2018

A Novel IEEE 802.11 Power Save Mechanism for Energy Harvesting Motivated Networks

Yigitcan Celik
yigitcancelik91@gmail.com

A NOVEL IEEE 802.11 POWER SAVE MECHANISM FOR ENERGY HARVESTING MOTIVATED NETWORKS

A thesis submitted to
the Graduate College of
Marshall University
In partial fulfillment of
the requirements for the degree of
Master of Science
In
Computer Science
by
Yigitcan Celik
Approved by
Dr. Cong Pu, Committee Chairperson
Dr. Wook-Sung Yoo
Dr. Jamil Chaudri

Marshall University
May 2018

APPROVAL OF THESIS

We, the faculty supervising the work of Yigitcan Celik affirm that the thesis, *A Novel IEEE 802.11 Power Save Mechanism for Energy Harvesting Motivated Network*, meets the high academic standards for original scholarship and creative work established by the M.S. in Computer Science and the College of Information Technology and Engineering. This work also conforms to the editorial standards of our discipline and the Graduate College of Marshall University. With our signatures, we approve the manuscript for publication.

Dr. Cong Pu, Division of Computer Science — Committee Chairperson — Date: April 26, 2018

Dr. Wook-Sung Yoo, Division of Computer Science — Committee Member — Date: April 30, 2018

Dr. Jamil Chaudri, Division of Computer Science — Committee Member — Date: 2018 Apr 26

ACKNOWLEDGMENTS

First of all, I would like to thank my advisor Dr. Cong Pu, Dr. Wook-Sung Yoo and Dr. Jamil Chaudri for supporting me and sharing their knowledge and experiences through the process of this master thesis.

Furthermore, I am grateful to my friends Cihan Secinti and Alper Yagiz Demirbas who supported me during stressful times throughout this master thesis process.

Lastly but most importantly, I am very grateful to my parents and my sister for their support and love. This master thesis would never have been completed without them.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

The spread of wirelessly connected computing sensors and devices and hybrid networks are leading to the emergence of an Internet of Things (IoT), where a myriad of multi-scale sensors and devices are seamlessly blended for ubiquitous computing and communication. However, the communication operations of wireless devices are often limited by the size and lifetime of the batteries because of the portability and mobility. To reduce energy consumption during wireless communication, the IEEE 802.11 standard specifies a power management scheme, called Power Saving Mechanism (PSM), for IEEE 802.11 devices. However, the PSM of IEEE 802.11 was originally designed for battery-supported devices in single-hop Wireless Local Area Networks (WLANs), and it does not consider devices that are equipped with rechargeable batteries and energy harvesting capability. In this thesis, the original PSM is extended by incorporating with intermittent energy harvesting in the IEEE 802.11 Medium Access Control (MAC) layer specification, and a novel energy harvesting aware power saving mechanism, called *EH-PSM*, is proposed. The basic idea of EH-PSM is to assign a longer contention window to a device in energy harvesting mode than that of a device in normal mode to make the latter access the wireless medium earlier and quicker. In addition, the device in energy harvesting mode stays active as far as it harvests energy and updates the access point of its harvesting mode to enable itself to be ready for receiving and sending packets or overhearing any on-going communication. The proposed scheme is evaluated through extensive simulation experiments using OMNeT++ and its performance is compared with the original PSM. The simulation results indicate that the proposed scheme can not only improve the packet delivery ratio and throughput but also reduce the packet delivery latency.

I. INTRODUCTION

With recent technological advances in portability, mobility, low-power microprocessors, and high speed wireless Internet, embedded computing devices capable of wireless communication are rapidly proliferating. The growing presence of WiFi and 4G Long-Term Evolution (LTE) enable users to pursue seemingly insatiable access to Internet services and information wirelessly. It is predicted that 30 billion wirelessly connected devices will be available by 2020, nearly triple the number that exists today (Intelligence, 2013). The spread of these devices and hybrid networks is leading to the emergence of an Internet of Things (IoT), where a myriad of multi-scale sensors and devices are seamlessly blended for ubiquitous computing and communication (Palattella et al., 2016). The prevalence of cloud, social media, and wearable computing and the reduced cost of processing power, storage, and bandwidth are fueling explosive development of IoT applications in major domains (i.e., personal and home, enterprise, utilities, and mobile), which have the potential to create an economic impact of $2.7 trillion to $6.2 trillion annually by 2025 (Gubbi, Buyya, Marusic, & Palaniswami, 2013; Al-Fuqaha, Guizani, Mohammadi, Aledhari, & Ayyash, 2015). It is envisioned that wirelessly connected smart devices under the IoT will not only enhance flexible information accessibility and availability but also improve our lives further.

To realize this vision, however, a limited lifetime of the battery to power wireless devices must be overcome. For example, the TMote™ Sky node consumes 64.68 mW in receive mode (Raymond, Marchany, Brownfield, & Midkiff, 2009). Using two standard 3,000 mAh AA batteries, the network lifetime is only 5.8 days if nodes are heavily utilized (Pu, Gade, Lim, Min, & Wang, 2014). In addition, rapidly proliferating wearable devices implanted to anywhere of user (e.g., glasses, clothes, shoes, accessories, or even under skin) are directly affected by the

lifetime of batteries ("Google Glass," n.d.; Richmond, 2013). In order to extend the lifetime of the batteries, energy harvesting from surrounding environmental resources (e.g., vibrations, thermal gradients, lights, wind, etc.) has been given considerable attention as a way to either eliminate replacing the batteries or at least reduce the frequency of recharging the batteries. For example, ambient vibration-based energy harvesting has been widely deployed because of the available energy that can be scavenged from an immediate environment, such as the pulse of a blood vessel, or the kinetic motion of walking or running. Piezoelectric-based energy harvesting is favored when vibration is the dominant source of environmental energy and solar light is not always available. Though energy harvesting from photo-voltaic cells is popular and well-studied, it is inefficient because of the unpredictable availability of solar irradiation, such as installed location (e.g., a shaded area), initial position (e.g., seasonal variations between the sun's angle and solar panel), weather conditions (e.g., a rainy season), and harvesting period (e.g., daytime only).

Thus, it is anticipated that energy harvesting will play a pivotal role in making possible self-sustainable wireless devices ranging from nano-scale sensors to hand-held mobile devices, and serve as a major building block for emerging IoT applications. Although environmental energy harvesting has been well studied in civil and mechanical engineering, research on energy harvesting sensitive communication algorithms and protocols embedded into link layer (i.e., IEEE 802.11 Medium Access Control (MAC)) is still in its infancy. Thus, key research motivations of this paper are summarized: (i) Prudent energy efficient mechanisms have been proposed to extend the network lifetime. Despite the prior best effort-based approaches, manually replacing the batteries or recharging the batteries is ultimately unavoidable. Disposable batteries can address this issue but they may pose a potential environmental hazard; (ii) Energy

harvesting from immediate environmental sources may radically shift the paradigm on energy management from reducing energy consumption to maximizing the utilization of opportunistic energy; (iii) With the increasing prevalence of wearable computing, the adoption of energy harvesting techniques to multi-scale wireless sensors and mobile devices is essential to the design of IoT applications.

In this thesis, a novel Power Saving Mechanism (PSM) of IEEE 802.11 for self-sustainable devices supported by intermittent energy harvesting is designed and proposed. The proposed research will shift the paradigm of energy management from conserving limited battery energy to maximizing the utilization of harvested energy, and provide design considerations to the broader IoT community seeking new applications. The major contribution is briefly summarized in two-fold:

- A novel energy harvesting aware power saving mechanism of IEEE 802.11, called *EH-PSM*, is designed and proposed incorporating with intermittent energy harvesting from environmental resources. The basic idea of EHPSM is to assign a longer contention window to a device in energy harvesting mode than that of a device in normal mode to make the latter access the wireless medium earlier and quicker. In addition, the device in energy harvesting mode stays awake as far as it harvests energy and updates the access point of its harvesting mode to enable itself to be ready for receiving and sending packets or overhearing any ongoing communication.
- A customized discrete event-driven simulation framework is developed by using OMNeT++ and the proposed scheme through extensive simulation experiments is evaluated in terms of packet delivery ratio, throughput, packet delivery latency, and total active time (Varga, 2014). The original PSM in energy harvesting environment is also

revisited and implemented for performance comparison. The simulation results indicate that the proposed scheme can not only improve the packet delivery ratio and throughput but also reduce the packet delivery latency.

The rest of thesis is organized as follows. The prior approaches are analyzed and presented in Related Work. A system model and the proposed energy harvesting aware power saving mechanism are presented in The Proposed Approach. Performance Evaluation presents extensive simulation results and their analyses. Finally, we conclude the thesis in Conclusion.

II. RELATED WORK

In the research conducted by Jung, Qi, Yu, and Suh (2014), an energy saving algorithm is proposed to reduce power consumption of tethering smartphone, which plays a role of mobile access point temporarily. The basic idea is that smartphone turns off its WiFi interface when there is no traffic in order to conserve battery power without increasing the packet delay substantially. Ding et al. (2012) propose a system, called *Percy*, to maximize the energy saving of static and dynamic PSM respectively while minimizing the delay of flow completion time. The Percy deploys a web proxy at the access point and suitably configures the PSM parameters, and is designed to work with unchanged clients running dynamic PSM, unchanged access points, and Internet servers. In the research reported by Gupta and Mohapatra (2007), a detailed anatomy of the power consumption by various components of WiFi-based phones has been provided. Through a measurement-based study of WiFi-based phones, the power consumption for various workloads at various components has been analyzed.

WiFi continues to be a prime source of energy consumption in mobile devices, and energy optimization has conventionally been designed with a single access point. However, network contention among different access points can dramatically increase client's energy

consumption. Thus, Manweiler and Choudhury (2011) design and propose an approach to achieve energy efficiency by evading network contention. The basic idea is that the access points regular the sleeping window of their clients in a way that different access points are active/inactive during non-overlapping time windows. In the research conducted by Liu and Zhong (2008), a micro power management is proposed to enable an IEEE 802.11 interface to enter unreachable power saving mode even between medium access control (MAC) frames, without noticeable impact on the traffic flow. In order to control data lost, the proposed scheme leverages the retransmission mechanism in IEEE 802.11 and controls frame delay to adapt to demanded network throughput with minimal cooperation from the access point. He, Yuan, Ma, and Li (2008) present an experimental study of the IEEE 802.11 power saving mechanism on personal digital assistant (PDA) in a Wireless Local Area Network (WLAN), where the power consumption of the PDA in both continuous active mode and power saving mode are measured under various traffic scenarios, beacon period, and background multicast traffic. In the research reported by Eu, Tan, and Seah (2011), the performance of different MAC schemes based on carrier-sense multiple access (CSMA) and polling techniques for wireless sensor networks which are solely powered by solar energy are studied. In the research conducted by Vithanage, Fafoutis Andersen, and Dragoni (2013), the feasibility of powering wireless metering devices (e.g., heat cost allocators) by thermal energy harvested from radiators is investigated.

In summary, there is a significant amount of research effort on the IEEE 802.11 power saving mechanism and its variants. However, to the best of author's knowledge, the proposed research focusing on designing energy harvesting aware power saving mechanism and integrating it with the original PSM of IEEE 802.11 is new.

III. THE PROPOSED APPROACH

In this section, first, the IEEE 802.11 Power Saving Mechanism (PSM) is reviewed. Then the system model is presented, and a novel energy harvesting aware PSM of IEEE 802.11 for self-sustainable devices supported by intermittent energy harvesting is proposed.

A. OVERVIEW OF THE IEEE 802.11 POWER SAVING MECHANISM

As proposed in the IEEE 802.11 standard, an IEEE 802.11 based wireless network interface can choose to stay in either awake state or sleep state at any time (IEEE Computer Society LAN MAN Standards Committee, 1997). In the awake state, the device turns on its wireless interface and performs normal data communications, for example receiving or sending packets, or just stays in idle. In order to save the residual energy, the device can switch to the sleep state, where the radio of a device is turned off and the wireless interface cannot detect or sense any wireless communication. Wireless interface in awake state usually consumes an order of magnitude more power than that in sleep state (He et al., 2008).

