International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-10 Issue-1, October 2020

Priority Arbiter for TinyOS: The need of renown OS for WSN and IoT

Rajashree V Biradar, Anita Patil

Abstract: ***In the present technical era, we are extremely dependent on technological applications such as internet, multimedia, social media, home automation, industrial automation, medical instrumentation, web technology so on and so forth. Moreover, as a backbone such applications are supporting the research related to science and technology in turn. There are certain technologies for example Wireless Sensor Network, IOTs, Artificial Intelligence, and Cloud Computing etc., working behind these applications as unseen hands. Nowadays in all these facilities, there is much more advancement and high demand for real-time applications to serve interactive services. Such necessities enforce the technologies to upgrade themselves to their next level. As such, in WSN, the existing Operating Systems also should upgrade in-terms of different concerns such as memory management, scheduling techniques, power supply scarcity issues and overall efficient utilization of available limited resources. In this regard, here is an attempt to improvise TinyOS, which is the popular OS for WSN and IOT. The survey on WSN applications reveals, what sort of improvisations are necessary to fulfill the requirements of varieties of applications. In that direction, for more efficient scheduling of tasks based on the situation, new technique is required. Being the best OS for low power devices of WSN and IOT, TinyOS hinders to support many application those need different type of scheduling than FCFS, which is the only scheduling technique for TinyOS. Hence, Integration of new Priority arbiter as a first step of main scheduler improvisation is the essence of this paper.***

Key words: Operating Systems for WSN, Applications of WSN, Scheduling techniques, Scheduling in TinyOS, Operating systems for IOT.

I. INTRODUCTION

Over the last five years, several operating systems (OSs) for wireless sensor network (WSN) have been developed to aid the developers [9, 10, 11, 12]. This novel work is led by survey of the current state of operating systems for WSN and comparative analysis of some popular OSs [3, 6, 7], in terms of main features namely architecture, programming model, scheduling, memory management, communication protocols, resource sharing and support for real-time applications. Consequently, the comparison is done for the additional features like, the file system, database support, security support, simulation support, language support, supported platforms and documentation support.

Revised Manuscript Received on November 10, 2020.
*Correspondence Authors
Rajashree V Biradar*, Dept of CSE, Ballari Institute of Technology and Management, Ballari-583104, Karnataka, India, rajashreebiradar@yahoo.com
Anita Patil, Department of CSE, Ballari Institute of Technology and Management, Ballari-583104, Karnataka, India, anitha.bijapur@gmail.com

Finally, it is found that, TinyOS is the open source, popular, widely used, highly documented and familiar operating system [7], whose limitations can be nullified, and such improved version of TinyOS can be used as all in one general purpose OS. As shown in below figure, the architecture [8] of TinyOS has the components suitable for sensor node subsystems, namely- sensing subsystem, communicating subsystem, processing subsystem and power management subsystem.

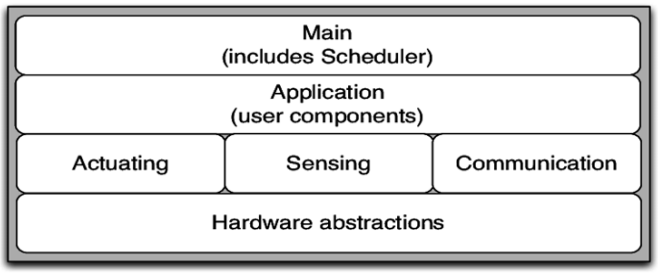

Fig 1.1.TinyOS Architecture

Like all other OSs, TinyOS also has hardware abstraction on one side and user on another side. The user side Main includes the scheduler as the main component. In between the actuator, sensor and communicating subsystems are there to support sensor node hardware. The rigid and constraint-full architecture of WSN challenges the design of OS critically by providing tiny structured node with limited amount of all the resources [2, 7, 9, 10, 12] like memory, flash memory, power supply etc. This challenge has made the OS designer to take care of few necessary restricted resources, keeping all others aside. In the sense, to take care of limited memory (which is just in-terms of kbs), the footprint of the OS itself should be as less as possible. Therefore, definitely the code written for processor management, memory management and storage management etc. also should be as less as possible. As a result of all these, limitations detected in TinyOS are, having only FCFS (First Come First Serve) scheduler and not supporting real time applications. This work justifies that if an OS becomes flexible in scheduling selection, then such OS will be one of the more efficient OSs. Moreover, WSN is the fundamental technology base for some advanced ones especially IOTs, thus it shares the set of OSs with IOT [1,2]. That is why, if we brows for "Operating systems for IOT", in the list we find TinyOS as the main one. At the same time author in [5] says, OS developers/researchers are rare due to the fact that it is a highly specialized field with a slow curve and tolerance for change. This is an alarm for the urgency of novel research in this area [5]. Thus, there is the scope for research in TinyOS and this empirical work proposal is for tasks scheduling in TinyOS. Here, the first section of paper introduces the work

