Coordinating Ping Events between Raspberry Pis using Ethernet

Michael Phillips: michael.phillips@progeny.net

October, 9th 2019

Abstract

This experiment consisted of two raspberry pi computers communicating over ethernet through a network switch. The raspberry pies are intended to model the processor and transmitter systems within the sonar array assembly. The ethernet-switch-ethernet connection represents a potential upgrade to the current discrete communication path between these systems. The results show that for two raspberry pies running the precision time protocol daemon (ptpd) in steady-state over the ethernet link, clock synchronization remains on the order of 10's of milliseconds, and the ethernet-scheduled ping transmission over 1000 transmit cycles never exceeded 5ms in timing offset from the allotted transmit time. Even though the offset limit was never exceeded in this experiment, the probability of higher priority processes occurring within the raspberry pi operating system is not zero, preventing 100% ping timing reliability for this setup.

Introduction

The overall goal was to see if two raspberry pi's without deterministic real-time capabilities, running only ptp could communicate over ethernet and make a simple transmission ping event consistently occur within 5ms of the planned event time.

The setup consists of a three raspberry pi computers: a processor, transmitter, and listener. The processor and transmitter are in communication over an *ethernet--switch--ethernet* link and the listener is discretely linked to both for data collection. The section which is intended to mimic the sonar array assembly is the *processor--ethernet--switch--ethernet--transmitter* section. Ptpd is running on the processor and the transmitter, using the ethernet connection to synchronize the clocks. Then with clocks in step, a transmit message is sent over the ethernet line from the processor to the transmitter containing the processor's current time plus 2.5ms. The transmitter receives the message and wait for its clock time to be greater than or equal to the received time. When this happens the transmitter calls the transmit subroutine. At the same time the processor is running the same program, it also calls the same transmit subroutine. These transmit subroutines set a discrete pin on the raspberry pi's GPIO pin-out to high, which lights up an LED (blue = transmitter, yellow = processor). These discrete voltages are fed into the listener raspberry pi GPIO pin-out. The listener pi is monitoring the GPIO pins with an interrupt program. When it detects a rising edge on one of the pins a timestamp is created (I have not considered the sample rate or the timestamp accuracy of the interrupt program, and from the results we know that it is not significant enough for us to worry about). The various timestamps are then fed into an events.txt file and analyzed by another plotter program. The plotter program plots the offset magnitude of each ping event, plots a histogram of all the offsets, calculates the average offset, calculates the standard deviation of the offset, and plots all the messages in each ping event's timeline. It was found that if ptpd is running in steady state and the CPU isn't experiencing a heavy workload, this method of making a ping event occur within 5ms of an allotted time is reliable 100% of the time. One major problem was keeping ptpd running in steady state longer than one hour.

The primary concern with regard to the obsolescence upgrade is the reliability of the ethernet linkage. For instance, the pre-ping-trigger (ppt) signal, which originates in the processor, must reach the transmitter and cause a ping transmission within five milliseconds to ensure consistent homing system performance. For the proposed ethernet linkage upgrade, if the ppt-signal is deemed low priority by one of the CPUs it may not arrive within the required time slot, resulting in poor homing signal performance. It has been suggested that this risk can be mitigated using a contractual-agreement ethernet protocol (IEEE 802.11as).

Another concern, which is was the focus of this experiment, is the clock synchronization between the processor and the transmitter. This becomes a concern because in the transformation of the ppt-signal from discrete to ethernet form, the nature of the signal is inherently changed. In the discrete case, the signal is simply a discrete voltage and the transmitter decides whether to ping by measuring the electric potential between the line and a reference (a deterministic signal,

not an encoded message). Whereas in the ethernet case, the signal is actually a message containing binary information that has to be interpreted by a CPU before a ping transmission can be performed. In order to make the ping transmission occur at the time intended by the processor, the ethernet lag time and CPU processing times must be accounted for. The simplest way to do this is by synchronizing the clocks of the transmitter and processor. Once synced, the transmit event can be scheduled to occur at a future time (less than 5ms away). So in this method the ppt-signal takes the form of a binary ethernet message containing the future time for the ping transmission to occur. The transmitter then just waits for its clock to reach the scheduled time to call the transmission subroutine. For this particular setup the raspberry pies are running a Linux operating system, so *deterministic real-time operation* isn't available and the risk of larger than 5ms variations in processing times is real and does impact the reliability of the ping signal, however, this risk can be mitigated in the real system by using a dedicated real-time microcontroller, for instance.