To reduce the energy consumption of IEEE 802.11 devices during wireless communications, IEEE Computer Society LAN MAN Standards Committee (1997) specifies a power management scheme, called Power Saving Mechanism (PSM). The basic idea of PSM is that the devices sleep most of the time and stay at a low power state (i.e., turn off wireless interface) but periodically wake up and switch to a high power state (i.e., turn on wireless interface) to receive the packets buffered at the access point (AP). In PSM, the AP buffers incoming packets destined for devices in low power state and periodically announces its buffering status through the traffic indication map (TIM) contained in the beacon frames. The device wakes up at the beginning of beacon interval periodically to listen to the beacon frames. If the corresponding bit of the association ID (AID) of the device is set in the TIM, the device will

stay awake, and wait for the AP to initialize a PS-Poll packet to retrieve data packet from it and/or send the buffered packet to it. The AP can issue multiple PS-Poll packets until all outstanding packets from the devices have been retrieved. As opposed to the continuously awake mode, a device applying power saving mechanism can often have opportunities to turn off its wireless interface to save energy when it has no packets buffered at the AP, or no packets need to be sent to the AP. Here, a snapshot of the power saving mechanism of IEEE 802.11 is shown in Figure 1. For example, the device S_A and S_B wake up at the beginning of beacon interval and listen to the TIM broadcasted by the AP. Suppose that the AP initializes a PS-Poll packet to S_A first and sends the buffered packet $DATA_a$ to it. After receiving the packet $DATA_a$, S_A replies an *ACK* packet to the AP after a short time period *SIFS*, and then switches to power saving mode and turns off its wireless interfaces to save energy.

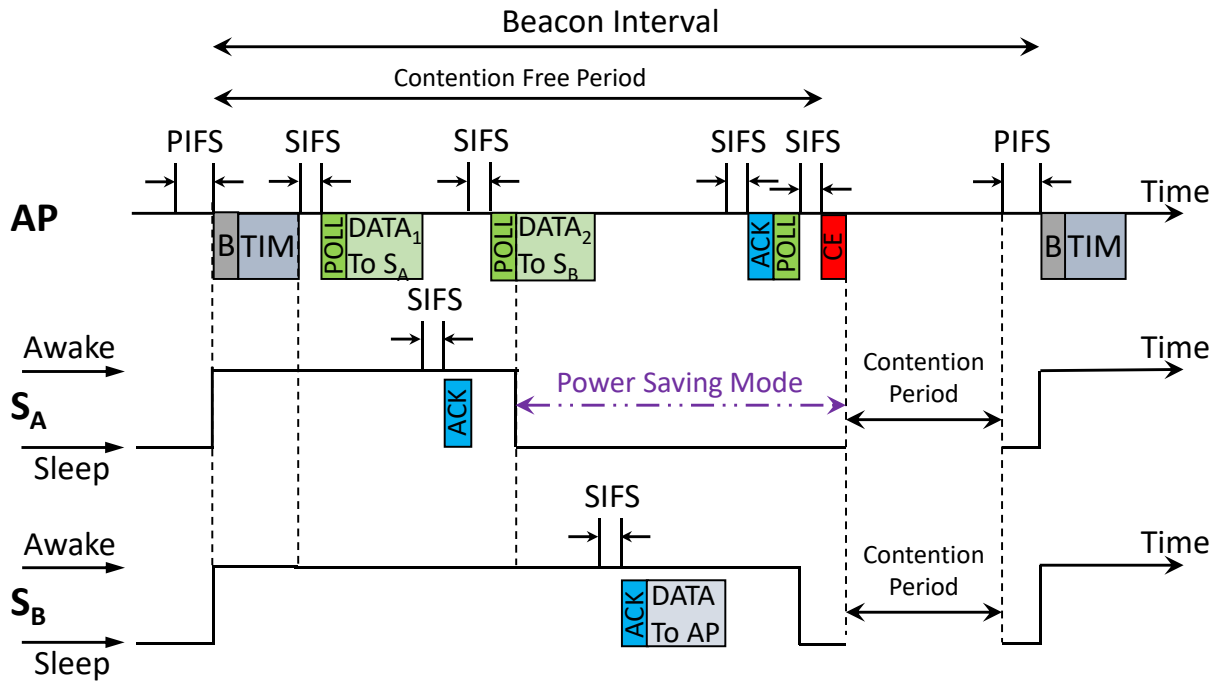


Figure 1. A snapshot of the power saving mechanism of IEEE 802.11.

B. SYSTEM MODEL

In this thesis, it is assumed that each IEEE 802.11 device is equipped with an energy harvesting component to replenish its rechargeable battery. For example, a piezoelectric fiber composite bi-morph (PFCB) W14 based energy harvesting from an immediate environment (e.g., disturbance or typical body movements) can generate sufficient power (i.e., 1.3 mW – 47.7 mW) for wireless sensors (Starner, 1996; Starner, & Paradiso, 2004.; Wang, 2011). It is envisaged that multi-scale piezo devices and integrated self-charging power cells (SCPCs) will enhance the efficiency of energy harvesting (Xue, Wang, Guo, Zhang, & Wang, 2012). An energy harvesting is modeled by a two-state Markov process with energy harvesting (S_h) and normal (S_n) modes (Pu et al., 2014). A device stays in S_n mode for a random amount of time, which is exponentially distributed with a mean λ_n, and changes its mode into S_h mode. After harvesting energy for some amount of time in S_h mode, which is also assumed to be exponentially distributed with a mean λ_h, the device changes its mode back to S_n mode. Both S_n and S_h modes are repeated.

C. ENERGY HARVESTING AWARE POWER SAVING MECHANISM

The PSM of IEEE 802.11 was originally designed for battery-supported devices in single-hop Wireless Local Area Networks (WLANs), and does not consider devices that are equipped with rechargeable batteries and energy harvesting component. In this thesis, the original PSM is extended by incorporating with intermittent energy harvesting in the IEEE 802.11 Medium Access Control (MAC) layer specification, and a novel energy harvesting aware power saving mechanism, called *EH-PSM*, is proposed. The basic idea of EH-PSM is to assign a longer contention window to a device in energy harvesting mode than that of a device in normal mode to make the latter access the wireless medium earlier and quicker. In addition, the device in energy harvesting mode stays awake as far as it harvests energy and updates the AP of its energy

harvesting mode to enable itself to be ready for receiving and sending packets, or overhearing any on-going communication. The rationale behind the EH-PSM is to shift the paradigm of energy management from conserving limited battery energy to maximizing the utilization of harvested energy. Thus, (i) how to assign contention window to a device in energy harvesting mode, and (ii) how to extend the awake time period of a device in energy harvesting mode are focused on.

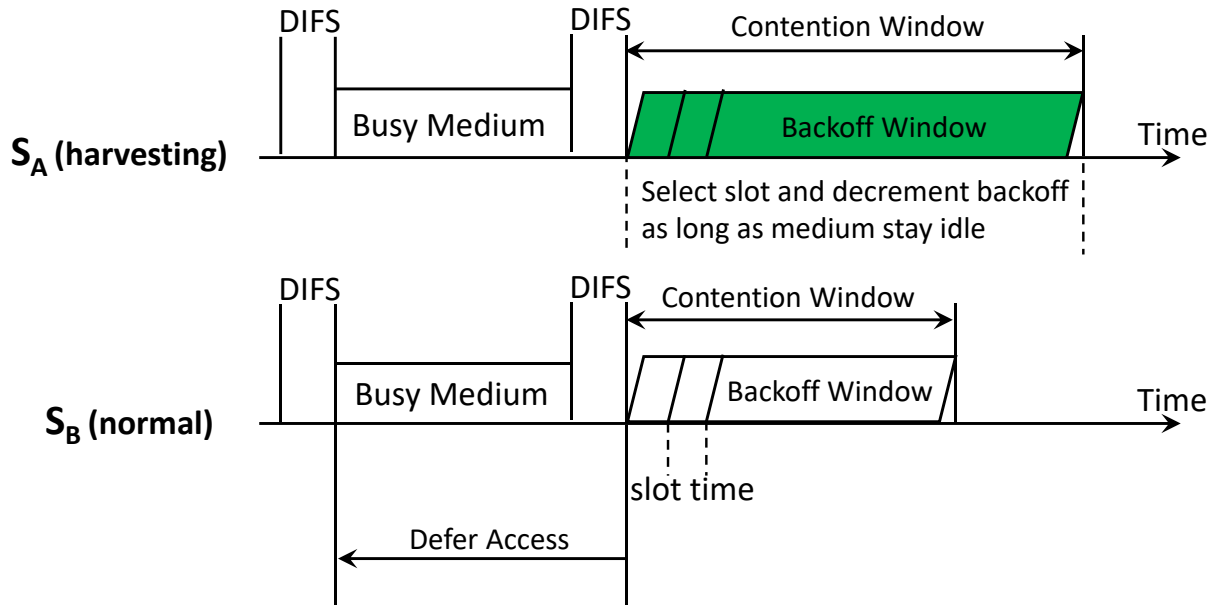


Figure 2. An example of longer contention window for a device in energy harvesting mode.

First, in the original PSM, each device uniformly chooses the contention window for backoff period before accessing the medium to avoid any potential collision. In the presence of energy harvesting, however, each device may need to adjust its contention window differently. The basic idea is to intentionally assign a longer contention window to a device in energy harvesting mode than that of a device in normal mode. Then a device containing the less amount of residual energy or staying in normal mode has more chances to choose a shorter backoff period to access the medium earlier and quicker, and then turns off its wireless interface, which

finally results in lower energy consumption. Since a device in energy harvesting mode may experience a longer delay before initiating the communication, it will have a shorter contention window later to access the medium for fairness when it is in the normal mode. For example, as shown in Figure 2, suppose that device S_A and S_B is in energy harvesting mode and normal mode, respectively. Since S_A is in energy harvesting mode, it intentionally sets a longer contention window and randomly chooses a backoff period. However, S_B is in normal mode, and it follows the original PSM and selects the backoff period from a normal contention window that is shorter than that of S_A. Thus, S_B has more chances to select a shorter backoff period from normal contention window, accesses the medium earlier, and finally consumes less amount of residual energy.

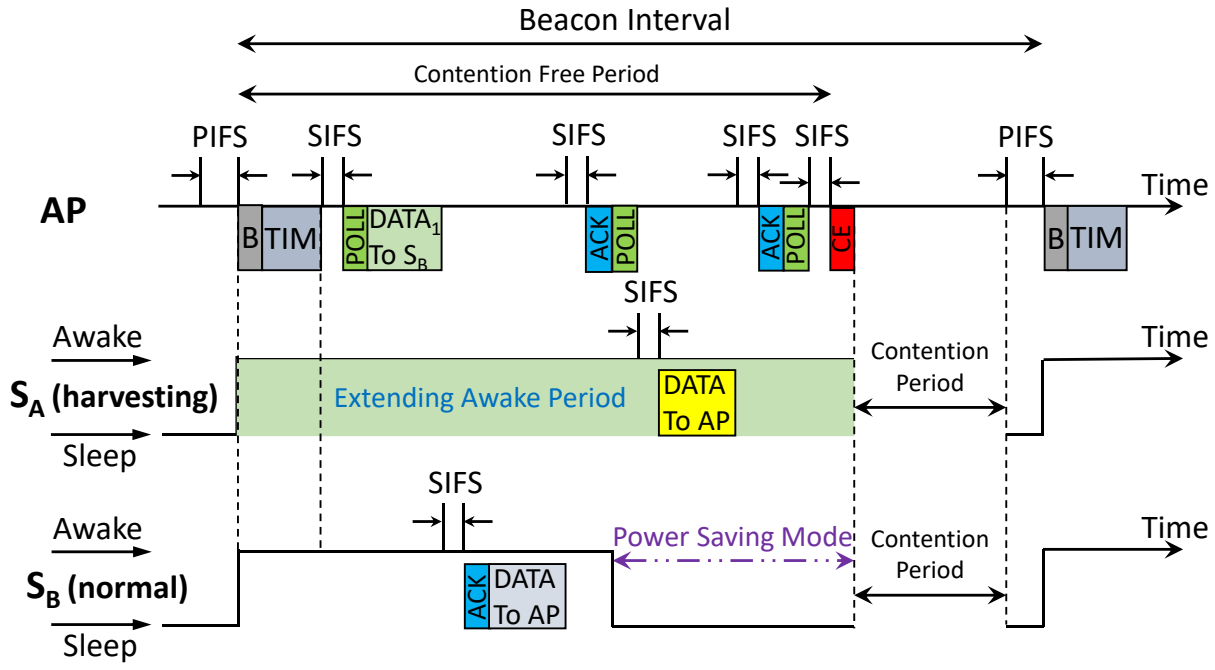


Figure 3. An example of node in energy harvesting mode extends awake period.

