18478	9.3	19.8
100	22372	20137	2.9	12.6

All the experiments done are by considering each nodes samples every minute for 24 hours resulting total of 1440 samples for ADC resolutions of 16 bits. The non fluctuating variable is temperature variable where simulation was done feeding from the graph obtained from real results. The range of variable was made low by attenuating magnitude of the graph equation and vice versa. The fluctuating variable is sound. The fluctuation was generated form generating random number within the considered audible range [12]. The range was considered by dividing the audible range to five parts. Very-High range here means the range from least audible sound decibel value to the largest [12].

From Table III, it can be clearly seen that Marcelloni's algorithm outperforms ours' as the environmental variables do not fluctuate. But from Table IV, our algorithm outperforms Marcellonis' as the variable fluctuates. It can be seen that in wider range both the algorithm's performance degrades. When considered lower range the difference between two consecutive readings is very low in the case of Table III. The difference between two samples is very low and the Huffman's table and digitization of difference result fewer bits of representation but as the range increases, the difference between two consecutive readings also increases and the code becomes larger. Hence the compression ratio decreases as the range increases.

The comparison on non-fluctuating variables with Bzip2 [11] and Gzip [11] is done on [8]. On similar simulation with compression output of Marcelloni's algorithm 66.99% (in our experiment 70.13%) ours had 64.93% and 42.19% and 43.05% of Gzip [11] respectively. For the fluctuating variables on similar conditions of simulation, compression outputs were 51.9%, 40.6%, 36.52% and 36.84% respectively for ours, Marcelloni's algorithm, Gzip [11]. In the case of fluctuating variable, as the readings do not have a trend or pattern, the consecutive variables may have a big difference between each other. Due to which the Marcelloni's algorithm does not performs well, but our algorithm stores the pseudo-mean value of the readings and stores a trend and makes the algorithm perform better. As the range increases, even with the pseudo-mean, the difference becomes larger and the algorithm cannot maintain its performance but still it can be seen that for both the fluctuating and non-fluctuating variables our algorithm is more consistent providing compression ratio from 64.9% max to 12.6% min. with 52.3% of compression range where as our experiment showed the Marcelloni's algorithm having compression range 67.23% making our algorithm more consistent.

The algorithm is as lossless as of Marcelloni's algorithm and performs similarly. The pseudo-mean value is similar to consecutive values hence demands similar memory. Implementation of either algorithm is almost identical. In case of less deviating values, we have seen from our experiments that in Marcelloni's algorithm, the difference nearly becomes zero or one but the pseudo –mean deviated a little more from consecutive reading, hence the Marcelloni's algorithm performs better on non fluctuating variables than ours. The reason our value deviation is that it stores trend. On less fluctuating variable the data being on trend already, the function of the trend differ from the trend that our pseudo mean stores hence giving more deviation. But on data with no trend, the trend generation yields a meaningful idea on restricting the consecutive deviation with replacing with the trend deviation which is concurred from Table III and Table IV**.**

V. CONCLUSION

There are many techniques being explored on the field of data compression and the increasing use of wireless sensor networks on the various fields has also increased the variables to be sensed. The variables that the node senses have different properties. Currently implemented fields of these kinds of network sense environment variables that do not fluctuate more over consecutive samples but the new requirement have emerged due to expanding implementation. The fluctuating data also are to be sensed in the current application trend of wireless sensor networks.

Here we present a modified version of data compression algorithm presented by Francesco Marcelloni in IEEE Communications Letters 2008. The algorithm Marcelloni presented performs best on variables having a trend of reading, implying having least fluctuation with previous reading. But as the data fluctuates the trend vanishes and algorithm becomes inefficacious. We have presented a simple solution to the performance problem of Marcelloni's algorithm on fluctuating data. Marcelloni's compression algorithm computes the difference $d_i = r_i - r_{i-1}$, for each new reading m_i, which is input to an entropy encoder (in order to compute d_0 it assume that r_{-1} is equal to the central value among the 2R possible discrete values). We studied that differentiating between current state and previous state on the fluctuating data yielded larger differences.

The Marcelloni's algorithm gives higher output as the n_i becomes smaller implying that as little magnitude in difference becomes smaller compression is higher. In order to minimize those differences we wanted to differentiate the current data with a value closer to the current reading. As we noticed that there can be established pseudo-trend on fluctuating data while evaluating large numbers of reading of randomly generated sound data. We tried to evaluate the mean of the all data but as the node is memory limited device and we wanted to make the new algorithm as good as that of Marcelloni, we came with the concept of pseudo-mean. As mean would imply the average of data and it is explained that we cannot evaluate mean with all the readings, we decided to generate a pseudo-mean of our current reading with previous mean or pseudo-mean. Pseudo-mean (pm_i) is the average of the previous mean with current reading (r_i). We add a pseudo-trend to those fluctuating data on referred algorithm and make it more efficacious on such scenarios. The pseudo-mean is a single variable hence required calculation is also very simple making our algorithm is almost identical on implementation with Marcellonis' algorithm. We changed the differentiation mode from previous reading to pseudo-mean of all readings. Our experimental results are shown in this article explaining how the existing Marcelloni's algorithm cannot perform its best at those conditions and how our modified algorithm outperforms. Marcelloni's algorithm performs best on the non-fluctuating variables but on fluctuating variable our algorithm outperforms by difference of 8.93% on average. When a data is being compressed by 2.9%, achieving a compression ratio up to 12.6% is more than three times. Our sensor-node data compression algorithms are very important on WSN because implementation not only saves the transmission power of data generating sensor node but also saves the energy of all nodes on the path of data routing on both reception and hop-transmission.

