CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Information Technology

Proceedings of the 10th Prague Embedded Systems Workshop

June 30 - July 2, 2022

Horoměřice

Czech Republic

ISBN 978-80-01-07015-4

Editors:

prof. Ing. Hana Kubátová, CSc.

doc. Ing. Petr Fišer, Ph.D.

Ing. Jaroslav Borecký, Ph.D.

Message from the Program Chairs

The Prague Embedded Systems Workshop is a research meeting intended for the presentation and discussion of students' results and progress in all aspects of embedded systems design, testing, and applications. It is organized by members of the Department of Digital Design at the Faculty of Information Technology and supported by the Czech Technical University in Prague. The workshop mainly focused on dependable and low power design, and it also has a special session on network security under the auspices of the Network Traffic Monitoring Lab. The workshop aims to enhance collaboration between different universities not only inside the EU. It will be based on oral presentations, mutual communication, and discussions.

There are three types of students' submissions and presentations:

- Full papers describing the students' original research. These papers were submitted to a standard reviewing process.
- Abstracts of authors' earlier published and successfully presented papers (at conferences, journals, etc.). These contributions were not reviewed; emphasis was put on the presentation and discussion.
- Student posters - abstracts of defended Bc. and MSc. Theses with subsequent poster presentation.

Twelve papers were accepted for PESW 2022 presentation, from which there were 10 full papers and two abstracts. Contributions from Austria, Belgium, Czech Republic, Poland, and Spain were accepted this year.

The technical program is also highlighted by three keynote speakers in the areas of network security and artificial intelligence:

- Networkmetrics for Network Monitoring and Security.
  Speaker: José Camacho Páez, University of Granada, Spain
- Reliability evaluation of general purpose and AI processing architectures.
  Speaker: Prof. Dr. Miloš Krstić, University of Potsdam, Germany, Leibniz Institute for High Performance Microelectronics (IHP)
- Advances in dependable image processing and deep learning.
  Speaker: Luca Cassano, Ph.D., Politecnico di Milano, Italy

Five technical sessions and two poster sessions were formed.

Last but not least we would like to thank to our sponsors CTU in Prague, Research Center for Informatics, ASICentrum, CESNET, Daiteq, Digiteq automotive, EATON, METIO Software, STMicroelectronics. Special thanks go to IEEE: IEEE Student Branch at Czech Technical University in Prague and IEEE Young Professionals, organizing student contest, and Czechoslovakia Section of IEEE.

Prague Embedded System Workshop is celebrating its 10^{th} anniversary this year in the same form of organization. Let's hope the tradition continues, therefore we wish all participants many heated discussions and possible establishment of mutual research cooperation at PESW 2022 in Horoměřice.

Hana Kubátová and Petr Fišer

Committees

Workshop Chairs

Hana Kubátová, CTU in Prague (CZ)

Petr Fišer, CTU in Prague (CZ)

Programme Committee

P. Bernardi, Politecnico di Torino (IT)

J. Bělohoubek, CTU in Prague, FIT (CZ)

L. Cassano, Politecnico di Milano (IT)

T. Čejka, CTU in Prague (CZ)

G. DiNatale, TIMA, Grenoble (FR)

R. Drechsler, University of Bremen (DE)

P. Fišer, CTU in Prague, FIT (CZ)

J.L. Gaudiot, University of California, Irvine (USA)

K. Jelemenská, STU Bratislava (SK)

M. Jenihhin, Tallinn Univ. of Technology (EE)

L. Kekely, Brno University of Technology, FIT (CZ)

P. Kitsos, TEI West. Greece (GR)

M. Krstić, IHP, Frankfurt (Oder) (DE)

H. Kubátová, CTU in Prague, FIT (CZ)

R. Kvaček, ASICentrum, Prague (CZ)

F. Leporati, Univ. di Pavia (GR)

I. Levin, Tel-Aviv University (IL)

R. Lórencz, CTU in Prague (CZ)

A. McEwan, University of Leicester (UK)

N. Mentens, KU Leuven (BE)

P. Mróz, University of Zielona Gora (PL)

M. Novotný, CTU in Prague, FIT (CZ)

A. Orailoglu, UC San Diego (USA)

Z. Plíva, TU Liberec (CZ)

M. Poupa, University of West Bohemia, FEE (CZ)

J. Raik, Tallinn Univ. of Technology (EE)

O. Ryšavý, BUT, Brno (CZ)

E. Sanchez, Politecnico di Torino (IT)

J. Schmidt, CTU in Prague, FIT (CZ)

M. Skrbek, CTU in Prague, FIT (CZ)

J. Sobotka, CTU in Prague, FEE (CZ)

B. Steinbach, TU Chemnitz (DE)

A. Steininger, Vienna University of Technology (AT)

R. Stojanovic, Univ. of Podgorica Montenegro (ME)

J. Strnadel, Brno University of Technology, FIT (CZ)

R. Ubar, Tallinn Univ. of Technology (EE)

Z. Vašíček, Brno University of Technology, FIT (CZ)

I. Vatajelu, TIMA - CNRS / Université Grenoble Alpes (FR)

P. Velan, ICS MUNI (CZ)

W. Zając, Jacob of Paradies University (PL)

Organizing Committee

H. Kubátová, CTU in Prague (CZ)

P. Fišer, CTU in Prague (CZ)

J. Borecký, CTU in Prague (CZ)

M. Novotný, CTU in Prague (CZ)

R. Kinc, AMCA (CZ)

E. Uhrová, AMCA (CZ)

Contents

Keynote 1: Networkmetrics for Network Monitoring and Security 1
José Camacho Páez, University of Granada, Spain

Keynote 2: Reliability evaluation of general purpose and AI processing architectures 2
Prof. Dr. Miloš Krstić, University of Potsdam, Germany, Leibniz Institute for High Performance Microelectronics (IHP)

Keynote 3: Advances in dependable image processing and deep learning 3
Luca Cassano, Ph.D., Politecnico di Milano, Italy

Design and Analysis of Hardware Multithreading in Embedded Systems 4
Florian Guggi

QRV – Design and Implementation of a Quasi Delay-Insensitve RISC V Processor 14
Michael Kurz

Logical vs physical time in Cyber-Physical Systems specified by Interpreted Petri Nets 24
Łukasz Sawicki, Iwona Grobelna, Piotr Mróz and Małgorzata Mazurkiewicz

Precision Landing of Unmanned Aerial Vehicles using Computer Vision 33
Phi V. Nguyen

Traffic Capture Infrastructure 43
Lukáš Hejcman and David Beneš

Classification of network traffic 52
Matej Hulák and Tomáš Čejka

Detection of Cryptomining in High-speed Networks 59
Richard Plný and Karel Hynek

Vision of Active Learning Framework Approach to NetworkTraffic Analysis Research 68
Jaroslav Pešek, Dominik Soukup and Tomáš Čejka

Dataset Quality Assessment in Autonomous Networks with PermutationTesting 73
Katarzyna Wasielewska, Dominik Soukup, Tomáš Čejka and José Camacho

LAURA: Lora enAbled secURe offgrid messAging 77
Dries Martin, Jethro Pans, Md Masoom Rabbani, Jo Vliegen and Nele Mentens

Automated Annotation of Network Traffic with Data from Web Browser 87
Jan Kala and Dominik Soukup

Enhancing Reactive Ad Hoc Routing Protocols with Trust 95
Yelena Trofimova

Sponsors 96

Partners 98

Keynotes

Networkmetrics for Network Monitoring and Security

Speaker: **José Camacho Páez**, *University of Granada, Spain*

Multivariate analysis has been recognized as an outstanding Machine Learning approach in several domains, including industrial monitoring, marketing, weather modeling, bioinformatics, and many more. In this methodology, visualization, interpretation, and data interaction are principal tools for an analyst to understand the problem the data reflects. This locates multivariate analysis as one approach within the area of interpretable machine learning, also referred to as eXplanaible AI (XAI), which has raised a lot of attention in recent years.

Prof. Camacho coined the term networkmetrics one decade ago, referring to the use of multivariate analysis and other interpretable machine learning tools in network applications. It stands for the combination of the application domain (network engineering) and the suffix "-metrics", inherited from other disciplines where interpretable multivariate analysis has been widely adopted, both in academia and industry. This is the case of psychometrics in psychology, econometrics in economy or chemometrics in chemistry. As for today, those areas are so mature that there is a wide body of research with internationally renowned journals, and companies demand specific job positions of psychometricians, econometricians and chemometricians, acknowledging their added value in problem solving. Data interpretability is key in all –metrics disciplines: the data analyst employs multivariate analysis to understand patterns and gain insights into the data, rather than as a black-box model. Network engineering shares with psychology, economy or chemistry the same need for interpretable multivariate analysis, since tasks like network monitoring and troubleshooting, network design and optimization, or network security, have essentially a multivariate nature, and require models that are interpretable by engineers in a network operator center (NOC) or network security center (NSC).

This talk will motivate the use of multivariate analysis in network problems using real examples.

José Camacho Páez

José Camacho is full professor with the Department of Signal Theory, Networking and Communications and researcher in the Information and Communication Technologies Research Centre, CITIC for its initials in Spanish, both at the University of Granada, Spain. He holds a degree in Computer Science from the University of Granada (2003) and a Ph.D. from the Technical University of Valencia (2007). His Ph.D. was awarded with the second Rosina Ribalta Prize to the best Ph.D. projects in the field of Information and Communication Technologies (ICT) from the EPSON Foundation, and with the D.L. Massart Award in Chemometrics from the Belgian Chemometrics Society. He worked as a post-doctoral fellow at the University of Girona, granted by the Juan de la Cierva program, and was a Fulbright fellow in 2018 at the Dartmouth College, USA. His research interests include exploratory data analysis, machine learning and inferential statistics with multivariate techniques applied to data of very different nature, including personalized medicine, molecular biology, chemistry and communication networks. He has authored more than 50 publications in impact journals (JCR), and a similar number of publications in conferences and workshops. He has participated in 20 research projects and collaborate actively in three open software projects in Github.

Reliability evaluation of general purpose and AI processing architectures

Speaker: **Prof. Dr. Miloš Krstić**, *University of Potsdam, Germany, Leibniz Institute for High Performance Microelectronics (IHP)*

In many applications, such as space or automotive, the reliability plays significant role. In addition to usual properties, such as performance and power consumption, such applications need to fulfil additional reliability requirements.
Such evaluation is usually connected with tedious fault injection campaigns that need to be performed at the netlist level. In this talk the alternatives will be discussed, including the use of machine learning for the extraction of reliability critical parameters from digital circuits.
Additionally, the reliability evaluation has been performed on the various processing architectures. Major focus is on AI accelerator architectures, and with this respect two orthogonal technologies have been investigated. On one hand, the classical digital approach has been investigated on open access NVDLA platform. As an alternative, emerging RRAM technology and corresponding crossbar architectures are frequently evaluated in the context of AI acceleration. In this talk, we will indicate the potential of those approach, possibilities for system level verification, but also the reliability threats, such as read disturb.

Miloš Krstić

Prof. Dr. Milos Krstic received the Dr-Ing. degree in electronics from Brandenburg University of Technology, Cottbus, Germany in 2006. Since 2001 he has been with IHP, Frankfurt (Oder), Germany, where he leads the department System Architectures. From 2016 he is also professor for "Design and Test Methodology" at the University of Potsdam. For the last few years, his work was mainly focused on fault tolerant architectures and design methodologies for digital systems integration. Prof. Krstic has been managing many international and national R & D projects (GALAXY, EMPHASE, BB-KI-Chips, IC-NAO, ENROL, RTU-ASIC, SEPHY, DIFFERENT, VHiSSi, RESCUE, etc.). He is also leading and coordinating space activities at IHP. He has published more than 200 journal and conference papers, and registered 9 patents.

Advances in dependable image processing and deep learning

Speaker: **Luca Cassano, Ph.D.**, *Politecnico di Milano, Italy*

There is great interest in employing Image Processing (IM) and Deep Lerning (DL) in a variety of application fields that expose safety- and mission-critical requirements. In these scenarios, it is crucial to assess the reliability of the computing system (composed of the application running on top of a processing platform), to adopt (if necessary) specific hardening techniques. Most of existing approaches for the reliability analysis and for the hardening of these applications still rely on the classical coccrect/corrupted output identification. Nevertheless, when dealing with IM and DL applications designers and safety-engineers have to keep in mind that the application itself may benefit from an intrinsic degree of resilience to faults. Indeed, cases may exist where the final application is still able to correctly carry out its task although some intermediate results are partially corrupted. Therefore, it may be beneficial to analyse the relibility of the application and to harden the system based on the more advanced usable/unusable output classification paradigm that has recently been proposed. The talk will present a number of advanced solutions for the reliability analysis and the hardening of IM and DL applications accelerated onto embedded CPUs and GPUs.

Luca Cassano

Luca Cassano is a Tenure-Track Assistant Professor at Politecnico di Milano, Italy. He received the B.S., M.S. and Ph.D. degrees in Computer Engineering from the University of Pisa, Italy. His research activity focuses on the definition of innovative techniques for fault simulation, testing, untestability analysis, diagnosis, and verification of fault tolerant and secure digital circuits and systems. He published more than 60 papers in refereed international journals and conferences. He served as program chair for DFTS 2021 and he is currently the program chair of DFTS 2022 and he serves in the organizing and program committees of several conferences on EDA, CAD and test (ETS, IOLTS, DDECS, DSD). He is associate editor of Integration, the VLSI Journal and of the Journal of Electronic Testing. With his Ph.D. thesis, titled "Analysis and Test of the Effects of Single Event Upsets Affecting the Configuration Memory of SRAM-based FPGAs", he won the European semifinals of the 2014 TTTC's E. J. McCluskey Doctoral Thesis Award.

Design and Analysis of Hardware Multithreading in Embedded Systems

Guggi Florian
TU Vienna, Embedded Computing Systems Group
Treitlstraße 3, 2nd Floor, A-1040 Wien

florian.guggi@tuwien.ac.at

Abstract. While hardware multithreading is common in desktop processors, its implementations in embedded systems are sparse. In this paper we aim to analyse such architectures by studying previous work and motivating advantages for embedded systems. Additionally, we design an interleaved multithreading RISC-V pipeline with eight thread contexts. The fully synthesizeable processor is evaluated on an FPGA by rendering a simple scene with a raytracing algorithm.

Keywords. multithreading, RISC, FPGA, embedded systems

1 Introduction

With the increasing demand for digital processing in almost all areas of our lives comes a demand for increasing performance of microprocessors. For a long time, this was made possible by advancements in chip manufacturing. The miniaturization of transistors allowed them to be packed more densely onto a die, whilst also decreasing their power consumption and granting higher clock frequencies. The number of transistors on a chip is still increasing, however the clock frequency, and with it the performance of a single-threaded CPU, has started to plateau in the early two thousands. This can be attributed to the issue of heat dissipation. The power consumption in synchronous digital circuits is roughly calculated with the following formula: $P = C \cdot f \cdot V^2 + P_{static}$, where C is the switching capacitance, f the clock frequency, V the supply voltage and P_{static} the leakage power. Keeping in mind, that an increasing frequency also requires a higher supply voltage for stable operation, it can be observed that the power consumption rises almost with the frequency cubed. Hence, alternative ways of increasing performance need to be explored. [1]

One way of achieving better performance lies in parallelization. By exploiting the distributable nature of many tasks, a performance increase can be achieved in multicore systems. This method is readily employed in general-purpose CPUs, where some even propose systems with thousands of cores. [2] However, the incurred overhead from intercore communication and cache coherence protocols demands large enough applications, whose achievable speedup outweighs such expenses.

