Fachprojekt for Embedded System: Design and Implement Your Own Embedded Systems (1)

Junjie Shi Mikail Yayla

LS 12, TU Dortmund

06, May 2020

Introduction

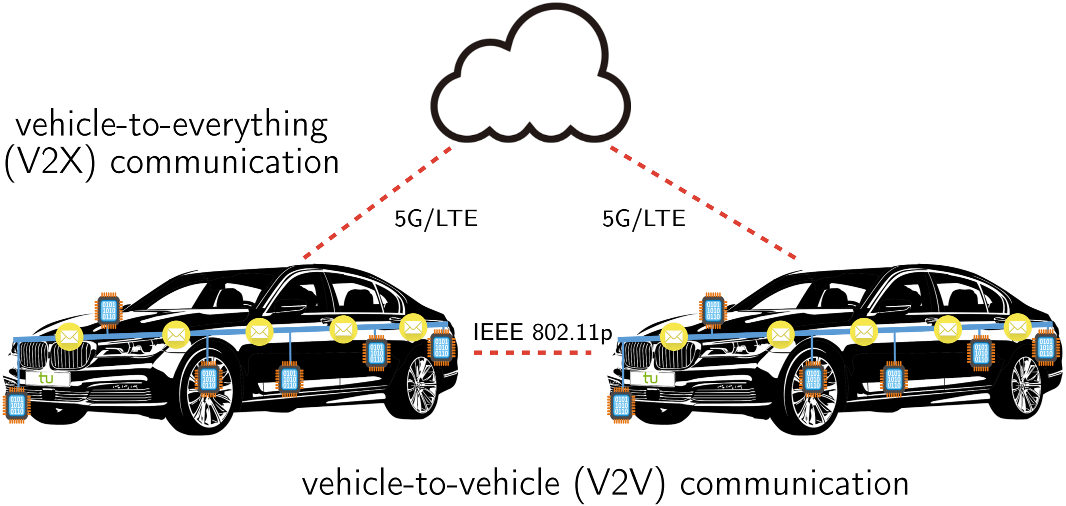
- The Fachprojekt
  - Different topics related to Embedded System, Real-Time, and Machine Learning
  - Group work: 3 Students per group (8 groups in total)
  - Hand in a final report + give a presentation
- The supervisors
  - Junjie Shi and Mikail Yayla
  - Lea Schönberger, Christian Hakert, and Kuan-Hsun Chen
- Introduce yourselves for grouping (After the presentation)

Topics

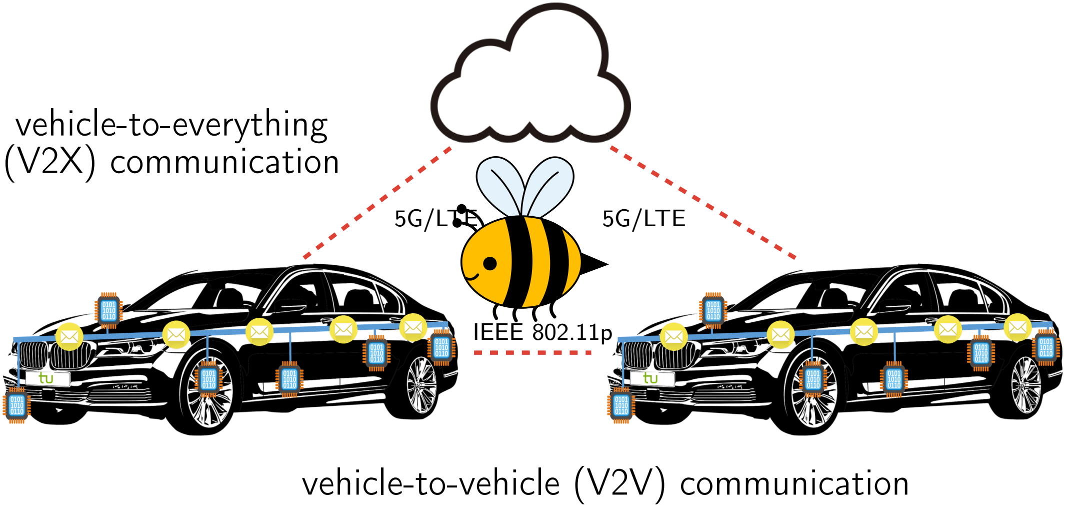
- Simulating Security Attacks on Embedded Systems (2)
- Real-Time Scheduler Implementation on Unikraft (2)
- The Effect of Input Binarization on Neural Network Accuracy (2)
- Implementing Non-volatile Memory Error Models in PyTorch Tensor using CUDA Kernels (2)
- Implement A Multiprocessor Event-driven Simulator (1)
- Interactive Real-Time Gaming: Design a few games to demonstrate the real-time properties (2)