Experimental Methods

The setup in Figure 1 shows the various Ethernet communication links within the sonar assembly. The raspberry pi setup shown in Figure 2 and Figure 3 was based upon this.

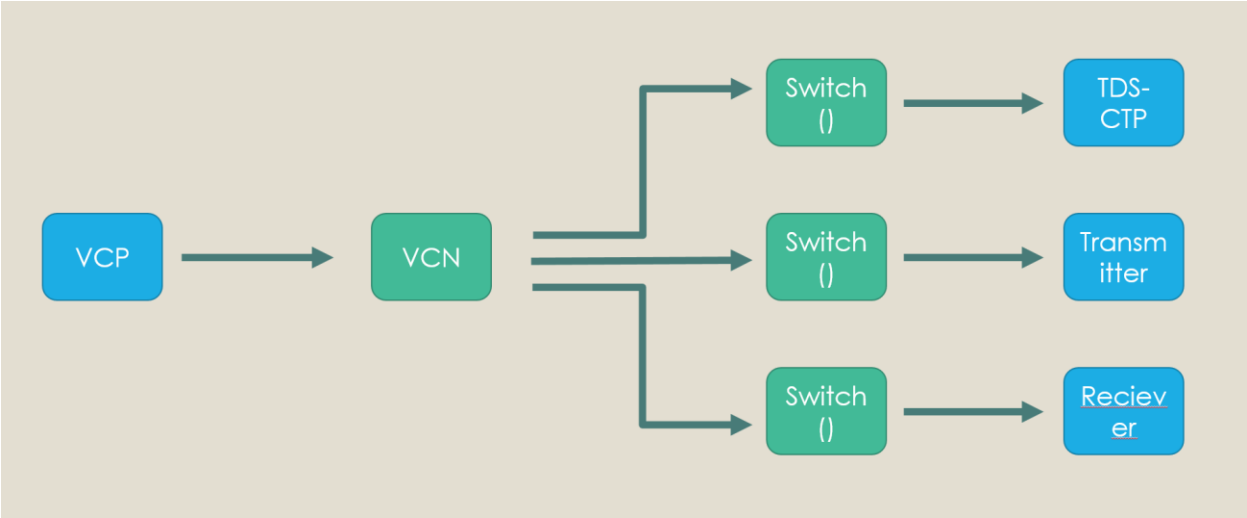

Figure 1: Sonar Assembly Ethernet Linkage Diagram