This can often not be fulfilled in embedded systems, as typical use-cases appear on a smaller scale and impose real-time constraints, which in turn need cycle-accurate and deterministic operation. Nevertheless, the concurrent execution of tasks is often needed and consequently accomplished through multithreading. Many contemporary solutions, such as commonly used RTOSs, rely on software multithreading in order to ensure portability. More sophisticated approaches forfeit this advantage and create highly specific solutions, such as offloading the scheduling of tasks onto an FPGA fabric, while still using a single-threaded processor. [3]

What both approaches have in common, is that they try to work around limitations imposed by a processor which might not primarily be designed for concurrency. The standard architectures seldom support concurrency in hardware, therefore offloading large amounts of work onto software.

The goal of this paper is to explore the potential of hardware multithreading in embedded systems. This is achieved by analysing diverse approaches found in literature and already existing processors. The results are applied to find and entertain use-cases in embedded systems. As a practical example, a processor with interleaved multithreading is implemented on an FPGA. Its architecture is then tested by rendering a simple 3D scene using a highly parallelizable raytracing algorithm.

2 An Overview of Hardware Multithreading Methods

As already briefly discussed in the introduction, the need for hardware multithreading stems from the need to mitigate the effect of events which block the execution path for multiple clock cycles. As most modern applications feature multiple tasks which need to be performed at the same time or can be separated into individual chunks, a simple solution is to continue executing a different task until the original latency cause is resolved. While operating systems can be used to perform such a switch through software, this comes with a significant overhead, even when done on a kernel level. Replacing thread resources combined with large amounts of bookkeeping, requires additional code and can amount to possibly tens of thousands of CPU cycles dedicated to these tasks. [4]

This not only limits the conceivable performance increase, it also completely prevents its application for short latency events. Such events are commonly generated by memory reads, as access times are often magnitudes slower than CPU cycles. As an example, the STMicroelectronics STM32F7x9 series of microcontrollers requires ten wait cycles, when accessing the onboard flash memory at full speed. The use of an instruction and data cache can reduce the penalty considerably, however this is not able to mitigate all events and makes the verification of real-time constraints difficult or rather pessimistic when caches are unused. [5]

Other sources of latency might be multi-cycle instructions or computations offloaded to hardware accelerators. Even if these events only take a small amount of time to resolve, they can easily bottleneck applications that are memory bound or rely heavily on the use of specific computations.

To lessen the software overhead of thread switching and make it viable for short latencies, additional hardware for context swapping is required. By altering the processors' architecture, intrinsic support can be achieved, which does not entail massive resource costs. [6, p. 432] The following sections will provide an overview of possible solutions.

2.1 Common Architectures

To highlight the specific characteristics of the different pipelines, as well as to allow for proper comparison between them, we will base our examples on a classic five stage processor pipeline shown in Figure 1. Rectangles with rounded corners signify a function block; rectangles with sharp corners are registers. In order to retain focus on the important aspects, detailed data and control paths, as well as hazard handling, are omitted but should be kept in mind, as exotic pipeline structures might also require complex control flow.

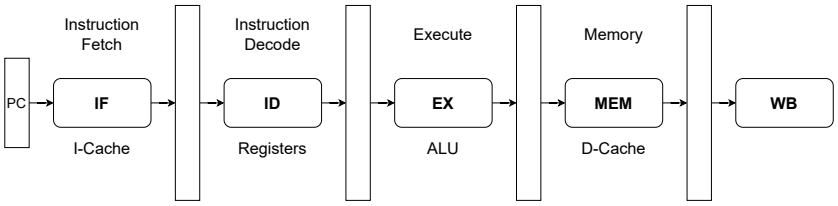


Figure 1: A simple 5-stage processor pipeline

2.1.1 Block Multithreading

Block or coarse-grain multithreading performs the aforementioned thread context switch in hardware. Any resources that define a programs' state, namely the program counter, register file contents, possibly the page-table base register and other flags are simply replicated. A thread selector then dispatches a thread into the pipeline, along with an identifier for selecting the appropriate resources. When the execution reaches a long latency event, said selector is notified, which in turn decides on another thread to run. Figure 2 shows only two individual threads, however many more could certainly be supported.

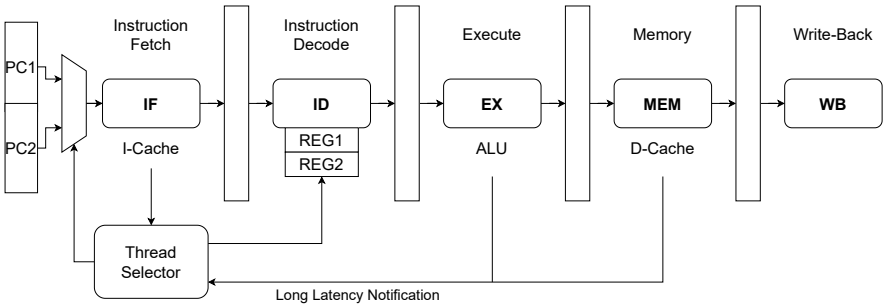


Figure 2: A simplified 2-way block multithreading pipeline

A context switch can now be performed much faster, yet still incurs some overhead. First, the instruction fetch block takes one cycle to load the new program counter. Secondly, depending on where the latency event occurs, parts of the pipeline need to be flushed to enforce proper execution order. In the case of a data cache miss, this includes all but the write-back stage. In total, 5 clock cycles are wasted. As expected, this value grows with the number of pipeline stages and can, therefore, lead to large performance losses with more aggressively pipelined systems.

A way of circumventing flushing costs is to notify the thread selector preemptively. In [7] Le Sueur and Heiser analyse a design in which any instruction that might lead to a long latency event, such as memory access or lock checking, invokes a thread switch. Consequently, the decision can be made right after the fetch stage when the instruction is received. It then progresses through the pipeline, while a different thread follows. Now the overhead only consists of a single cycle which is taken to apply the new program counter. Especially in systems with little or no cache at all, preemptive measures will often be correct and supersede late notifications.

One of the block multithreading architectures' big advantages is its versatility. In a system with many threads it facilitates fast context switching at possible zero cost, asymptotically approaching one instruction per cycle. Even when a specific application only offers a single thread that is ready to run, the performance is not worse compared to a pipeline without block multithreading. [8]

Because a thread can only be scheduled when the currently executing one encounters a

latency event or vacates its spot voluntarily, starvation is possible. Especially with numerous threads, great care needs to be taken to ensure all are executed within their requirements. With appropriate scheduling hardware, it is possible to implement a real-time system. This can be expanded to intrinsically support deadlines, therefore offloading RTOS work into the processor itself. [9]

2.1.2 Interleaved Multithreading

Interleaved multithreading differs slightly from block multithreading. Again, the processor needs to contain program counters, register files and other resources multiple times. The switch between threads however, occurs after each and every clock cycle. This means, that context swapping does not entail any overhead in terms of clock cycles. Depending on the complexity of the design, different scheduling algorithms can be used, ranging from a simple round-robin which ensures predictable execution, to dynamic dispatching from a large pool of threads.

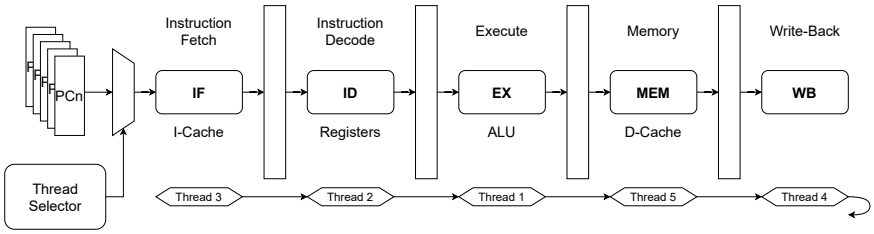


Figure 3: A interleaved multithreading pipeline with 5 threads

This design choice has a big impact on the architecture's behaviour. When the threads' states are ignored, and they are simply dispatched one after another, the number of clock cycles per instruction for each thread approaches n where n is also the total number of threads. While such a design can offer high throughput, it requires all threads to constantly be supplied with work as to not waste their execution slot. Combined with the large sacrifice of single thread performance, this kind of scheduling might only be appropriate for niche applications with large amounts of parallelism. [6, p. 441]

If the architecture contains enough threads, it is possible to simplify the processor pipeline, due to the elimination of data hazards. This occurs, when a thread does not enter the pipeline often enough to create the need for forwarding mechanisms, as any results for previous instructions will already be written back to registers. Additionally, branches come without penalties for the same reason.

In [10] Hoover et al. attempt to design an interleaved multithreaded microcontroller, focused on its applicability for sensor nets. The result is a 6 stage pipeline which supports up to 8 dynamically dispatched threads. The decision on which thread to execute is taken based on the threads' readiness and availability of resources. This gives the microcontroller the ability to appropriately react to the unpredictable nature of many real world tasks and even service interrupts without the otherwise often costly setup and teardown.

2.1.3 Simultaneous Multithreading

The approach taken in SMT varies greatly from other multithreading approaches, as it is possible to issue multiple instructions each cycle. This is achieved by adapting the stages responsible for fetching and decoding instructions, while also having multiple units responsible for calculations and memory access. The result is a processor with the possibility of completing more than 1 instruction per clock cycle. Context swapping is therefore not directly applicable, as different

threads are already executed simultaneously. [11] As complex calculations and memory reads can take a different number of clock cycles to complete, different threads are generally allowed to progress independently of each other. In out-of-order processors, it may even be possible to prefetch instructions for a thread and execute them before the previous ones are retired. This culminates in run-ahead execution, where an otherwise stalled thread continues to speculatively perform instructions in order to preload memory it might need later on. [12]

Figure 4 only shows a very abstracted diagram, as the concrete control mechanisms required for such a design vary greatly, are complex and go beyond the scope of this paper.

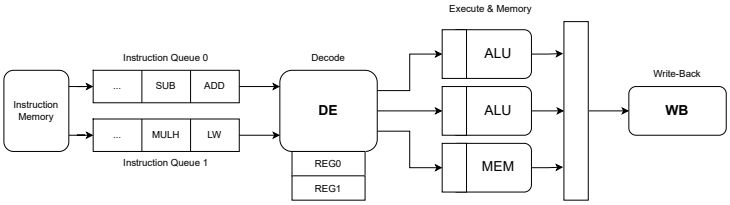


Figure 4: Simultaneous Multithreading

The aforementioned intricacies possible within such designs, are generally difficult to implement in the context of embedded systems, as they take away predictability and complicate verification. Especially the choice of which threads' instructions to fetch and issue, and consequently deciding which thread receives pipeline resources, can massively influence performance. Cazorla et al. show in [13] that this influence is hard to predict as it depends on the individual program that is executed. Nevertheless, much previous work has been done to tackle these shortcomings.

In order to support hard real-time requirements in SMT systems, Siqueira and Kreutz propose an architecture in [14], which guarantees a certain amount of processor resources to every critical thread. This is achieved by scheduling these threads in a round-robin fashion, akin to what is discussed in Section 2.1.2. Additional non-critical threads can only be executed, when the critical one whose turn it is, has to stall. This allows processor resources to be used when they would otherwise be sitting idle. A problem with such design is the starvation of non-critical thread, which the authors are aware of, but do not address.

2.2 Examples of Application

The following section aims to provide an overview on the usage of hardware multithreading in past and current processors. As such architectures are not common in the embedded systems space, the discussion will not be limited to the ones therein.

2.2.1 Historical Uses

Some of the earliest computers readily employed multithreading mechanisms. For example the DYSEAC [15], built in 1954 by the U.S. Army, featured a system akin to block multithreading. It allowed the programmer to choose between two program counters, either explicitly through an instruction or automatically on certain I/O operations. As these operations might include things like printing with a mechanized typewriter, such events take a significant amount of time, compared to sub-millisecond arithmetic instructions.

In order to hide memory access times, the CDC 6000 from the Control Data Corporation used a form of interleaved multithreading. In this design a selector rotates through ten unique peripheral processing units, allowing the faster central processor to access slow memory and I/O

peripherals in a round-robin fashion. As shown in Figure 5 the design is not equal to the one described in Section 2.1.2, but still follows the same underlying principles. [16]

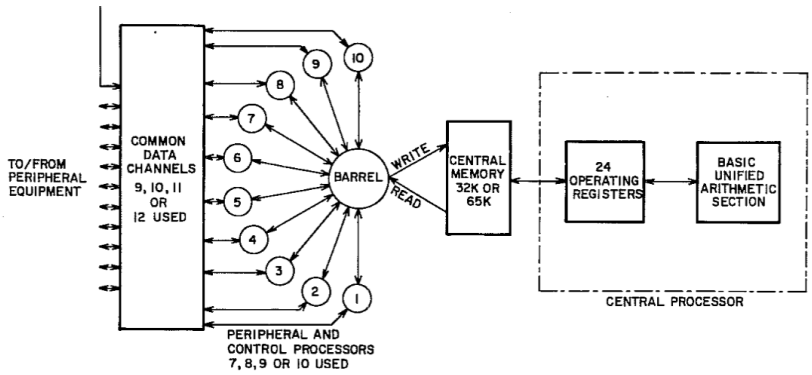


Figure 5: The CDC 6000s block diagram. Taken from [16]

2.2.2 Modern Multithreading

The defacto standard for modern desktop processors is simultaneous multithreading. Intel coined the term hyperthreading for their implementation of SMT in the early two thousands, which since then, is highly used in their products. Each processor core is augmented with an additional thread context, allowing the programmer to treat it as if it were two separate cores. However, unlike the large amount of resources needed for a whole separate core, the die area increases only by about 5% for the extra thread context. The system can issue multiple instructions from two different threads with every clock cycle, which leads to much greater utilization of execution resources. [17]

The counterparts from AMD use a similar form of SMT, with two threads per core as well. Unfortunately, not much information about their products can be found publicly. [18]

3 Implementation of a Barrelprocessor Architecture

In this section an implementation of the previously discussed interleaved multithreading architecture is presented and analysed. The design features a RISC-V base instruction set, extended with application specific custom instructions. Written in VHDL code, the result is a fully synthesizeable processor, which is evaluated on an FPGA, by rendering a simple 3D scene with a raytracing algorithm. The following sections will outline the processors' architecture and describe how its performance was evaluated.

3.1 Processor Architecture

The architecture chosen has an eight-stage pipeline, following the general scheme found in Figure 3. Eight threads are supported in hardware, which enter the pipeline in a strict round-robin fashion. Each has its own program counter and register file, but shares all the other resources of the pipeline. A block diagram can be found in Figure 6.

Having eight pipeline stages and eight threads means, that each thread will only be present once within the pipeline. This simplifies the design considerably, as no data or control flow hazards can occur and therefore no forwarding or other special control needs to be implemented.

