A Distributed Virtual Time System for Embedded Linux Devices

Yuan-An Liu
Illinois Institute of Technology
yliu301@hawk.iit.edu

Christopher Hannon
Illinois Institute of Technology
channon@hawk.iit.edu

Abstract:

The successful operations of modern electric power infrastructure are dependent on reliable and efficient communication networks. This calls for a simulation-based platform that provides robust features and controllability for evaluating network application designs, as well as facilitating the transition from in-house research ideas to deployment on real systems. This work focuses on creating a hybrid testing platform combining electric power simulation with Linux network devices such as hosts and switches. This system supports a distributed communication network using real networking hardware to support high fidelity analysis of communication network applications and their impacts on the power systems. The challenge in designing such a hybrid system is in the synchronization of combining real networking devices to an electric power simulator. We implement a solution for this synchronization challenge through a virtual time system.

1. Introduction

Our main contributions are in the design of a distributed virtual time system for Linux devices with precision controllability of the execution of the systems. In other words, pausing and resuming any specified processes in the perception of their own virtual clocks. An efficient system clock has to minimize the overhead of its own operation, ensuring the accuracy of the system. We demonstrate how to utilize hardware interrupts to communicate between linux devices for rapid synchronization of the distributed virtual time system. We also show that this approach is efficient, and scales linearly with the number devices. Additionally, we evaluate the system performance to measure the impact of the virtual time system and synchronization overhead on the throughput and latency of the communication devices.

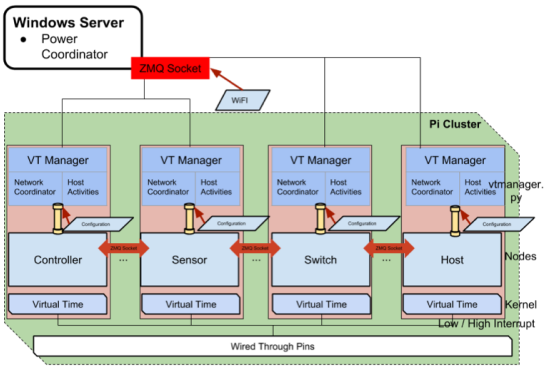
2. Architecture

2.1 Distributed Architecture

To achieve the our simulation, we come up with design of using multiple embed devices to create and distributed system. The relationship between nodes as shown in Fig. 2.1. Each node has 2 major layers:

- Virtual Time Manager
- Virtual Time kernel

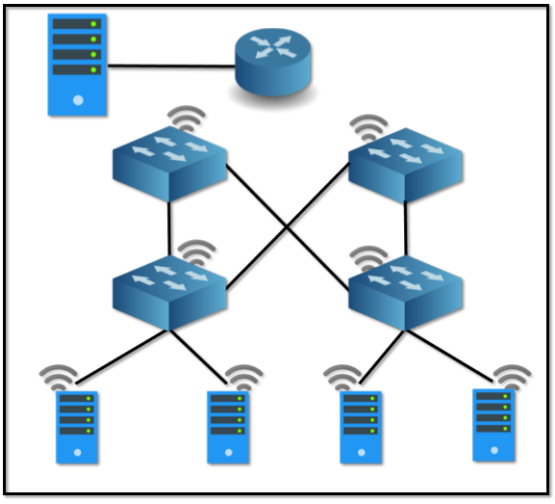
They are all connected using Ethernet, wireless networking and direct hardware connection.



↑ Fig. 2.1. The relationship between distributed systems

2.2 Design

The distributed system composed of 3 communication channels: Ethernet, wireless management and direct hardware connection with general purpose and router Linux hardware.



↑ Fig. 2.2. Architecture of distributed system:

- The top node:

Monitoring machine which is not in Virtual Time.

- The bottom nodes:

Embedded Linux devices. For the switch, we use Open vSwitch to create the flow for the network [4].

