Steal Task Scheduling from OS: Enabling Task-Network Co-schedule for Time-critical Traffic

Xuyan Jiang, Wenwen Fu, Xiangrui Yang, Yinhan Sun, Zhigang Sun
College of Computer, National University of Defense Technology
Changsha, Hunan, China
{jiangxuyan,fuwenwen16,yangxiangrui11,sunyinhan20,sunzhigang}@nudt.edu.cn

CCS CONCEPTS

• **Networks** → **End nodes**; • **Computer systems organization** → **Real-time systems**.

KEYWORDS

co-scheduling, real-time system, Real-Time Ethernet

ACM Reference Format:
Xuyan Jiang, Wenwen Fu, Xiangrui Yang, Yinhan Sun, Zhigang Sun. 2021. Steal Task Scheduling from OS: Enabling Task-Network Co-schedule for Time-critical Traffic. In *SIGCOMM '21 Poster and Demo Sessions (SIGCOMM '21 Demos and Posters), August 23–27, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3472716.3472868

1 INTRODUCTION

Real-Time Ethernet (RT Ethernet) such as Time-Sensitive Networking and Time-Triggered Ethernet is widely deployed in the distributed real-time systems such as aerospace, automotive and industrial domains[2]. Typically, there are many sensoring-computing-actuating Control Loops (CL) requiring real-time and deterministic end-to-end communication. From the network perspective, these CLs work as follow. The sensor generates sensoring traffic periodically to the computing node (i.e. the end system) via RT Ethernet. Then the computing node executes the corresponding task and generates actuating traffic to the actuator. Finally, the actuator receives the traffic and executes the actions accordingly. Sensoring and actuating traffic are both Time-critical (TC) traffic. In order to achieve end-to-end determinism in the loop, the deterministic and real-time process on the end system must be guaranteed. In another word, the task running on the Operating System (OS) of the end system, which processes TC traffic, must be scheduled with tight dependency with the underlying network scheduling[2].

Traditionally, the task scheduling strategy on the OS is asynchronous with the underlying network (i.e. frame scheduling). This may incur great uncertainty in the scenario where end-to-end determinism must be achieved. The main reason behind is that the local time of the OS is out of sync with the network whose time is synchronized by sophisticated time synchronization protocols like PTP [3].

SIGCOMM '21 Demos and Posters, August 23–27, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8629-6/21/08…$15.00
https://doi.org/10.1145/3472716.3472868

In order to address this issue, a natural solution would be synchronizing the OS time with the whole network (e.g., integrating PTP into kernel). Moreover, a special scheduling mechanism should also be implemented which takes the packet arrival info into account. However, this requires massive modifications to the kernel and may incur compatibility issues when integrating with different types of the underlying network [5]. What's more, Commercial-off-the-shelf (COTS) end systems are more adopted in modern real-time system to reduce the cost and shorten the development period [4]. So it is hard to introduce fundamental changes to the key components of COTS systems.

In this poster, we aim to tackle this problem from a different perspective. Specifically, we suggest that by carefully scheduling packets from the ingress engine of the NIC, a task-network co-scheduling can be achieved, which eliminates the necessity of the modification to the task scheduling mechanism on the OS completely. For this purpose, we introduce Frame-Task Co-Scheduling (FTCS), a mechanism that enables co-scheduling of task and network on COTS end systems. FTCS steals task scheduling from OS through time-triggered frame schedule and submission. The potential benefits are as follows. Firstly, FTCS achieves tight co-scheduling of critical task and traffic without any modification of software and hardware on COTS devices. Secondly, OS time on the end system does not need to synchronize with the network. Because the entity executing FTCS will synchronize with network global time. Thus, even if the frequency and phase shift are unmatched , co-scheduling can still be achieved.