Simulating Security Attacks on Embedded Systems

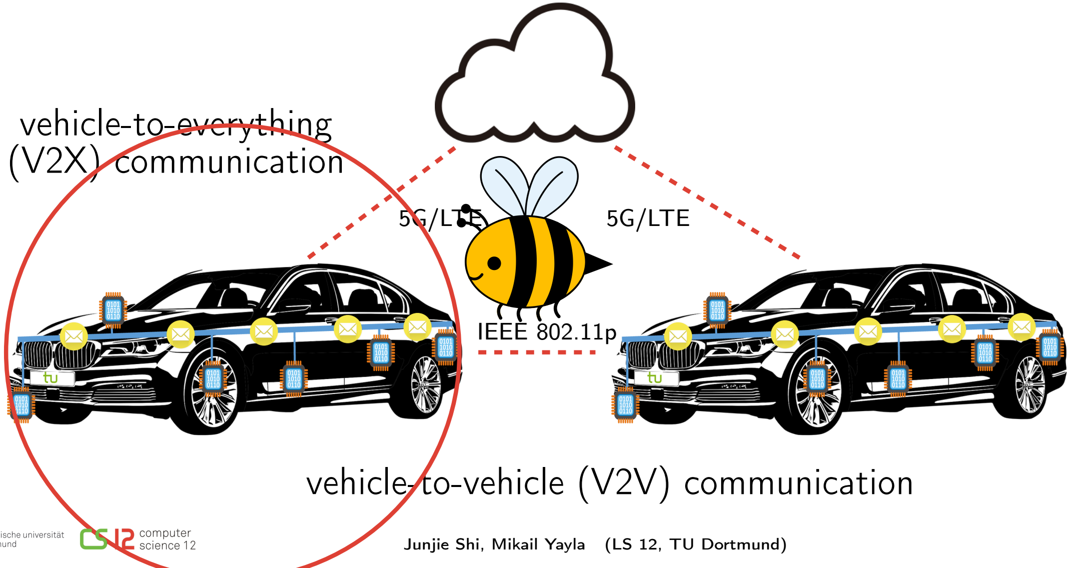
Motivational Example



Motivational Example

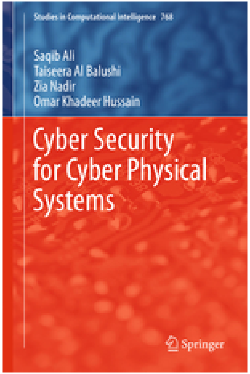


Motivational Example



Which threats and attacks exist?

- denial of service attack (DOS)
- distributed denial of service attack (DDOS)
- hidden vehicle attack
- manipulation of sensor information
- privilege escalation attack
- ...



Your Job

- experimentally investigate the impact of different security attacks on cyber-physical systems
- choose or create a model in the robot simulator Gazebo[1]
- create a simple Controller Area Network (CAN) with the discrete event simulation library OMNeT++[2]
- connect your model and your network
- try out various security attacks, investigate their impact, document the results

[1] http://gazebosim.org/
[2] https://omnetpp.org/

Required Skills

- knowledge of C++ (or the strong motivation to learn it)

Why real-time scheduling?