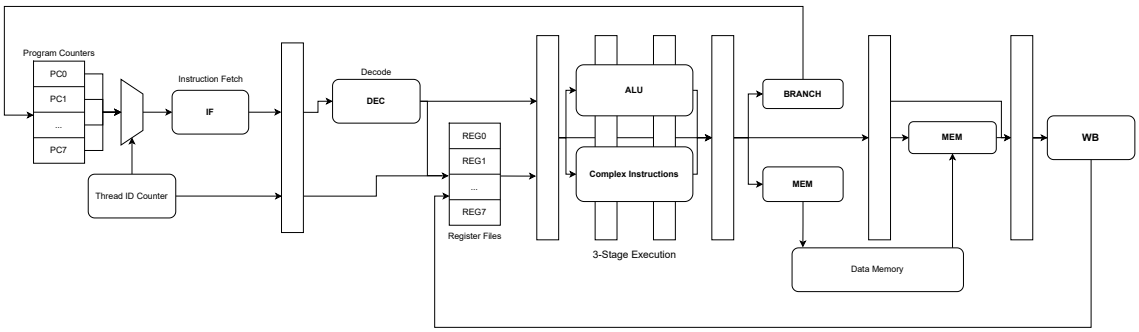


Figure 6: Block diagram for the barrelprocessor

In order to speed up arithmetic calculations, a fixed point (Q16.16 format) multiply and divide instruction is realized in hardware. While the former can fit into the three-phased execution, the latter needs more than 40 cycles to fit onto the target platform. To avoid stalling the pipeline in our architecture, this latency is masked through the help of the strict scheduling algorithm and fast memory system. The divider is pipelined to perform a division within exactly 42 clock cycles, which is the time it takes a thread to execute five instructions. This can be exploited by having the software start the division and afterwards inserting an appropriate number of NOPs or other instructions before fetching the result.

In the instruction fetch stage, the next instruction is fetched from memory and the program counter incremented. An identification for the thread, named thread id, is produced and passed onto the next stage. The decoding and accessing of registers happens in the following cycle. Here reside the 32 registers for each thread whose relevant contents are moved onwards to a three-phase execution. The two extra cycles allow for complex instructions to be executed without the need to stall the pipeline.

Memory access is performed over two stages. Provided access only takes one clock cycle, which is guaranteed in this design, the desired address can be applied, and the result read immediately thereafter. In order to allow inter-thread communication and supply each thread with its own data memory, the pipeline calculates the addresses according to a special layout, which includes a shared and individual space. Any memory access up to a certain constant reaches the same memory (i.e. the shared memory), from there on, each thread accesses its individual block.

The last stage is used to write back results to the registers.

3.2 Platform Design

As discussed in Section 2.1.2, an interleaved multithreading design requires a highly parallelizable task to properly display its advantages. The chosen raytracing algorithm can be computed separately for each pixel of the desired scene, therefore making it a good candidate. In order to simplify development, and because it is sufficient for the required observations, the renderer only supports spheres and a limited number of materials. The desired result can be seen in Figure 7.

The system is synthesized using Quartus and fitted for the Altera Cyclone IV 4CE115 FPGA, using a 100MHz clock. As the implemented design only has very rudimentary debugging features, a NIOS II softcore processor is used to bootstrap and control it. Once the system is running, the barrelprocessor writes its results directly to an external SDRAM chip which is used to store the rendered scene. The execution time is measured by an instantiated timer which is stopped once the NIOS core detects a completion flag from every thread. The outlines of the entire system are shown in Figure 8.

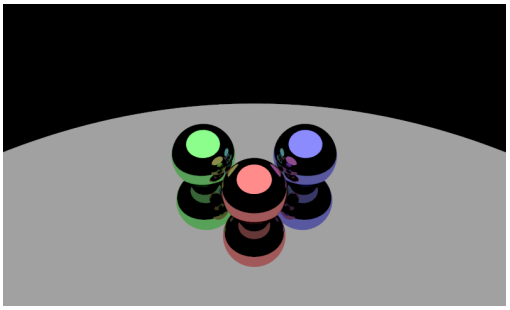


Figure 7: A simple 3D scene to render with raytracing

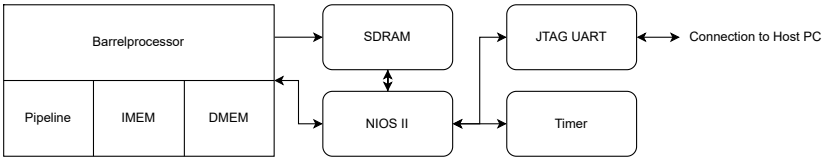


Figure 8: A diagram showing the complete system

3.3 Results

The performance of the barrelprocessor is evaluated by measuring the runtime of the discussed raytracing algorithm and analysing its FPGA resource usage. The same measurements are made for the NIOS 2 softcore processor, which is used as a single-threaded reference, as it is similar in features. For proper comparison, the NIOS 2 is extended with custom instructions containing the exact same logic, as the hardware multiplier and divider of the barrelprocessor. [19]

3.3.1 Resource Usage

A true comparison of the FPGA resources used is difficult, as the features of the two designs differ and are not treated equally by the synthesizer and fitter. Table 1 shows a summary extracted from the Quartus analysis and synthesis report. This was chosen to allow for a more general comparison than the values provided in the fitter report, which are specific to the used FPGA. Additional entries are provided which contain values that are adjusted to more accurately reflect resource usage. They are calculated by assuming that both designs implement their registers in dedicated memory blocks and removing caches and debugging features from the NIOS 2. This shows a moderate increase of ALUTs and the expected eight-fold increase in memory bits that are required for register duplication. While these values can indicate a certain trend in resource usage, their accuracy is questionable, as the processors do not implement the same instruction set and the adjusted values are only an approximation.

3.3.2 Runtime

The time required to render the raytracing scene is measured over 100 times and varying the camera angle and position slightly with each run. The average execution time for the barrelprocessor results to 6.4 seconds, while the NIOS 2 takes 14.06 seconds. This depicts a speedup of 2.2 with the multithreaded architecture and can be attributed to penalty-free branching and the reduced cost of divisions and multiplications. During the 42 clock cycles it takes to complete a division, the NIOS pipeline is stalled. Even though the barrelprocessor

	Barrelprocessor	NIOS 2
Combinational ALUTs	12096	5772
Dedicated Logic Registers	12770	5270
Memory Bits	0	221440
Combinational ALUTs (adjusted)	6180	5479
Dedicated Logic Registers (adjusted)	4834	4999
Memory Bits (adjusted)	8192	1024

Table 1: Resource usage comparison between the barrelprocessor and NIOS 2

needs the same amount of cycles, it can continue to execute instructions from a different thread throughout and even issue new divisions.

It is important to note, that the performance gain is only possible, because only the highly parallelizable raytracing algorithm is benchmarked. Any strictly sequential computation would incur massive penalties due to the bad single-thread performance of such an interleaved multithreading design. However, for such workloads a barrelprocessor is certainly not the right choice, Hence, it seems justified to pick a benchmark scenario that favors multithreaded architectures.

4 Conclusion

The results of our work show, that multithreading architectures can offer great benefits in terms of performance and resource utilization at only a small to moderate increase in hardware costs. While block and simultaneous multithreading architectures can easily provide single-threaded performance close or equal to their non-multithreaded counterparts, interleaved multithreading architectures enter a trade-off between single-thread performance and increased throughput. However, such designs are only useful for applications where parallel execution dominates sequential.

While the market for desktop processors is dominated by simultaneous multithreading architectures, other hardware multithreading designs might prove useful for embedded systems, as they allow for good parallel performance, whilst preserving deterministic behaviour and having low hardware costs. Previous work has shown, that especially combinations of different archetypes can be beneficial, as they compensate each other's disadvantages. The drawback of such approaches is the highly specific programming model, which requires greater effort in creating a software toolchain, than standard architectures.

References

[1] I. L. Markov, "Limits on fundamental limits to computation," *Nature*, vol. 512, pp. 147–154, Aug 2014.

[2] S. Borkar, "Thousand core chips: A technology perspective," in *Proceedings of the 44th Annual Design Automation Conference*, DAC '07, (New York, NY, USA), p. 746–749, Association for Computing Machinery, 2007.

[3] I. Bahri, M. Benkhelifa, and E. Monmasson, "Hw-sw real-time operating system for ac drive applications," in *International Symposium on Power Electronics Power Electronics, Electrical Drives, Automation and Motion*, pp. 194–199, 2012.

[4] L. Sawalha, M. Tull, and R. Barnes, "Hardware thread-context switching," *Electronics letters*, vol. 49, no. 6, pp. 389–391, 2013.

[5] STMicroelectronics, *RM0401 Reference Manual*, 2018.

[6] M. Dubois, M. Annavaram, and P. Stenstrm, *Parallel Computer Organization and Design*. USA: Cambridge University Press, 2012.

[7] Y. Lu, S. Sezer, and J. McCanny, "Design and analysis of an advanced static blocked multithreading architecture," in *23rd IEEE International SOC Conference*, pp. 169–173, 2010.

[8] H. V. Caprita and M. Popa, "Design methods of multithreaded architectures for multicore microcontrollers," in *2011 6th IEEE International Symposium on Applied Computational Intelligence and Informatics (SACI)*, pp. 427–432, 2011.

[9] A. Pulka and A. Milik, "Multithread risc architecture based on programmable interleaved pipelining," in *2009 16th IEEE International Conference on Electronics, Circuits and Systems - (ICECS 2009)*, pp. 647–650, 2009.

[10] G. Hoover, F. Brewer, and T. Sherwood, "A case study of multi-threading in the embedded space," in *Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, CASES '06, (New York, NY, USA), p. 357–367, Association for Computing Machinery, 2006.

[11] M. Nemirovsky and D. Tullsen, *Multithreading Architecture*. Synthesis Lectures on Computer Architecture, Morgan & Claypool, 2013.

[12] A. Naithani, J. Feliu, A. Adileh, and L. Eeckhout, "Precise runahead execution," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 397–410, 2020.

[13] F. Cazorla, P. Knijnenburg, R. Sakellariou, E. Fernandez, A. Ramirez, and M. Valero, "Enabling smt for real-time embedded systems," in *2004 12th European Signal Processing Conference*, pp. 1341–1344, 2004.

[14] H. Siqueira and M. Kreutz, "A simultaneous multithreading processor architecture with predictable timing behavior," in *2018 VIII Brazilian Symposium on Computing Systems Engineering (SBESC)*, pp. 62–66, 2018.

[15] A. L. Leiner, "System specifications for the dyseac," *Journal of the ACM (JACM)*, vol. 1, no. 2, pp. 57–81, 1954.

[16] C. D. Corporation, *Control Data 6000 Series Coputer Systems*, 1979.

[17] D. T. Marr, F. Binns, D. L. Hill, G. Hinton, D. A. Koufaty, J. A. Miller, and M. Upton, "Hyper-threading technology architecture and microarchitecture.," *Intel Technology Journal*, vol. 6, no. 1, 2002.

[18] I. Advanced Micro Devices, "AMD "Zen 3" Core Architecture." https://www.amd.com/en/technologies/zen-core-3, 2022.

[19] Altera, *Nios II Classic Processor Reference Guide*, 2016.

QRV – Design and Implementation of a Quasi Delay-Insensitive RISC-V Processor

Michael Kurz
Embedded Computing Systems, TU Wien
Silbergasse 8, 7+8, 1190 Vienna, Austria

michael.kurz@tuwien.ac.at

Abstract. This paper presents an FPGA-synthesizable asynchronous processor implementing a basic RISC-V instruction set. The proposed design follows the quasi delay-insensitive design paradigm, which offers high robustness with regard to process, voltage and temperature variations and only requires relatively few timing constraints. The asynchronous processor core is able to interface with the synchronous on-chip memory of the target FPGA and allows for in-system programming and communication over jtag-uart.

Keywords. QDI, Asynchronous Circuits, RISC-V

1 Introduction

Integrated circuit design is pushing its boundaries especially with the ongoing trend for smaller feature sizes and voltages leading to greater timing uncertainties and an increasing likelihood of upsets caused by (cosmic) radiation. One possible way to tackle these issues are asynchronous and especially Quasi Delay Insensitive (QDI) circuits, which are known for their robustness against timing variants. For studies regarding their fault tolerance a framework called Python Production Rules (PYPR) has been developed, which is capable of generating QDI circuits [1]. To show the capabilities of this tool and at the same time also make asynchronous design more accessible a processor has been developed according to the goals presented in Section 1.1.

1.1 Design Goals

- Simplicity
- Support of basic RISC-V instruction set
- Allow fault injection studies on core
- Ability to run in hardware with as little modifications as possible
- Peripheral devices for illustrations purposes accompanying to lectures

Simplicity led to a single token implementation as it is by concept hazard free and avoids pipelining. Necessary adjustments to run the design on an FPGA are outlined in Section 5.

1.2 Background and Related Work

Asynchronous circuits exist in many flavours, in the context of this paper Bundled Data (BD) and QDI circuits are important. While synchronous circuits use a global clock network, asynchronous circuits

utilise handshaking. Channels bundle data signals with an acknowledge and (if not implicitly encoded into data) also a request signal. Data signals are used to transport information from sources to sinks (buffers for example). Sources notify sinks about completeness by asserting the request signal and sinks notify sources about reception using the acknowledge signal.

	d.t	d.f
Empty (”E”)	0	0
Valid ”0”	0	1
Valid ”1”	1	0
Not used	1	1

Table 1: Dual Rail (DR) symbol mapping

a	b	y
0	0	0
0	1	no change
1	0	no change
1	1	1

Table 2: Truth table of a C gate

BD is more closely related to synchronous circuits as delay matching is performed for each request path similar to synthesis tools trying to adjust skew but explicitly.
QDI on the other hand encodes the request signal into the data lines enabling the receiving end to detect completeness of data by looking at data lines only. Among many possible encodings, DR is commonly chosen for the sake of simplicity by encoding symbols as shown in Table 1. Several DR signals can be grouped into a channel while each signal is either zero or valid. Channels in the context of QDI are valid if all associated signals are valid and zero if all of them are zero. With this encoding it is possible to separate data tokens (valid channel) by zero tokens / spacers (all signals are zero).
The purpose of completion detectors is to assert the output only if a channel transitioned to a token and deassert if the channel transitioned to a spacer requiring a truth table as shown in Table 2. To implement such behaviour the C gate[1] is used which is an important circuit element in asynchronous design.
The architecture of the processor core introduced in this paper will be presented and discussed on the data-flow level. This circuit representation corresponds to the register-transfer level of the synchronous domain and allows to abstract away certain implementation details (like e.g., the handshaking protocol or the actual data encoding), which makes circuit design more efficient. The asynchronous circuit framework PYPR, we are using in this paper, supports this design entry. PYPR is an open source python framework developed to accelerate/enable studies on fault tolerances of asynchronous circuits. Internally PYPR uses a circuit representation inspired by Production Rule Set (PRS) enabling inspection of generated circuits for debugging purposes [1]. The tool is actually built around this representation simplifying the generation of asynchronous circuits and is also able to convert them into VHDL for logic simulation and also FPGA synthesis. Furthermore it is able to generate combinational logic circuits from Verilog (leveraging yosys for synthesis and technology mapping internally).
On the data-flow level the system is viewed as a network of operations connected by channels. Data flows through this system in the form of tokens. An operation can e.g., be a buffer (which can store tokens), a combinational transformation or a so called control-flow element. The latter allows to direct the flow of tokens through the network, either unconditionally (fork/join) or conditionally (MUX/DEMUX) based on the value of some token. In this paper we are using the notation shown in Figure 1. Channels are always drawn as black arrows.
The source connected to a fork may only issue a new token if all of the sinks have acknowledged token reception. Only by then the fork is allowed to assert acknowledge for the source too.
Join elements on the other hand combine all signals of the input channels into one output channel. Therefore a token at the sink can only be detected as complete if all input tokens are complete. Sink's acknowledge is concurrently forwarded to all sources.
Multiplexers include another channel called *select*. A token on the output is only issued after there are tokens on both *select* and the selected input channel, an acknowledge from source needs to be forwarded

[1]The Muller C-element or short C gate is a fundamental gate in asynchronous logic. Its function is to output the logic level seen at its inputs when these match, and to retain the last valid output state otherwise.