2 OVERVIEW

There are three assumptions for FTCS. 1) The execution flow of TC task is receiving-processing-transmitting. 2) Task is developed using a blocking socket channel to transfer data. A blocking socket will not return control back to the task until it has received or sent all the data, which means that the task will be suspended (i.e. non-schedulable), and is in state of TASK_UNINTERRUPTABLE. Once the socket is done, task state shifts to TASK_RUNNING, indicating that the task is ready to run (i.e., schedulable). All tasks are suspended and wait for reception data at the beginning. 3) TC tasks only run on the specified processors, and these processors do not handle any hardware interrupts so the execution of TC task will not be interrupted arbitrarily. [1] It is easy to implement because COTS OS provides the interface to set task-CPU and interrupt-CPU affinities, such as `sche_setaffinity()` in Linux.

What is the meaning of *steal*? *Steal* means that shifting the task scheduling from OS to network. Network takes the initiative to decide when and which task to schedule, which originally belongs

[1] That is because hard interrupts are triggered randomly and have the highest priority.

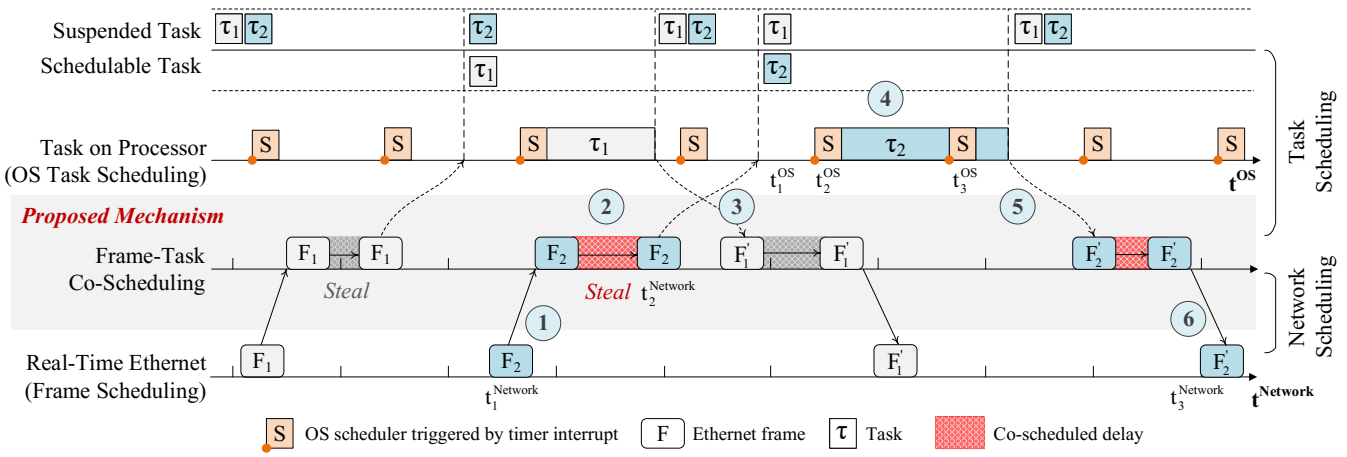

Figure 1: How to steal task scheduling from OS. The idle process (Process 0) is not shown here.

to the OS scheduler (or more specifically the *schedule()* function in Linux).

Why can FTCS steal task scheduling from OS? The frame's status in the end system can be reasoned and deduced when there is no queuing delay from receiving path to the transmitting path. Fortunately, zero queuing delay can be guaranteed by the control of submission time to the OS. Because there are no frame processing queue on the NIC-OS interface. Upon the software side, task and interrupt affinities ensure that the execution time of task can be estimated by Worst Case Execution Time (WCET) tools[1]. And the Point-in-Time (PIT) the frame arrives at socket between the PIT the corresponding task executes have upper bound too, which is the time slice of OS.

Therefore, the total delay in the end system has its upper bound, which is the insight of the proposed FTCS mechanism.

How to *steal* task scheduling from OS? The key point of FTCS to dominate task scheduling is to ensure that there is **no more than one schedulable task** for OS scheduler at any time. Thus, the scheduler has no alternative but to schedule the only one task specified by the network. In other words, FTCS *steals* the task scheduling from OS.