- wireless and wired

indicate the connection between switches, router and nodes

2.3 System

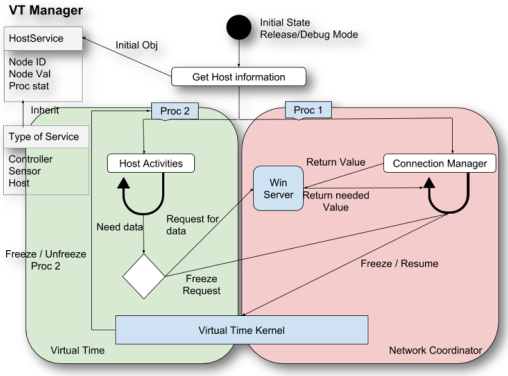
- Linux Debian 8 (Bananian 16.04)
- Kernel 3.4.113+
- OpenvSwitch 2.3.0
- Python 2.7.9
- GCC 4.9.2
- Banana Pi M1
- Banana Pi R1

3. Implementation

3.1 Virtual Time Manager

Virtual Time Manager contains two sub-processes Fig. 3.1.:
The green part of the diagram represents the host activity.The host can be either a Controller, a Switch or a Sensor.

The red part of the diagram represents the Network Coordinator (NC) which handles all pausing, resuming and other network communications.

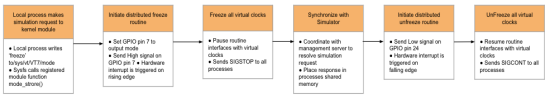


↑ Fig. 3.1. State Diagram of Virtual Time Manage

3.2 VTGPIO and Virtual Time

Building on previous work by C. Hannon and J. Yan [1] we extend the virtual time kernel for distributed environments. The methods we do this by is a hardware bus

that connects the devices. Communication is binary over hardware interrupts that enable the devices to indicate to each other in a distributed fashion to initiate the pausing of the virtual clocks. The synchronization algorithm is kicked by a process that creates an event to the power simulator. Upon this time, the kernel module on the local process triggers the distributed pause using the hardware interrupt. At this time each device can freeze the required virtual clocks. When the simulator is finished processing events it places the result in the calling processes buffer and the kernel module can begin the unfreeze routine similar to the freeze one. The method was originally presented in [2].



↑ Fig. 3.2. The flow diagram represents the overview of distributed algorithm within the kernel module, VTGPIO layer.

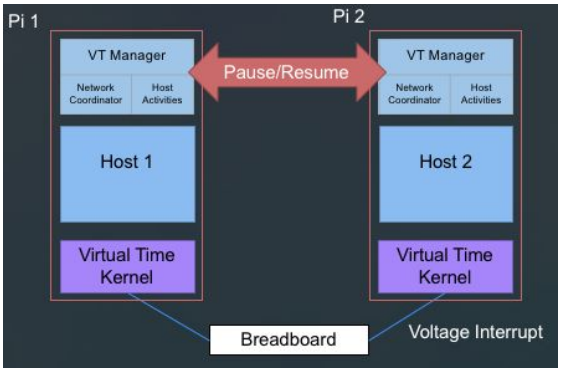
```
root@bananapi1 ~ # gpio readall
 +----------+-Rev3-+------+--------+------+-------+
 | wiringPi | GPIO | Phys | Name   | Mode | Value |
 +----------+------+------+--------+------+-------+
 |      0   |  17  |  11  | GPIO 0 | ALT4 | Low   |
 |      1   |  18  |  12  | GPIO 1 | IN   | Low   |
 |      2   |  27  |  13  | GPIO 2 | ALT4 | Low   |
 |      3   |  22  |  15  | GPIO 3 | ALT4 | Low   |
 |      4   |  23  |  16  | GPIO 4 | IN   | Low   |
 |      5   |  24  |  18  | GPIO 5 | ALT2 | Low   |
 |      6   |  25  |  22  | GPIO 6 | ALT4 | Low   |
 |      7   |   4  |   7  | GPIO 7 | IN   | Low   |
 |      8   |   2  |   3  | SDA    | ALT5 | Low   |
 |      9   |   3  |   5  | SCL    | ALT5 | Low   |
 |     10   |   8  |  24  | CE0    | ALT2 | Low   |
 |     11   |   7  |  26  | CE1    | IN   | Low   |
 |     12   |  10  |  19  | MOSI   | IN   | Low   |
 |     13   |   9  |  21  | MISO   | IN   | Low   |
 |     14   |  11  |  23  | SCLK   | IN   | Low   |
 |     15   |  14  |   8  | TxD    | ALT0 | Low   |
 |     16   |  15  |  10  | RxD    | ALT0 | Low   |
 |     17   |  28  |   3  | GPIO 8 | IN   | Low   |
 |     18   |  29  |   4  | GPIO 9 | ALT4 | Low   |
 |     19   |  30  |   5  | GPIO10 | OUT  | High  |
 |     20   |  31  |   6  | GPIO11 | ALT4 | Low   |
 +----------+------+------+--------+------+-------+
```

↑ Fig. 3.3. GPIO Information for Our Hardware.