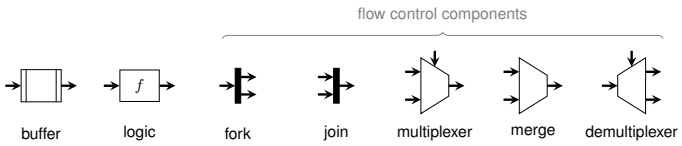


Figure 1: Data Flow Graph (DFG) Elements

to both of them.
Merge is a special case of a multiplexer with mutually exclusive input channels and therefore *select* can be omitted.
Lastly, demultiplexers perform the reverse operation of multiplexers forwarding input tokens to the selected channel by *select*. Again a token on the output can be issued only if there are tokens on both the input itself and *select*.
For a more formal definition and in-depth discussion we refer to [2].
There are also other processors implemented in QDI, Vortex [7] or DD1 [6] to name a few examples.

2 Architecture

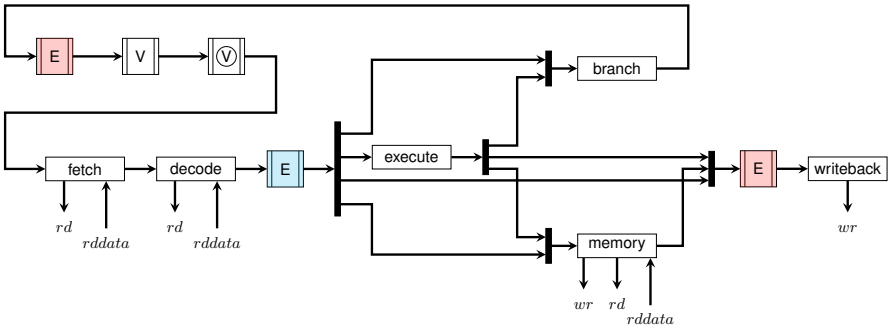


Figure 2: Core overview

The overall design is of our processor is kept close to a traditional, in-order, five-stage execution pipeline architecture (fetch, decode, execute, memory, writeback)[5]. However, as illustrated in Figure 2 we don't use buffers after each stage, as we don't strive to implement pipelining. The processor only executes one instruction at a time, which corresponds to a single token rotating in the three buffer stages shown on the top-left of the figure.
Processors are in general comprised of the core and the interfaces to its environment (ROM, RAM and peripherals). Using a DFG to design the core presents a challenge here as there is no inherent possibility to directly specify and generate large memories. In principle one could implement memory using a ring of buffers which would compare to register based memories in the synchronous world. However, this strategy consumes a lot of logic (mainly in the form of C gate) and does not present an efficient use of FPGA resources.
Hence, we decided to keep the memories (including the register file) outside of the core. The core then uses input/output channels to communicate with these memories and e.g., fetch an instruction or read the register file. This approach leave the core as a standalone platform independent entity, which facilitates porting it other possible hardware targets. Moreover, it can easily be used for fault injection/masking studies (one of our goals for the core) while at the same time allowing to illustrate the workflow and get hands-on experience by extending it with peripherals.

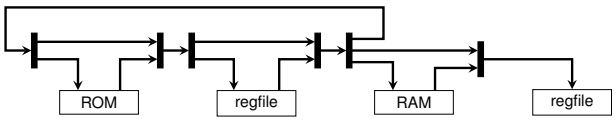
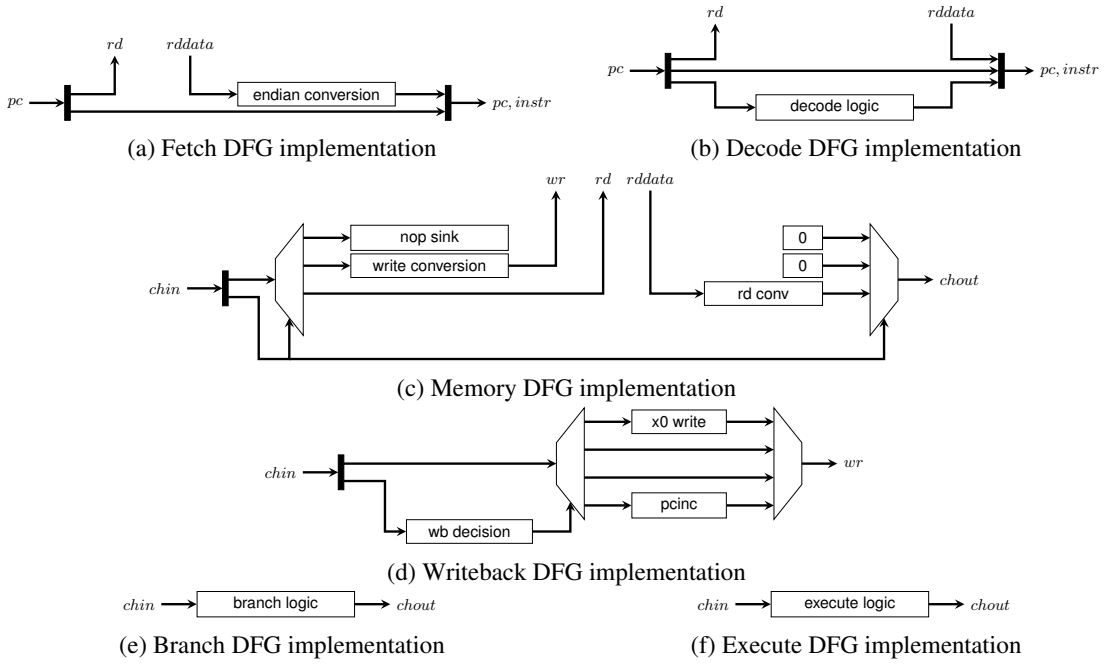


Figure 3: Abstract overview



(a) Fetch DFG implementation

(b) Decode DFG implementation

(c) Memory DFG implementation

(d) Writeback DFG implementation

(e) Branch DFG implementation

(f) Execute DFG implementation

Figure 4: Core modules

As a result this project consists of the core and its interfaces to the ROM (Section 3.1), RAM (Section 3.2) and regfile (Section 3.3). For simulating memories only the core itself needs to be simulated and the memories/peripherals are implemented in the testbench. For synthesis (platform-dependent) wrappers for FPGA internal memories are used which are entirely written in VHDL without going through PYPR.

The logic of the Execute and Decode stage is implemented in Verilog and converted to a DR QDI circuit using PYPR's integrated logic synthesis tools, which enormously speeds up the design process.

Analysing the topology of the core's DFG results in the flow graph presented in Figure 3, giving an abstract insight on the token flow. Note that for used handshaking protocol every loop requires at least three buffers in its path [2]. Furthermore, there is a chain of memories (i.e., I/O channels) generating and consuming tokens. However, they don't form loops and, therefore, no necessity for buffers arises.

In principle the three required buffers can be placed arbitrarily as long as they are in the loop described earlier. Though, Section 3 will point out that this is not the case here. Depending on the actual placement a deadlocked circuit could be the result. At least the buffer containing the start-address token should be located directly ahead fetch to conveniently set reset vectors (which in this case consists of the start address only). As this part of the loop contains the least amount of bits it is at least resource wise desirable to also place the other buffers here. Two functional placements are shown in Figure 2 and discussed in detail in Section 3.3 as the chosen implementation of the regfile plays an important role in determining the placement.

2.1 Fetch

Figure 4a shows the implementation of fetch. After token reception (containing the program counter) it forks and issues a query to ROM before joining the program counter and endian converted read result. Two channels, namely *rd* and *rddata* are used whereas *rd* acts as token sink requiring *rddata* to act as token source to prevent a deadlock. Endian conversion has been written in Verilog and is converted to DR QDI before it is added as combinational function block into fetch's DFG.

2.2 Decode

Decode heavily relies on automated logic conversion from Verilog. An implementation using data flow components is (regarding speed) not beneficial here as speed gains in asynchronous circuits are mostly due to average case performing better than constant worst case (which is the case for synchronous designs). The instruction to be analysed is forwarded from fetch and directly used to query regfile memories as implemented in Figure 4b. Decode is responsible for generating control signals telling all subsequent modules what actions to perform.
The interface to the regfile is slightly different as two registers have to be queried at once as outlined in Table 3.

2.3 Execute

For the sake of simplicity execute is entirely written in Verilog too. This module definitely has lots of optimisation potential leveraging the advantages of asynchronous circuits. By using the DFG flow elements MUX and DEMUX it is possible to alter the flow and adding optimised logic for specific instructions. For now execute is implemented as shown in Figure 4f and features ALU operations as well as calculation of branching decisions.

2.4 Memory

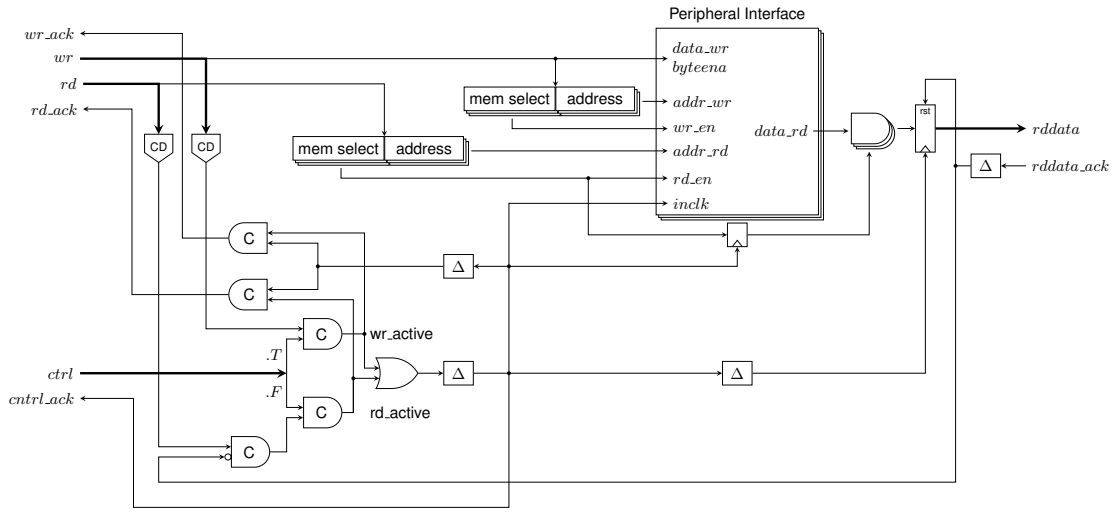
Memory finally shows what asynchronous design is capable of. Figure 4c illustrates token flow and makes clear that there are three different cases to consider:

1. On **mem_nop** no memory operations is performed. Input token consumption as well as output token generation take place simultaneously as soon as *select* is complete. An output token could even be generated before its corresponding input token is consumed if *select* transitions to data phase faster than input data of DEMUX.
2. **mem_wr** requires data to be converted first as specified by mem_op. The corresponding output token on the other hand can be issued immediately. Input channel acknowledge is delayed by memory access time though.
3. **mem_rd** is most expensive. Both tokens have to be converted and acknowledge/token output are dependent on memory access time (actually delayed by the matched delay).

In comparison to synchronous design where worst case timing determines maximum frequency it is crucial to note that asynchronous design has average case performance whereby average depends on the code which is executed as can be seen in Section 5

2.5 Writeback

Writeback selects one of its data inputs depending on *wb_op* and writes to the regfile unconditionally. In case of a nop writeback operation it writes zero to regfile register 0 which by the specification of RISC-



CD blocks represent completion detectors as outlined in Section 1.2

Figure 5: Converting QDI to BD: RAM-Interface

V will get discarded leading to an expensive nop. Section 3.3 justifies this design decision depicted in Figure 4d.

2.6 Branch

Based on *br_op* branch increments the current program counter by four or returns *pc_new* calculated by execute. Figure 4e shows the implementation which again is implemented in Verilog and instantiated as function block.

3 Interfaces

For simulation purposes all the interfaces (ROM, Regfile, RAM) are emulated by the testbench in contrast to FPGA synthesis where they connect actual peripherals to the CPU. Device specific components are: On-chip ROM [3] or a JTAG interface. By using instantiation templates [4] it might be possible to avoid some of these platform dependencies. Although memories could theoretically be generated using buffer rings, this variant is not feasible at all for larger memories. On-chip resources (memory) are made for synchronous designs which much better relates to BD-channels than to DR encoded QDI channels. Recall: BD utilises (single rail) data lines with matched delays.
The delays have to be set in a way that fulfil timing requirements for the used resources (t_{SU}, t_{H} and if the circuit was fast enough also f_{max}). In case of an FPGA those delay elements do not exist and need to be created from gate and wire delays whereas an assembly thereof is usually called *delay-chain*. Without constraining location of the gates used, the actual delay is rather unpredictable but works surprisingly well. Although some speed in terms of maximum frequency is lost it contributes to simplicity - one of the main design goals. Moreover using configurable length delay-chains it is possible to speed up the CPU and allows fine tune delays after the design is routed.
Conversion is done for all interfaces as depicted in Figure 5 with variances in control logic and memory selection. All QDI to BD conversion modules provide the same interface for peripheral connection as described in Table 3. Which peripheral is accessible at which address is configured by a specification file. For read operation the module itself also converts BD data back to QDI.

Signal	Dir.	Type	Description
reset	in	std_logic	Reset (high active)
inclk	in	std_logic	Clock for rd/wr operation
wr_en	in	std_logic	Write on rising_edge(inclk), mutually exclusive with rd_en
rd_en	in	std_logic	Read on rising_edge(inclk)
addr_rd	in	std_logic_vector	Read address
data_rd	out	std_logic_vector	Data read from peripheral
addr_wr	in	std_logic_vector	Write address
data_wr	in	std_logic_vector	Data to write
byteena	in	std_logic_vector	Byte-enable for sub-word writes
<io-signal>	<dir>	std_logic(_vector)	Peripheral i/o, forwarded to top entity

Table 3: Peripheral Interface Description

3.1 ROM

This module only allows read access featuring *rd* and *rddata* channels as interfaces to the core. Implementation of ROM interface is similar to that of RAM shown in Figure 5 omitting the *wr*-Channel. Section 3.2 presents an in-depth explanation.

3.2 RAM

This module provides both read and write access using *rd*, *wr* and *rddata* channels as interfaces to the core. Notice that RAM does not use the *cntrl* channel as presented in Figure 5 due to read and write access being mutually exclusive which allows to remove the C gates *cntrl* is connected to. Alternatively it is also possible to wire the Completion Detector (CD) inputs to both inputs of their respective C gates.

3.2.1 Write Access

If the *mem-select* part of a given write address matches *wr_en* will get asserted for the respective memory and the delayed *inclk* pulse will lead to a write operation. Access to unmapped memory is ignored as *wr_en* will not get asserted at all. An *outclk* pulse will be generated although due to exclusive *rd* or *wr* access there currently has to be a spacer at *rd* which will mask all memory outputs as *rd_en* is deasserted for all memories. Furthermore *wr_ack* is delayed relative to *inclk* to satisfy peripheral's hold times. Recall that conversion from DR signals to BD is done by simple stripping false rails.

3.2.2 Read Access

Reading is quite similar to writing although the need for an output multiplexer arises. This is required because each memory continuously outputs the data from the (read) address accessed lastly (or the reset address in case it has never been read from yet). There is no guarantee that the input data is still valid on arrival of *outclk* as *rd* could get an acknowledge earlier than *outclk* is asserted thereby requiring to register *rd_en* for selecting the correct memory. At the output register the retrieved data is converted back to DR by either asserting true or false rail for each bit.