Acknowledgement

We would like to express our sincere thanks to Dr. Chen Ming, Dr. Feng Guofu and Dr. Chi Tao for their valuable help, support and reviews. This work was funded by National High-tech Research and Development Plan (863 program) of People's Republic of China, no, 2012AA101905.

References

[1] C. S. Raghavendra, K. M. Sivalingam, and T. F. Znati, *Wireless Sensor Networks*, Boston: Kluwer Academic Publishers, 2004.

[2] G. Anastasi, M. Conti, M. D. Francesco, and A. Passarella, "How to prolong the lifetime of wireless sensor networks," Mario Di Francesco, Department of Information Engineering, 2006.

[3] E. Fasolo, M. Rossi, J. Widmer, and M. Zorzi, "In-network aggregation techniques for wireless sensor networks: a survey," *IEEE Trans. Wireless Commun.*, vol. 14, pp. 70-87, April 2007.

[4] H. Chan, A. Perrig, and D. Song, "Secure hierarchical in-network aggregation in sensor networks," in *Proc. the 13th ACM Conference on Computer and Communications Security*, ACM Press, New York, NY, USA, 2006 , pp. 278–287.

[5] K. Wu, D. Dreefa, B. Sunb, and Y. Xiao, "Secure data aggregation without persistent cryptographic operations in wireless sensor networks," *Ad Hoc Networks*, vol. 5, pp. 100–111, 2006.

[6] L. Hu and D. Evans, "Secure aggregation for wireless networks," in *Proc. Workshop on Security and Assurance in Ad Hoc Networks*, pp. 384-391, 2003.

[7] S. Peter, K. Piotrowski, and P. Langendoerfer, "On concealed data aggregation for wireless sensor networks," in *Proc. Consumer Communications and Networking Conference*, vol. 5, 2007.

[8] F. Marcelloni, "A Simple Algorithm for Data Compression in Wireless Sensor Networks," *IEEE Communications Letters*, vol. 12, June 2008.

[9] K. C. Barr and K. Asanovi ́c, "Energy-aware lossless data compression," *ACM Trans. Comput. Syst.*, vol. 24, pp. 250–291, August, 2006.

[10] W. B. Pennebaker and J. L. Mitchell, *JPEG Still Image Data Compression Standard*, 1992.

[11] Gzip. (2013). [Online]. Available: http://www.gzip.org/.

[12] Wikipedia. (2013). [Online]. Available: http://en.wikipedia.org/wiki/File:Audio23.jpg.

Pawan Acharya was born in Dang, Nepal, He received his bachelor's degree in engineering from Acme Engineering College, affiliated to Purbanchal University, Nepal in electronics and communication in 2010 and He is studying master's degree in applied computer technology from Shanghai Ocean University, Shanghai, China, focusing on wireless sensor network, data mining, neural network etc.

He worked as an instructor of science, mathematics and programming in Nepal at a secondary school for three years later he joined a software Development Company in Nepal as software engineer for one and half years. He is currently volunteering as software engineer at an International Company in Shanghai. His current research work is wireless sensor network, programming and data-mining.

Mr. Acharya received scholarship-A from Shanghai government for further study and researches. He is registered Engineer of Nepal's Engineering Council and also has earned kudos competitions like National Robotic Competitions in Nepal for first runner-up.

Uma Pun was born in Hong-Kong, China, She received her bachelor's degree in information technology in Nepal, She received her master's in economics from Trivuwan University, Nepal and is studying master's degree in applied computer technology from Shanghai Ocean University, Shanghai, China, focusing on wireless sensor network, data mining, neural network etc.

She worked as an instructor of computer science in Nepal at a institute and later worked in Australia for two years. Her current research work is wireless sensor network, programming and data-mining.

Mrs. Pun received scholarship-A from Shanghai government for further study and researches.

Chen Ming was born in Hunan, China, He received his received the research and cybernetics degree from Shanghai University in 1995, China, and received the PhD in computer software and its theory, in 2001. His research interests are data mining, wireless sensor network, and the agriculture information system.

In 2001, he has been an assistant professor at Department of computer science, Shanghai Fishery University. From 2005 to 2008, he worked as professor in the college of information, Shanghai Fishery University. In 2008, because of the university renamed as Shanghai Ocean University, he worked as Vice dean of information college, Shanghai Ocean University.

Mr. Chen is the editor of the Journal of Shanghai Ocean University, and the member of AIPA (IFIP, TC5 special interest group on advanced information processing for agriculture). At the same time, Mr. Chen is the vice director of Shanghai digital agriculture engineering and research center, and the vice director of Shanghai internet of thing for agriculture research center.