- Some systems rely on computing tasks to complete within a given time
  - Automotive and aeronautic systems
  - Flight control, emergency systems
- "If a task misses its deadline, a catastrophy happens"

Real-Time Scheduler Implementation on Unikraft

Why real-time scheduling?

- Some systems rely on computing tasks to complete within a given time
  - Automotive and aeronautic systems
  - Flight control, emergency systems
- "If a task misses its deadline, a catastrophy happens"

Why Unikraft?

- Real-time systems are controlled by embedded system nowadays
- Unikraft is a operating system, designed for embedded systems
- Library structure makes Unikraft configurable, support many libraries and apps

Real-Time Scheduler Implementation on Unikraft

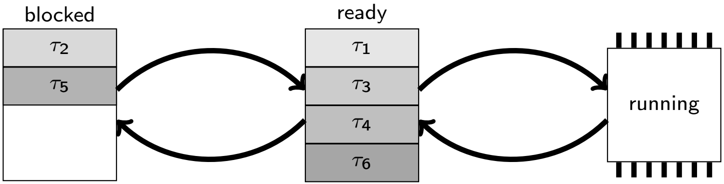
Real-time scheduling in a nutshell

- A timer generates an interrupt every x ms
- Real-time tasks are either ready (want to compute), running (currently computing) or blocked (do not want to compute)
- At each timer tick, tasks are moved between read, running and blocked state
- The running task is chosen from the ready list according to the highest priority

Real-Time Scheduler Implementation on Unikraft

Real-time scheduling in a nutshell

- A timer generates an interrupt every x ms
- Real-time tasks are either ready (want to compute), running (currently computing) or blocked (do not want to compute)
- At each timer tick, tasks are moved between read, running and blocked state
- The running task is chosen from the ready list according to the highest priority



Real-Time Scheduler Implementation on Unikraft

What to do?

- Hardware implementation
  - Setup the timer interrupt
  - Implement the context switch
- Scheduler implementation
  - Keep tasks in data structures
  - Move them on every tick between the lists
  - Choose according to priority
- Task Scheduler interface
  - Tasks may be created, started, stopped
  - Tasks may want to go to sleep

DON'T PANIC

- we only implement a minimal real-time scheduler
- NO advanced functions like mutextes, semaphores, ...
- NO early completion, yielding, ...
- NO device driver interaction

Implement NVM Error Models in PyTorch Tensor using CUDA Kernels

Why NVM as main memory?

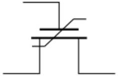
- DRAM is reaching a plateau
- NVMs use no power during idle time and retain information, no refreshs needed
- NVM performance/cost is getting closer to DRAM

The NVM-OMA Project: NVM as both main memory and storage

- OMA-OS
- NVM error models and application level error tolerance

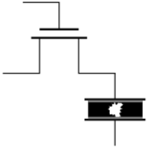
Emerging NVMs: Introduction

FeFET



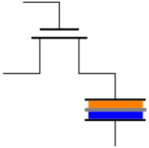
- State: Permanent polarisation in ferroelectric
- Polarization reversed by external electric field

RRAM



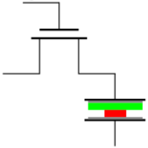
- Form/reset defects in material
- More cuurent can flow when defect formed

STT-RAM



- State: Spin orientation of electrons in magnetic layer
- Change orientation by spin polarization

PCM



- Material state amorphous or crystalline
- Applying heat changes state and resistance

Bit Errors in Emerging NVMs

Emerging NVMs have bit errors

- FeFET: High temperature $\rightarrow p_{01} = 2.1\%$ and $p_{10} = 1.1\%$ (Amrouch, 2020)
- RRAM: Low power programming setting $\rightarrow$ 3.3% (Hirtzlin et al., 2019)
- STT-RAM: Lower programming energy $\rightarrow$ higher bit error rate (Hirtzlin et al., 2019)
- MLC-PCM: Resistance drift $\rightarrow$ after t_d, cell drifts to other programming state (Papandreou et al., 2010)