3.2.3 Memory Selection

Memories are selected by their offset specified in the memory mapping specification file. Read and write addresses are split into address and select part whereas address part is forwarded to the respective memory and the select part is compared against the given offset in the specification file. Only multiples of memory word lengths are permitted as offset to avoid interference of addressing with offset comparison.

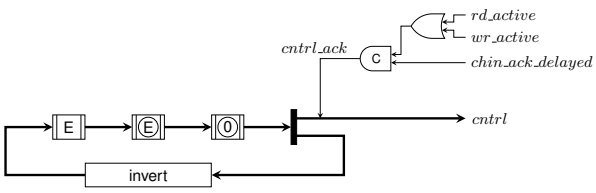


Figure 6: Control Signal for Regfile

Peripherals act as memory devices and are instantiated directly by this conversion module according to the specification file.

3.3 Regfile

The regfile is special in several aspects:

- Two addresses are read at once and, therefore, also two signals for the read result are needed
- Two different modules (decode, writeback) need access in specific order

The first issue can easily be solved by instantiating two memories as peripherals, omitting memory selection and using additional signals for (read) address and (read) result to interface with both memories at once. The second issue is a bit tricky as it, in fact, implies two requirements:

- Regfile write of current instruction must always happen before regfile read of next instruction
- *rd* and *wr* tokens can occur concurrently since transition of *rd* to spacer depends on ack from regfile. Though unlikely, *wr* from writeback could arrive before *rd* transitions to spacer. Just as well as the next *rd* token could arrive while *wr* is in progress. It is guaranteed that a *rd* token arrives first, though!

Using round robin access is a simple solution to tackle in order regfile access which comes at a cost: Writeback needs to issue a write token in any case regardless of *wb_op*. The chosen implementation extends QDI to BD conversion as shown in Figure 5 by the external control token generator as illustrated in Figure 6. It has one limitation tough: Any *rd* or *wr* token's spacer must have been acknowledged before access can be transferred to the other channel. An additional C gate (shown in Figure 6) enforces this requirement as otherwise delayed *inclk* for *rd_ack* or *wr_ack* could still be asserted resulting in an immediate acknowledge of the next token.
This design also influences buffer placement and leads to the examples presented in Figure 2. To raise the abstraction level, any solution with a buffer between decode and writeback works by allowing the buffer to acknowledge *rddata* early thus allowing *rd* to transition into spacer phase and thereby handing over control to the *wr* channel. Transitioning control back to the *rd* channel follows naturally with *wr_ack* issued from regfile.

4 Verification

Verifying complex designs is hard but having a working reference can make this task significantly simpler. RISC-V[5] is getting a lot of attention both in research and industry and as a result many (open source) implementations are available. qemu-riscv has been chosen as reference implementation due to its widespread use lowering the risk of bugs.
On program execution the outcome of the current instruction depends on the data stored in the regfile, the instruction itself and eventually on data from RAM. Taking a closer look it is clear that the core

```
Instruction(address=0x20000000, instruction=0x80040137, wb_reg=0x02, wb_val=0x80040000)
Instruction(address=0x20000004, instruction=0xffc10113, wb_reg=0x02, wb_val=0x8003fffc)
```

Figure 7: Example output of verification script

itself has no state as the regfile is an external memory too. The main idea of our testing approach is to log instructions (including their ROM address) in combination with the writeback command (if any) and compare this data with the output produced by qemu-riscv.
By stepping through a program with gdb all necessary information can be extracted:

- program counter, instruction: by issuing `disas /r $pc, $pc+4`
- writeback address, value: comparing all register values (`i r`) before and after execution of instructions

The only case that cannot be differentiated is writing back the same value in comparison to not writing back at all. In this case writeback information of the logic simulation is discarded too.
The same executable (actually its .text and .data sections) is now run on the UUT by a VHDL simulator, GHDL in this case. Report statements allow to keep a log of ROM reads and Regfile accesses.
A python script is used to automate this process executing both riscv-gdb and GHDL and allows dumping the results to files. By using `diff` for example differences are easily detectable easing localisation of the first faulty instruction including program counter and the (faulty) writeback operation. This enormously supports debugging as the misbehaving instruction can be identified immediately. Using gdb-riscv-objdump and navigating to the address given yields a human readable version of the instruction to further narrow down the part causing the failure.

	Address	Notes
DMEM_BASE_ADDRESS	0x8000_0000	
RODATA_BASE_ADDRESS	0x2004_0000	logic simulator just maps dmem to this region too
FLASH_BASE_ADDRESS	0x2000_0000	
UART_BASE_ADDRESS	0x1000_0000	
HALT_EMULATOR_ADDRESS	0x0010_0000	writing 0x5555 stops emulator

Table 4: Address layout specified by qemu-riscv

By using the same memory map in logic simulation as qemu-riscv (see Table 4) it is possible to compare instructions and regfile access by simply diffing the log files. qemu-riscv also maps a UART peripheral and uses a special address to stop the processor. The testbench therefore not only has to emulate memory but also the UART interface and stop command. The script produces an output similar to Figure 7.

5 Results

PYPR generates a netlist consisting mainly of C gates and other basic gates. The C gates are mapped to single LUTs of the target FPGA using a handcrafted and platform dependent instantiation template. It is important to prevent the FPGA synthesis from making any changes to the circuit. As the synchronous synthesis tool has no understanding of asynchronous design principles, changes may lead to the violation of design rules. Although mapping C gates to cells was enough to get the core functional, all logic functions are mapped for completeness and to prevent the synthesis tool from optimising logic possible generating logic violating the asynchronous fundamentals.
Synthesis of combinational DR logic blocks can, among others, be implemented using DIMS[2] or NCLX[3] design style. Table 5 compares resource usage and processing speed whereas only resources for the core

[2]Delay Insensitive Minterm Mapping, for more information see [2]
[3]Variant of Null Convention Logic with explicit completion detection, see [1] for details

itself are taken into consideration. Average speed is determined by measuring runtime of an md5 algorithm similar to the one used for verification but with more iterations. Additionally maximum frequency is measured by examining acknowledge toggle rate using a simple infinite loop to avoid memory access.

Design Style	Cell Library	Resource Usage (LUTs)	Maximum Frequency [MHz]	Average Runtime [ms]
DIMS	default[1]	16036	4.2	4.04
NCLX	default[1]	13892	5.5	3.13
NCLX	FA[2]	14394	5.5	3.17

[1] using only *and*, *xor* variants for specified logic style
[2] Full Adder using *and*, *xor* and an NCLX full adder

Table 5: Comparing Metrics of different technologies

Table 5 confirms that the DIMS version is outperformed both resource and speed wise compared to NCLX. The slightly faster execution of the default NCLX variant could be due to measurement inaccuracies or slightly faster execution not detectable by frequency measurement. Area wise the default variant is slightly better than the one incorporating full adders.
Frequency improvements might be possible by fine tuning the delay chains in the memories. By optimising regfile access (e.g. no regfile lookup on unconditional jumps) execution speed for short loops could be improved.

Acknowledgement

I am really grateful for being supervised by Florian Huemer who is also the author of PYPR. I'm thankful not only for the opportunity to get into the asynchronous world but also for his enthusiasm especially about these topics and he always has some papers and references in mind. Therefore thank you very much not only for the support and constructive discussions but also for the patience and for adding additional handy features to PYPR while working on the core.

References

[1] Huemer Florian: Contributions to Efficiency and Robustness of Quasi Delay-Insensitive Circuits (not yet published), 2022

[2] Sparsø Jens: Introduction to Asynchronous Circuit Design, 2020, ISBN 978-87-643-2001-5

[3] Internal Memory (RAM and ROM) - UG-01068-4.3, Altera Corporation, November 2013

[4] Recommended HDL Coding Styles (QII51007-13.1.0), Altera Corporation, November 2013

[5] John L. Hennessy, David A. Patterson: Computer Organization and Design RISC-V Edition: The Hardware Software Interface (The Morgan Kaufmann Series in Computer Architecture and Design), 978-0128203316

[6] S. Keller and A. J. Martin and C. Moore: DD1: A QDI, Radiation-Hard-by-Design, Near-Threshold 18uW/MIPS Microcontroller in 40nm Bulk CMOS, 21st IEEE International Symposium on Asynchronous Circuits and Systems, 10.1109/ASYNC.2015.15

[7] A. Lines: The Vortex: A Superscalar Asynchronous Processor, 13th IEEE International Symposium on Asynchronous Circuits and Systems, 10.1109/ASYNC.2007.28

Logical vs. Physical Time in Cyber-Physical Systems Specified by Interpreted Petri Nets

Łukasz Sawicki, Iwona Grobelna,
Piotr Mróz, Małgorzata Mazurkiewicz
University of Zielona Góra
65-516 Zielona Góra, Poland

Corresponding author: i.grobelna@iee.uz.zgora.pl

Abstract. Interpreted Petri nets are a mathematical formalism to specify the control part of a cyber-physical system. A formal model can be then analyzed and verified to check some basic properties before implementation. The aspect of time is viewed differently in the logical specification and in the physical implementation. In this paper, we discuss the differences between logical and physical time, basing on a case study of automatic control system for coupling of a tractor unit with a semi-trailer. It is shown how the concept of time changes during the following stages of system development, which may cause some unexpected problems.

Keywords. Control system, Cyber-physical system, Formal specification, Petri net.

1. Introduction

Cyber-physical systems (CPSs) [1] integrate sensing, computation, control and networking into physical objects and infrastructure, connecting them with each other and to the Internet. A lot of different issues become then significant, including software, hardware and the link between them [2]. The design process of CPSs enforces the collaboration between engineers from various disciplines – computer science, automation and control, mechatronics, etc. The rapid progress in the industry creates also new challenges that have to be overcome [3].

Petri nets (PNs), as a general-purpose graphical and mathematical modeling tool [4] can be used for describing information processing systems and their analysis [5]. Currently, they are applied in many domains, including manufacturing systems [6], automatic control [7], freight logistics and transportation systems [8] or cyber-physical systems [9]. Interpreted Petri nets, following the notion introduced in [10], apart from the description of events and reactions, take into account the presence of input and output signal. This allows the communication with the environment. The deterministic design methodology for CPSs realized as distributed or integrated system has already been introduced in [11].

This paper continues our previous works [10, 11] and considers the timing aspects of systems modelled with the proposed methodology [11], taking into account logical and physical time. The case study of automatic control system for coupling of a tractor unit with a semi-trailer illustrates the possible problem with the perception of time. In the specification phase, the logical time is considered, while in the implementation phase we have to deal with the physical (real) time [12], where both times may differ from each other.

In this paper, we discuss the perception of time and point out important aspects that should be taken into account during the development of cyber-physical systems. In a synchronous system with some concurrent processes specified as an interpreted Petri net, it may happen that two or more transitions fire at the same time [10]. This directly may cause some problems in the distributed implementation, where the target devices may operate in different clock domains, considering e.g., FPGA, Raspberry Pi, or STM32 processor.

The main contribution of this paper is the discussion of time perception in various phases of the development of cyber-physical systems, where the specification is formally delivered in form of an interpreted Petri net. Due to the other perceptions of time, some problems may occur during the implementation, that were not possible to be detected in the model. We identify such a problem using an implemented case study of automatic control system for coupling of a tractor unit with a semi-trailer.

The rest of the paper is structured as follows. Section 2 introduces some basic definitions, which will be used in the further part of the article. Section 3 provides a case study of an automatic control system for coupling of a tractor unit with a semi-trailer. Section 4 considers the time in modelling methodology, distinguishing between logical and physical time, discussing also some experimental verification results. Finally, Section 5 summarizes and concludes the paper.

2. Basic Definitions

For easier understanding and reading, some preliminaries are introduced, considering mainly the theory of Petri nets. The following definitions are based on [4], [5], and [10].

Definition 1 (Petri net). A Petri net is a four-tuple $PN = (P, T, F, M_0)$, where P is a finite non-empty set of places, T is a finite non-empty set of transitions, $F \subseteq (P \times T) \cup (T \times P)$ is a finite set of arcs and M_0 is an initial marking.

Definition 2 (marking). A marking involves all places that contain a token.

Definition 3 (enabled and fired transition). A transition is enabled in marking M, if each of its input places contains a token. A transition can be fired, if it is enabled. Then, a token is removed from all its input places and added to all its output places.

Definition 4 (reachable marking). A marking is reachable from any other marking, if it can be reached by a sequence of transition firings.

Definition 5 (live PN). A Petri net is live, if from any reachable marking it is possible to fire any transition by a sequence of firings of other transitions.

Definition 6 (safe PN). A Petri net is safe, if there is no reachable marking such that any place contains more than one token.

Definition 7 (interpreted Petri net). An interpreted Petri net is a six-tuple $IN=(P, T, F, M_0, X, Y)$, where the first four elements describe a Petri net PN that is live and safe; X is a finite set of logic input signals, Y is a finite set of logic output signals. A transition in an interpreted net can be fired, if it is enabled and all the conditions of its input signals assigned are fulfilled.

A sample of an interpreted Petri net is shown in Figure 1. A short sequential process is modelled using three places (*p1, p2, p3*) and three transitions (*t1, t2, t3*) connected alternatively with each other, together with three input signals assigned to transitions (*start, endTask1, endTask2*) and three output signals assigned to places (*ready, task1, task2*). The initial marking involves place *p1*. And so, initially place *p1* is marked and output signal *ready* is active. After input signal *start* becomes *true*, transition *t1* fires – and as a result it consumes a token from place *p1* and adds a token to place *p2*. Then, output signal *task1* becomes active (and task 1 is executed), until input signal *endTask1* is received, etc. The interpreted Petri net is live and safe.

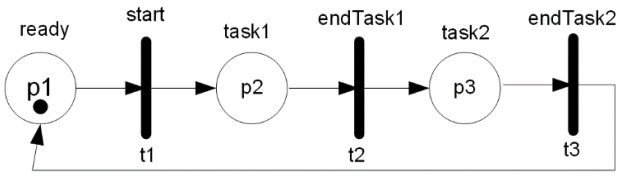


Figure 1. A sample of an interpreted Petri net

3. Automatic Control System for Coupling of a Tractor Unit with a Semi-trailer

Let us illustrate the diverse aspects of time perception with a case study of an automatic control system for coupling of a tractor unit with a semi-trailer. The prototype system has been implemented at the University of Zielona Góra, Poland. The structure of the control system is shown on Figure 2. There are two cooperating systems: (1) image gathering and processing subsystem and (2) tractor unit control subsystem. These systems communicate with each other via the asynchronous UART interface. The tractor unit control subsystem is responsible for communication with the manipulator that controls the truck model and all vehicle functions. More details about the automatic control system are can be found in [13].

