TOA Ranging Using Real Time Application Interface (RTAI) in IEEE 802.11 Networks

Jian Fang[1], Alvin Lim, and Qing Yang[2]

[1] Computer Science and Software Engineering,
Auburn University, Auburn AL 36849 USA
{fangjia,limalvi}@auburn.edu
[2] Department of Computer Science
Montana State University, Bozeman MT 59717 USA
qing.yang@cs.montana.edu

Abstract. Ranging and positioning of wireless mobile devices using time-of-arrivals (TOA) method is becoming an increasingly interesting and challenging research topic. There are various TOA-based ranging algorithms and positioning systems, but most of them require either specially designed hardware or modifications to existing firmware. Using only off-the-self hardware, we present a novel software-based TOA ranging approach which accurately measures TOAs using the Real Time Application Interface (RTAI) operating system. A prototype system is implemented which provides precise measurements of round trip time (RTT) using IEEE 802.11b MAC layer ACK frames and the real-time communication mechanism provided by RTAI. Experiments show that using RTAI can achieve a ranging result with precision close to the accuracy obtained by hardware based methods.

Keywords: IEEE 802.11, Ranging, Localization, RTT, TOA, RTAI.

1 Introduction

In today's fast-paced and technology-centric world, positioning and tracking through mobile devices is extremely useful for applications such as disaster rescue missions, travel guidance systems, fire fighting, and healthcare services. Although GPS positioning system is widely used, it does not work correctly in indoor or metropolitan areas where tall buildings may block GPS signals. On the other hand, Wi-Fi networks are designed mainly for indoor use, so an alternative solution is localizing mobile devices through widely populated IEEE 802.11 wireless networks. In this article, we are interested in the time-of-arrival (TOA) based ranging technique which is the foundational component of IEEE 802.11 localization system.

For TOA-based ranging techniques, accurate round trip time (RTT) measurement is the most important problem. However, it is not an easy task to obtain accurate RTTs because many factors may contribute to errors in RTT measurements such as multipath effect, signal interference, operating system scheduling, and CPU clock throttling. All these factors could introduce variations or jitters to the RTT measurement procedure. This implies that the longer the execution path that the data travels in a system, the more difficult it is to obtain a stable RTT, and the higher the errors that are introduced to the ranging results.

To address this issue, a TOA ranging system needs to measure the RTT as close to the system's bottom (physical) layer as possible. If time stamping cannot be provided at the bottom layers, a deterministic execution of instructions along the data path needs to be guaranteed, which is usually considered a difficult task due to hardware and software latency and jitters [10]. Another important factor affecting the accuracy of RTT measurements is the resolution of system timer. For instance, the programmable interval timer (PIT) on most Intel x86 CPUs has a coarse resolution of $1ms$ (1000Hz) which corresponds to a computed distance of more than $30km$. Therefore, to achieve a viable TOA ranging system, three key factors must be taken into account: 1) short data path, 2) deterministic execution of instructions, and 3) high-resolution timer.

To satisfy the three requirements for accurate RTT measurements, Real Time Application Interface (RTAI) [1] is selected in our implementation. By taking full control of hardware and task scheduling, RTAI can achieve a much faster response to outside events, e.g., interrupts of network activities, than general purpose operating systems. Therefore, more accurate time stamps can be recorded when a packet is sent or received by network devices, which is essential for obtaining precise RTT measurements.

There are mainly four contributions of this paper. First, the variation of RTT measurements caused by the processing delays at system and process levels is completely investigated. Second, we design and implement an accurate TOA based ranging system using RTAI for standard IEEE 802.11 networks. Third, a prototype system is implemented and tested using off-the-shelf wireless cards. From experimental results, we find the proposed system can achieve a precise ranging between two wireless devices with an error $< 5m$ in indoor and an error $< 8m$ in outdoor scenarios. Fourth, since the RTAI based ranging system does not require specific hardware, it can be easily installed (as kernel modules) on existing IEEE 802.11 network devices. In fact, the same methodology can be utilized in other wireless networks, e.g., sensor network and vehicular networks.

The rest of the paper is organized as follows: Section 2 presents methods that are related to our work. Section 3 analyzes the task latency at the system and process levels, and then describes the RTT ranging method in detail. Section 4 and 5 give the setup of our experiments and result analysis, respectively. Finally, conclusions are provided in Section 7.