by explaining the scope of the work and giving proper work flow in-terms of paper structure itself. The motivation behind this new priority arbiter idea is described in section 2. Section 3 details about implementation that includes the hardware resources employed and other software, including the OS installed in this empirical study. The results are discussed in section 4. Finally, concluding highpoints are in Section 5.

II. MOTIVATION FOR PRIORITY ARBITER

As there are numerous OSs exists [7], each with one or the other specific purpose, OS selection becomes difficult for the application developer. In that case application developer should become expert in OS first, rather than concentrating on application development side. At the same time, author in [13] says, tinyOS is preliminary designed for Wireless Sensor Networks (WSN) and distinctively the popular among the research community for many years. However, nowadays researchers are not using this much, due to lack of active development. Hence, there is the need of research in TinyOS to improvise this as a general purpose OS for WSN. Appropriate scheduling keeps the processor always busy and utilize the memory efficiently, so that it can achieve more work done in less amount of time i.e. more throughput, less turnaround time and less response time. Above all this energy efficiency is surplus advantage achieved. This reduces the impact of the scarcity of power supply which is the major problem in sensor node, due to which nodes die sooner and network topology changes frequently. Hence, for any OS, if scheduling is efficient and dynamically adaptive, then overall OS will be efficient from both user and system viewpoints [16]. To attain this, both hardware and software technologies must work hand in hand. As per surveys, this resource constrained tiny sized node limits the size of OS, in-turn restricts the services. However, nowadays hardware technology development is faster, for example, mini SIM cards (mobile phone sim) are replaced by micro SIM and later by nano SIM, as well as the size of storage devices also has extensively reduced i.e. earlier in what sized device 1Gb storage was available, in that sized or even the smaller sized device (pen drive) 16 Gb storage is available nowadays. Consequently, software (here OS) also has to grow with the contender i.e. hardware developments as counterpart. Thus, here is a plan to keep OS ready for under up-gradation hardware that has to relieve the resource constrained architecture of node, by providing more capabilities in same or reduced sized node. This innovative idea motivates to improvise an OS, and then work proceeds in specific direction as follows. TinyOS-2.1.2, being one of the leading and traditional OS for low power devices in WSN and IOT, even then it is lagging behind, as it has only FCFS scheduler. Even it seems to be a single and simple limitation, it hinders the OS by not allowing to support diversified applications with varieties if requirements. There are some proposals for different scheduling techniques for TinyOS [16]. Here is the plan to incorporate an additional scheduling algorithm. In this experiment, instead of directly implementing the plan in scheduler, implementation is done on arbiter. Arbiter is the intermediary for tasks to get processed by processor in some particular fashion. Multiple tasks need an efficient arbiter for efficient utilization of processor and other resources of sensor node. As such, TinyOS already has the arbiters working in FCFS and RR (Round Robin) fashion to process the tasks, where the idea of new priority arbiter can be implemented. The existing FCFS arbiter forwards the tasks to the processor as per their arrival. The RR arbiter gives equal amount of opportunity to each task to get processed by the processor in each round. At any movement of time if any task needs highest preference of processing, for such one TinyOS don't have arbiter. For example, in under water study T-sunami detection could have been kept at highest level of sensing and information processing than any other tasks processing to reduce the impact of that disaster to the maximum extent. For this, there is the need of Priority arbiter and hence the scheduler.

III. IMPLEMENTATION DETAILS

The behavior of the new arbiter is well defined for different prioritized tasks, at the same time the code developed for Priority arbiter works satisfactorily.