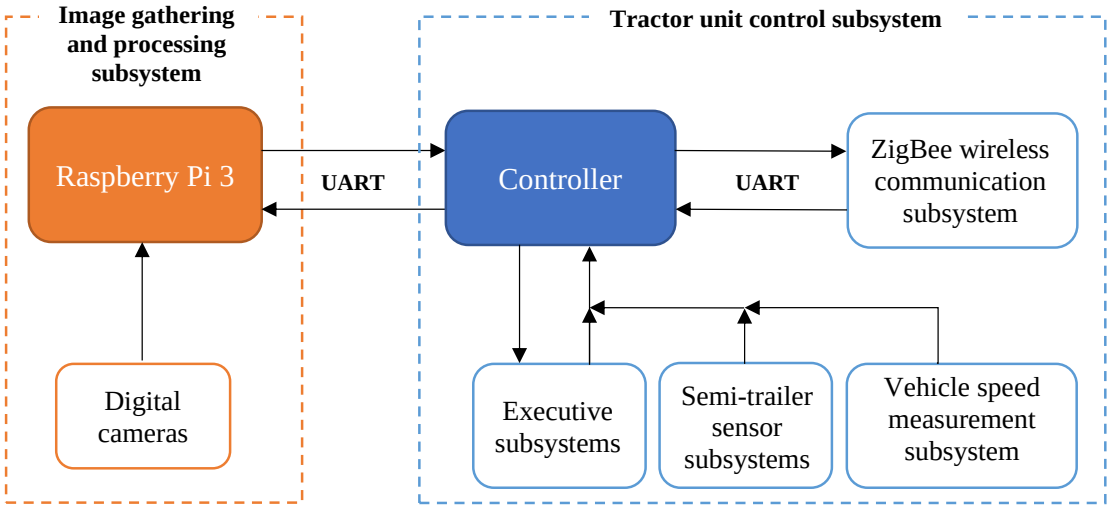


Figure 2. Structure of the real control system

On the front of the semi-trailer there is a synthetic square marker composed of a wide black border and an inner binary matrix (ArUco marker) with the coded identification number (id) of the semi-trailer. ArUco markers (an example is shown in Figure 3a) are currently commonly used in various detection systems, including, e.g., drones [14], health systems [15], or augmented reality [16]. The image gathering and processing subsystem is responsible for taking the image from the camera, processing it, reading the code and its coordinates from the ArUco marker. Finally, it generates the steering vector for the truck. The detection of ArUco marker is illustrated in Figure 4b, with identified code Id = 40 and marked one of the corners for which the coordinates are designated. The steering vector contains information about setting the speed and the steering angle of the front wheels. Both the processes of determining the ArUco code and the steering vector are initiated by the tractor unit control subsystem by sending an appropriate command via the UART interface. The recognized ArUco code and the steering vector are returned in the same way.

(a) (b)

Figure 3. Exemplary ArUco markers

4. Perception of Time

During various stages of cyber-physical system development, the perception of time may be changed. At the beginning, no time is considered at all, and the informal specification is mostly focused on the basic functionality of a designed system, describing in a natural language its main operations. In formal specification, here delivered in form of an interpreted Petri net, logical time is considered. During analysis, verification and optional decomposition, which are also based on Petri net models, logical time is considered too. In the distributed implementation, the physical time may already differ from the logical time. Schematic illustration of time perception during various stages of system development is shown in Figure 4.

Figure 4. Schematic illustration of time perception during various stages of system development

Let us first introduce some general concepts of logical and physical time (Subsection 4.1). Then, formal specification with logical time is presented (Subsection 4.2), followed by interpretation of physical time after implementation (Subsection 4.3).

4.1 Concepts of Time in Interpreted Petri Nets

Let us consider a cyber-physical system with some concurrent processes that will be implemented as a distributed system (models are shown in Figure 5). In the specification phase, where the interpreted Petri net models are prepared, the logical time is used, and we may assume that the independent transitions fire at the same time, e.g., transitions *ta1* and *tb1* may fire exactly at the same point in time. Then, logically, firing of transitions *ta1* and *tb1* changes marking as follows – tokens are consumed from places *pa1* and *pb1* and added to places *pa2* and *pb2* – also exactly at the same point in time. As a result, both output signals *ya* and *yb* become active simultaneously.

In the case of an implementation integrated system, i.e., in a single device (e.g., FPGA device), both transitions *ta1* and *tb1* are connected to the same clock signal and the physical time corresponds then to logical time. However, in case of implementation as a distributed system, i.e., where both processes are executed by different devices (e.g., by Raspberry Pi and STM processor), the particular clock periods vary, and thus transitions *ta1* and *tb1* fire at different points in time. Each device operates in its own clock domain, therefore, the logical time does not correspond to the physical one anymore. This may be problematic.

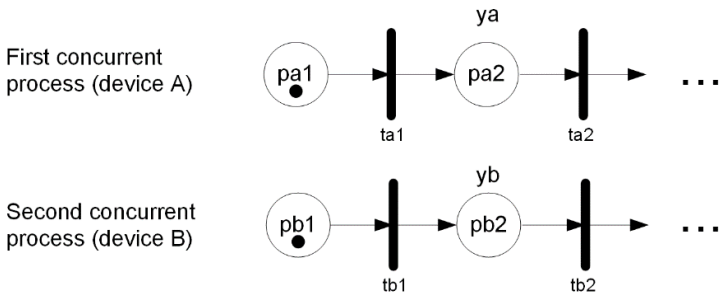


Figure 5. Interpreted Petri net models for two concurrent processes

Formal specification in form of an interpreted Petri net is usually a basis for the next steps of system development, i.e., analysis [17, 18] and verification (also using model checking technique [19]), followed by implementation [20, 21]. Aiming at a distributed implementation, the PN model needs to be first decomposed into parts (components), each one then to be implemented as a separate device. Additional signals may then need to synchronize the processes, when necessary. This aspect is however not treated further in the paper. For more details on the modelling methodology please refer to [10].

4.2 Formal Specification as an Interpreted Petri Net and Logical Time

The automatic control system for coupling of a tractor unit with a semi-trailer has been modeled as an interpreted Petri net, as shown in Figure 6. The net consists of 13 places and 12 transitions.

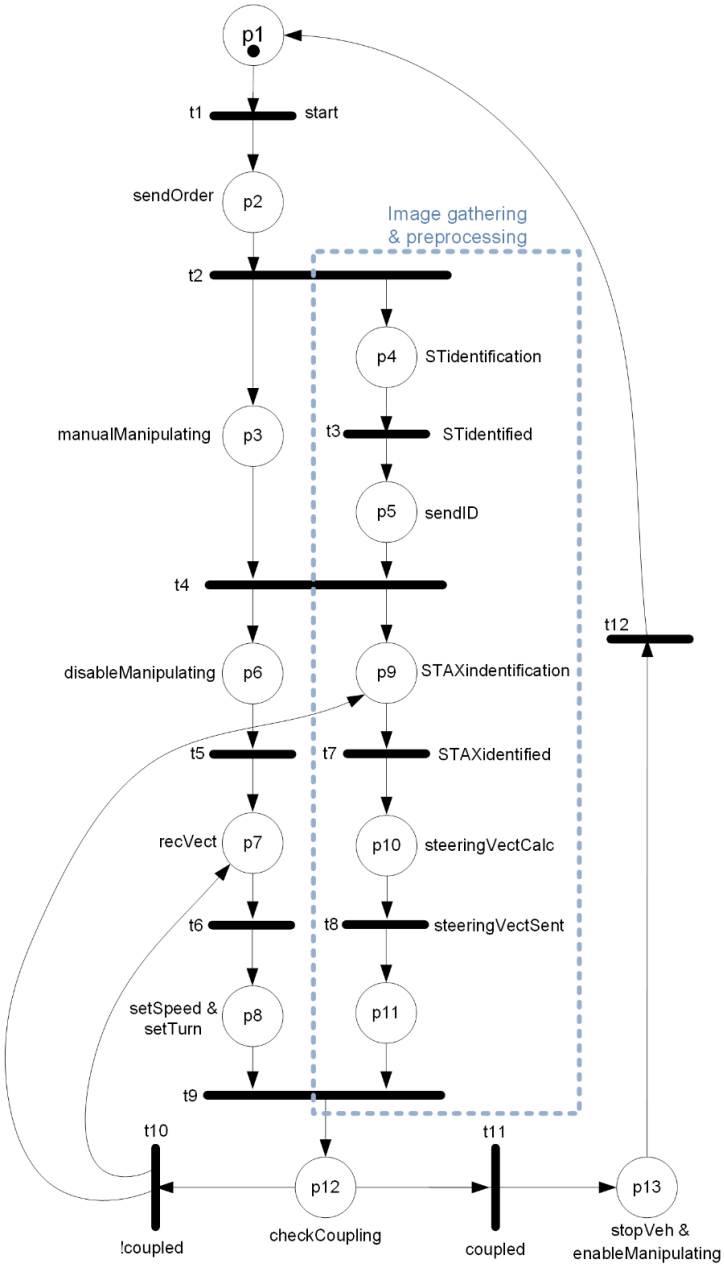


Figure 6. Interpreted Petri net model

The PN model considers both – controlling vehicle movement (left part) and image gathering and preprocessing (right part marked with a dashed border around it). The first part has then been implemented in STM32F409 processor, while the second one in Raspberry Pi 3+. The details of the implementation can be found in [13]. In the interpreted PN model various input and output signals are used for the communication with the environment. The signals are described in Table 1.

Table 1. Input and output signals used in the interpreted PN model

	Signal name	**Signal description**
Inputs	*start*	Start the control algorithm
	STidentified	Semi-trailer is identified
	STAXidentified	Semi-trailer axle is identified
	steeringVectSent	Vehicle steering vector is sent to tractor unit
	coupled	True, if tractor unit and semi-trailer are coupled, otherwise – false
Outputs	*sendOrder*	Send order to start the automatic approach
	manualManipulating	Manual manipulating of the vehicle is enabled and used
	STidentification	Semi-trailer is being identified
	sendID	ID of recognized semi-trailer is being sent
	disableManipulating	Manual manipulating of the vehicle is disabled
	STAXidentification	Semi-trailer axle is being identified
	steeringVectCalc	Vehicle steering vector calculation
	recVect	Receiving steering vector
	setSpeed	Set speed of the vehicle (used directly for coupling)
	setTurn	Set turn of the vehicle (used directly for coupling)
	checkCoupling	Check if coupling ended successful
	stopVeh	Stop the vehicle (automatic coupling process finished)
	enableManipulating	Manual manipulating of the vehicle is enabled

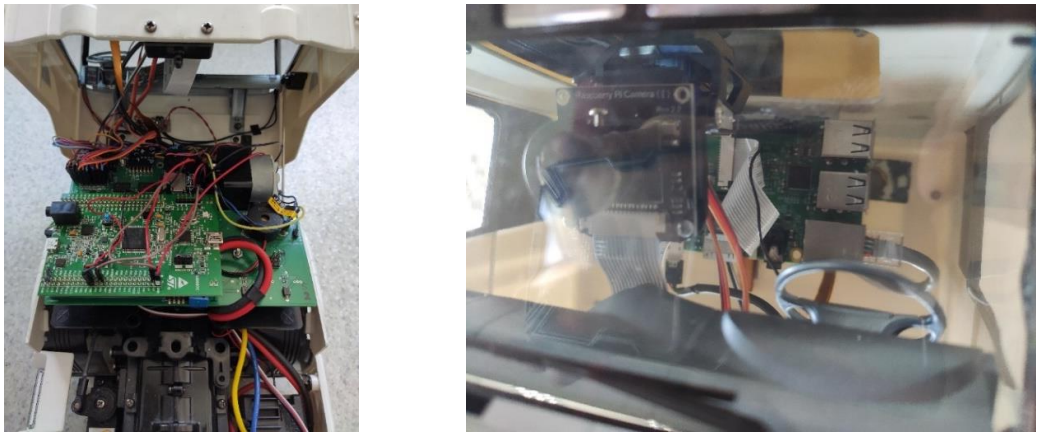
The interpreted Petri net model is not a timed model, which means that no time details are directly added into the model. It uses logical time, and firing time of a transition is infinitely short. In particular, when considering the concurrent processes, some transitions may fire at the same point in time, e.g., transition *t5* and *t7* or *t5* and *t8*. This will not be possible after the distributed implementation, where the target devices operate in different time domains. The physical time at which the ticks occur is irrelevant to the logical time in the PN model.

4.3 Distributed Implementation and Physical Time

The interpreted Petri net model from Figure 6 has been implemented as a distributed systems. The tractor unit control subsystem was implemented in the EmBitz environment on a STM32 processor (Figure 7a) and the image gathering and processing subsystem was implemented in the CodeBlocks environment on a Raspberry Pi computer (Figure 7b).

The autonomous drive of the tractor to the semi-trailer begins after the command is sent from the transmitter to the tractor. This is denoted by the *t1* transition (Figure 6). Then, the tractor unit control subsystem sends to the image gathering and processing subsystem the command to start detection and read the id code from the ArUco marker (transition *t2*). Subsequently, the tractor unit control subsystem enables further manual tractor control (place *p3*). During this time, the image gathering and processing subsystem identifies the ArUco marker and reads the code written on it (place *p4*). The next step is to send the semi-trailer id (place *p5*) to the tractor unit control subsystem. An autonomous driveway to the semi-trailer begins from transition *t4*. Then, the speed and turn settings received from the manipulator are blocked (place *p6*). During this time, gathering and processing subsystem determines the coordinates of the ArUco marker and on their basis determines the steering vector (place *p10*). This vector is sent to the tractor unit control subsystem (transition *t8*), which sets the tractor speed and turn to the value

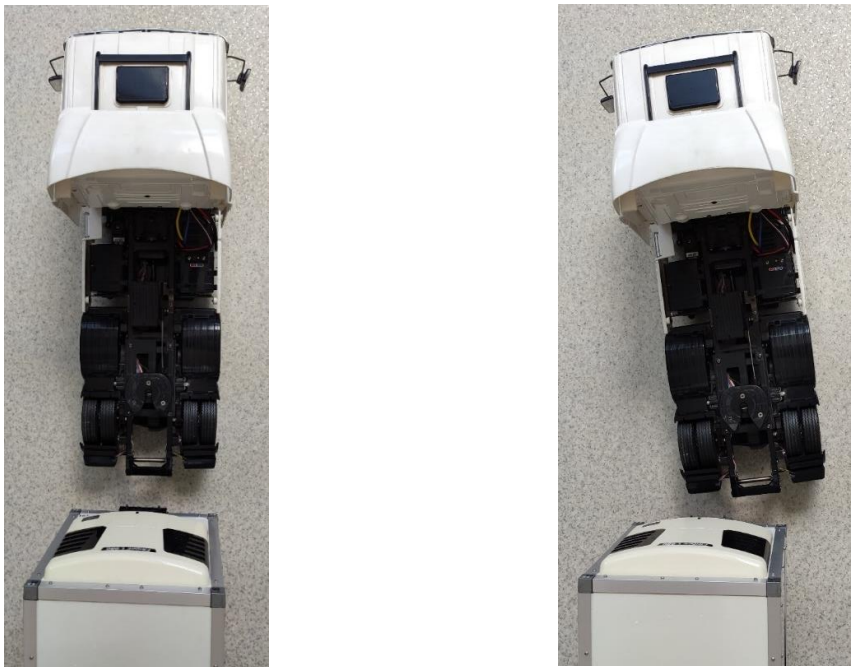
determined by the steering vector (places *p7* and *p8*). The process of determining the control vector and setting the speed of the tractor is repeated until the tractor unit control subsystem reads the information that the tractor has reached the semi-trailer (place *p12*).



(a) (b)

Figure 7. View of subsystems: (a) tractor unit control and (b) image gathering and processing

When correctly approaching the semi-trailer, the tractor drives as shown in Figure 8a. In practice, it was noticed that after sending to the tractor unit control subsystem of the semi-trailer (place *p5*), the tractor is for some time controlled by the transmitter and not by the image gathering and processing subsystem. This causes disturbances, and as the result the tractor misses the semi-trailer (Figure 8b). The problem may be the time difference between the two subsystems.