Second, in the original PSM, as shown in Figure 1, a device immediately sleeps back after receiving the buffered packets from the AP or sending generated packets to the AP in order

to reduce energy consumption. In the EH-PSM, however, the basic idea is that a device in energy harvesting mode stays awake as far as it harvests energy and updates the AP of its energy harvesting mode. This approach enables the device to be ready for receiving and sending packets, or overhearing any on-going communication. In addition, in order to reduce the energy consumption of a device in normal mode, the AP can specify non-harvesting device as early polled device in the Traffic Indication Map (TIM) and send the PS-Poll packet and buffered packets immediately after broadcasting the TIM. After receiving buffered packets from AP or sending the outstanding packet to AP, the device in normal mode can switch to power saving mode and turn off its wireless interface to save energy. For example, as shown in Figure 3, S_A is in energy harvesting mode while S_B is in normal mode. In order to reduce the energy consumption of S_B, the AP first polls S_B's packet and sends the buffered packet to it after broadcasting the TIM. After receiving the buffered packet from AP and replying the ACK with the outstanding packet to AP respectively, S_B switches to power saving mode and turns off its wireless interface to reduce energy consumption. However, S_A stays awake as far as it harvests energy in energy harvesting mode, and extends awake time period to overhear any on-going communication, e.g., PS-Poll packet. Since S_A extends awake time period, whenever it has a newly generated packet for AP, it can directly send the packet after overhearing the PS-Poll packet. Here, major operations of the EH-PSM are summarized in Figure 4.

Notations:

• B_{int}, T_{CFP}, T_{CP}, CW_i, t_i^{boff}, and δ: Beacon interval, contention-free period, contention period, contention window of device n_i, backoff period of device n_i, and extension of contention window.

• $pkt[src, des, type]$: A packet with source id, src, destination id, des, and packet type, $type$. Here, $type$ is *Data*, *PS-Poll* or *ACK*.

• *TIM*, S_h, S_n: Defined before.

◇ At the beginning of B_{int}, device n_i wakes up to listen to *TIM*:

```
/* Contention free period T_CFP begins */
if i ∈ TIM
  if n_i is in S_n
    /* n_i is in normal mode */
    Wait for pkt[AP, i, PS-Poll] from AP;
    Reply pkt[i, AP, ACK] to AP with potential pkt[i, AP, Data];
    Turn off wireless interface;
  else
    Wait for pkt[AP, i, PS-Poll] from AP;
    Reply pkt[i, AP, ACK] to AP with potential pkt[i, AP, Data];
    Keep wireless interface on; Overhear on-going communication;
else
  if n_i is in S_n
    Turn off wireless interface;
  else
    Keep wireless interface on; Overhear on-going communication;
```

◇ When T_{CFP} ends, and T_{CP} begins:

```
/* Contention period T_CP begins */
if n_i is in S_n
  t_i^boff = rand(CW_i);
else
  t_i^boff = rand(CW_i + δ);
```

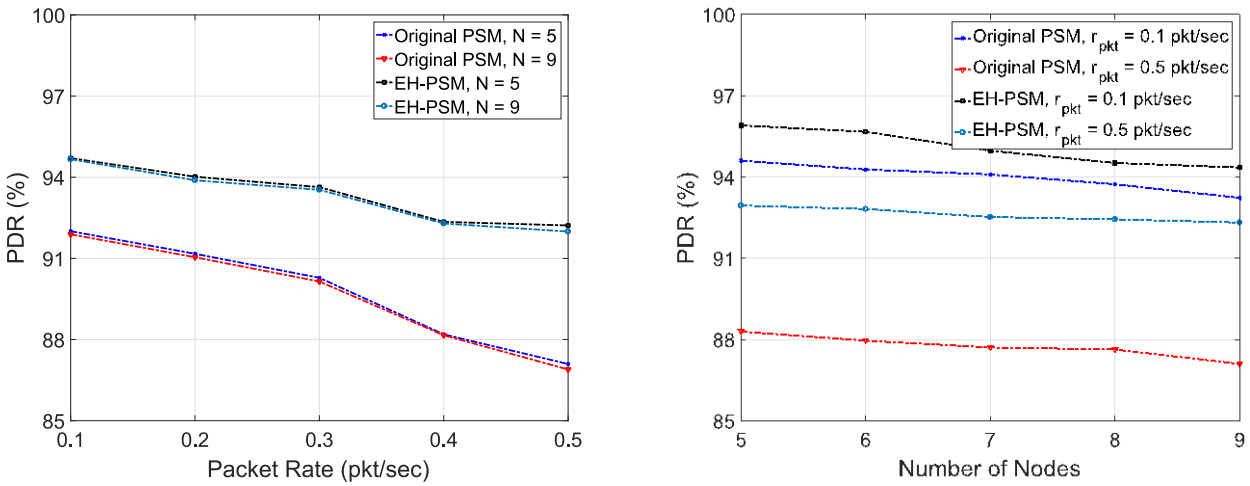
Figure 4. The pseudocode of EH-PSM.

IV. PERFORMANCE EVALUATION

A. SIMULATION TESTBED

A customized discrete-event driven simulator using OMNeT++ is developed to conduct our experiments. A 600×400 m^2 network area is considered, where 5 to 9 devices which belong to one access point (AP) are distributed in the network. The communication range of each device is 500 (m). The radio model simulates CC2420 with a normal data rate of 2 Mbps. The access point generates data traffic with packet injection rate 0.1 to 1.0 pkt/sec and the packet size is 60 Bytes. The periods of energy harvesting and normal states are assumed to be exponentially distributed with mean λ_h (50 seconds) and λ_n (25 seconds), respectively. The total simulation

time is 5000 seconds. In this thesis, the performance is measured in terms of packet delivery ratio (PDR), throughput, packet delivery latency, and total awake time by changing key simulation parameters, including number of nodes (N) and packet rate (r_{pkt}). For performance comparison, the proposed scheme EH-PSM is compared with the original PSM of IEEE 802.11.



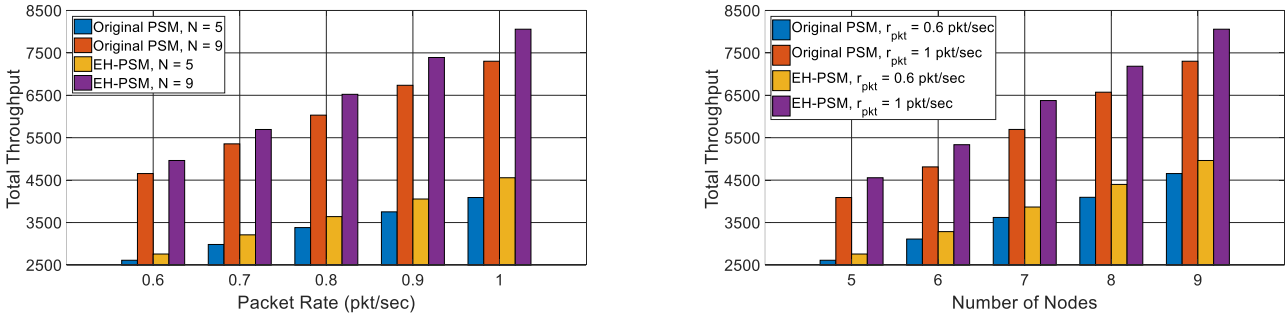
(a) (b)

Figure 5. The performance of packet delivery ratio (PDR) against packet rate and number of nodes.

B. PERFORMANCE COMPARISON

First, the packet delivery ratio (PDR) is measured by varying packet rate and number of nodes in Figure 5. In Subfigure 5(a), as the packet rate increases from 0.1 to 0.5 pkt/sec, the PDR of EH-PSM and original PSM decreases from 95% and 92% to 93% and 87%, respectively. Since each node generates more packets with larger packet rate, packets could have more chances to collide with each other, and the PDR decreases. The EH-PSM shows the higher PDR than that of original PSM because each node stays awake as far as it is in energy harvesting mode and overhears on-going communication, it can forward the packets to AP directly with a shorter contention window based on the overhearing network traffic. However, the PDR is not

sensitive to the number of nodes in the network, and thus, a slightly lower PDR is observed with larger number of nodes. In Subfigure 5(b), the overall PDR of EH-PSM and original PSM decreases as the number of nodes increases, because more number of nodes will content for the medium for sending packets to the AP during the contention period, packets have more chances to collide with each other, and the number of packets received by the AP is reduced. As the packet rate increases, a lower PDR is observed, because more number of packets are generated and forwarded to the AP, more packets collide with each other at the AP, which results in a lower PDR.

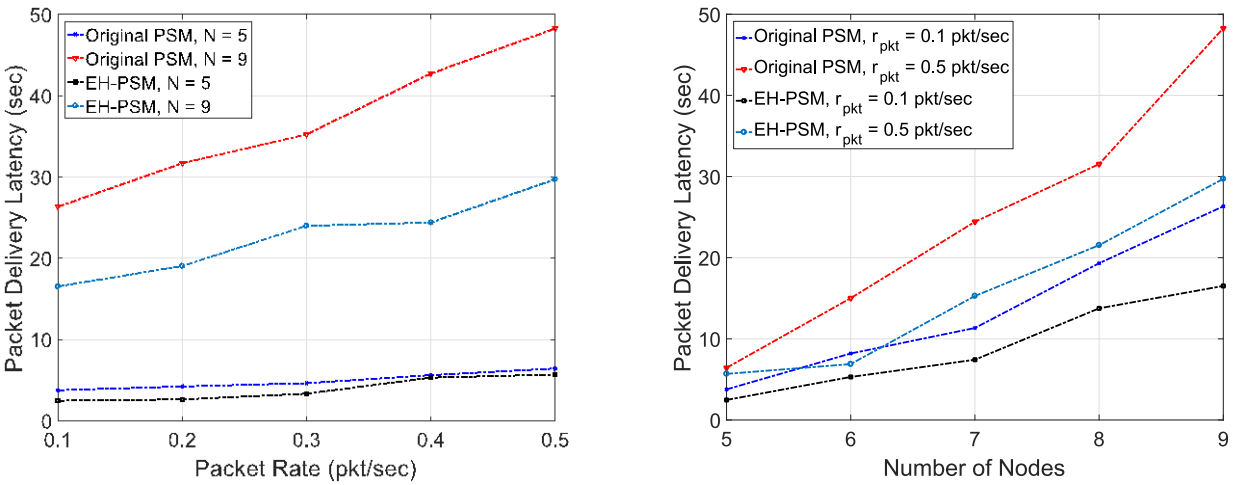


(a) (b)

Figure 6. The performance of throughput against packet rate and number of nodes.

