CSE 237B Final Project
Google Go Programming Language and Real-Time Applications

Hao Zhuang, hao.zhuang@cs.ucsd.edu, CSE Dept., UC San Diego
Xinan Wang, xinan@ucsd.edu, CSE Dept., UC San Diego
June 11, 2015

1 Introduction

In this project, we implement XBee transportation among BeagleBone Black (BBB) devices. First, we implement the communication. Then, we create environment to mimic a heavy communication traffic. We anticipate the overhead from timer implemented by pure Go programming language. Then, we improve the timer by using lower-level C programming language, which is more closed to hardware-level.

2 Settings

2.1 Setting A

We have two BeagleBone Black (BBB) [1] devices with XBee Xsticks on each.

- One of machine is set as Coordinate AT (CAT), the other is END Device AT(EAT).
- Use xctu [2] from Digi to set the Xstick. The baud rate is 9600.
- Use “stty -F /dev/ttyUSB0 speed 9600 cs8 -cstopb crtscts -parenb raw ” at each device for the serial communication in Linux at BBB.
- The communication programs are implemented in Go language [3].
- The timer directly utilizes Go’s Time library [4].

2.2 Setting B

Build on the Setting A, we further create a busy distributed application. The hardware level timer implementation is done using C API [5] [6] and invoked by Go. The main functions we call as follows,

- Set timer "timerfd_settime(timerfd, 0, &new_value, NULL) == -1"
- Create timer "timerfd_create(CLOCK_MONOTONIC, 0)";
- Invoke from sleeping mode by "read(timerfd, &exp, sizeof(uint64_t))"

We assume

3 Implementation of communication program and the benchmark for characterization of time latency/jitter

In this work, we profile the latency/jitter from stack of Xbee (ZigBee) communication stack up to Golang Runtime. I also devise a tiny benchmark for Go's timer in order to capture more accurate result by removing its overhead on the top of OS.

3.1 Design

The flow of my benchmark is listed as follow:

- Coordinate AT (CAT): **xbeeTimingRecvCor.go**, **xbeeTimingSend.go** and **Go's serial package** [7], which are under CoordinateAT in my tar package.
- END Device AT (EAT): **xbeeTimingRecv.go** and **Go's serial package** [7], which are under EndDeviceAT in my tar package.

Flow of benchmark

1. First, launch **xbeeTimingRecv.go** at EAT, which waits for the time stamp sent from CAT (After **Item 3**, once it receives the time stamp, it immediately returns to CAT).
2. Then, Launch **xbeeTimingRecvCor.go** at CAT. (It receives the time stamp back from **Item 1**, subtracts it from current time and obtain the round trip time number (RTT). It keeps receiving the data to profile the delay/jitter number.)
3. Launch **xbeeTimingSend.go** at CAT to send the time stamps every 15 seconds.

The code can be found in the submitted tar package correspondingly.

3.2 Data and Observations

The time latency contains from Xbee (ZigBee) communication stack up to Golang runtime performance. Table 1 shows the data obtained by the benchmark.

Based on the limited data set, the **average delay** is $0.299258s$. The **maximum delay** $0.315851s$ and the **minimum delay** $0.280589s$. The jitter is $0.035262s$. The statistics is shown in Table 1.

3.3 Tiny Benchmark for the characterization of Golang timer overhead on the top of the OS-level

The procedure is:

At each iteration, I issue two timer using Go's time.Now, between the two timer, there is $5s$ sleep by Go's time.Sleep. I repeat this process by 10 times and obtain $50008918967ns$.

Table 1: Characterizations of Latency/Jitter of Xbee (ZigBee) communication and Golang Runtime (the time stamps are from the system date of BeagleBone Black **Coordinate AT**).

Test	TimeStamp Sent	TimeStamp Received	RTT(μs)	delay (μs)
1	23:APR:2014:23:16:07:720864	23:APR:2014:23:16:08:342818	621954	310977.0
2	23:APR:2014:23:16:22:728927	23:APR:2014:23:16:23:312285	583358	291679.0
3	23:APR:2014:23:16:37:732581	23:APR:2014:23:16:38:346838	614257	307128.5
4	23:APR:2014:23:16:52:737204	23:APR:2014:23:16:53:303785	566581	283290.5
5	23:APR:2014:23:17:07:742727	23:APR:2014:23:17:08:374429	631702	315851.0
6	23:APR:2014:23:17:22:747460	23:APR:2014:23:17:23:318574	571114	285557.0
7	23:APR:2014:23:17:37:750225	23:APR:2014:23:17:38:374054	623829	311914.5
8	23:APR:2014:23:17:52:753309	23:APR:2014:23:17:53:314487	561178	280589.0
9	23:APR:2014:23:18:07:765073	23:APR:2014:23:18:08:377745	612672	306336.0

Table 2: The statistics of Table 1

Average Delay	0.299258s
Jitter	0.035262s
Maximum Delay	0.315851s
Minimum Delay	0.280589s

I subtract 50s (10 $\times$ 5s) from 50.008918967s, and divide it by 10. The result 0.0008918967s is the overhead by using Golang timer OS-level.

3.4 More accurate Delay/Jitter number without Golang timer overhead

From Sec. 3.3, we can have more accurate profiles about the Xbee communication stacks on BBB up to the Golang runtime.

Since I send the time stamp after the timer at CAT side, and invoke another timer after receiving time stamp, there are two units of the overhead there. To be more accurate, I need to subtract the delay by 0.001784s ($\approx$ 2 $\times$ 0.0008918967s). Therefore, We have another Table 3 to calculate the delay/jitter by removing the timer overhead in Go. (The benchmark is designed in **golang_timer_overhead.go** in my tar package).

Table 3: The statistics of Table 1 without Golang Timer Overhead

Average Delay	0.297474s
Jitter	0.035262s
Maximum Delay	0.314067s
Minimum Delay	0.278805s

3.5 Other Founding

Use cat “/dev/ttyUSB0” and “echo "XXX" > /dev/ttyUSB0” to communicate via XBee, where XXX is any string.

- The data sent from End Device AT to Coordinate AT is good.
- The data sent from Coordinate AT to End Device AT often loses message. Or Coordinate AT not response at the side of Coordinate AT.

This leads to the workaround in this report. I use a running pool at both of CAT and EAT and another program to inject the time stamp. In this way, the messages are pretty reliable and almost never miss a byte.

4 Summary

I implement the message communication programs via Go in BeagleBone Black boards with Xbee (Xstick). All the programs, including the public serial package of Go [7], are included in the attachment. The programs can be simply copied to two configured BeagleBone Black boards correspondingly, and launched to communicate as discussed in Sec. 3.1. The benchmarks profile the round trip time, delay and jitter when the two BeagleBone Black boards communicated via Xbee. I think the above report and code can achieve the main goal of Lab 2. I can do more profiling when there is enough time.

References

[1] “Beaglebone black.” `http://beagleboard.org/BLACK`.

[2] Digi, “XCTU.” `http://knowledge.digi.com/articles/Knowledge\_Base\_Article/X-CTU-XCTU-software`.

[3] “Go language.” `http://golang.org/`.

[4] “Time package in go language.” `http://golang.org/pkg/time/`.

[5] Z. Fang, “Private communication,” 2014.

[6] Z. Fang, “ligcgo,” 2014.

[7] “Go serial package.” `https://github.com/tarm/serial/`.