2 Related Works

Since different applications may require different location accuracy or positioning precision, various wireless ranging methods are used by positioning and tracking systems. These techniques can be classified along different dimensions, such as hardware or software-based, time or signal based [2,3,4]. Besides the widely used

GPS system which does not work well for indoor environments, there exist several other positioning techniques such as systems based on infrared, ultrasonic, or received signal strength (RSS) [2]. Although they can overcome the shortcoming of GPS systems, these techniques are not widely adopted either because of their reliance on special hardware, offline trained radio maps, or complex design.

Another group of ranging methods are time-based techniques such as the time of arrival (TOA) and time difference of arrival (TDOA) ranging. TDOA ranging can provide accurate results but require perfect synchronization of APs, which adds a great amount of complexity to the positioning system. TOA is similar to TDOA but does not require synchronization. Synchronization error between two nodes can be eliminated by measuring the RTTs of network packets. For example, the RTS/CTS (request-to-send and clear-to-send) packets in MAC layer are used to measure RTTs in [5]. Using phase matching and shifting of received signals, [6] reports a ranging precision of less than $5m$. In [7], a debug version of Intel ABG WLAN card and an external FPGA card are used for accurately time stamping transmit and receive signals. All the above-mentioned systems achieve high precision ranging results but require auxiliary hardware or modifications to existing wireless devices' firmware.

Unlike those hardware-based method, a software based TOA ranging method is presented in [8]. It achieves an indoor ranging error of a few meters by modifying the driver of existing wireless devices. Our work is different from [8] because we use a RTAI extended system which largely minimizes the time variations in RTT measurements. Moreover, in the proposed system, the MAC layer's status information can be easily obtained from the application's kernel modules or user space using services provided by RTAI. Thus, it is easier to develop real-time localization systems with our approach.

3 Design and Implementation

RTT of a packet can be measured on different levels, e.g., hardware, firmware, driver, OS and application. Among these available approaches, measuring TOA in the software layer may be the most feasible choice because software can be more easily updated than hardware.

3.1 Latency at System Level

Time stamping a packet can be done on different levels, e.g., in the user space or at the driver layer. However, theoretically, a precise and accurate ranging can only be achieved by measuring the time when a data packet leaves or enters the transmitter/air boundary. Unfortunately, it is impossible to accomplish this task by a pure software based approach. For instance, the SoftTOA [8] measure RTT of network packets in the network driver layer which is above the transmitter/air boundary. That means the obtained RTT between the sender and receiver is:

$$RTT = t_{tx\_proc} + 2 \times TOA + t_{rx\_proc} \tag{1}$$

where t_{tx_proc} is the time for the sender to transmit a packet and process the corresponding ACK message. t_{rx_proc} is the time for the receiver to process the received data packet, generate and transmit an ACK message. Equation 1 implies the ranging precision of RTT depends on the precise measurements of t_{tx_proc} and t_{rx_proc}, which are determined by the hardware and software used in the ranging system.

The hardware and software of a system affect the measurements of t_{tx_proc} and t_{rx_proc} in two ways: latency and jitter. Generally, hardware latency determines the system response time but may not affect the precision of RTT, e.g., a precise RTT can be obtained by hardware with longer but constant latency. However, jitter of latency is detrimental to precise RTT measurement. Thus, TOA ranging techniques need to focus on reducing jitters.

Generally, time jitters in the hardware are small, e.g., the time is relatively stable for a NIC (network interface controller) transmitting a fixed number of bits. However, large jitters exist in software due to the internal non-deterministic execution of instructions in a general-purpose OS. For instance, the RTT measurement process may be preempted by other process with a higher priority. To eliminate these jitters, a software-based ranging method needs to timestamp the data as close to the physical layer as possible.

3.2 Latency at Process Level

To obtain a precise RTT measurement, it is important to identify the points (in a process) where time stamps should be put. As shown in Fig. 1, a typical process (e.g., the RTT measuring process) is initiated within the user space. Then, to transmit a packet, it may enter the kernel space which may be later interrupted by a hardware signal. Obviously, measuring RTT at the interrupt level will give the best measurement accuracy because it is the level closest to the hardware.