This newly designed and developed priority arbiter implementation is carried out as follows.

- Here in this work TinyOS is installed in Ubuntu 18.04. Whereas it can be installed in both Windows and Linux systems. TinyOS has footprint of less than 400bytes i.e. core or base code of OS. This has to fit in node memory, wherein complied code of application and other required software also should fit.
- For this empirical work Telosb platform is taken, which is one of the suitable sensor boards for tinyOS. Like any sensor node the size of Telosb is also tiny i.e 2.55*1.24*0.24 inches, within which 10kb RAM, 48kb Flash Memory, 2*AA Batteries, 8MHz MSP430 microcontroller, 3 sensors and 3 LEDs etc. objects are soldered, which all together weighs 23 grams(excluding batteries weight)[18].
- There are several simulators for WSN for example Avrora, WSim, TOSSIM, MSPSIM, JTossim etc. Telosb has MSP430 microcontroller. Hence, Simulator used here is MSPSIM [19]. Actually in tinyOS, TOSSIM simulator is inbuilt. But, to use that additional interfaces in C++ or Python are compulsory, which is an extra overhead again. The language nesC [20] is used to code priority arbiter and then it has to be integrated with other arbiters in the "arbiter" directory of tinyos.

No doubt, that the total time taken for the processing all the processes is not less than the sum of their individual processing times, it is the fact which can't be changed. Here the concern is not only about processor being efficiently utilized but also about tasks completion more efficiently with the least possible response time, waiting time and turnaround time i.e. the overall good throughput for a sensor node. The sensor node with FCFS arbiter processes the events as per their occurrence or arrival. The events itself are the load in experimentation. As already said above in scope of the work, change in order of tasks completion itself is reflects the node throughput.

For n number of tasks in queue,

$$TTn \geq \sum_{i=1}^{n} Tti \quad \text{for} \quad i=1 \text{ to } n \qquad (1)$$

where,

TTn→ Total time taken for processing n tasks.
t → task Tti → processing time of i^{th} task
Ttn → Processing time of n^{th} task.

Based on the requirement, the order of processing the tasks can be changed with the help of different scheduling algorithms. The readily available scheduler in TinyOS executes the tasks in FCFS order, with the below described impact on tasks completion.

For any i^{th} task the response time Rt, waiting time Wt and turnaround time Tt, depend on its own arrival time, own processing time and processing time of other tasks arrived before it's arrival.

In case, if lengthy tasks are before the shorter tasks in the queue, then the efficiency of those shorter tasks get hindered by much higher Rt, Wt and Tt of the lengthy tasks before themselves.

For the task t_i and next task $t_{(i+1)}$, if the arrival time and processing time are like,

A_t →arrival time, P_t →processing time

$$At(i) < At(i+1) \quad \text{and} \quad Pt(i) >> Pt(i+1)$$

Then for the task $t_{(i+1)}$ the Rt, Wt and Tt will be definitely much higher resulting in poor performance. Such a cumulative effect of all tasks decreases the overall system performance.

The problem can be solved by assigning the priority for each task based on the least processing time or any other criteria such as task with least resource requirements first, the shortest task first, sensing task first, communication task first etc, then accordingly arranging them in queue for processing.

Pr → Priority

$$ti = \Pr i \quad \text{for all processes i=1 to n.} \qquad (2)$$

$$\text{if } \Pr i \geq \Pr(i+1), \text{ then } Q[\text{head}] = \text{ti} \qquad (3)$$

$$\text{else } Q[head] = t(i+1) \qquad (4)$$

$$\text{and } Q[head+1] = ti \qquad (5)$$

IV. RESULTS AND DISCUSSIONS

The component based and event driven TinyOS is developed in NesC language [20]. The TinyOS applications also need nesC for coding. NesC language has its own programming structure, where each application must have 3 components wrapped in a folder:

1. Module file: Implements the components.
2. Configuration file: Wires the components.
3. Make file: Compiles and executes the application.