Second, Figure 6 shows the throughput of EH-PSM and original PSM with varying packet rate and number of nodes. As shown in Subfigure 6(a), the overall throughput increases as the packet rate increases, because each node can generate more number of packets and send them to the AP. However, the EH-PSM shows a higher throughput than that of original PSM with different number of nodes, because each node in energy harvesting mode can extend their awake time period to be ready for receiving and sending more packets. When the number of nodes increases, a higher throughput is observed by EH-PSM and original PSM, respectively, because more number of nodes could send more packets to the AP, the throughput is increased. However,

the EH-PSM still performs better than original PSM, because more nodes could stay awake for a longer time period in energy harvesting mode and overhear on-going communication, and more packets can be delivered to the AP when the medium is free. In Subfigure 6(b), as the number of nodes increases, the throughput of EHPSM and original PSM increases, respectively. Since more number of nodes are associated with the AP and could generate and send more packets, finally a higher throughput is achieved. However, our scheme still achieves a better performance than original PSM, because more number of nodes can switch to energy harvesting mode and then stay awake as far as they harvest energy, more packets can be generated and sent to the AP, which result in a higher throughput.



(a) (b)

Figure 7. The performance of packet transmission latency against packet rate and number of nodes.

Third, the packet delivery latency is measured by varying packet rate and number of nodes in Figure 7. Overall, the packet delivery latency increases as the packet rate increases in Subfigure 7(a). With a larger packet rate, the AP can generate more packets for each node in the network. However, packet receiver could be in normal mode and switch to power saving mode

(i.e., turn off wireless interface) after receiving all buffered packets from the AP. The newly generated packets at the AP have to be buffered until the next beacon period, thus, a higher packet delivery latency is observed. On the other hand, EH-PSM scheme shows a lower packet delivery latency than original PSM, because each node could be in energy harvesting mode, extend awake time period, and then receive more newly generated packets from the AP quickly. As shown in Subfigure 7(b), the overall packet delivery latency of EH-PSM and original PSM increases as the number of nodes increases. This is because more number of nodes could be in normal node and switch to power saving mode after receiving all buffered packets from the AP, the newly generated packets have to be buffered at the AP and experience a longer packet delivery latency. However, the EH-PSM still provides the better performance than that of original PSM because the node in energy harvesting mode extends its awake time period and enables itself to be ready for receiving packets. Thus, the AP can directly send the newly generated packets quickly, which results in a lower packet delivery latency.

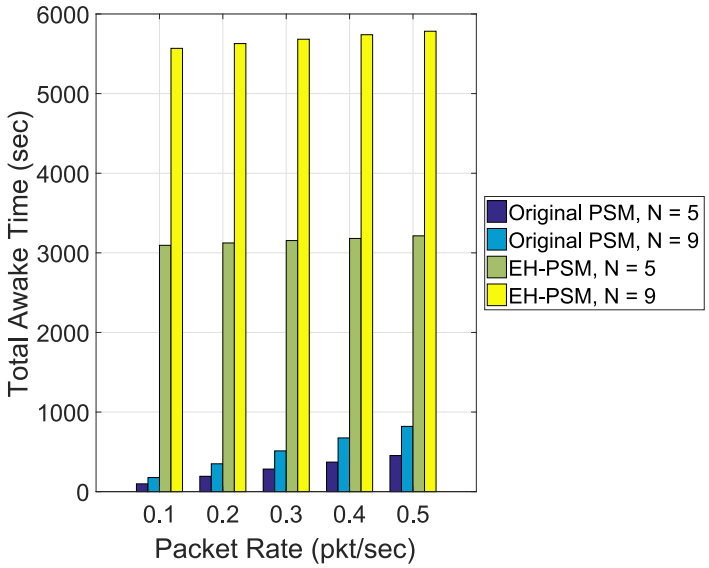


Figure 8. The performance of total active time period against packet rate and number of nodes.

Fourth, the total active time against packet rate and number of nodes are measured in Figure 8. Overall, the proposed EH-PSM achieves a much higher total active time than the original PSM, because the EH-PSM enables each node in energy harvesting mode to extend its awake time period, a larger total active time period can be observed compared to that of original PSM. As the number of nodes increases, the total active time is increased because more number of nodes could stay in energy harvesting mode and extend their active time period.

V. CONCLUSION

In this thesis, the power saving mechanism incorporating with intermittent energy harvesting in the IEEE 802.11 Medium Access Control (MAC) layer specification was investigated, a novel energy harvesting aware power saving mechanism, called EH-PSM, was proposed. In the EH-PSM, a longer contention window is assigned to a device in energy harvesting mode than that of a device in normal mode to make the latter access the wireless medium earlier and quicker. In addition, the device in energy harvesting mode stays awake as far as it harvests energy and updates the access point of its energy harvesting mode to enable itself to be ready for receiving and sending packets, or overhearing any on-going communication. The performance of the proposed scheme was evaluated through extensive simulation experiments, and compared with the original PSM of IEEE 802.11. Extensive simulation results indicate that the proposed scheme achieves better performance in terms of packet delivery ratio, throughput, and packet delivery latency.

REFERENCES

Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., & Ayyash, M. (2015). Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys & Tutorials*, *17*(4), 2347-2376.

Ding, N., Pathak, A., Koutsonikolas, D., Shepard, C., Hu, Y. C., & Zhong, L. (2012, March). Realizing the full potential of psm using proxying. In *INFOCOM, 2012 Proceedings IEEE*(pp. 2821-2825). IEEE.

Eu, Z. A., Tan, H. P., & Seah, W. K. (2011). Design and performance analysis of MAC schemes for wireless sensor networks powered by ambient energy harvesting. *Ad Hoc Networks*, *9*(3), 300-323.

Google Glass, http://www.google.com/glass/start/.

Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. *Future generation computer systems*, *29*(7), 1645-1660.

Gupta, A., & Mohapatra, P. (2007, June). Energy consumption and conservation in wifi based phones: A measurement-based study. In *Sensor, Mesh and Ad Hoc Communications and Networks, 2007. SECON'07. 4th Annual IEEE Communications Society Conference on* (pp. 122-131). IEEE.

He, Y., Yuan, R., Ma, X., & Li, J. (2008, March). The IEEE 802.11 power saving mechanism: An experimental study. In *Wireless Communications and Networking Conference, 2008. WCNC 2008. IEEE* (pp. 1362-1367). IEEE.

IEEE Computer Society LAN MAN Standards Committee. (1997). Wireless LAN medium access control (MAC) and physical layer (PHY) specifications. *IEEE Standard 802.11-1997*.

Intelligence, A. B. (2013). More than 30 billion devices will wirelessly connect to the internet of everything in 2020. *New York, NY, USA.: Allied Business Intelligence (ABI) Research. Retrieved November*, *10*, 2014.

Jung, K. H., Qi, Y., Yu, C., & Suh, Y. J. (2014, April). Energy efficient wifi tethering on a smartphone. In *INFOCOM, 2014 Proceedings IEEE* (pp. 1357-1365). IEEE.

Liu, J., & Zhong, L. (2008, June). Micro power management of active 802.11 interfaces. In *Proceedings of the 6th international conference on Mobile systems, applications, and services* (pp. 146-159). ACM.

Manweiler, J., & Roy Choudhury, R. (2011, June). Avoiding the rush hours: WiFi energy management via traffic isolation. In *Proceedings of the 9th international conference on Mobile systems, applications, and services* (pp. 253-266). ACM.

Palattella, M. R., Dohler, M., Grieco, A., Rizzo, G., Torsner, J., Engel, T., & Ladid, L. (2016). Internet of things in the 5G era: Enablers, architecture, and business models. *IEEE Journal on Selected Areas in Communications*, *34*(3), 510-527.

Pu, C., Gade, T., Lim, S., Min, M., & Wang, W. (2014, October). Lightweight forwarding protocols in energy harvesting wireless sensor networks. In *Military Communications Conference (MILCOM), 2014 IEEE* (pp. 1053-1059). IEEE.

Raymond, D. R., Marchany, R. C., Brownfield, M. I., & Midkiff, S. F. (2009). Effects of denial-of-sleep attacks on wireless sensor network MAC protocols. *IEEE transactions on vehicular technology*, *58*(1), 367-380.

Richmond, S. (2013). Wearable computing is here already: How hi-tech got under our skin. *The Independent*.

Starner, T. (1996). Human-powered wearable computing. *IBM systems Journal*, *35*(3.4), 618-629.

Starner, T., & Paradiso, J. A. (2004). Human generated power for mobile electronics. *Low power electronics design*, *45*, 1-35.

Varga, A. (2014). OMNeT++. http://www.omnetpp.org/.

Vithanage, M. D., Fafoutis, X., Andersen, C. B., & Dragoni, N. (2013, July). Medium access control for thermal energy harvesting in advanced metering infrastructures. In *EUROCON, 2013 IEEE* (pp. 291-299). IEEE.

Wang, Z. L. (2011). Nanogenerators for self-powered devices and systems.

Xue, X., Wang, S., Guo, W., Zhang, Y., & Wang, Z. L. (2012). Hybridizing energy conversion and storage in a mechanical-to-electrochemical process for self-charging power cell. *Nano letters*, *12*(9), 5048-5054.

APPENDIX A: APPROVAL LETTER

MARSHALL UNIVERSITY.
www.marshall.edu

Office of Research Integrity

November 9, 2017

Yigitcan Celik
Weisberg Division of Computer Science
Marshall University
Huntington, WV 25755

Dear Yigitcan:

This letter is in response to the submitted thesis abstract entitled *"A Novel IEEE 802.11 Power Save Mechanism for Energy Harvesting Motivated Networks"*. After assessing the abstract, it has been deemed not to be human subject research and therefore exempt from oversight of the Marshall University Institutional Review Board (IRB). The Code of Federal Regulations (45CFR46) has set forth the criteria utilized in making this determination. Since the information in this study does not involve human subjects as defined in the above referenced instruction, it is not considered human subject research. If there are any changes to the abstract you provided then you would need to resubmit that information to the Office of Research Integrity for review and a determination.

I appreciate your willingness to submit the abstract for determination. Please feel free to contact the Office of Research Integrity if you have any questions regarding future protocols that may require IRB review.