In IEEE 802.11 networks, when a device receives a packet, the MAC layer of its NIC will generate an interrupt which notifies the OS that a packet matching its MAC address is received. This time instance in which the interrupt is generated should be a perfect time stamping point since it is the closest to the physical layer. However, the CPU in a non-real-time OS may not be able to respond to this interrupt immediately, so an accurate time stamping cannot be achieved.

For example, Task 1 in Fig. 1 may be executing in a critical region where an interrupt (of receiving a packet) is generated at time t_3. The interrupt can be responded only after t_4 or after Task 1 exits the critical region. In other words, a ranging process can timestamp this event of receiving a packet only after t_4. $t_4 - t_3$ is the latency for interrupt handling and it will introduce jitters into the measured RTT. Even worse, the time length of $t_4 - t_3$ is non-deterministic, i.e., it depends on the running process, the scheduler, and the resolution of the timer. Therefore, at the process level, a precise ranging system requires a fully preemptable kernel if time stamping needs to be provided in the upper layers of the system.

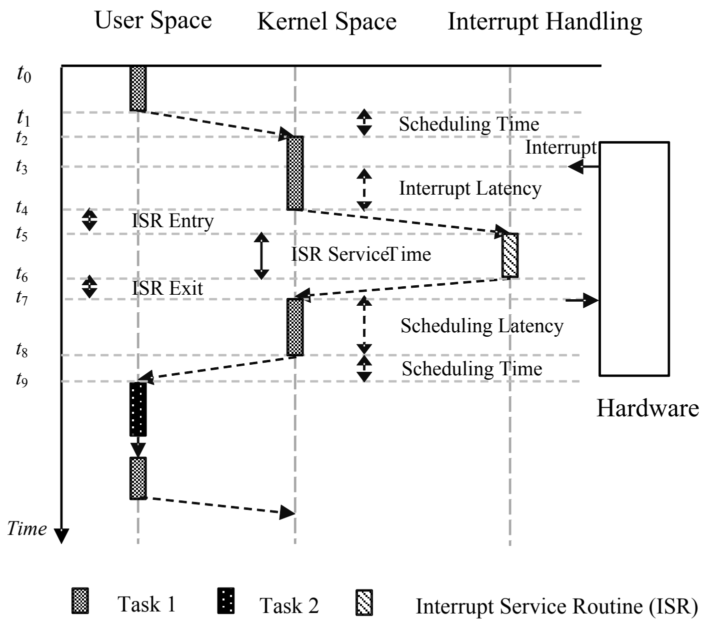


Fig. 1. Task response time and latency in general purpose Linux system

3.3 RTAI Based Ranging System

Taking into account the latency and jitter described above, we select RTAI as the operating system for our TOA ranging system. RTAI is a high performance real-time extension to general-purpose Linux. It achieves real-time task executions by implementing a real-time hardware abstraction layer (RTHAL) on which the real-time application interface is mounted. With this sub-layer, RTAI takes over the system hardware management completely.

Once the RTAI modules are loaded, all hardware interrupts will be intercepted and dispatched by the RTAI and the scheduler in the general-purpose Linux core is also taken over by RTAI's real-time scheduler, which provides simultaneous one-shot and periodic scheduling. RTAI can provide scheduling with a much higher precision than general-purpose Linux because it can schedule tasks based on time stamp counter (TSC) readings. Thus, the theoretical ranging resolution in meters a RTAI sytem can achieve is:

$$d_r = \frac{c}{TSC_{freq}} \tag{2}$$

where d_r is the theoretical minimum distance a RTAI system can measure although the ranging resolution achieved in practice depends on multiple factors, such as channel conditions and hardware response time, c is the speed of light and TSC_{freq} is the TSC frequency which is 1.2G Hz in our ranging system. Besides task scheduling with timers of finer resolutions, RTAI has more advantages as a ranging system over general purpose operating systems. Generally, a RTAI