4. Evaluation

4.1 Method

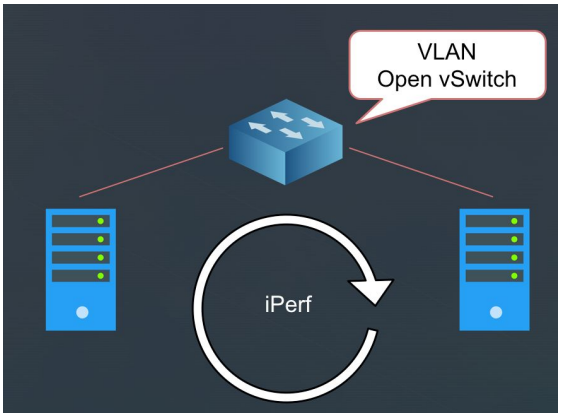
In this section, we evaluate the simulation performance and the overhead. It was first necessary to ensure the device clocks were synchronized. To complete the synchronization, we use the default NTP server from Debian and we input the starting date-time for the VT manager to start the simulation at the same time. The concept of the evaluation is shown Fig. 4.1.



↑ Fig. 4.1. Evaluation Concept

Basically, let the machine do the pausing and resuming multiple times. And we log the timestamp from layers to layers to compute the overhead.

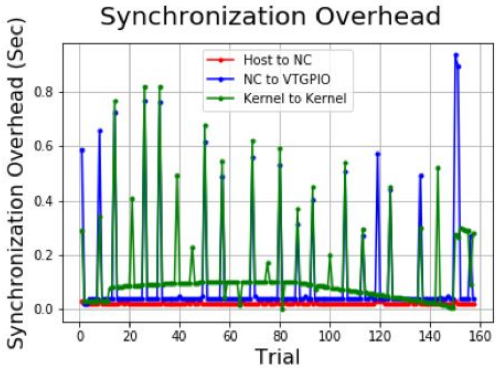
Second, to benchmark the network performance with our hardwares with and without virtual time. We are using IPerf to evaluate both with and without using the virtual time. After VLAN tagging the open source router BPi-R1, since all the ports for R1 share one interface, and the installation of Open vSwitch, we use IPerf to benchmark by sending TCP packets as shown in Fig. 4.2.



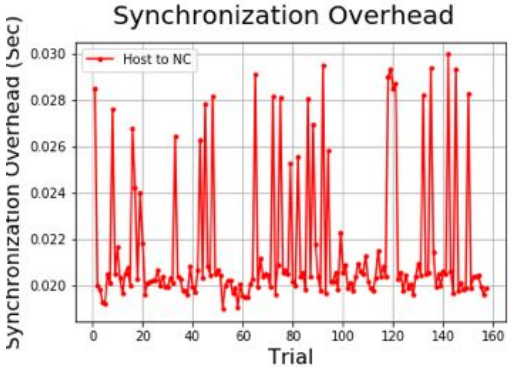
↑ Fig. 4.2. Evaluation using iPerf Benchmarking

4.2 Result

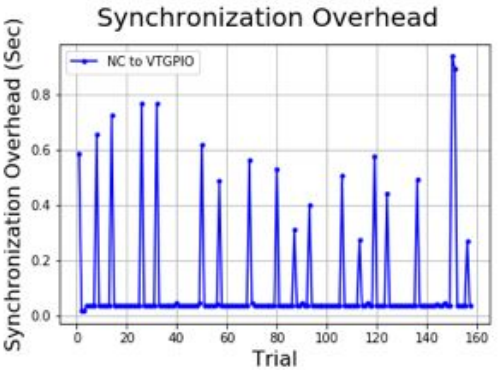
The results of measuring the performance as shown below. In Fig. 4.3, it shows all ΔT from layers to layers. From Fig. 4.4 - 4.6, They are represented individually. Fig. 4.7, 4.8 are the CDF of the overhead.