Sincerely,

Bruce F. Day, ThD, CIP
Director

APPENDIX B: SIMULATION SOFTWARE SOURCE CODE

1) network.ini

```
[General]
network = wireless

# status
seed-set = 1
sim-time-limit = 5000s
simtime-resolution = ps
**.hasStatus = true

**.networkLayer.configurator.networkConfiguratorModule = "configurator"
*.host*.networkLayer.arpType = "GlobalARP"

*.host*.wlan[0].typename = "Ieee80211Nic"
wireless[*].mgmtType = "Ieee80211MgmtSTA"


*.host*.wlan[*].radioType = "Ieee80211ScalarRadio"
*.host*.wlan[*].radio.transmitter.bitrate = 2Mbps
*.host*.wlan[*].radio.transmitter.preambleDuration = 0s
*.host*.wlan[*].radio.transmitter.headerBitLength = 100b
*.host*.wlan[*].radio.transmitter.communicationRange = 500m
*.host*.wlan[*].radio.transmitter.interferenceRange = 500m
*.host*.wlan[*].radio.transmitter.detectionRange = 500m
*.host*.wlan[*].radio.receiver.ignoreInterference = false

# access point
**.ap.wlan[*].mac.address = "10:00:00:00:00:00"
**.host*.**.mgmt.accessPointAddress = "10:00:00:00:00:00"
**.mgmt.frameCapacity = 10

*.host1.numPingApps = 1
*.host2.numPingApps = 1
*.host3.numPingApps = 1
*.host4.numPingApps = 1
*.host5.numPingApps = 1
*.host6.numPingApps = 1
*.host7.numPingApps = 1
*.host8.numPingApps = 1
*.host9.numPingApps = 1
*.hostReciever.numPingApps = 10
*.hostReciever.pingApp[0].destAddr = "host1"
*.hostReciever.pingApp[1].destAddr = "host2"
*.hostReciever.pingApp[2].destAddr = "host3"
*.hostReciever.pingApp[3].destAddr = "host4"
*.hostReciever.pingApp[4].destAddr = "host5"
*.hostReciever.pingApp[5].destAddr = "host6"
*.hostReciever.pingApp[6].destAddr = "host7"
*.hostReciever.pingApp[7].destAddr = "host8"
*.hostReciever.pingApp[8].destAddr = "host9"
*.host*.pingApp[*].destAddr = "hostReciever"
*.host*.pingApp[*].printPing = true
```

```
*.host1.pingApp[*].sendInterval = uniform(0s,1s)
*.host2.pingApp[*].sendInterval = uniform(0s,1s)
*.host3.pingApp[*].sendInterval = uniform(0s,1s)
*.host4.pingApp[*].sendInterval = uniform(0s,1s)
*.host5.pingApp[*].sendInterval = uniform(0s,1s)
*.host6.pingApp[*].sendInterval = uniform(0s,1s)
*.host7.pingApp[*].sendInterval = uniform(0s,1s)
*.host8.pingApp[*].sendInterval = uniform(0s,1s)
*.host9.pingApp[*].sendInterval = uniform(0s,1s)
*.hostReciever.pingApp[0].sendInterval = uniform(0s,1s)
*.hostReciever.pingApp[1].sendInterval = uniform(0s,1s)
*.hostReciever.pingApp[2].sendInterval = uniform(0s,1s)
*.hostReciever.pingApp[3].sendInterval = uniform(0s,1s)
*.hostReciever.pingApp[4].sendInterval = uniform(0s,1s)
*.hostReciever.pingApp[5].sendInterval = uniform(0s,1s)
*.hostReciever.pingApp[6].sendInterval = uniform(0s,1s)
*.hostReciever.pingApp[7].sendInterval = uniform(0s,1s)
*.hostReciever.pingApp[8].sendInterval = uniform(0s,1s)

*.host*.interfaceTable.displayAddresses = true

**.wlan[*].bitrate = 54Mbps
**.mac.dataBitrate = 54Mbps
**.mac.basicBitrate = 2Mbps
**.mac.multicastBitrate = 54Mbps
**.mac.controlBitrate = 54Mbps

*.host*.**.bitrate = 1Mbps

*.visualizer.mediumVisualizer.signalPropagationAnimationSpeed = 5.000001
*.visualizer.mediumVisualizer.signalTransmissionAnimationSpeed = 5.0005
*.visualizer.mediumVisualizer.displayTransmissions = true # enables displaying radio signals propagating through
the radio medium
*.visualizer.mediumVisualizer.displayReceptions = true # enables displaying radio signals propagating through the
radio medium
*.visualizer.mediumVisualizer.displaySignals = true # enables displaying radio signals propagating through the
radio medium
```

2) network.ned

```
package inet.examples.mynetwork.testap;
import inet.networklayer.configurator.ipv4.IPv4NetworkConfigurator;
import inet.node.wireless.AccessPoint;
import inet.visualizer.integrated.IntegratedCanvasVisualizer;
import inet.physicallayer.ieee80211.packetlevel.Ieee80211ScalarRadioMedium;
import inet.node.inet.WirelessHost;
import inet.common.lifecycle.LifecycleController;
import inet.common.figures.DelegateSignalConfigurator;

network wireless
{
   parameters:
      string hostName = default("");
      **.mgmt.numChannels = 2;
```

```
    @display("bgb=561,444");
submodules:
    radioMedium: Ieee80211ScalarRadioMedium {
        @display("p=51,203");
    }
    visualizer: IntegratedCanvasVisualizer {
        @display("p=51,49");
    }
    configurator: IPv4NetworkConfigurator {
        @display("p=51,121");
        config = xml("<config><interface hosts='*' address='145.236.x.x' netmask='255.255.0.0'/></config>");
    }
    host1: WirelessHost {
        parameters:
            @display("p=503,281");
    }
    host2: WirelessHost {
        parameters:
            @display("p=437,333");
    }
    host3: WirelessHost {
        parameters:
            @display("p=327,356");
    }
    host4: WirelessHost {
        parameters:
            @display("p=229,341");
    }
    host5: WirelessHost {
        parameters:
            @display("p=160,291");
    }
    host6: WirelessHost {
        parameters:
            @display("p=503,356");
    }
    host7: WirelessHost {
        parameters:
            @display("p=437,404");
    }
    host8: WirelessHost {
        parameters:
            @display("p=327,412");
    }
    host9: WirelessHost {
        parameters:
            @display("p=220,404");
    }
    hostReciever: WirelessHost {
        parameters:
            @display("p=327,84");
    }
    lifecycleController: LifecycleController {
        @display("p=52,286");
    }
    figureHelper: DelegateSignalConfigurator {
```

```
        @display("p=52,376");
    }
    ap: AccessPoint {
        @display("p=327,202");
    }
}
```

3) IEEE80211Frame_m.h

```
class INET_API Ieee80211Frame : public ::omnetpp::cPacket
{
 protected:
  short type;
  bool toDS;
  bool fromDS;
  bool retry;
  bool moreFragments;
  ::omnetpp::simtime_t duration;
  short AID;
  MACAddress receiverAddress;
  ::omnetpp::simtime_t MACArrive;

 private:
  void copy(const Ieee80211Frame& other);

 protected:
  // protected and unimplemented operator==(), to prevent accidental usage
  bool operator==(const Ieee80211Frame&);

 public:
  int packagePriority = 0;
  bool packetLoss = false;

  Ieee80211Frame(const char *name=nullptr, short kind=0);
  Ieee80211Frame(const Ieee80211Frame& other);
  virtual ~Ieee80211Frame();
  Ieee80211Frame& operator=(const Ieee80211Frame& other);
  virtual Ieee80211Frame *dup() const override {return new Ieee80211Frame(*this);}
  virtual void parsimPack(omnetpp::cCommBuffer *b) const override;
  virtual void parsimUnpack(omnetpp::cCommBuffer *b) override;

  // field getter/setter methods
  virtual short getType() const;
  virtual void setType(short type);
  virtual bool getToDS() const;
  virtual void setToDS(bool toDS);
  virtual bool getFromDS() const;
  virtual void setFromDS(bool fromDS);
  virtual bool getRetry() const;
  virtual void setRetry(bool retry);
  virtual bool getMoreFragments() const;
  virtual void setMoreFragments(bool moreFragments);
  virtual ::omnetpp::simtime_t getDuration() const;
  virtual void setDuration(::omnetpp::simtime_t duration);
  virtual short getAID() const;
```

```
    virtual void setAID(short AID);
    virtual MACAddress& getReceiverAddress();
    virtual const MACAddress& getReceiverAddress() const {return const_cast<Ieee80211Frame*>(this)-
>getReceiverAddress();}
    virtual void setReceiverAddress(const MACAddress& receiverAddress);
    virtual ::omnetpp::simtime_t getMACArrive() const;
    virtual void setMACArrive(::omnetpp::simtime_t MACArrive);
};

class INET_API Ieee80211DataFrame : public ::inet::ieee80211::Ieee80211DataOrMgmtFrame
{
  protected:
    MACAddress address4;
    uint16_t qos;
    int ackPolicy;
    uint8_t tid;
    bool aMsduPresent;

  private:
    void copy(const Ieee80211DataFrame& other);

  protected:
    // protected and unimplemented operator==(), to prevent accidental usage
    bool operator==(const Ieee80211DataFrame&);

  public:
    Ieee80211DataFrame(const char *name=nullptr, short kind=0);
    Ieee80211DataFrame(const Ieee80211DataFrame& other);
    virtual ~Ieee80211DataFrame();
    Ieee80211DataFrame& operator=(const Ieee80211DataFrame& other);
    virtual Ieee80211DataFrame *dup() const override {return new Ieee80211DataFrame(*this);}
    virtual void parsimPack(omnetpp::cCommBuffer *b) const override;
    virtual void parsimUnpack(omnetpp::cCommBuffer *b) override;

    // field getter/setter methods
    virtual MACAddress& getAddress4();
    virtual const MACAddress& getAddress4() const {return const_cast<Ieee80211DataFrame*>(this)-
>getAddress4();}
    virtual void setAddress4(const MACAddress& address4);
    virtual uint16_t getQos() const;
    virtual void setQos(uint16_t qos);
    virtual int getAckPolicy() const;
    virtual void setAckPolicy(int ackPolicy);
    virtual uint8_t getTid() const;
    virtual void setTid(uint8_t tid);
    virtual bool getAMsduPresent() const;
    virtual void setAMsduPresent(bool aMsduPresent);

    bool active;
};
```

4) IEEE80211Frame_m.cc

```
Ieee80211Frame::Ieee80211Frame(const char *name, short kind) : ::omnetpp::cPacket(name,kind)
{
   this->setByteLength(LENGTH_ACK / 8);

   this->type = 0;
   this->toDS = false;
   this->fromDS = false;
   this->retry = false;
   this->moreFragments = false;
   this->duration = -1;
   this->AID = -1;
   this->MACArrive = 0;

   srand(time(NULL));
   this->packetLoss = (rand() % 100) < 5 ? true : false;

   std::string temp = this->getOwner()->getClassName();
   if(temp == "inet::ieee80211::Ieee80211MgmtSTA")
   {
      Ieee80211MgmtSTA *module = check_and_cast<Ieee80211MgmtSTA *>(this->getOwner());
      if(!module->harvestingState)
      {
         this->packagePriority = 1;
      }
   }
}

void Ieee80211Frame::copy(const Ieee80211Frame& other)
{
   this->type = other.type;
   this->toDS = other.toDS;
   this->fromDS = other.fromDS;
   this->retry = other.retry;
   this->moreFragments = other.moreFragments;
   this->duration = other.duration;
   this->AID = other.AID;
   this->receiverAddress = other.receiverAddress;
   this->MACArrive = other.MACArrive;
   this->packagePriority = other.packagePriority;
   this->packetLoss = other.packetLoss;
}

Ieee80211DataFrame::Ieee80211DataFrame(const char *name, short kind) :
::inet::ieee80211::Ieee80211DataOrMgmtFrame(name,kind)
{
   this->setByteLength(DATAFRAME_HEADER_MINLENGTH / 8);
   this->setType(ST_DATA);

   this->qos = 0;
   this->ackPolicy = 0;
   this->tid = 0;
   this->aMsduPresent = false;
   this->active = true;
```

```
}
void Ieee80211DataFrame::copy(const Ieee80211DataFrame& other)
{
   this->address4 = other.address4;
   this->qos = other.qos;
   this->ackPolicy = other.ackPolicy;
   this->tid = other.tid;
   this->aMsduPresent = other.aMsduPresent;
   this->active = other.active;
}
```

5) Contention.cc

```
void Contention::startContention(int cw, simtime_t ifs, simtime_t eifs, simtime_t slotTime, ICallback *callback)
{
   startTime = simTime();
   ASSERT(ifs >= 0 && eifs >= 0 && slotTime >= 0 && cw >= 0);
   Enter_Method("startContention()");
   cancelEvent(channelGrantedEvent);
   ASSERT(fsm.getState() == IDLE);
   this->ifs = ifs;
   this->eifs = eifs;
   this->slotTime = slotTime;
   this->callback = callback;

   if(this->frame->packagePriority > 0){
      cw = 1;
   }

   backoffSlots = intrand(cw + 1);         //change backoffslots here
   EV_DETAIL << "Starting contention: cw = " << cw << ", slots = " << backoffSlots << endl;
   handleWithFSM(START);
}
```

6) IEEE80211MgmtFrames_m.h

```
class INET_API Ieee80211BeaconFrame : public ::inet::ieee80211::Ieee80211ManagementFrame
{
 protected:
   Ieee80211BeaconFrameBody body;

 private:
   void copy(const Ieee80211BeaconFrame& other);

 protected:
   // protected and unimplemented operator==(), to prevent accidental usage
   bool operator==(const Ieee80211BeaconFrame&);

 public:
   Ieee80211BeaconFrame(const char *name=nullptr, short kind=0);
   Ieee80211BeaconFrame(const Ieee80211BeaconFrame& other);
   virtual ~Ieee80211BeaconFrame();
   Ieee80211BeaconFrame& operator=(const Ieee80211BeaconFrame& other);
   virtual Ieee80211BeaconFrame *dup() const override {return new Ieee80211BeaconFrame(*this);}
   virtual void parsimPack(omnetpp::cCommBuffer *b) const override;
```

```
    virtual void parsimUnpack(omnetpp::cCommBuffer *b) override;
    simtime_t atim[20];

    // field getter/setter methods
    virtual Ieee80211BeaconFrameBody& getBody();
    virtual const Ieee80211BeaconFrameBody& getBody() const {return
const_cast<Ieee80211BeaconFrame*>(this)->getBody();}
    virtual void setBody(const Ieee80211BeaconFrameBody& body);
};
```