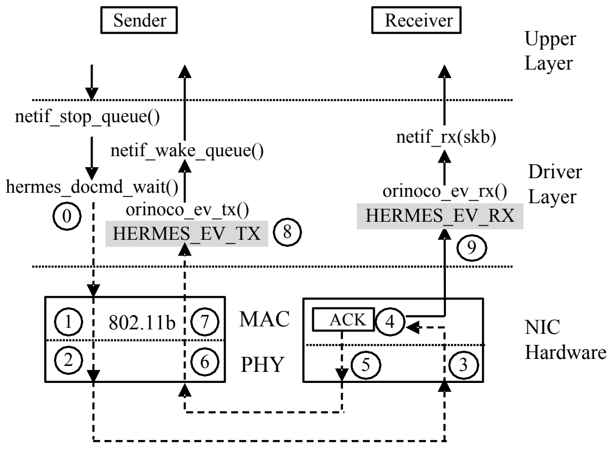


Fig. 2. Tx and Rx path in the WLAN card and driver layer

application is a single-threaded process with a fixed priority. The RTAI thread can be assigned a high priority and run to completion without being preempted. Thus, in comparison with general-purpose operating systems, it can greatly reduce or eliminate the time $t_4 - t_3$ and $t_8 - t_7$ in Fig. 1. Implementation details of RTAI can be found in [9].

Taking into account the above discussion on latency and jitter, the stamping points for RTT in our ranging system are shown in Fig. 2. For easy explanation, we use the same function names as the driver code of our RTT measurement system, which uses an IEEE 802.11b Orinoco Gold card with a Hermes chipset. Drivers for other cards may use different function names, but the same methodology can be used.

Fig. 2 shows the shortest round trip path of a data packet in a TOA ranging system. At the sender, when the driver transmits a packet, it tells the upper layer to stop feeding packet by calling *netif_stop_queue()*. Then, it calls *hermes_docmd_wait()* to transfer data to the MAC layer. In IEEE 802.11 systems, a data frame from the sender requires an ACK frame to be returned to the sender. When the packet is received at the receiver, an ACK frame is assembled and sent to the sender after one SIFS. This ACK is generated by the MAC firmware and is not passed to the upper layer. The MAC layer at the sender raises the *HERMES_EV_TX* interrupt only after it receives this ACK. Thus, the shortest data path for RTT measurement is 0 through 8 (the path with dashed lines in Fig. 2). The receive interrupt, *HERMES_EV_RX* (9 in Fig. 2), is generated only after the data packet is copied to the socket buffer.

To be precise, RTT in our RTAI-based ranging system starts at the time when *hermes_docmd_wait()* is completed; it ends at the time when the interrupt *HERMES_EV_TX* is raised. Hence, Equation 1 can be rewritten as:

$$\begin{aligned} RTT =& t_{tx\_data\_trans} + 2 \times TOA + t_{rx\_ACK\_proc} \\ &+ SIFS + t_{rx\_ACK\_trans} + t_{tx\_ACK\_proc} \end{aligned} \quad (3)$$

where $t_{tx\_data\_trans}$ is the time to transmit a data frame including the preamble, frame header, data payload, and frame extension. $t_{tx\_ACK\_proc}$ and $t_{rx\_ACK\_proc}$ are the time for processing the ACK at the sender and receiver, respectively. $t_{rx\_ACK\_trans}$ is the time to transmit the ACK frame by the receiver.

Equation 3 not only gives the lower bound of RTT, but also identifies where jitters are introduced into ranging results. Particularly, jitters come from the interrupt handling $(0, 8)$, the MAC logic execution $(1, 7, 4)$, and transmitting and receiving $(2, 3, 5, 6)$. These jitters will be eliminated or alleviated by specific data processing algorithms which will be introduced in latter sections.

4 Experiment Setting

A prototype system implementing the ranging logic above is developed on two Dell laptops with a $1.2GHz$ Pentium III mobile processor with no CPU scaling capability. The communication between the sender and receiver is set in ad hoc mode in order to control the jitters at the receiver. The WLAN card used is the Orinoco Gold PCMCIA card which has been set to transmit with the maximum data rate of $11Mbps$. The software used is a vanilla Linux 2.6.23 kernel patched with RTAI 3.6. The test program is implemented in kernel space as kernel modules. RTT data is collected as CPU ticks from the kernel printing buffer. In the data analysis phase, the CPU ticks are converted to nanoseconds according to the following equation:

$$t_{ns} = \frac{Ticks}{TSC_{freq}} = \frac{Ticks}{1.2} \tag{4}$$

where $Ticks$ is time stamp value read from the TSC and TSC_{freq} is the frequency of the TSC in gigahertz.