Eclipse editor or terminal with any regular text editor of Ubuntu can be used for application coding and manipulations. As already said, more suitable platform for tinyOS is Telosb, on which applications need to be deployed and executed. The application Blink is similar to a "Hello World" program for any programming language. By this Blink and other example applications, nesC programming components namely tasks, events, commands, interfaces, modules, configurations are analyzed. As a result, further coding in implementation became easier. The blinking LEDs in the Telosb board show how the arbiters perform in different fashion such as FCFS, RR and this new Priority arbiter. This new Priority arbiter can assign the priority to different tasks based on any criterion like, task with least resource requirements first, shortest task first, sensing task first, communication task first etc. Such a surrounding environment based dynamic prioritizing is the basis for various applications. For better experimentation more events are considered, for which the available only 3 LEDs of MSPSIM SkyGui are not sufficient. At the same time, the correct result analysis is not that easy with the blinking LEDs, so in every task some statements like "red blinks" are displayed using printf. The statements displayed at the USART port output window give the order of task execution.

A. Result 1

Individually for the arbiters FCFS, Round Robin and Priority, results are respectively shown in figures 4.1, 4.2 and 4.3 below. Simulation results show execution by printing the statements like "red blinks", "green blinks", "blue blinks" "pink blinks", "yellow blinks" and "orange blinks", as per the order of execution decided by respective arbiter. The code is also visible in editor of the screenshots, calling different Resources in some order like Resource4.request(), Resource3.request(),Resource2.request(), Resource0.request(),Resource1.request(), accordingly their code execution displays the respective color blink statements.

The colors taken in code for Resources from Resource0 to Resource4 are respectively red, green, blue, yellow and pink.