7) IEEE80211MgmtFrames_m.cc

```
void Ieee80211BeaconFrame::copy(const Ieee80211BeaconFrame& other)
{
    this->body = other.body;
    for(int i = 0; i < 20; i++)
    {
        this->atim[i] = other.atim[i];
    }
}
```

8) IEEE80211MgmtSTA.h

```
class INET_API Ieee80211MgmtSTA : public Ieee80211MgmtBase, protected cListener
{
  public:
    //
    // Encapsulates information about the ongoing scanning process
    //
    struct ScanningInfo
    {
        MACAddress bssid;    // specific BSSID to scan for, or the broadcast address
        std::string ssid;    // SSID to scan for (empty=any)
        bool activeScan;    // whether to perform active or passive scanning
        simtime_t probeDelay;    // delay (in s) to be used prior to transmitting a Probe frame during active scanning
        std::vector<int> channelList;    // list of channels to scan
        int currentChannelIndex;    // index into channelList[]
        bool busyChannelDetected;    // during minChannelTime, we have to listen for busy channel
        simtime_t minChannelTime;    // minimum time to spend on each channel when scanning
        simtime_t maxChannelTime;    // maximum time to spend on each channel when scanning
    };

    //
    // Stores AP info received during scanning
    //
    struct APInfo : public cObject
    {
        int channel;
        MACAddress address;    // alias bssid
        std::string ssid;
        Ieee80211SupportedRatesElement supportedRates;
        simtime_t beaconInterval;
        double rxPower;

        bool isAuthenticated;
```

```
        int authSeqExpected;    // valid while authenticating; values: 1,3,5...
        cMessage *authTimeoutMsg;    // if non-nullptr: authentication is in progress

        APInfo()
        {
            channel = -1;
            beaconInterval = rxPower = 0;
            authSeqExpected = -1;
            isAuthenticated = false;
            authTimeoutMsg = nullptr;
        }
    };

    //
    // Associated AP, plus data associated with the association with the associated AP
    //
    struct AssociatedAPInfo : public APInfo
    {
        int receiveSequence;
        cMessage *beaconTimeoutMsg;

        AssociatedAPInfo() : APInfo() { receiveSequence = 0; beaconTimeoutMsg = nullptr; }
    };

  protected:
    cModule *host;
    IInterfaceTable *interfaceTable;
    InterfaceEntry *myIface;

    // number of channels in RadioMedium -- used if we're told to scan "all" channels
    int numChannels;

    // scanning status
    bool isScanning;
    ScanningInfo scanning;

    // APInfo list: we collect scanning results and keep track of ongoing authentications here
    // Note: there can be several ongoing authentications simultaneously
    typedef std::list<APInfo> AccessPointList;
    AccessPointList apList;

    // associated Access Point
    bool isAssociated;
    cMessage *assocTimeoutMsg;    // if non-nullptr: association is in progress
    AssociatedAPInfo assocAP;

  public:
    Ieee80211MgmtSTA() : host(nullptr), interfaceTable(nullptr), myIface(nullptr), numChannels(-1),
isScanning(false), isAssociated(false), assocTimeoutMsg(nullptr) {}

  protected:
    virtual int numInitStages() const override { return NUM_INIT_STAGES; }
    virtual void initialize(int stage) override;

    /** Implements abstract Ieee80211MgmtBase method */
    virtual void handleTimer(cMessage *msg) override;
```

```
/** Implements abstract Ieee80211MgmtBase method */
virtual void handleUpperMessage(cPacket *msg) override;

/** Implements abstract Ieee80211MgmtBase method */
virtual void handleCommand(int msgkind, cObject *ctrl) override;

/** Utility function for handleUpperMessage() */
virtual Ieee80211DataFrame *encapsulate(cPacket *msg);

/** Utility method to decapsulate a data frame */
virtual cPacket *decapsulate(Ieee80211DataFrame *frame);

/** Utility function: sends authentication request */
virtual void startAuthentication(APInfo *ap, simtime_t timeout);

/** Utility function: sends association request */
virtual void startAssociation(APInfo *ap, simtime_t timeout);

/** Utility function: looks up AP in our AP list. Returns nullptr if not found. */
virtual APInfo *lookupAP(const MACAddress& address);

/** Utility function: clear the AP list, and cancel any pending authentications. */
virtual void clearAPList();

/** Utility function: switches to the given radio channel. */
virtual void changeChannel(int channelNum);

/** Stores AP info received in a beacon or probe response */
virtual void storeAPInfo(const MACAddress& address, const Ieee80211BeaconFrameBody& body);

/** Switches to the next channel to scan; returns true if done (there wasn't any more channel to scan). */
virtual bool scanNextChannel();

/** Broadcasts a Probe Request */
virtual void sendProbeRequest();

/** Missed a few consecutive beacons */
virtual void beaconLost();

/** Sends back result of scanning to the agent */
virtual void sendScanConfirm();

/** Sends back result of authentication to the agent */
virtual void sendAuthenticationConfirm(APInfo *ap, int resultCode);

/** Sends back result of association to the agent */
virtual void sendAssociationConfirm(APInfo *ap, int resultCode);

/** Utility function: Cancel the existing association */
virtual void disassociate();

/** Utility function: sends a confirmation to the agent */
virtual void sendConfirm(Ieee80211PrimConfirm *confirm, int resultCode);

/** Utility function: sends a management frame */
```

```
    virtual void sendManagementFrame(Ieee80211ManagementFrame *frame, const MACAddress& address);

    /** Called by the signal handler whenever a change occurs we're interested in */
    virtual void receiveSignal(cComponent *source, simsignal_t signalID, long value, cObject *details) override;
    virtual void receiveSignal(cComponent *source, simsignal_t signalID, cObject *obj, cObject *details) override;

    /** Utility function: converts Ieee80211StatusCode (->frame) to Ieee80211PrimResultCode (->primitive) */
    virtual int statusCodeToPrimResultCode(int statusCode);

    /** @name Processing of different frame types */
    //@{
    virtual void handleDataFrame(Ieee80211DataFrame *frame) override;
    virtual void handleAuthenticationFrame(Ieee80211AuthenticationFrame *frame) override;
    virtual void handleDeauthenticationFrame(Ieee80211DeauthenticationFrame *frame) override;
    virtual void handleAssociationRequestFrame(Ieee80211AssociationRequestFrame *frame) override;
    virtual void handleAssociationResponseFrame(Ieee80211AssociationResponseFrame *frame) override;
    virtual void handleReassociationRequestFrame(Ieee80211ReassociationRequestFrame *frame) override;
    virtual void handleReassociationResponseFrame(Ieee80211ReassociationResponseFrame *frame) override;
    virtual void handleDisassociationFrame(Ieee80211DisassociationFrame *frame) override;
    virtual void handleBeaconFrame(Ieee80211BeaconFrame *frame) override;
    virtual void handleProbeRequestFrame(Ieee80211ProbeRequestFrame *frame) override;
    virtual void handleProbeResponseFrame(Ieee80211ProbeResponseFrame *frame) override;
    //@}

    /** @name Processing of different agent commands */
    //@{
    virtual void processScanCommand(Ieee80211Prim_ScanRequest *ctrl);
    virtual void processAuthenticateCommand(Ieee80211Prim_AuthenticateRequest *ctrl);
    virtual void processDeauthenticateCommand(Ieee80211Prim_DeauthenticateRequest *ctrl);
    virtual void processAssociateCommand(Ieee80211Prim_AssociateRequest *ctrl);
    virtual void processReassociateCommand(Ieee80211Prim_ReassociateRequest *ctrl);
    virtual void processDisassociateCommand(Ieee80211Prim_DisassociateRequest *ctrl);
    //@}

  public:
     virtual MACAddress getMyAddress();
     bool harvestingState;
     simtime_t stateStartTime;
     simtime_t stateEndTime;
     simtime_t awakeTime;
     simtime_t goWakeTime;
     simtime_t sleepingTime;
     simtime_t goSleepTime;
     simtime_t delay = SIMTIME_ZERO;
     simtime_t isDelayed = SIMTIME_ZERO;
     int recievedPackageCount;
     bool active;
     void checkForState();
};
```