To verify the performance of the ranging system, tests are conducted both indoor and outdoor. The indoor tests are conducted in a straight aisle inside a building about $3m$ wide. In the outdoor tests, the shortest distance from the test devices to the surrounding buildings is about $10m$. In both situations, the system is placed about 1.5 meters above the ground to preserve the Fresnel zone. Test is repeated with a LOS distance of $0, 15, 30, 45, 60, 75$ feet between the sender and receiver, respectively. In each test, 1000 RTT samples were collected. The RTT collected at distance $0ft$, denoted as RTT_0 is considered as the processing overhead in the ranging system, which corresponds to the summation of the right hand side terms, except for the term $2 \times TOA$, in Equation 3. The distance between the sender and receiver is then computed as:

$$d_i = c \times TOA = c \times \frac{RTT_{d_i} - RTT_0}{2} \tag{5}$$

where RTT_{d_i} is the RTT obtained with distance d_i.

5 Result Analysis

5.1 Data Filtering

Even with time stamping being implemented close to the physical layer, jitters could still be found in the RTT measurement results due to various reasons such as hardware noise and multipath affects. However, with the RTAI ranging system, the data samples, after filtering, are concentrated into a narrow time band of about 10000*ns* for both indoor and outdoor situations.

To remove data outliers, the data is filtered in two steps. The first step is to find the distribution of the data samples. Samples collected are distributed into a series of buckets within a fixed interval (from 95000 ticks to 125000 ticks); any sample falling outside this interval are discarded. The filtered RTT data is shown in Fig. 3.

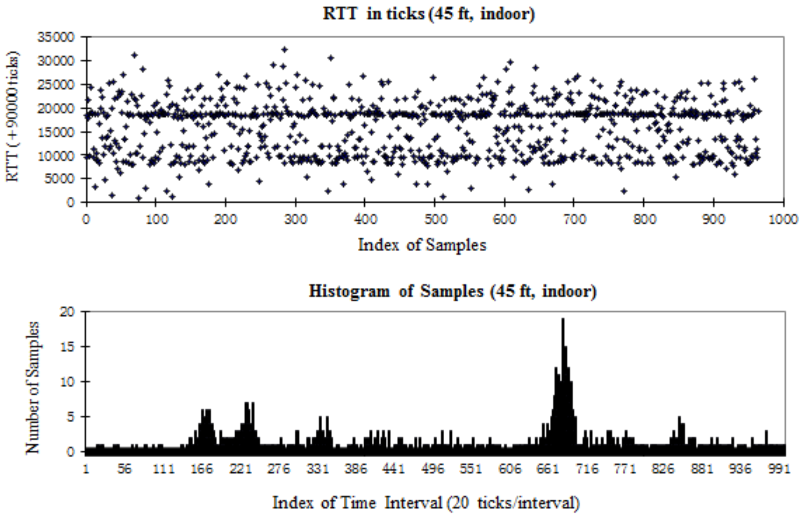


Fig. 3. RTT in ticks at $45ft$ indoor test

Two characteristics can be observed from Fig. 3. First, RTT samples generally follow a *Normal* or a *Gaussian* distribution around the peak value. Thus, statistical data filtering algorithms for *Normal* distribution data can be used for further analysis. Second, several sample clusters exist. We conjecture that the first small clusters on the left are caused by multipath effect.

The second step in data filtering uses a smaller filter window to find the data samples for the final RTT calculation, which are shown as the samples falling within the intervals surrounding the peak interval 683 in Fig. 3. The size of the filter window is experimentally determined. Windows of different sizes are slided around the peak interval and the RTTs that are calculated based on samples within this

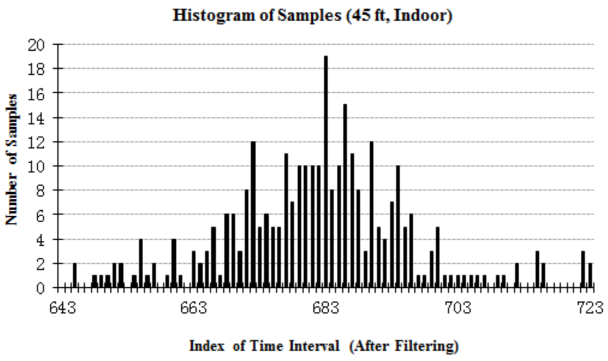


Fig. 4. RTT intervals filtered by sliding window

window are compared. The window size resulting in the smallest error of RTTs are used. For our experiments, a window size of 81 is finally selected (40 intervals to each side of the peak interval). The intervals in Fig. 3 that are filtered this way are shown in Fig. 4, where the peak interval index is 683 and the window boundary intervals are 643 and 723 (±40 intervals around the peak interval).