Bit error tolerant applications

- Error tolerant applications have lower requirements on memory
- Some algorithms can be optimized for error tolerance (Artificial Neural Networks)

PyTorch and CUDA GPU Kernels

PyTorch

- Open source machine learning library (primarily developed by FAIR)
- Tensor computing like in Numpy, but with extensive use of GPU
- Deep neural network tools
- Used by a large fraction of researchers that work with neural networks

Nvidia CUDA Kernels

- PyTorch relies on CUDA for parallel computing tasks
- CUDA is a parallel computing model and framework for parallel computing
- For orchestrating the processing of parallel workloads with CUDA kernels

Implement NVM Error Models in PyTorch Tensor using CUDA Kernels

Tasks and goals

- Goal: Provide a collection of efficient NVM bit error model implementations in PyTorch Tensor
- Conduct literature review on error models of different NVMs
- Prepare a simulation framework, in which bit flips are injected according to the bit error models into PyTorch tensors, using custom CUDA kernels

The Effect of Input Binarization on Neural Network Accuracy

Artificial neural networks

- Broadly applied in numerous fields
- In ES and CPS, NNs are deployed on resource-constrained devices, such as battery-powered and mobile systems
- Managing the resource demand of NNs is a challenge, especially the memory (too many weights)

Efficient NNs

- Quantization and binarization of weights
- Convolutions become XNOR operations
- Realization in hardware is simple and efficient
- What about the first layer ?

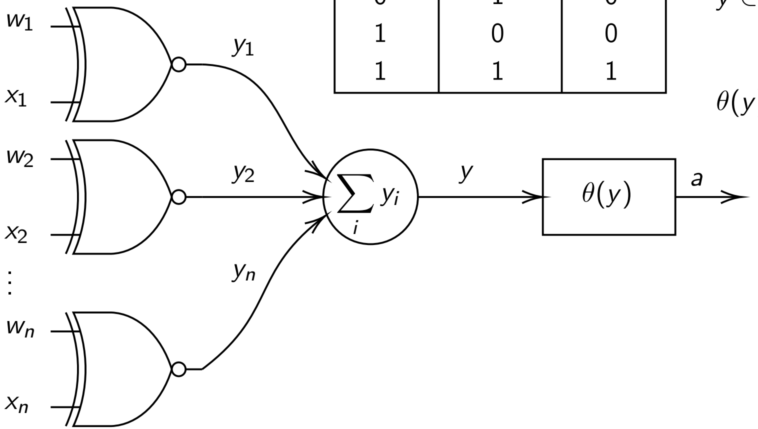
Binarized Neural Networks: Basics (Hubara et al., 2016)

w_1	x_1	y_1
0	0	1
0	1	0
1	0	0
1	1	1

$w_i, x_i, y_i, a \in \{0, 1\}$

$y \in \mathbb{Z}$

$$\theta(y) = \begin{cases} 1 & y > s \\ 0 & \text{else} \end{cases}$$



For XNOR in First Layer, Input Binarization is Needed

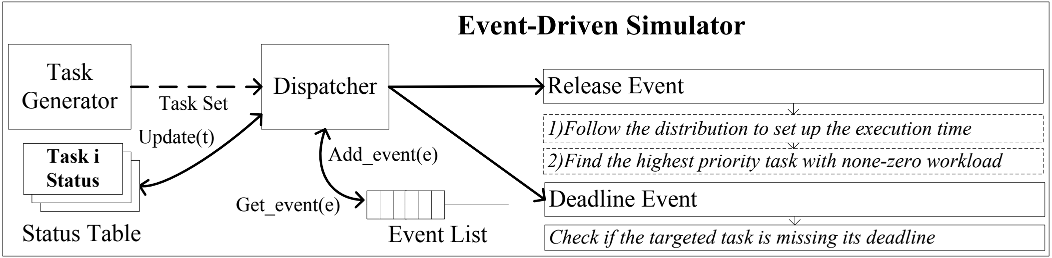
Tasks and goals

- Goal: Create an overview of different binarization techniques and their corresponding effects on NN accuracy
- Binarization algorithms for the input (stochastic, otsu, thresholding, ...)
- Datasets (MNIST, Fashion, CIFAR10, SVHN, ...)
- Prepare own training and evaluation scripts, NN models, and input binarization methods