In order to explain the design principle in detail, take task τ_2 and its traffic F_2 and $F_2^{'}$ for example in Fig. 1. 1) The RT Ethernet schedules F_2 to the end system at PIT $t_1^{Network}$. The PIT $t_1^{Network}$ has already been pre-planned by RT Ethernet thanks to its network schedule. 2) After the end system receives F_2, FTCS delays the frame for a specific amount of time and then schedule & submit it to processor at PIT $t_2^{Network}$. The delay, namely co-scheduled delay in Fig. 1, is vital to *stealing* and co-scheduling, which will be discussed later. 3) F_2 goes through the NIC-OS interface and received by the blocking socket of τ_2 at t_1^{OS}. Once socket receive F_2, it wakes up τ_1 and shifts its state from TASK_UNINTERRUPTABLE to TASK_RUNNING. That is to say, τ_2 have the opportunity to be scheduled by OS scheduler in the most recent Timer Interrupt. 4) When the periodic timer interrupt triggers a context switch at PIT t_2^{OS}, the OS scheduler checks if there are schedulable tasks and finds out that only τ_2 is ready to run. **So no matter what scheduling policy the scheduler is performing**, τ_2 is scheduled to process F_2 and get $F_2^{'}$. It is worth noting that if the current OS time slice has expired before the task completion, OS scheduler will re-schedule this task, the same as the case at t_3^{OS}. This is because there is no other task to schedule at the moment for OS. What's more, since this processor does not handle any hardware interrupt, so τ_2 will not be interrupted. 5) Then τ_2 sends $F_2^{'}$ and shifts to suspended state again. 6) $F_2^{'}$ goes through the OS-NIC interface and is captured by FTCS. If it is not the PIT for network to schedule, FTCS will delay $F_2^{'}$ again until it is the pre-planned departure PIT, which is $t_3^{Network}$ in Fig. 1. To sum up, the network-task co-scheduling is achieved through FTSC such that the delay of receiving-processing-transmitting on the end system meets the requirement of RT Ethernet scheduling.

Determining the Co-Scheduled Delay (CSD) is one of the key points in FTCS. As mentioned above, **the length of CSD determines when to *steal* task scheduling**, like the $t_2^{Network}$ in Fig. 1, because the arrival PIT is determined by RT Ethernet. Hence, the algorithm to compute each frame's CSD is the key to enable FTCS. The principle of CSD algorithm is to avoid the latter frame interfering the former one. A possible method is to split receiving-processing-transmitting, i.e., step 3, 4, 5, into parallel stage according to the use of shared resources. And the algorithm needs to ensure it is contention-free at every stage. Apparently, the potential algorithm should meet the constraint of every frame's arrival and departure PIT pre-planned by RT Ethernet.

3 FUTURE WORK

Our future work will focus on the detailed design & verification of FTCS, including the entity and the algorithm. Moreover, the deploying position of FTCS is another direction that potentially affects the performance of FTCS.

ACKNOWLEDGMENTS

This work was funded by the National Natural Science Foundation of China (91938301) and the Open Project of Zhejiang Lab (2020LE0AB01).

REFERENCES

[1] Jaume Abella, Damien Hardy, Isabelle Puaut, Eduardo Quinones, and Francisco J Cazorla. 2014. On the comparison of deterministic and probabilistic WCET estimation techniques. In *2014 26th Euromicro Conference on Real-Time Systems*. IEEE, 266–275.

[2] Silviu S Craciunas and Ramon Serna Oliver. 2016. Combined task-and network-level scheduling for distributed time-triggered systems. *Real-Time Systems* 52, 2 (2016), 161–200.

[3] T. Neagoe, V. Cristea, and L. Banica. 2006. NTP versus PTP in Com puter Networks Clock Synchronization. In *IEEE International Symposium on Industrial Electronics*.

[4] Rodolfo Pellizzoni, Bach D Bui, Marco Caccamo, and Lui Sha. 2008. Coscheduling of cpu and i/o transactions in cots-based embedded systems. In *2008 Real-Time Systems Symposium*. IEEE, 221–231.

[5] Jinli Yan, Wei Quan, Xiangrui Yang, Wenwen Fu, Yue Jiang, Hui Yang, and Zhigang Sun. 2020. TSN-Builder: enabling rapid customization of resource-efficient switches for time-sensitive networking. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.