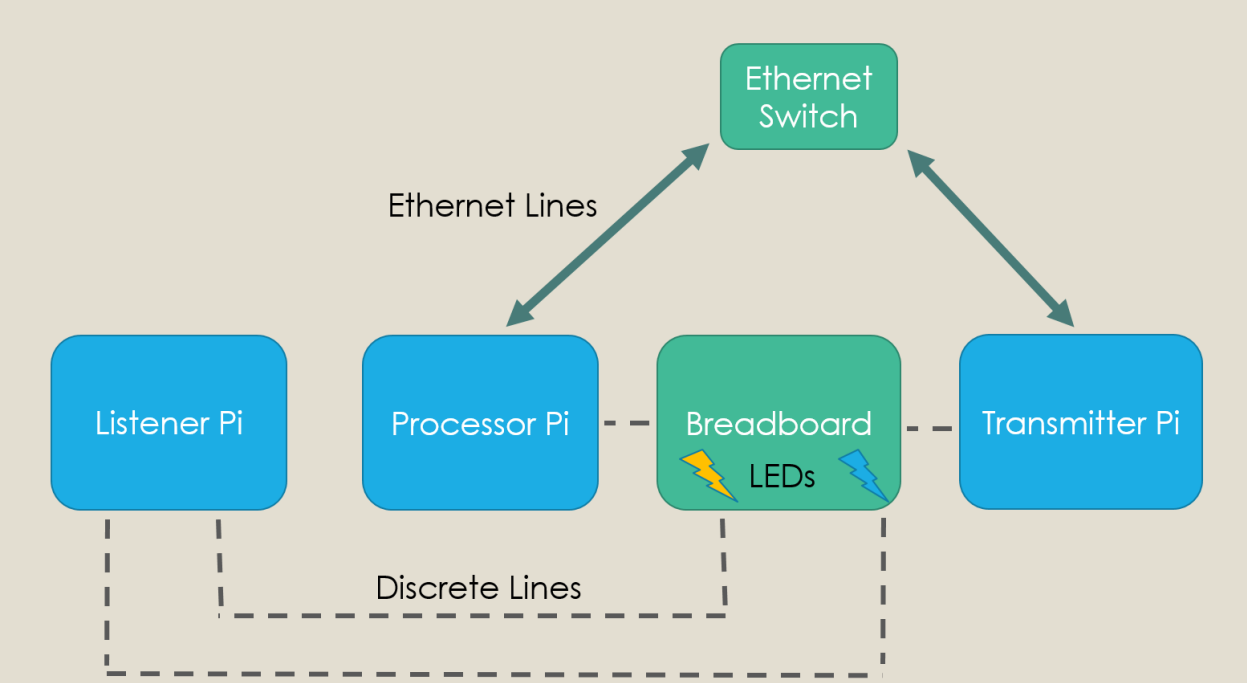

Figure 2: Experiment Setup

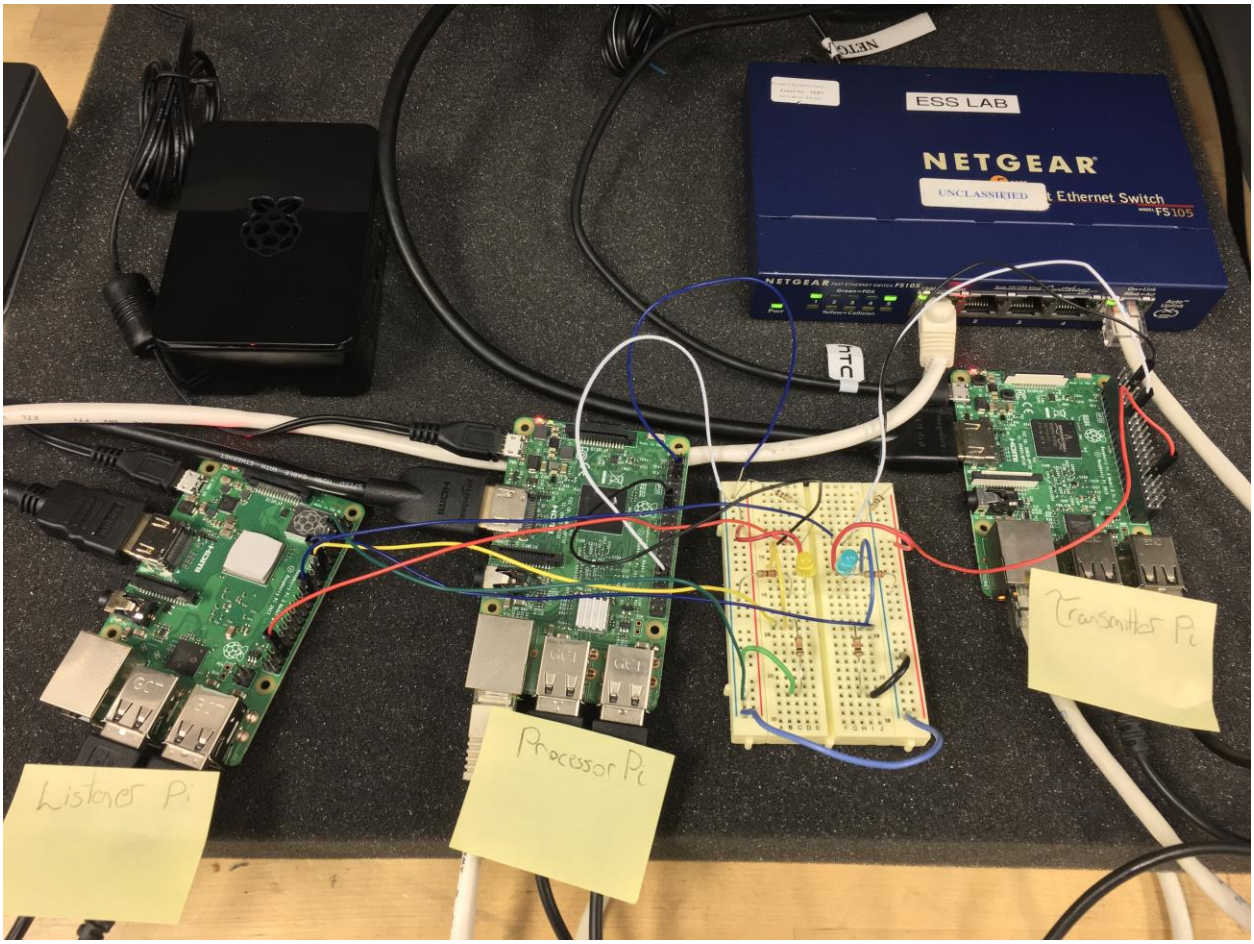

Figure 3: Actual Bench Setup

Raspberry Pis Pinging.mp4