(a) (b)

Figure 8. View of the tractor approaching the semi-trailer: (a) correct and (b) incorrect

The reason of incorrect approaching (Figure 8b) may be some communication delays between processors. After recognizing the id of the semi-trailer, it is sent to the tractor unit control subsystem (place *p5* in PN model in Figure 6). Then the image analysis and steering vector generation in image gathering and processing subsystem begins. After receiving the id by the tractor unit control subsystem, the remote control is blocked (place *p6*) and the system waits for the steering vectors from the gathering and processing subsystem (place *p7*). As a result of the delays between the clocks of both systems, there is a delay which makes it possible to receive the control vector in absence of a remote control lock. The consequence may be therefore incorrect driving of the tractor to the semi-trailer. What has not been a problem at the specification stage with logical time, may not operate correct with physical time in a distributed implementation.

5. Conclusions

In this article we have shown how the time perception changes in the various stages of cyber-physical system development. At the specification stage, logical time is used, with no delays, and the concurrent processes work in the same time domain. Logical time appears also during analysis, verification and optional decomposition of an interpreted Petri net model. Although it is also possible that the concurrent processes will work in the same time domain after implementation as an integrated system (e.g., in an FPGA device), it is no longer the case in a distributed system. Then, various devices operate in their own time domains (in the implemented and discussed example – STM processor and Raspberry Pi), and therefore the physical time does not correspond to the logical one anymore.

A case study of an automatic control system for coupling of a tractor unit with a semi-trailer was used to illustrate the possible problems that are related to time perception. Having a formal specification that is operating correctly in the logical time, a not fully correct implementation can be achieved, where the various time domains of target platforms and additional time delays introduced by real devices change significantly the perception of time. The physical time in the working prototype system is irrelevant to the logical time in the interpreted Petri net model.

Plans for the future include further development of the presented automatic control system to ensure the proper operation in real time. The work is continuously carried out at the Faculty of Computer, Electrical and Control Engineering, University of Zielona Góra, Poland.

Acknowledgment

The work is supported by the National Science Centre, Poland (grant number 2019/35/B/ST6/01683).

Conflicts of Interest

The authors declare no conflict of interest.

References

[1] Wolf, W. Cyber-physical systems. Computer, 42 (03), 88-89, 2009.
[2] Khaitan, S. K., McCalley, J. D. Design Techniques and Applications of Cyberphysical Systems: A Survey, IEEE Systems Journal, vol. 9, no. 2, pp. 350–365, Jun. 2015.
[3] Quadri, I. et al, Modeling Methodologies for Cyber-Physical Systems: Research field study on inherent and future challenges, ADA USER, vol. 36, no. 4, pp. 246, 2015.
[4] Murata, T. Petri nets: properties, analysis and applications. Proc. of the IEEE, 77, 541-580, April 1989.
[5] David, R., Alla, H. Discrete, continuous, and hybrid Petri nets. 1, 17-130, Berlin: Springer, 2010.
[6] Grobelna, I., Karatkevich, A. Challenges in Application of Petri Nets in Manufacturing Systems. Electronics 2021, 10, 2305, doi: 10.3390/electronics10182305.

[7] Giua, A., Silva, M. Petri nets and Automatic Control: A historical perspective. Annual Reviews in Control, 45, 223-239, 2018).
[8] Cavone, G., Dotoli, M., Seatzu, C. A survey on Petri net models for freight logistics and transportation systems. IEEE Transactions on Intelligent Transportation Systems, 19(6), 1795-1813, 2017.
[9] Wisniewski, R., Bazydło, G., Szcześniak, P., Grobelna, I., Wojnakowski, M. Design and Verification of Cyber-Physical Systems Specified by Petri Nets—A Case Study of a Direct Matrix Converter. Mathematics 2019, 7, 812, doi: 10.3390/math7090812.
[10] Grobelna, I., Wiśniewski, R., Wojnakowski, M. Specification of Cyber-Physical Systems with the Application of Interpreted Nets, IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society, 2019, pp. 5887-5891, doi: 10.1109/IECON.2019.8926908.
[11] Wisniewski, R., Grobelna, I., Karatkevich, A. Determinism in Cyber-Physical Systems Specified by Interpreted Petri Nets. Sensors 2020, 20, 5565, doi: 10.3390/s20195565.
[12] Lee, E. A., Seshia, S. A. Introduction to embedded systems: a cyber-physical systems approach, Second edition. Cambridge, Massachusetts: MIT Press, 2017.
[13] Jędrzejczak, O., Mróz, P., Mazurkiewicz, M. System of automatic approach control for a tractor unit's approach to a semi-trailer, Proceedings of the 9th Prague Embedded Systems Workshop - PESW 2021, Horoměřice, Czechy, Prague: Czech Technical University, pp. 23-31, 2021.
[14] Sani, M. F., Karimian, G. Automatic navigation and landing of an indoor AR. drone quadrotor using ArUco marker and inertial sensors. In international conference on computer and drone applications (IConDA), pp. 102-107, November 2017, IEEE.
[15] Koeda, M., Yano, D., Shintaku, N., Onishi, K., Noborio, H. Development of wireless surgical knife attachment with proximity indicators using ArUco marker. In International Conference on Human-Computer Interaction, pp. 14-26, July 2018, Springer, Cham.
[16] Avola, D., Cinque, L., Foresti, G. L., Mercuri, C., Pannone, D. A practical framework for the development of augmented reality applications by using ArUco markers. In International Conference on Pattern Recognition Applications and Methods, Vol. 2, pp. 645-654, February 2016, SciTePress.
[17] Karatkevich, A. Dynamic analysis of Petri net-based discrete systems, Vol. 356, Springer Science & Business Media, 2007.
[18] Wojnakowski, M. et al. Analysis of safeness in a Petri net-based specification of the control part of cyber-physical systems, International Journal of Applied Mathematics and Computer Science, 31,4, 2021.
[19] Grobelna, I., Wiśniewski, R., Grobelny, M., Wiśniewska, M. Design and Verification of Real-Life Processes With Application of Petri Nets, IEEE Transactions on Systems, Man, and Cybernetics: Systems, vol. 47, no. 11, pp. 2856-2869, Nov. 2017, doi: 10.1109/TSMC.2016.2531673.
[20] Grobelny, M., Grobelna, I., Karatkevich, A. C code generation from Petri-net-based logic controller specification, Proc. SPIE 10445, Photonics Applications in Astronomy, Communications, Industry, and High Energy Physics Experiments 2017, 1044525 (7 August 2017), doi: 10.1117/12.2280959.
[21] Wisniewski, R. Design of Petri Net-Based Cyber-Physical Systems Oriented on the Implementation in Field Programmable Gate Arrays, Energies, 14(21):7054, 2021, doi: 10.3390/en14217054.

Precision Landing of Unmanned Aerial Vehicles using Computer Vision

Van Phi NGUYEN
Faculty of Military Technology, University of Defence
Kounicova 65, Brno, Czech Republic
vanphi.nguyen@unob.cz

Abstract. Precision landing is considered to be the last and most critical stage of navigation. Autonomous precision landing Unmanned Aerial Vehicle (UAV) can be widely used in many industries and various applications. Previously, most designs were based on the Global Positioning System (GPS) and the Inertial Navigation Systems using inertial sensors such as 3-axis accelerometers, gyroscopes, and magnetometers. However, the position data provided by GPS is often inaccurate, so more accurate sensors are required. In this work, a solution for high-precision landing is presented, which is based on the use of a camera and ArUco markers as the landing platform. Once the marker is detected by image processing, the output of this process obtains the relative position between the marker and the drone. Thanks to MAVLink protocol, the NVIDIA Jetson Nano board can communicate with the flight controller to control the drone's position. This way, it can fly to the marker's location and then land in the exact position. The algorithm was evaluated and validated in the ROS/Gazebo simulation environment, as well as in real-world tests.

Keywords. ArduPilot, ArUco markers, drone, MAVLink protocol, NVIDIA Jetson Nano, OpenCV, precision landing, ROS/Gazebo, UAVs.

1 Introduction

Over the past few years, the interest in Unmanned Aerial Vehicles (UAVs) has been growing rapidly. Their wide applications make those drones or UAVs be tagged as the hottest technology on the planet nowadays. They have appeared in military drones, other commercial and industrial ones. In terms of the military, drones are used as target decoys for combat missions, research and development, and supervision.

Landing a UAV is acknowledged as the last and most critical stage of navigation [1] especially when 80 % of the hazard UAVs' accidents are related to the landing process [12]. Therefore, estimating the precise position and orientation of UAVs plays an important role in UAV safe self-landing.

Previously, most designs relied on the Global Positioning System (GPS) and Inertial Navigation Systems (INS), using inertial sensors. However, these methods have shown significant inaccuracies. The result in a landing position typically deviates from the intended one by 1 to 3 m, sometimes even up to 10 m.

In this work, the desire is to design, implement, and practical verification of the hardware and software system solution, which will enable the drone to autonomously land on the drone at a defined location.

The main aim of this work is to develop a vision-based landing system to make a UAV land in a specific place with high precision. The proposed landing algorithm uses the drone's built-in image processing system. These processing processes take place on the onboard computer NVIDIA Jetson Nano. The landing target is ArUco markers (Section 3.3) with certain advantages in identification and localization in 3D space. The MAVLink protocol (Section 3.4) will handle all the communication, information exchange, and control of the drone from the onboard computer. This project also offers simulation environment solutions. The tests in the simulation environment helps the process of checking the correctness and safety of the algorithm. In addition, the article gives readers the first look at the ability of programming for drones.

2 Related Work

Recently, different types of UAV landing approaches have been studied. In particular, a number of computer vision-based approaches have presented, such as follows.

Oualid Ararr and his colleagues [3] succeeded in autonomous tracking and landing on a moving platform. This indoor experiment showed that the UAV can successfully land on the target landing platform (that contains multiple fiducial markers). With a landing height of about 80 cm and a landing platform moving at speeds between 40 and 180 cm/s, the accuracy is impressive within 13 cm tolerance.

Chen et al. [5] developed a vision approach and the use of LiDAR sensors for tracking and localization systems. The landing target platform is a sample with a red rectangle and a green circle inside. The fundamental purpose of image processing is to determine the landing destination via color analysis. The drone can track the landing target's position (in horizontal plane *x*,*y*-coordinates) and the altitude parameter is provided by the LiDAR sensor. This technology works reliably and quickly for autonomous landing; however, the LiDAR appears to be extremely pricey when applied to a variety of drones.

According to Thien Hoang Nguyen, et al. [7], even though vision-based navigation technologies are sophisticated, a UAV may fail to target its landing pad if GPS error prevents its field of view (FOV) from viewing the target. To circumvent this, an Ultra – wideband (UWB) system was fitted on the drone to ensure that it was led to the target landing pad within the drone's field of view (FOV), after which visual navigation took over and completed the operation. The method worked well, but it required the usage of two onboard computers, one mini-LiDAR sensor, and a RealSense Camera, to mention a few. They consume a lot of power, so this approach may not be a practical solution.

De Souza et al. [8] created an autonomous landing system using an Artificial Neural Network (ANN) with Fuzzy Mamdani Logic as a supervisor. Their solution reduced computational complexity while preserving the fuzzy logic controller's properties and intelligence. They validated their method for static and dynamic landing locations using simulation and real-world testing.

One difference in this work is that the UAV can detect the landing area when it is high above the ground (height > 15 m). The proposed landing algorithm compensates for potentially high GPS error values while using inexpensive sensors (only a Jetson camera is required) and achieving very low landing accuracy (less than 20 cm). To do this, the ArduPilot firmware in the Pixhawk 4 flight controller is employed. Previously, the majority of the ArduPilot community used an IR-beacon (infrared) to land a UAV from a height of 15 m, reliably in all lighting situations and with a maximum offset of just 30 cm. While this method produces amazing results, ours exceeds it in terms of accuracy, altitude, and cost.

3 System Architecture

System architecture is depicted in Fig. 1. To enable precision landing of the drone by image processing, UAV onboard system must contain the following essential parts: the flight controller; a camera attached to the onboard computer and other general components as Electronic Speed Controller (ESC), brushless DC motors, Li-Po battery, telemetry, etc. The drone's flight controller uses the MAVLink protocol to connect to the onboard computer through GPIO pins. A router-based network is established to communicate between this onboard computer and the ground control station with the help of the Wi-Fi adapter. There is also additional telemetry-based connection between the flight controller and the ground control station also using the MAVLink protocol.

3.1 Flight Controller

Pixhawk 4 is the latest update of the successful Pixhawk flight controllers' family. It is in charge of not only receiving the information from different sensors such as GPS, barometer, and LiDAR, etc. but also communicating with the NVIDIA Jetson Nano (presented below). This communication is based on the MAVLink protocol, which has output data from Jetson to stabilize and control UAV (motor speed via ESC, flight surfaces, camera triggers, etc.) at a low level.

3.2 Onboard Computer NVIDIA Jetson Nano

The NVIDIA Jetson Nano platform meets three main purposes in this work. Firstly, it runs a Python application that processes the image acquisition from camera. On account of using the ArUco marker library in the open-source tool OpenCV, this process is relatively achieved. Secondly, integrated with DroneKit-Python, this Python application allows NVIDIA Jetson Nano to communicate with the flight controller over MAVLink protocol. Therefore, the onboard computer can make commands in Pixhawk4, making the UAV fly to the marker's location and land in its exact position. Finally, this platform can build up a network (ad-hoc network or router-based network) to facilitate communication with a ground station (laptop).

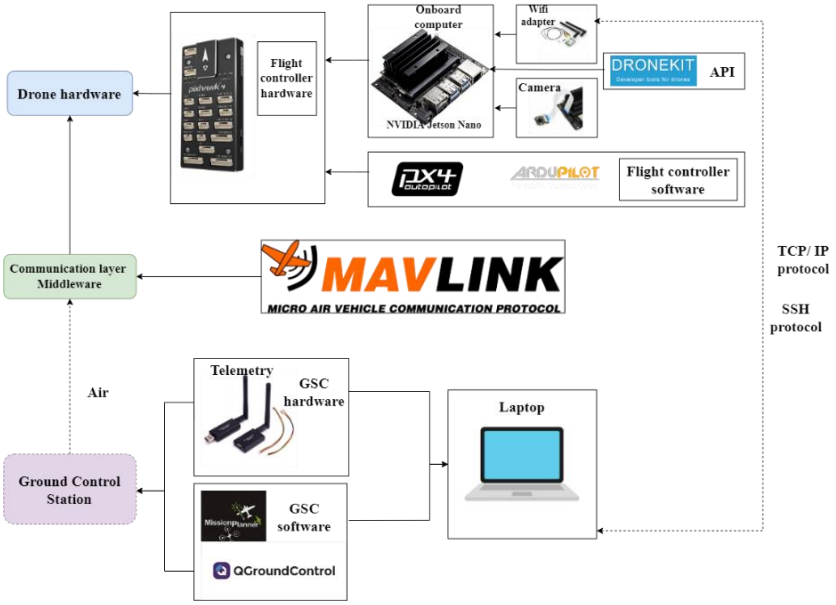


Figure 1. Overview of hardware and software structure for autonomous landing.

A wide-angle fish-eye camera connects to the NVIDIA Jetson Nano by a Camera Serial Interface (CSI). The value of Frames per Second (FPS) and resolution of this camera can be adjusted in case of the latency in image processing, leading to an inaccurate landing. Moreover, calibrating the camera is essential to avoid distortion and improve the performance of the precision landing algorithm. This process contains estimating the parameters to define a camera's focal length, optical sensor, and lens distortion. It works effectively to prevent image distortion, especially radial and tangential distortion.

3.3 Landing Platform – ArUco markers

Software structure is in Fig. 2. All applied components will be described below.