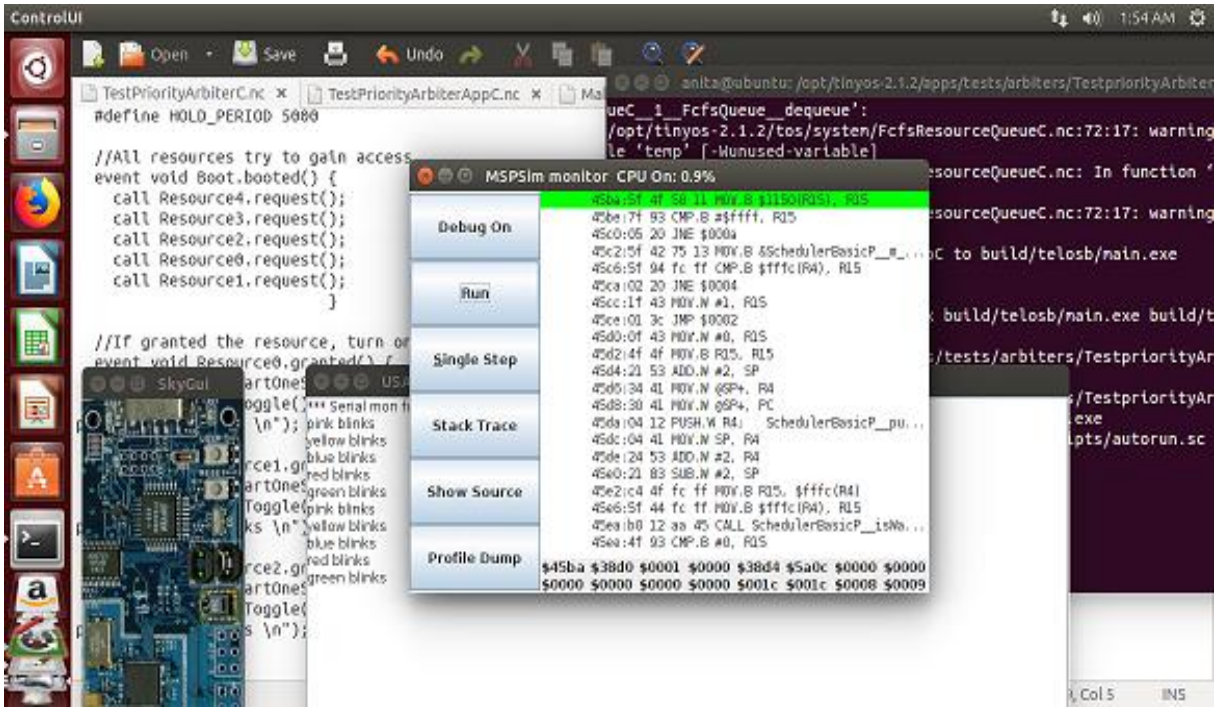

Fig. 4.1- FCFS arbiter result

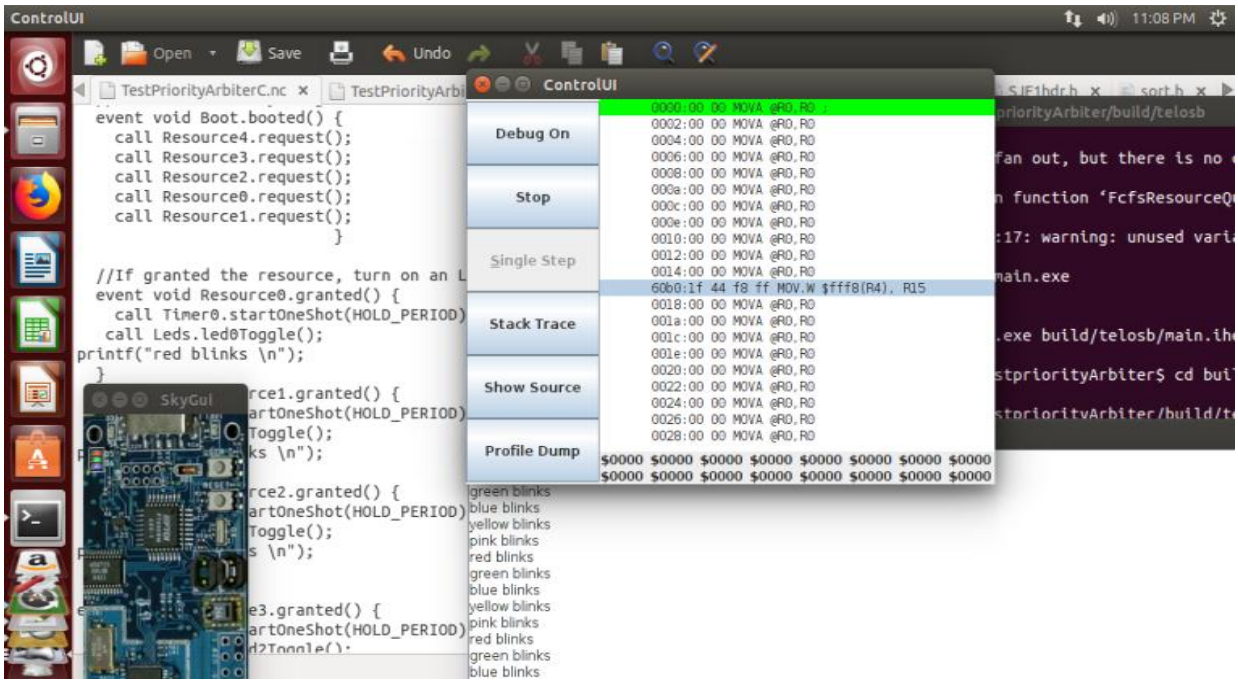

Fig. 4.2- RR arbiter result

Retrieval Number: 100.1/ijeat.A18391010120
DOI: 10.35940/ijeat.A1839.1110120
Journal Website: www.ijeat.org