5.2 Data Analysis

After the RTT sample intervals in the histogram are identified, statistical methods are applied to the samples falling within these intervals. RTT is computed as follows: First, the mean and standard deviation σ are computed for samples falling within the window:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2} \tag{6}$$

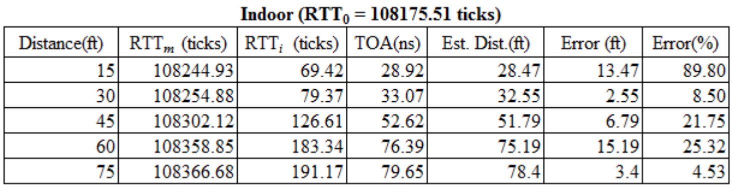
where x_i is the RTT value of sample i, and μ is the mean of all samples within the selected intervals. Estimators of RTT are computed as $\mu \pm \sigma$, $\mu \pm 2\sigma$ and $\mu \pm 3\sigma$, respectively.

Our Experiments show that simply using the mean value as the estimator of RTT has slightly smaller errors. The table in Fig. 5 shows the results of both indoor and outdoor tests using the mean of RTTs:

In the table, RTT_m is the RTT measured in the test, RTT_i is the estimated RTT value for distance i, which is the estimation of the term $2 \times TOA$ in Equation 3. TOA(ns) is computed by Equation 4, and errors are computed as:

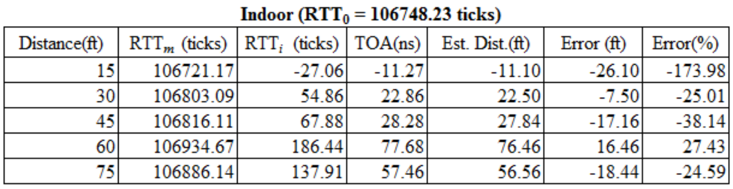
$$Err = c\mu/2 - d \tag{7}$$

where d is the actual distance being measured and c is the speed of light. The table in Fig. 5 shows our RTAI ranging system achieves a result with an error less than $15.19ft$ ($4.63m$) for indoor ranging and $26.1ft$ ($7.95m$) for outdoor ranging. The ranging results compared with the true distances are plotted in Fig. 6.



Indoor (RTT_0 = 108175.51 ticks)

Distance(ft)	RTT_m (ticks)	RTT_i (ticks)	TOA(ns)	Est. Dist.(ft)	Error (ft)	Error(%)
15	108244.93	69.42	28.92	28.47	13.47	89.80
30	108254.88	79.37	33.07	32.55	2.55	8.50
45	108302.12	126.61	52.62	51.79	6.79	21.75
60	108358.85	183.34	76.39	75.19	15.19	25.32
75	108366.68	191.17	79.65	78.4	3.4	4.53



Indoor (RTT_0 = 106748.23 ticks)

Distance(ft)	RTT_m (ticks)	RTT_i (ticks)	TOA(ns)	Est. Dist.(ft)	Error (ft)	Error(%)
15	106721.17	-27.06	-11.27	-11.10	-26.10	-173.98
30	106803.09	54.86	22.86	22.50	-7.50	-25.01
45	106816.11	67.88	28.28	27.84	-17.16	-38.14
60	106934.67	186.44	77.68	76.46	16.46	27.43
75	106886.14	137.91	57.46	56.56	-18.44	-24.59