9) IEEE80211MgmtSTA.cc

```
void Ieee80211MgmtSTA::initialize(int stage)
{
   Ieee80211MgmtBase::initialize(stage);
```

```
    if (stage == INITSTAGE_LOCAL) {
        isScanning = false;
        isAssociated = false;
        assocTimeoutMsg = nullptr;
        myIface = nullptr;
        numChannels = par("numChannels");

        host = getContainingNode(this);
        host->subscribe(NF_LINK_FULL_PROMISCUOUS, this);

        WATCH(isScanning);
        WATCH(isAssociated);

        WATCH(scanning);
        WATCH(assocAP);
        WATCH_LIST(apList);
    }
    else if (stage == INITSTAGE_LINK_LAYER_2) {
        IInterfaceTable *ift = findModuleFromPar<IInterfaceTable>(par("interfaceTableModule"), this);
        if (ift) {
            myIface = ift->getInterfaceByName(utils::stripnonalnum(findModuleUnderContainingNode(this)-
>getFullName()).c_str());
        }
    }

    harvestingState = 0;
    stateStartTime = simTime();
    active = true;
    awakeTime = SIMTIME_ZERO;
    goWakeTime = simTime();
    sleepingTime = SIMTIME_ZERO;
    goSleepTime = simTime();
    recievedPackageCount = 0;
}

void Ieee80211MgmtSTA::handleUpperMessage(cPacket *msg)
{
    if (!isAssociated || assocAP.address.isUnspecified()) {
        EV << "STA is not associated with an access point, discarding packet" << msg << "\n";
        delete msg;
        return;
    }


    Ieee80211DataFrame *frame = encapsulate(msg);
    frame->active = active;
    sendDown(frame);
    if(!harvestingState)
    {
        if(active)
        {
            goSleepTime = simTime();
            awakeTime = awakeTime + (simTime() - goWakeTime);
        }
        active = false;
```

```
    }
}

void Ieee80211MgmtSTA::handleDataFrame(Ieee80211DataFrame *frame)
{
    // Only send the Data frame up to the higher layer if the STA is associated with an AP,
    // else delete the frame
    if (isAssociated){
        std::string name = frame->getName();
        if(frame->getTransmitterAddress().getInt() == 17592186044416 && name.find("reply") == std::string::npos)
        {
            if(active && !frame->packetLoss)
            {
                recievedPackageCount++;
                if(!harvestingState)
                {
                    active = false;
                    goSleepTime = simTime();
                    awakeTime = awakeTime + (simTime() - goWakeTime);
                }
            }
            else
            {
                isDelayed = simTime();
            }
        }// can get rid of lastBeaconTime dont need it anymore
        //directly check if it is active or not and count package according to it

        PingApp *app;
        if(getMyAddress().getInt() == 11725260718081)
        {
            app = check_and_cast<PingApp *>(getModuleByPath("wireless.host1.pingApp[0]"));
            app->recievedPackageCount = recievedPackageCount;
            app->awakeTime = awakeTime;
            app->sleepingTime = sleepingTime;
            app->delay = app->delay + (frame->getDuration());
            app->delayActual = delay;
        }
        if(getMyAddress().getInt() == 11725260718082)
        {
            app = check_and_cast<PingApp *>(getModuleByPath("wireless.host2.pingApp[0]"));
            app->recievedPackageCount = recievedPackageCount;
            app->awakeTime = awakeTime;
            app->sleepingTime = sleepingTime;
            app->delay = app->delay + (frame->getDuration());
            app->delayActual = delay;
        }
        if(getMyAddress().getInt() == 11725260718083)
        {
            app = check_and_cast<PingApp *>(getModuleByPath("wireless.host3.pingApp[0]"));
            app->recievedPackageCount = recievedPackageCount;
            app->awakeTime = awakeTime;
            app->sleepingTime = sleepingTime;
            app->delay = app->delay + (frame->getDuration());
            app->delayActual = delay;
        }
```

```
if(getMyAddress().getInt() == 11725260718084)
{
   app = check_and_cast<PingApp *>(getModuleByPath("wireless.host4.pingApp[0]"));
   app->recievedPackageCount = recievedPackageCount;
   app->awakeTime = awakeTime;
   app->sleepingTime = sleepingTime;
   app->delay = app->delay + (frame->getDuration());
   app->delayActual = delay;
}
if(getMyAddress().getInt() == 11725260718085)
{
   app = check_and_cast<PingApp *>(getModuleByPath("wireless.host5.pingApp[0]"));
   app->recievedPackageCount = recievedPackageCount;
   app->awakeTime = awakeTime;
   app->sleepingTime = sleepingTime;
   app->delay = app->delay + (frame->getDuration());
   app->delayActual = delay;
}
if(getMyAddress().getInt() == 11725260718087)
{
   app = check_and_cast<PingApp *>(getModuleByPath("wireless.host6.pingApp[0]"));
   app->recievedPackageCount = recievedPackageCount;
   app->awakeTime = awakeTime;
   app->sleepingTime = sleepingTime;
   app->delay = app->delay + (frame->getDuration());
   app->delayActual = delay;
}
if(getMyAddress().getInt() == 11725260718088)
{
   app = check_and_cast<PingApp *>(getModuleByPath("wireless.host7.pingApp[0]"));
   app->recievedPackageCount = recievedPackageCount;
   app->awakeTime = awakeTime;
   app->sleepingTime = sleepingTime;
   app->delay = app->delay + (frame->getDuration());
   app->delayActual = delay;
}
if(getMyAddress().getInt() == 11725260718089)
{
   app = check_and_cast<PingApp *>(getModuleByPath("wireless.host8.pingApp[0]"));
   app->recievedPackageCount = recievedPackageCount;
   app->awakeTime = awakeTime;
   app->sleepingTime = sleepingTime;
   app->delay = app->delay + (frame->getDuration());
   app->delayActual = delay;
}
if(getMyAddress().getInt() == 11725260718090)
{
   app = check_and_cast<PingApp *>(getModuleByPath("wireless.host9.pingApp[0]"));
   app->recievedPackageCount = recievedPackageCount;
   app->awakeTime = awakeTime;
   app->sleepingTime = sleepingTime;
   app->delay = app->delay + (frame->getDuration());
   app->delayActual = delay;
}
if(getMyAddress().getInt() == 11725260718091)
{
```

```
            app = check_and_cast<PingApp *>(getModuleByPath("wireless.host10.pingApp[0]"));
            app->recievedPackageCount = recievedPackageCount;
            app->awakeTime = awakeTime;
            app->sleepingTime = sleepingTime;
            app->delay = app->delay + (frame->getDuration());
            app->delayActual = delay;
        }

        sendUp(decapsulate(frame));
    }
    else {
        EV << "Rejecting data frame as STA is not associated with an AP" << endl;
        delete frame;
    }
}

void Ieee80211MgmtSTA::handleBeaconFrame(Ieee80211BeaconFrame *frame)
{
    checkForState();

    if(isDelayed != SIMTIME_ZERO)
    {
        delay += (simTime() - isDelayed);
        isDelayed = SIMTIME_ZERO;
    }

    if(harvestingState && !active)
    {
        active = true;
        goWakeTime = simTime();
        sleepingTime = sleepingTime + (simTime() - goSleepTime);
    }
    else if(!harvestingState)
    {
        bool tempActive = active;
        int64_t temp = simTime().raw() + 100000000000;
        int64_t now = simTime().raw();
        if(getMyAddress().getInt() == 11725260718081)
        {
            if(((frame->atim[0].raw() < temp) && (frame->atim[0].raw() > now)) || ((frame->atim[5].raw() < temp) &&
(frame->atim[5].raw() > now)))
                active = true;
            else
                active = false;
        }

        if(getMyAddress().getInt() == 11725260718082)
        {
            if(((frame->atim[1].raw() < temp) && (frame->atim[1].raw() > now)) || ((frame->atim[6].raw() < temp) &&
(frame->atim[6].raw() > now)))
                active = true;
            else
                active = false;
        }

        if(getMyAddress().getInt() == 11725260718083)
```

```
    {
      if(((frame->atim[2].raw() < temp) && (frame->atim[2].raw() > now)) || ((frame->atim[7].raw() < temp) &&
(frame->atim[7].raw() > now)))
        active = true;
      else
        active = false;
    }

    if(getMyAddress().getInt() == 11725260718084)
    {
      if(((frame->atim[3].raw() < temp) && (frame->atim[3].raw() > now)) || ((frame->atim[8].raw() < temp &&
(frame->atim[8].raw() > now))))
        active = true;
      else
        active = false;
    }

    if(getMyAddress().getInt() == 11725260718085)
    {
      if(((frame->atim[4].raw() < temp) && (frame->atim[4].raw() > now)) || ((frame->atim[9].raw() < temp) &&
(frame->atim[9].raw() > now)))
        active = true;
      else
        active = false;
    }
    if(getMyAddress().getInt() == 11725260718087)
    {
      if(((frame->atim[10].raw() < temp) && (frame->atim[10].raw() > now)) || ((frame->atim[15].raw() < temp)
&& (frame->atim[15].raw() > now)))
        active = true;
      else
        active = false;
    }

    if(getMyAddress().getInt() == 11725260718088)
    {
      if(((frame->atim[11].raw() < temp) && (frame->atim[11].raw() > now)) || ((frame->atim[16].raw() < temp)
&&  (frame->atim[16].raw() > now)))
        active = true;
      else
        active = false;
    }

    if(getMyAddress().getInt() == 11725260718089)
    {
      if(((frame->atim[12].raw() < temp) && (frame->atim[12].raw() > now)) || ((frame->atim[17].raw() < temp)
&& (frame->atim[17].raw() > now)))
        active = true;
      else
        active = false;
    }

    if(getMyAddress().getInt() == 11725260718090)
    {
      if(((frame->atim[13].raw() < temp) && (frame->atim[13].raw() > now)) || ((frame->atim[18].raw() < temp
&& (frame->atim[18].raw() > now))))
```

```
            active = true;
         else
            active = false;
      }

      if(getMyAddress().getInt() == 11725260718091)
      {
         if(((frame->atim[14].raw() < temp) && (frame->atim[14].raw() > now)) || ((frame->atim[19].raw() < temp)
&& (frame->atim[19].raw() > now)))
            active = true;
         else
            active = false;
      }

      if(tempActive && !active)
      {
         goSleepTime = simTime();
         awakeTime = awakeTime + (simTime() - goWakeTime);
      }
      else if(!tempActive && active)
      {
         goWakeTime = simTime();
         sleepingTime = sleepingTime + (simTime() - goSleepTime);
      }
   }
   EV << "Received Beacon frame\n";
   storeAPInfo(frame->getTransmitterAddress(), frame->getBody());

   // if it is out associate AP, restart beacon timeout
   if (isAssociated && frame->getTransmitterAddress() == assocAP.address) {
      EV << "Beacon is from associated AP, restarting beacon timeout timer\n";
      ASSERT(assocAP.beaconTimeoutMsg != nullptr);
      cancelEvent(assocAP.beaconTimeoutMsg);
      scheduleAt(simTime() + MAX_BEACONS_MISSED * assocAP.beaconInterval,
assocAP.beaconTimeoutMsg);

      //APInfo *ap = lookupAP(frame->getTransmitterAddress());
      //ASSERT(ap!=nullptr);
   }

   delete frame;
}
MACAddress Ieee80211MgmtSTA::getMyAddress()
{
   return myAddress;
}

void Ieee80211MgmtSTA::checkForState()
{
   stateEndTime = simTime();
   int timeDif = ((stateEndTime.raw() - stateStartTime.raw())/1000000000000);
   if(harvestingState && timeDif > 50)
   {
       harvestingState = 0;
       stateStartTime = simTime();
       EV << "Harvesting for " << timeDif << " addr:" << this->myAddress.getInt() << std::endl;
```

```
    }
    else if( !harvestingState && timeDif > 25)
    {
       harvestingState = 1;
       stateStartTime = simTime();
       EV << "Active for " << timeDif << " addr:" << this->myAddress.getInt() << std::endl;
    }
}
```