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication

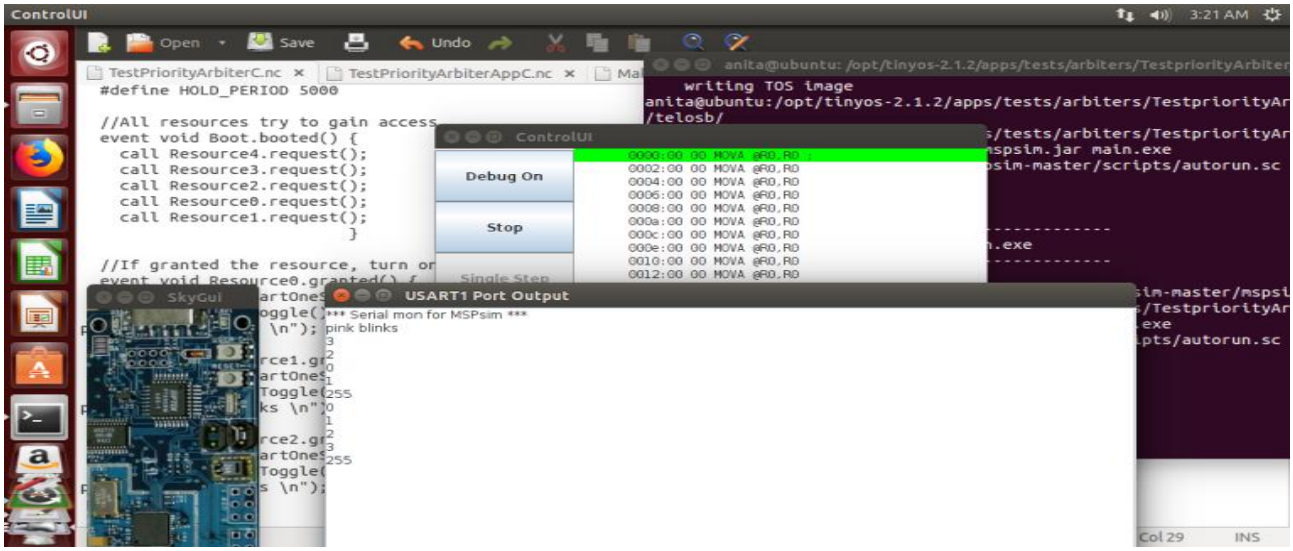

Fig. 4.3- Priority arbiter result

B. Result 2

nesC allows, integration of additional components as well as conditional selection of any component is same as that of C. Thus, multiple arbiters are offered for the application in the same program. Any arbiter can be invoked as per tasks requirement, accordingly results are found satisfactory as shown in Fig-4.4 below.

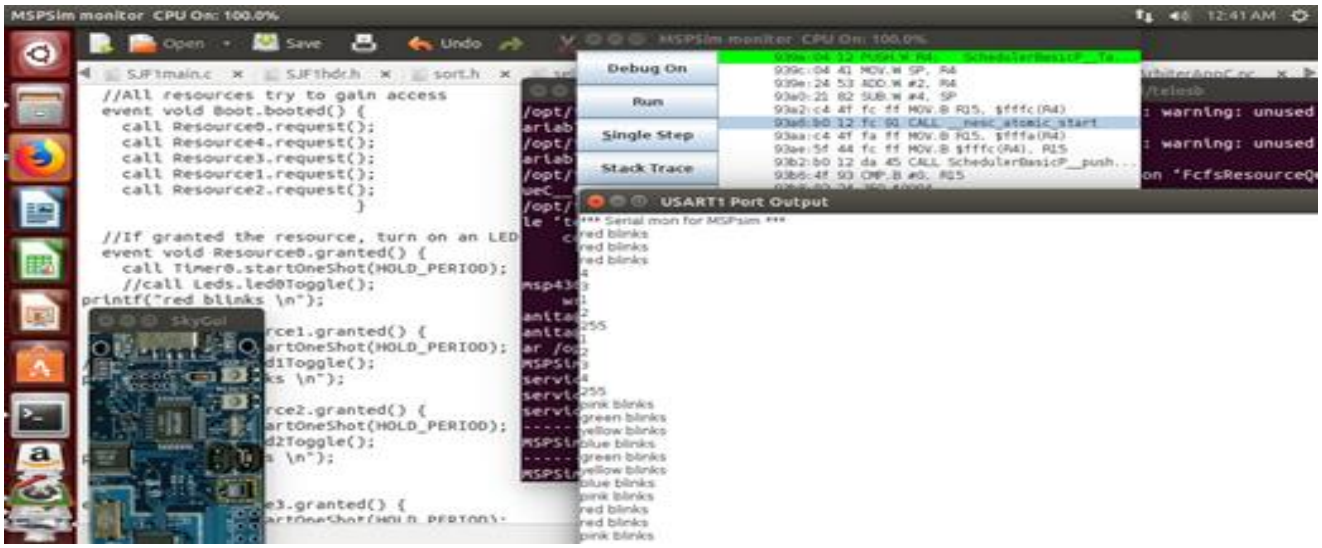

Fig. 4.4-FCFS, RR, Priority arbiters for the same application result

The orders of tasks completion in FCFS and Priority arbiters are as shown in below graph.