Fig. 5. Estimated distance for indoor and outdoor experiments

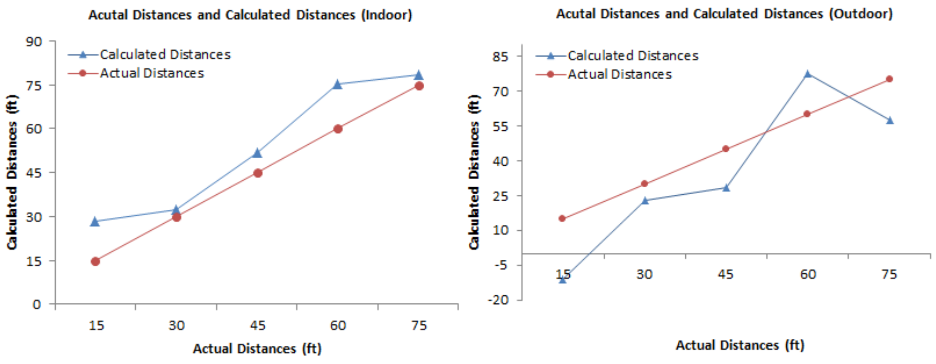


Fig. 6. Distances measured and actual distances for indoor and outdoor experiments

6 Discussion

In our test, RTAI is implemented at both the sender and receiver while the communication mode is set in ad hoc mode. In this configuration, data processing time at the receiver can be well controlled. However, in real world applications, a sender will more likely communicate with an AP. The response of AP depends on many factors that the sender cannot control such as workload change. This may introduce more errors to the ranging result than our configuration. Besides, our experiments are done in a limited number of scenarios, to make this system viable in general application environments, tests in more rigorous situations are needed, such as under different workloads both at the sender and receiver, or under severe channel interference situations. We will conduct these tests in our future work.

7 Conclusion

In this paper, a novel TOA ranging approach using real-time system RTAI is presented. The implemented system is software-based without any special hardware support or modifications to system firmware. RTAI provides real-time guarantees for task executions and has faster response to hardware interrupt than general purpose Linux system. Our experimental results show, with the RTAI real-time system, errors less than $(4.63m)$ for indoor ranging and errors less than $(7.95m)$ for outdoor ranging can be achieved. Our work demonstrated that by exploiting data packet transmission and receive at the instruction execution level at bottom layers of the system, a software based system can achieve good ranging accuracy. Our future work will test the system under more rigorous application conditions.

References

1. Dozio, L., Mantegazza, P.: Real time distributed control systems using RTAI. In: Sixth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, pp. 11–18 (May 2003)
2. Koyuncu, H., Yang, S.H.: A survey of indoor positioning and object locating system. International Journal of Computer Science and Networks Security 10(5), 121–128 (2010)
3. Liu, H., Darabi, H., Banerjee, P., Liu, J.: Survey of wireless indoor positioning techniques and systems. IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews 37(6), 1067–1080 (2007)
4. Bill, R., Cap, C., Kofahl, M., Mundt, T.: Indoor and outdoor positioning in mobile environments – a review and some investigation on WLAN positioning. Geographic Information Sciences 10(2), 91–98 (2004)
5. Izquierdo, F., Ciurana, M., Barcelo, F., Paradells, J., Zola, E.: Performance evaluation of a TOA-based trilateration method to locate terminals in WLAN. In: 1st International Symposium on Wireless Pervasive Computing, pp. 1–6 (January 2006)
6. Karalar, T., Rabaey, J.: An RF tof based ranging implementation for sensor networks. In: IEEE International Conference on Communications, vol. 7, pp. 3347–3352 (June 2006)
7. Golden, S., Bateman, S.: Sensor measurements for Wi-Fi location with emphasis on time-of-arrival ranging. IEEE Transactions on Mobile Computing 6(10), 1185–1198 (2007)
8. Ciurana, M., López, D., Barceló-Arroyo, F.: SofTOA: Software Ranging for TOA-Based Positioning of WLAN Terminals. In: Choudhury, T., Quigley, A., Strang, T., Suginuma, K. (eds.) LoCA 2009. LNCS, vol. 5561, pp. 207–221. Springer, Heidelberg (2009)
9. `http://www.aero.polimi.it/~rtai`
10. Muthukrishnan, K., Koprinkov, G., Meratina, N., Lijding, M.: Using time-of-flight for WLAN localization: feasibility study. Center for Telematics and Information Technology (WLAN) technical report, TR-CTIT-06-28 (June 2006)