10) IEEE80211MgmtAP.cc

```
void Ieee80211MgmtAP::sendBeacon()
{
   EV << "Sending beacon\n";
   Ieee80211BeaconFrame *frame = new Ieee80211BeaconFrame("Beacon");
   Ieee80211BeaconFrameBody& body = frame->getBody();
   body.setSSID(ssid.c_str());
   body.setSupportedRates(supportedRates);
   body.setBeaconInterval(beaconInterval);
   body.setChannelNumber(channelNumber);
   body.setBodyLength(8 + 2 + 2 + (2 + ssid.length()) + (2 + supportedRates.numRates));

   frame->setByteLength(28 + body.getBodyLength());
   frame->setReceiverAddress(MACAddress::BROADCAST_ADDRESS);
   frame->setFromDS(true);

   //getSimulation()->getModule(50) for every host(STA) type of PingApp * (includes nextPingTime)
   //put that times into beacon frame
   PingApp *app;

   //send ping times
   cModule *mod;

   mod = getSimulation()->getModuleByPath("wireless.host1.pingApp[0]");
   if(mod)
   {
      app = check_and_cast<PingApp *>(mod);
      frame->atim[0] = app->nextPingTime;
   }

   mod = getSimulation()->getModuleByPath("wireless.host2.pingApp[0]");
   if(mod)
   {
      app = check_and_cast<PingApp *>(mod);
      frame->atim[1] = app->nextPingTime;
   }

   mod = getSimulation()->getModuleByPath("wireless.host3.pingApp[0]");
   if(mod)
   {
      app = check_and_cast<PingApp *>(mod);
      frame->atim[2] = app->nextPingTime;
   }

   mod = getSimulation()->getModuleByPath("wireless.host4.pingApp[0]");
```

```
if(mod)
{
   app = check_and_cast<PingApp *>(mod);
   frame->atim[3] = app->nextPingTime;
}

mod = getSimulation()->getModuleByPath("wireless.host5.pingApp[0]");
if(mod)
{
   app = check_and_cast<PingApp *>(mod);
   frame->atim[4] = app->nextPingTime;
}

mod = getSimulation()->getModuleByPath("wireless.host6.pingApp[0]");
if(mod)
{
   app = check_and_cast<PingApp *>(mod);
   frame->atim[10] = app->nextPingTime;
}

mod = getSimulation()->getModuleByPath("wireless.host7.pingApp[0]");
if(mod)
{
   app = check_and_cast<PingApp *>(mod);
   frame->atim[11] = app->nextPingTime;
}

mod = getSimulation()->getModuleByPath("wireless.host8.pingApp[0]");
if(mod)
{
   app = check_and_cast<PingApp *>(mod);
   frame->atim[12] = app->nextPingTime;
}

mod = getSimulation()->getModuleByPath("wireless.host9.pingApp[0]");
if(mod)
{
   app = check_and_cast<PingApp *>(mod);
   frame->atim[13] = app->nextPingTime;
}

mod = getSimulation()->getModuleByPath("wireless.host10.pingApp[0]");
if(mod)
{
   app = check_and_cast<PingApp *>(mod);
   frame->atim[14] = app->nextPingTime;
}
//recieve ping times

mod = getSimulation()->getModuleByPath("wireless.hostReciever.pingApp[0]");
if(mod)
{
   app = check_and_cast<PingApp *>(mod);
   frame->atim[5] = app->recieveNextPingTime;
}
```

```
mod = getSimulation()->getModuleByPath("wireless.hostReciever.pingApp[1]");
if(mod)
{
   app = check_and_cast<PingApp *>(mod);
   frame->atim[6] = app->recieveNextPingTime;
}

mod = getSimulation()->getModuleByPath("wireless.hostReciever.pingApp[2]");
if(mod)
{
   app = check_and_cast<PingApp *>(mod);
   frame->atim[7] = app->recieveNextPingTime;
}

mod = getSimulation()->getModuleByPath("wireless.hostReciever.pingApp[3]");
if(mod)
{
   app = check_and_cast<PingApp *>(mod);
   frame->atim[8] = app->recieveNextPingTime;
}

mod = getSimulation()->getModuleByPath("wireless.hostReciever.pingApp[4]");
if(mod)
{
   app = check_and_cast<PingApp *>(mod);
   frame->atim[9] = app->recieveNextPingTime;
}

mod = getSimulation()->getModuleByPath("wireless.hostReciever.pingApp[5]");
if(mod)
{
   app = check_and_cast<PingApp *>(mod);
   frame->atim[15] = app->recieveNextPingTime;
}

mod = getSimulation()->getModuleByPath("wireless.hostReciever.pingApp[6]");
if(mod)
{
   app = check_and_cast<PingApp *>(mod);
   frame->atim[16] = app->recieveNextPingTime;
}

mod = getSimulation()->getModuleByPath("wireless.hostReciever.pingApp[7]");
if(mod)
{
   app = check_and_cast<PingApp *>(mod);
   frame->atim[17] = app->recieveNextPingTime;
}

mod = getSimulation()->getModuleByPath("wireless.hostReciever.pingApp[8]");
if(mod)
{
   app = check_and_cast<PingApp *>(mod);
   frame->atim[18] = app->recieveNextPingTime;
}
```

```
    mod = getSimulation()->getModuleByPath("wireless.hostReciever.pingApp[9]");
    if(mod)
    {
        app = check_and_cast<PingApp *>(mod);
        frame->atim[19] = app->recieveNextPingTime;
    }

    sendDown(frame);
}

void Ieee80211MgmtAP::handleDataFrame(Ieee80211DataFrame *frame)
{
    // check toDS bit
    if (!frame->getToDS()) {
        // looks like this is not for us - discard
        EV << "Frame is not for us (toDS=false) -- discarding\n";
        delete frame;
        return;
    }

    // handle broadcast/multicast frames
    if (frame->getAddress3().isMulticast()) {
        EV << "Handling multicast frame\n";

        if (isConnectedToHL)
            sendToUpperLayer(frame->dup());

        distributeReceivedDataFrame(frame);
        return;
    }

    // look up destination address in our STA list
    auto it = staList.find(frame->getAddress3());
    if (it == staList.end()) {
        // not our STA -- pass up frame to relayUnit for LAN bridging if we have one
        if (isConnectedToHL) {
            sendToUpperLayer(frame);
        }
        else {
            EV << "Frame's destination address is not in our STA list -- dropping frame\n";
            delete frame;
        }
    }
    else {
        // dest address is our STA, but is it already associated?
        if (it->second.status == ASSOCIATED)
        {
            distributeReceivedDataFrame(frame); // send it out to the destination STA

            std::string name = frame->getName();
            if(name.find("reply") == std::string::npos && frame->active && !frame->packetLoss)
            {
                PingApp *app;
                if(frame->getTransmitterAddress().getInt() == 11725260718081)
                {
```

```
            app = check_and_cast<PingApp *>(getSimulation()-
>getModuleByPath("wireless.hostReciever.pingApp[0]"));
            app->recievedPackageCount++;
            app->delay = app->delay + (frame->getDuration());
        }
        if(frame->getTransmitterAddress().getInt() == 11725260718082)
        {
            app = check_and_cast<PingApp *>(getSimulation()-
>getModuleByPath("wireless.hostReciever.pingApp[1]"));
            app->recievedPackageCount++;
            app->delay = app->delay + (frame->getDuration());
        }
        if(frame->getTransmitterAddress().getInt() == 11725260718083)
        {
            app = check_and_cast<PingApp *>(getSimulation()-
>getModuleByPath("wireless.hostReciever.pingApp[2]"));
            app->recievedPackageCount++;
            app->delay = app->delay + (frame->getDuration());
        }
        if(frame->getTransmitterAddress().getInt() == 11725260718084)
        {
            app = check_and_cast<PingApp *>(getSimulation()-
>getModuleByPath("wireless.hostReciever.pingApp[3]"));
            app->recievedPackageCount++;
            app->delay = app->delay + (frame->getDuration());
        }
        if(frame->getTransmitterAddress().getInt() == 11725260718085)
        {
            app = check_and_cast<PingApp *>(getSimulation()-
>getModuleByPath("wireless.hostReciever.pingApp[4]"));
            app->recievedPackageCount++;
            app->delay = app->delay + (frame->getDuration());
        }
        if(frame->getTransmitterAddress().getInt() == 11725260718087)
        {
            app = check_and_cast<PingApp *>(getSimulation()-
>getModuleByPath("wireless.hostReciever.pingApp[5]"));
            app->recievedPackageCount++;
            app->delay = app->delay + (frame->getDuration());
        }
        if(frame->getTransmitterAddress().getInt() == 11725260718088)
        {
            app = check_and_cast<PingApp *>(getSimulation()-
>getModuleByPath("wireless.hostReciever.pingApp[6]"));
            app->recievedPackageCount++;
            app->delay = app->delay + (frame->getDuration());
        }
        if(frame->getTransmitterAddress().getInt() == 11725260718089)
        {
            app = check_and_cast<PingApp *>(getSimulation()-
>getModuleByPath("wireless.hostReciever.pingApp[7]"));
            app->recievedPackageCount++;
            app->delay = app->delay + (frame->getDuration());
        }
        if(frame->getTransmitterAddress().getInt() == 11725260718090)
        {
```

```
                app = check_and_cast<PingApp *>(getSimulation()->getModuleByPath("wireless.hostReciever.pingApp[8]"));
                app->recievedPackageCount++;
                app->delay = app->delay + (frame->getDuration());
            }
            if(frame->getTransmitterAddress().getInt() == 11725260718091)
            {
                app = check_and_cast<PingApp *>(getSimulation()->getModuleByPath("wireless.hostReciever.pingApp[9]"));
                app->recievedPackageCount++;
                app->delay = app->delay + (frame->getDuration());
            }
        }

    }
    else {
        EV << "Frame's destination STA is not in associated state -- dropping frame\n";
        delete frame;
    }
  }
}
```

11) PingApp.h

```
class INET_API PingApp : public cSimpleModule, public ILifecycle
{
  protected:
    // parameters: for more details, see the corresponding NED parameters' documentation
    L3Address destAddr;
    L3Address srcAddr;
    std::vector<L3Address> destAddresses;
    int packetSize = 0;
    cPar *sendIntervalPar = nullptr;
    cPar *sleepDurationPar = nullptr;
    int hopLimit = 0;
    int count = 0;
    int destAddrIdx = -1;
    simtime_t startTime;
    simtime_t stopTime;
    bool printPing = false;
    bool continuous = false;

    // state
    int pid = 0;    // to determine which hosts are associated with the responses
    cMessage *timer = nullptr;    // to schedule the next Ping request
    NodeStatus *nodeStatus = nullptr;    // lifecycle
    simtime_t lastStart;    // the last time when the app was started (lifecycle)
    long sendSeqNo = 0;    // to match the response with the request that caused the response
    long expectedReplySeqNo = 0;
    simtime_t sendTimeHistory[PING_HISTORY_SIZE];    // times of when the requests were sent

    // statistics
    cStdDev rttStat;
    static simsignal_t rttSignal;
    static simsignal_t numLostSignal;
```

```
    static simsignal_t numOutOfOrderArrivalsSignal;
    static simsignal_t pingTxSeqSignal;
    static simsignal_t pingRxSeqSignal;
    long sentCount = 0;    // number of sent Ping requests
    long lossCount = 0;    // number of lost requests
    long outOfOrderArrivalCount = 0;    // number of responses which arrived too late
    long numPongs = 0;    // number of received Ping requests

  protected:
    virtual void initialize(int stage) override;
    virtual int numInitStages() const override { return NUM_INIT_STAGES; }
    virtual void handleMessage(cMessage *msg) override;
    virtual void finish() override;
    virtual void refreshDisplay() const override;

    virtual void parseDestAddressesPar();
    virtual void startSendingPingRequests();
    virtual void stopSendingPingRequests();
    virtual void scheduleNextPingRequest(simtime_t previous, bool withSleep);
    virtual void cancelNextPingRequest();
    virtual bool isNodeUp();
    virtual bool isEnabled();
    virtual std::vector<L3Address> getAllAddresses();
    virtual void sendPing();
    virtual void processPingResponse(PingPayload *msg);
    virtual void countPingResponse(int bytes, long seqNo, simtime_t rtt);

    virtual bool handleOperationStage(LifecycleOperation *operation, int stage, IDoneCallback *doneCallback)
override;

  public:
    PingApp();
    virtual ~PingApp();
    simtime_t nextPingTime;
    simtime_t recieveNextPingTime;
    int recievedPackageCount = 0;
    simtime_t sleepingTime = SIMTIME_ZERO;
    simtime_t awakeTime = SIMTIME_ZERO;
    simtime_t delay = SIMTIME_ZERO;
    simtime_t delayActual = SIMTIME_ZERO;
};
```

12) PingApp.cc

```
void PingApp::scheduleNextPingRequest(simtime_t previous, bool withSleep)
{
    simtime_t next;
    if (previous < SIMTIME_ZERO)
        next = simTime() <= startTime ? startTime : simTime();
    else {
        next = previous + sendIntervalPar->doubleValue();
        if (withSleep)
            next += sleepDurationPar->doubleValue();
    }
    if (stopTime < SIMTIME_ZERO || next < stopTime)
```

```
        scheduleAt(next, timer);
    nextPingTime = next;
    if(this->getOwner()->getFullPath() == "wireless.hostReciever")
    {
        recieveNextPingTime = next;
    }
}
void PingApp::finish()
{
    if (sendSeqNo == 0) {
        if (printPing)
            EV_DETAIL << getFullPath() << ": No pings sent, skipping recording statistics and printing results.\n";
        return;
    }

    lossCount += sendSeqNo - expectedReplySeqNo;
    // record statistics
    recordScalar("Pings sent", sendSeqNo);
    recordScalar("ping loss rate (%)", 100 * lossCount / (double)sendSeqNo);
    recordScalar("ping out-of-order rate (%)", 100 * outOfOrderArrivalCount / (double)sendSeqNo);

    // print it to stdout as well
    if (printPing) {
        cout << "--------------------------------------------------------" << endl;
        cout << "\t" << getFullPath() << endl;
        cout << "--------------------------------------------------------" << endl;

        cout << "sent: " << sendSeqNo << "   received: " << numPongs << "   loss rate (%): " << (100 * lossCount /
(double)sendSeqNo) << endl;
        cout << "round-trip min/avg/max (ms): " << (rttStat.getMin() * 1000.0) << "/"
             << (rttStat.getMean() * 1000.0) << "/" << (rttStat.getMax() * 1000.0) << endl;
        cout << "stddev (ms): " << (rttStat.getStddev() * 1000.0) << "   variance:" << rttStat.getVariance() << endl;
        cout << "Recieved Package COUNT = " << recievedPackageCount << endl;
        cout << "Awake Time = " << awakeTime << endl;
        cout << "Sleeping Time = " << sleepingTime << endl;
        cout << "Delay = " << delay << endl;
        cout << "Actual Delay = " << delayActual << endl;
        cout << "--------------------------------------------------------" << endl;
    }
}
```