Table1. Tasks for FCFS and priority arbiters

Tasks	Order of arrival	Priority assigned(smallest the number highest the priority)
T1	0	4
T2	1	1
T3	2	3
T4	3	2

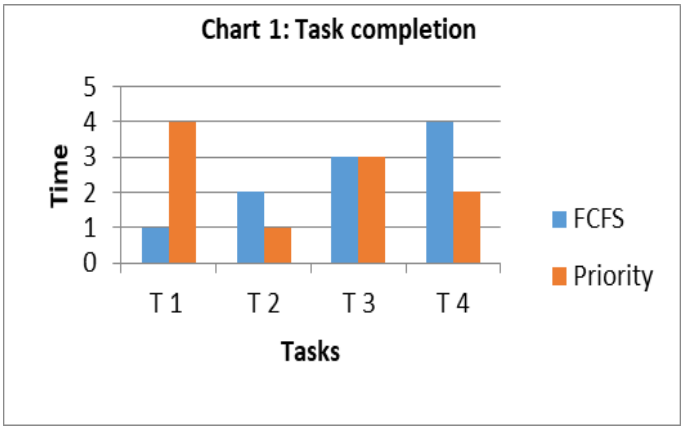

Chart 1: Tasks completion as per arbitration order

The tasks are completed in FCFS arbiter in the order T1, T2, T3 and T4 as per their arrival order, where as in Priority arbiter the tasks completion order is T2, T4, T3 and T1 as per their assigned priorities.

V. CONCLUSION

This work concludes that, when compared with other OSs of WSN, tinyOS-2.1.2 is a best one for low power devices specifically sensor nodes and low power devices of IOT. Even then TinyOS also has limitations like having only FCFS scheduler[16]. To rebuild this OS stronger, here is an attempt to nullify the short comings found while supporting various applications. In this regard, this novel work is an initiative for scheduling related improvisation, by including new arbiter. In today's real world the diversified applications need diversified requirements such as dynamically adaptive scheduler for real-time services, interactive services, different levels of prioritized tasks etc. This work is in that roadmap. This new Priority arbiter can assign the priority to different tasks based on any criteria like, task with the least resource requirements first, the shortest task first, sensing task first, communication task first etc. In results snapshot we can see the sorted tasks based on their assigned priorities or IDs.

REFERENCES

1. Roberto Rodriguez-Zurrunero , Ramiro Utrilla , Alba Rozas and Alvaro Araujo, "Process Management in IoT Operating Systems:Cross-Influence between Processing and Communication Tasks in End-Devices". Sensors 2019, 19, 805; doi:10.3390/s19040805 www.mdpi.com/journal/sensors Published: 16 February 2019
2. Rebin B Khoshnaw¹, Dana Farhad Doghramachi, Mazin S. Al-Hakeem, "A Review on Internet of Things' Operating Systems, Platforms and Applications", Conference Paper • February 2017 DOI:10.23918/iec2017.06
3. Hicham Aberbach, Sabri Abdelouahed, Adil Jeghal, H. Tairi, "A Comparative Study between Operating Systems (Os) for the Internet of Things (IoT)", DOI: 10.14738/tmlai.54.3192 Publication Date: 15th August 2017 URL: http://dx.doi.org/10.14738/tmlai.54.3192
4. Arslan Musaddiq1, Yousaf Bin Zikria1, (Senior Member, Ieee), Oliver Hahm2, Heejung Yu1, Ali Kashif Bashir 3, (Senior Member, Ieee), And Sung Won Kim , " A Survey on Resource Management in IoT Operating Systems", IEEEAccess, publication February 21, 2018, date of current version March 12, 2018. DOI 10.1109/ACCESS.2018.2808324
5. Salahuddin M. ElKazak, Cairo University, Masters in Computer Engineering, "GEN600 Final Technical Report:Research in Internet of Things' Operating Systems (IoT OS's)", Research in IoT OS's, GEN600: Final Technical Report
6. Anita Patil (1), Dr: Rajashree.V.Biradar(2), "Comparative study of Operating Systems for Wireless Sensor Networks",NCRTCSE-12, pages 232-242
7. Muhammad Amjad, Muhammad Sharif, Muhammad Khalil Afzal, and Sung Won Kim, "TinyOS-New Trends, ComparativeViews, and Supported Sensing Applications: A Review", IEEE SENSORS JOURNAL, VOL. 16, NO. 9, MAY 1, 2016
8. Waltenegus Dargie, Christian Poellabauer, Book, "Fundamentals of wireless sensor networks theory and practice",2010 John Wiley & Sons Ltd.
9. Adi Mallikarjuna Reddy V AVU Phani Kumar, D Janakiram, and G Ashok Kumar, "Operating Systems for Wireless Sensor Networks: A Survey Technical Report", May 3, 2007 pages 1-30