Results, Analysis, and Data

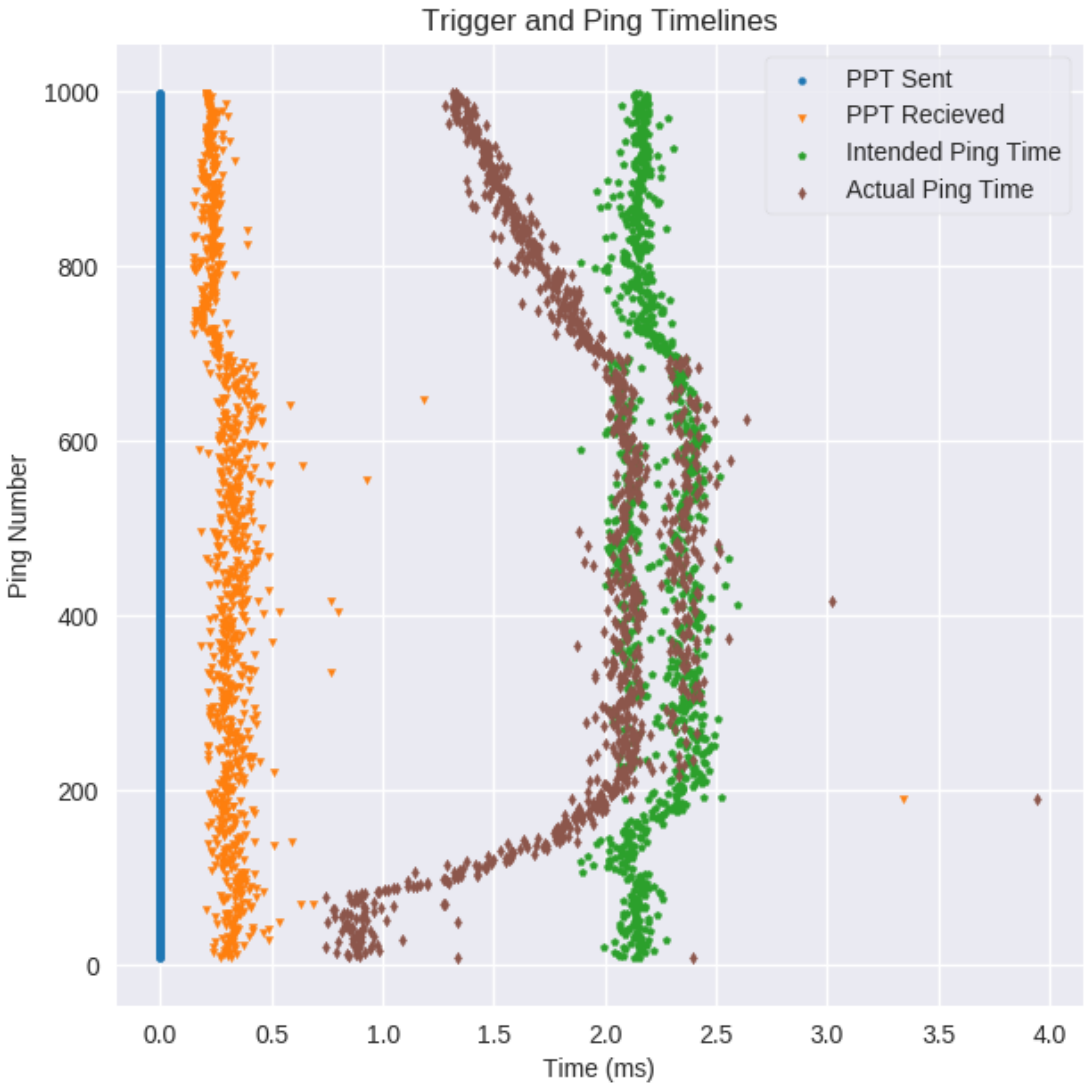

Figure 4: 1000 Pings, Timelines

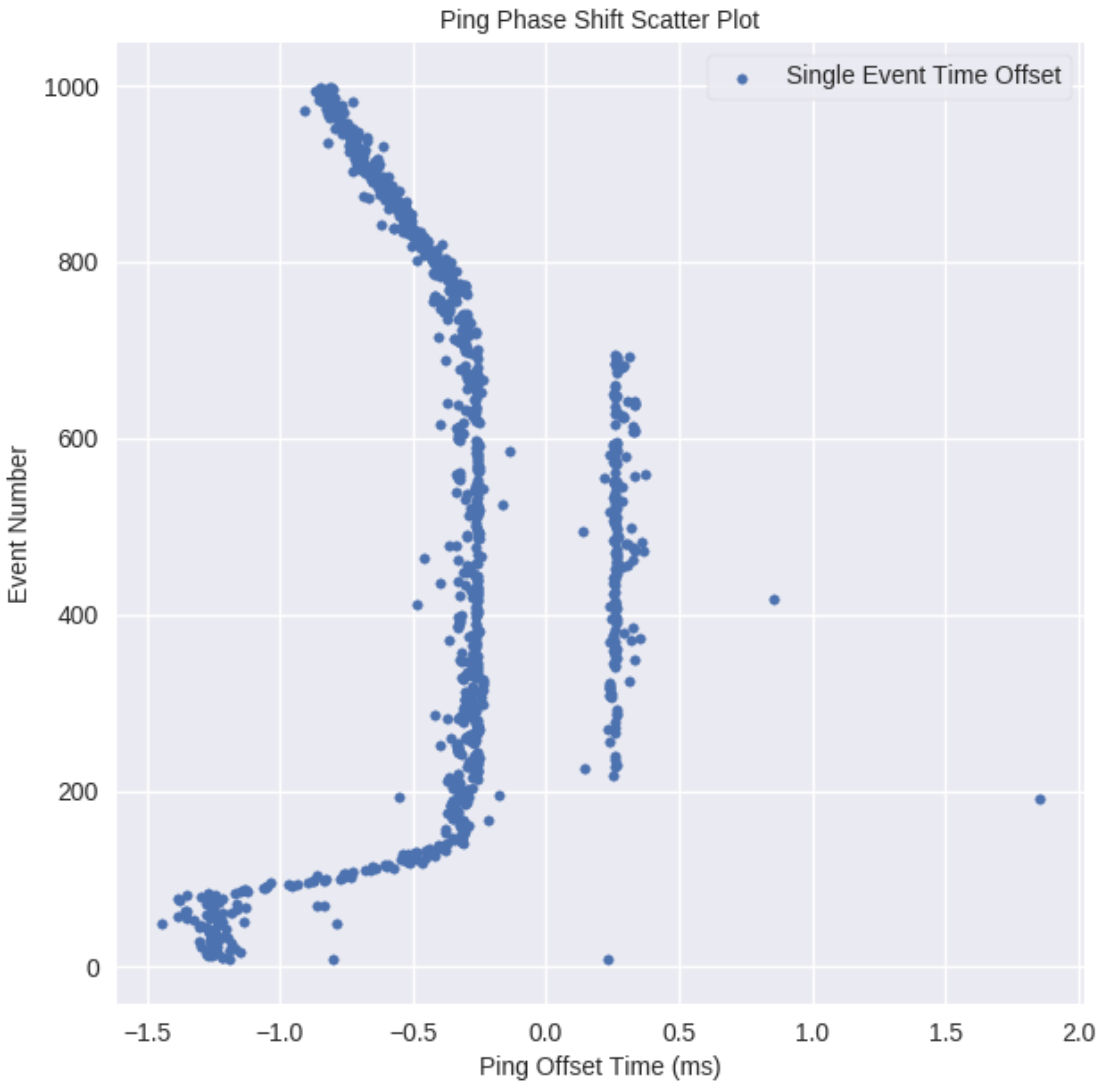