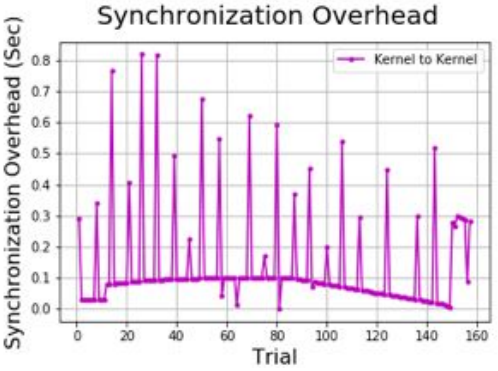
↑ Fig. 4.3. Synchronization Overhead: multi-layer



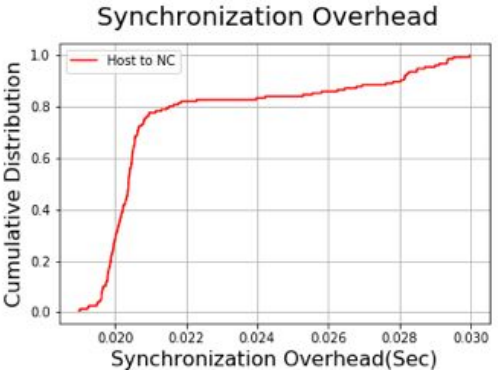
↑ Fig. 4.4. Synchronization Overhead: Host to NC



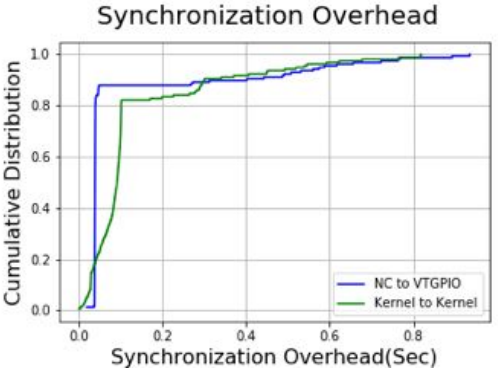
↑ Fig. 4.5. Synchronization Overhead: NC to VTGPIO



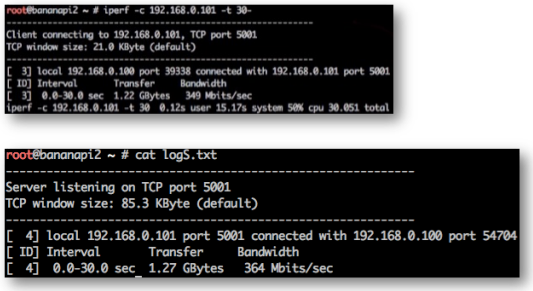
↑ Fig. 4.6. Synchronization Overhead: Kernel to kernel



↑ Fig. 4.7. Synchronization Overhead: CDF



↑ Fig. 4.8. Synchronization Overhead: CDF



↑ Fig. 4.9. IPerf results:

- Top: without Virtual time.
- Bottom: with Virtual Time

5. Conclusion and Future Work

As we can see from the results, there are approximately 15% chances to have a large overhead and there are some connections between layers to layers with the significantly large noises. Our assumption is due to the cache of the disk when we are trying to log the timestamp of each operations. To improve the logging method, we could store the log messages in the memory first and dump it to the disk when it is possible.

We also plan to finish the Windows server and implement this Simple Queue Structure to deal with race condition during the synchronization. After the implementation of the monitoring server, we are going to add more nodes to our system to increase the complexity so that it can be for reality and closer to the real-life senorio.

Above all, it is necessary and our primary goal to improve evaluation by increasing the accuracy and minimize the overhead.

6. Acknowledgements

For all the source code and instructions, Please read my Github repository [5] . Thanks to Professor Kevin Jin for his support and funding the testbed.

References:

[1] Christopher Hannon, JiaqiYan, Dong Jin: DSSnet: A Smart Grid Modeling Platform Combining Electrical Power Distribution System Simulation and Software Defined Networking Emulation

[2] Christopher Hannon, Neil Getty: Heterogeneous Distributed Embedded Linux System for Hardware-in-the-Loop Smart Grid Testbed

[3] Diego L. C. Dutra, Edilson C. Corrêa and Claudio L. Amorim: An efficient virtual system clock for the wireless raspberry pi computer platform

[4] OpenvSwitch on BPi-R1 with Bananian http://blog.br1an.space/2018/04/openvswitch-on-bpi-r1-with-bananian.html#more

[5] Github Repository: emb-vt https://github.com/Br1an6/emb-vt/tree/master/src/mgmt