A. K. Dwivedi, M. K. Tiwari, O. P. Vyas, "Operating Systems for Tiny Networked Sensors: A Survey", International Journal of Recent Trends in Engineering, Vol. 1, No. 2, May 2009, pages 153-15

10. W. Dong, C. Chen, X. Liu, and J. Bu, "Providing OS support for wireless sensor networks: Challenges and approaches", IEEE Commun.Surveys Tuts., vol. 12, no. 4, pp.519–530, Nov. 2010.
11. Muhammad Omer Farooq and Thomas Kunz,Article, " Operating Systems for Wireless Sensor Networks: A Survey", Sensors 2011, 11, 5900-5930; doi:10.3390/s110605900.
12. Yousaf Bin Zikria , Sung Won Kim , Oliver Hahm , Muhammad Khalil Afzal and Mohammed Y. Aalsalem , "Internet of Things (IoT) Operating Systems Management: Opportunities, Challenges, and Solution", Sensors 2019, 19, 1793; doi:10.3390/s19081793 www.mdpi.com/journal/sensors
13. Farhana Javed, Muhamamd Khalil Afzal, Muhammad Sharif and Byung-Seo Kim, "Internet of Things (IoTs) Operating Systems Support, Networking Technologies, Applications and Challenges: A Comparative Review", 1553-877X (c) 2018 IEEE
14. Roberto Rodriguez-Zurrunero, Ramiro Utrilla, Elena Romero, and Alvaro Araujo, Research Article-"An Adaptive Scheduler for Real-Time Operating Systems to Extend WSN Nodes Lifetime", Hindawi, Wireless Communications and Mobile Computing. Volume 2018, Article ID 4185650, 10 pages. https://doi.org/10.1155/2018/4185650
15. Anita Patil, Rajashree Biradar, "Scheduling Techniques for TinyOS: A Review", DOI: 10.1109/CSITSS.2016.7779420 Publisher: IEEE, Date of Conference: 6-8 Oct. 2016
16. https://en.wikipedia.org/wiki/TinyOS
17. https://en.wikipedia.org/wiki/List_of_wireless_sensor_nodes
18. http://tinyos.stanford.edu/tinyos-wiki/index.php/MSPSim
19. https://en.wikipedia.org/wiki/NesC
20. http://tinyos.stanford.edu/tinyos-wiki/index.php/TEPs
21. https://github.com
22. https://stackoverflow.com
23. http://www.tinyos.net

AUTHORS PROFILE

Dr. Rajashree V Biradar received her BE (E&CE) in the year 1991 from Karnataka University Darwad, MS(Software systems) in the year 1999 from BITS Pilani and Ph.D in the year 2012 from Shivaji University Kolhapur. Madam did Ph.D on routing in Self Organizing Wireless Sensor Networks(WSN). She has published 38 research papers in various international journals. Madam's two students have already awarded with Ph.D and six students are perusing their Ph.D in various fields of WSN including operating system, cross layer design, secured cloud computing etc. She has more than 29 years of experience in teaching. Currently, she is working as Professor in the department of CSE, Ballari Institute of Technology and Management, Ballri, India.

Anita Patil received her Bachelor degree in Computer Science & Engineering from Visvesvaraya Technological University, Belgaum, Karnataka, India in 2006. She received her Master's degree in Computer Network & Engineering from the same Visvesvaraya Technological University, Belgaum, Karnataka, India in 2008. Her area of specialization is wireless sensor networks. She is doing research in the area of Operating Systems for Wireless Sensor Network under the guidance of Dr.Rajashree V Biradar. She has published research papers in various international journals. She has more than 14 years of experience in teaching. Currently, she is working as an assistant professor in the department of CSE in Ballari Institute of Technology and Management, Ballri, India.