Figure 5: 1000 Pings, Offset Magnitudes

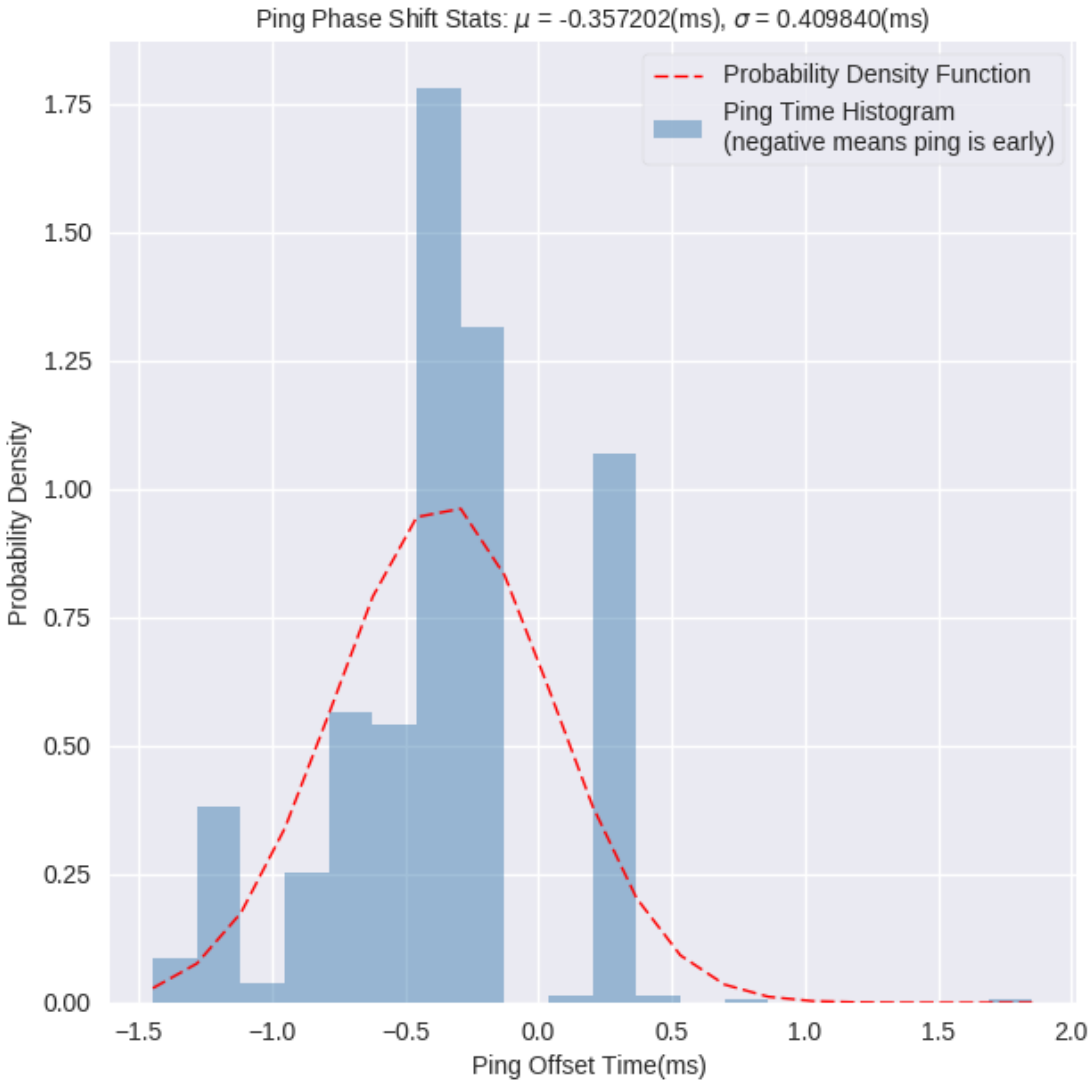

Figure 6: 100 Pings, Offset Histogram, Avg Offset, Offset Std Dev