Implement A Multiprocessor Event-driven Simulator

- Timeliness is an important feature for many embedded systems.
- Simulate deadline miss rate for a specific sporadic real-time task under fixed-priority preemptive scheduling.
- Consider randomly generated task sets with an execution behavior that simulates jobs that are subjected to soft errors incurred by hardware transient faults under a given fault rate.

Implement A Multiprocessor Event-driven Simulator



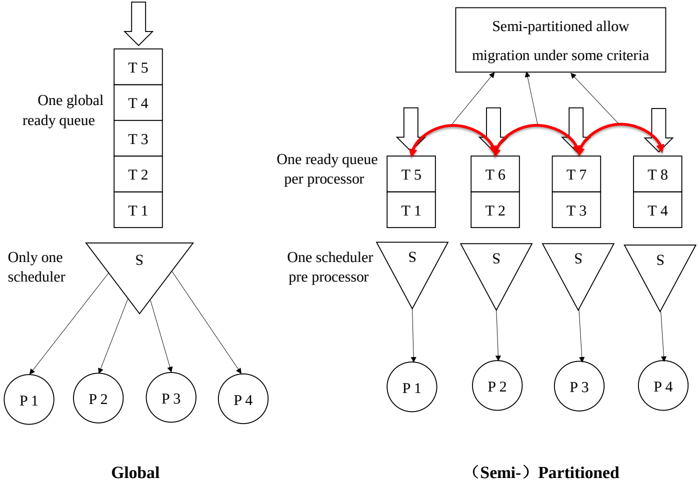
- based on Python2.7
- Only support uni-processor environment
- two basic events, i.e., release and deadline

Implement A Multiprocessor Event-driven Simulator

- Task Generator: Outputs a set of generated tasks under a given utilization value.
- Dispatcher: It checks if the number of released jobs from the targeted task is equal to the targeted number. If not, it continues to dispatch the next event from the event list.
- Event List: This linked list keeps track of the following events in the simulated task system. When a new event is inserted by another `release` event, the events in the list are sorted by their future occurring time.
- Status Table: It records the number of deadline misses, the number of released jobs, and the remaining work- load for each unfinished job of a task.

Implement A Multiprocessor Event-driven Simulator

Schedule algorithms for multi-processor real-time systems:



Implement A Multiprocessor Event-driven Simulator

What you have to do:

- Convert the python version from 2.7 to 3.6
- Extend the uni-processor environment to multiprocessor environment
- Support different type of multiprocessor schedule algorithms

Interactive Real-Time Gaming

Design game(s) based on FreeRTOS emulator to demonstrate the real-time properties.

- FreeRTOS:
  - Tiny, power-saving kernel
  - Support 40+ architectures, i.e., RISC-V and ARMv8-M
  - Modular libraries
  - IoT Reference Integrations
  - MIT licensed, free for teaching
- Emulator:
  - hardware or software that enables one computer system (host) to behave like another computer system (guest)
  - enables the host system to run software or devices designed for the guest system

Interactive Real-Time Gaming

FreeRTOS Emulator:

- An implementation of POSIX based FreeRTOS with the combination of SDL2 graphics. Aimed at providing an x86 emulation solution for teaching FreeRTOS to students without the need of embedded hardware
- Based on the FreeRTOS (V5.X) simulator developed by William Davy. Updated to use FreeRTOS V9.0.0
- More details can be found in: https://github.com/alxhoff/FreeRTOS-Emulator

Interactive Real-Time Gaming

Examples:

Interactive Real-Time Gaming

Examples:

- Demo

Interactive Real-Time Gaming

Examples:

- Demo
- Pong Game