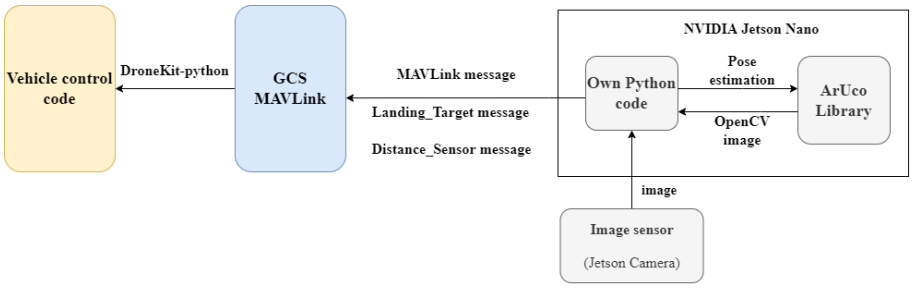


Figure 2. Software architecture.

ArUco markers (the target landing pad) are binary square fiducial markers that can be used for camera pose estimation. They come with enormous potential for both identification and localization in 3D space [10]. These markers are similar to the QR-codes, but they carry less information, only their identifiers. There are various types of configurations for ArUco markers: $N \times N$ (N = 4; 5; 6; 7), but they have one thing in common: a black border, which facilitates its fast detection in the image and identifies the code inside.

In this work, the landing pads belong to the ArUco marker library's DICT_ARUCO_ORIGINAL, which contains markers 7×7. The border's columns and rows are black, so data is encoded into a 5×5 matrix inside (Fig. 3). A white square corresponds to a value of 1 and a black one corresponds to the value 0. The first, third, and fifth columns are for parity and the second and fourth columns represent the data, and store the marker identifier in natural bin code.

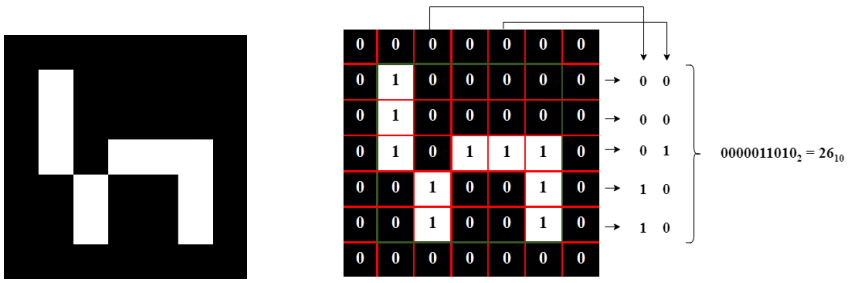


Figure 3. Examples of ArUco markers and decoding it.

OpenCV is employed as an infrastructure to realize algorithms in the Python programming language. Video input from the Jetson camera is processed by OpenCV to detect the ArUco marker. Moreover, after visualizing parameters and the bounding box of the detected markers, OpenCV will estimate the poses and the attitudes of the markers relative to the camera frame. After image processing, the final obtained output is either the ArUco marker's position in the image frame (x_{pixel}, y_{pixel}) in pixels or the relative position $P(x, y, z)$ of the marker with respect to the camera in meters.

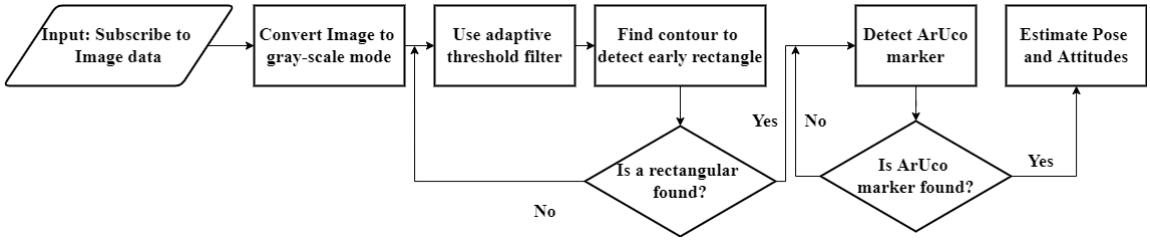


Figure 4. The main image-process algorithm by OpenCV.

3.4 MAVLink Protocol

The MAVLink protocol fully supports communication with the UAV. This protocol is a very lightweight messaging protocol (serial protocol) for communication (sending data and command) between vehicles (particularly drones) and ground stations [9], and also between the drone's flight controller and onboard computer NVIDIA Jetson Nano.

The most important MAVLink message used in this work is *Landing_Target*. The MAVLink version 2 can send two kinds of parameters in this message to control drone to move to the landing platform. The first one is the relative position $P(x, y, z)$ of the marker with respect to the camera from the image processing's output. The second one are two key parameters *angle_x* and *angle_y* (x, y-axis angular offset off the target from the center of the image) and one more parameter of distance from the drone to the marker – parameter z in $P(x, y, z)$.

Where

$$angle\_x = \frac{(x_{pixel} - horizontal\_resolution/2) \cdot horizontal\_fov}{horizontal\_resolution} \quad (1)$$

$$angle\_y = \frac{(y_{pixel} - vertical\_resolution/2) \cdot vertical\_fov}{vertical\_resolution} \quad (2)$$

and

- *horizontal_fov*, *vertical_fov* [rad] – the horizontal, vertical camera FOV angles,
- *horizontal_resolution*, *vertical_resolution* [pixel] – the horizontal, vertical resolution of the camera,
- x_{pixel}, y_{pixel} [pixel] – the position of the marker center in the image frame, on the *x*-axis, *y*-axis, in pixels.

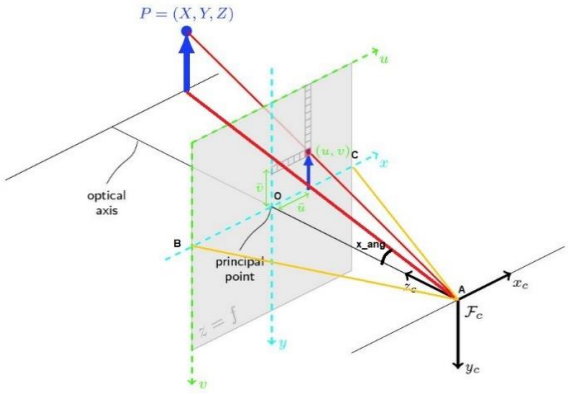


Figure 5. The relative position of the target on the captured image, adapted from [13].

Precision_Landing mode needs a rangefinder to be activated. There are two ways to do this: using a proper rangefinder or creating a virtual "*MAVLink rangefinder*". To solve this problem and take advantage of the depth parameter from the ArUco marker, the proposed algorithm uses a virtual distance sensor based on another message form of the MAVLink protocol: *Distance_Sensor*. The first step is configuring the rangefinder type's specifications, including its minimum and maximum range distance. The current altitude is measured by estimating pose and attitudes of ArUco markers in meters – parameter *z* in *P*(*x*, *y*, *z*). One will be transmitted through MAVLink message during the precision landing.

The required broadcast rate for sending MAVLink message depends on the landing speed and desired accuracy. According to the MAVLink documentation, the two messages *Landing_Target* and *Distance_Sensor* require broadcast rates of 10-50 Hz and 5-20 Hz respectively.

Following the research and flight tests, the suitable camera's resolution for this algorithm is 640×480 at 30 FPS. This resolution meets two requirements of detecting the landing pad 30×30 cm at a height of 15 m, and sending MAVLink messages in the loop at the required broadcast rates. It should be noted that changing image resolution also requires recalibrating the camera.

4 Landing Procedure

The drone's precision landing is mostly based on the *Precision_Landing* mode, an advanced feature by Copter firmware in ArduPilot. There are two conditions to detect a marker: (1) the marker must be completely inside the picture and (2) each square in this marker must be uniquely identified and analyzed (black or white).

During the landing, the drone's altitude will be the most noticeable. If the drone is too close to the landing pads (i.e., $z_3 = 0.5$ m), the marker will be too big to be fully inside the image frame. In other words, the markers exceed the camera's field of view. On the other hand, the camera cannot detect the target when the drone is flying too high (e.g., 15 m height and the marker size is only 30×30 cm). However, the maximum allowable height, at which the camera can detect the marker, can be improved by increasing the

size of the marker on the ground. As long as the camera can detect and identify the marker, the precision landing mode will be activated.

However, using a too large landing pad to increase target detection altitude can result in a larger error. Therefore, drone landing at the center of a large-sized marker is challenging. The provided solution here is combining multiple markers of different sizes. If the camera is able to detect a smaller-sized marker, it will change its destination to the smaller ones, and adjust its direction accordingly.

In this work, two ArUco markers of differrent sizes are employed, the bigger one – marker *id*_1 and the smaller one – marker *id*_2. The camera will start searching for markers *id*_1 or *id*_2 at altitudes below z_1 or below z_2 respectively. Depending on the drone's current altitude, it will decide to find the larger marker or smaller one (line 4-8 of the following algorithm). If no marker is detected, the flight mode of the UAV will be changed to mode *Loiter* (line 11) to maintain the current location, heading and altitude. If there are over 30 seconds of searching, the mission is aborted, and the drone will land by using GPS (line 13). Otherwise, if any marker is detected, the NVIDIA Jetson Nano will get the relative position $P(x, y, z)$ of the marker with respect to the camera by the pose estimation of the ArUco markers. The drone changes to mode *Land*. This onboard computer calculates the parameters *angle_x*, *angle_y* and sends two MAVLink messages (Sec. 3.4) to the flight controller. Until the altitude of the drone is starting to be smaller than z_3, the drone will only descend and disarm.

The algorithm for *Precision_Landing* mode is represented with the strategy below.

	Precision landing algorithm with using MAVLink protocol
1:	start timer 30 s
2:	*alt* ← drone.location.global_relative_frame.altitude
3:	**while** *alt* > z_3 **do**
4:	**if** *alt* > z_2 **then**
5:	*id_to_find* ← *id*_1*; marker_size* ← *id*_1*_size; marker_height* ← z_1
6:	**elseif** *alt* < z_2 **then**
7:	*id_to_find* ← *id*_2*; marker_size* ← *id*_2*_size; marker_height* ← z_2
8:	**end if**
9:	detect marker *id_to_find*
10:	**if not** detected **then**
11:	Change to *Loiter_mode*
12:	**if** timer exceeded then
13:	abort landing by GPS
14:	**end if**
15:	**else**
16:	Change to *Land_mode*
17:	get $P(x, y, z)_{id\_to\_find}$
18:	get(*angle_x, angle_y*)
19:	Send MAVLink *Distance_Sensor* message (integer (z_m))
20:	Send MAVLink *Landing_Target* message (*angle_x*, *angle_y*)
21:	**end if**
22:	**end while**
23:	descend and disarm UAV

5 Results

The landing platforms are placed in different locations. The position of the marker relative to the last waypoint of the drone is shown in Fig. 6. These positions are 4-5 m away from the drone to ensure that the marker is always in the landing area where the UAV is searching for the markers. Landing tests are performed repeatedly at each position both in the simulation and in the real-world.

A successful landing requires the UAV to land within 20 cm of the intended marker's center. All test flights were completed in the simulated environment to validate system operation. These tests were 100% successful in the simulation ROS/Gazebo. Table 2 shows the outcomes of the tests and the accuracy of the landing technique in the simulated environment. The position error is calculated by comparing the UAV position to the center of the marker.

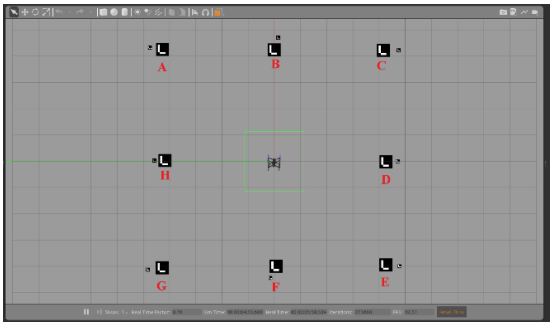


Figure 6. Position of the markers respect to the position of the last drone's waypoint before landing.

Table 1. The parameters of the markers for the precision landing procedure.

Parameter		Value in simulation	Value in real flight tests
Bigger marker	*Identifier: id_1*	3	26
	Size	50 × 50 cm	26 × 26 cm
	z_1	30 m	8 m
Smaller marker	*Identifier: id_2*	6	3
	Size	15 × 15 cm	15 × 15 cm
	z_2	6 m	2 m
z_3		0.5 m	0.5 m
Landing speed		30 cm/s	20 cm/s

Table 2. Precision landing results with position accuracy on *x*,*y*-axis.

Position of the landing platforms	**In the simulation environment ROS/GAZEBO**		**In the real-world flight tests**	
	Position accuracy *x*-axis [cm]	**Position accuracy *y*-axis [cm]**	**Position accuracy *x*-axis [cm]**	**Position accuracy *y*-axis [cm]**
A	3.83 ± 1.82	3.47 ± 1.81	2.38 ± 0.77	1.90 ± 0.38
B	6.25 ± 1.73	7.26 ± 2.37	3.05 ± 1.22	0.92 ± 0.34
C	8.80 ± 1.31	8.84 ± 1.85	4.05 ± 1.36	2.25 ± 0.92
D	2.95 ± 2.60	3.04 ± 2.34	3.96 ± 0.77	2.12 ± 1.86
E	7.51 ± 0.78	8.57 ± 0.97	5.12 ± 1.29	4.12 ± 0.71
F	2.09 ± 1.46	1.63 ± 0.93	2.59 ± 0.39	1.29 ±0.43
G	6.81 ± 1.21	6.57 ± 1.38	3.27 ± 1.60	3.22 ± 1.22
H	1.52 ± 0.59	1.72 ± 0.52	1.97 ± 0.76	2.84 ± 0.69
Average	4.97 ± 2.98	5.14 ± 3.26	3.30 ± 1.46	2.33 ± 1.36

From Table 2, the average deviation from the marker center on the *x*-axis was 4.97 ± 2.98 cm and on the *y*-axis was 5.14 ± 3.26 cm in the simulation environment. In the real-world flight tests, from the landing height of 8 m, the results were with the error only 3.30 ± 1.46 cm on the *x*-axis and 2.33 ± 1.36 cm on the *y*-axis.

In the simulation environment, during the first 10 seconds, the drone tries to move to the center of the landing pad while gradually lowering its altitude. Due to the inertia, *angle_x* and *angle_y* oscillate around zero. The drone altered course to land a smaller marker at a height of 6 m in 78^{th}-79^{th} s. *angle_x* and *angle_y* are recomputed, and the drone achieves equilibrium on the smaller marker after about a few seconds. In the real-world flight test, the inertia has a greater effect, *angle_x* and *angle_y* go to zero but do not stay there, oscillate until these two angles are small enough (Fig. 7a).

During the precision landing process, the drone will have to adjust its velocity in the horizontal plane. The Pixhawk will compute and select the suitable velocity based on the angular offset of the target from the center of the image (Figs 7a, 7d) and the relative position between the markers and the drone (Figs 7c,

7f). When the deviation angle is not too wide and the distance between the drone and the markers on the *x* and *y* axes is not too great, the drone will barely move.

In the real-world flight tests, when the UAV begins to fly toward the target and reaches the desired position, it levels to land on the target. However, because of its inertia, its speed does not equal zero when it reaches the destination. Therefore, the UAV overshoots and begins braking behind the target, then begins flying in the opposite direction, overshoots again, and so on, until the target's angular offset from the image center is small enough (Figs 7d, 7e, 7f).

Figs 7c and 7f showed that the UAV followed a smooth landing trajectory to move toward the landing pads.

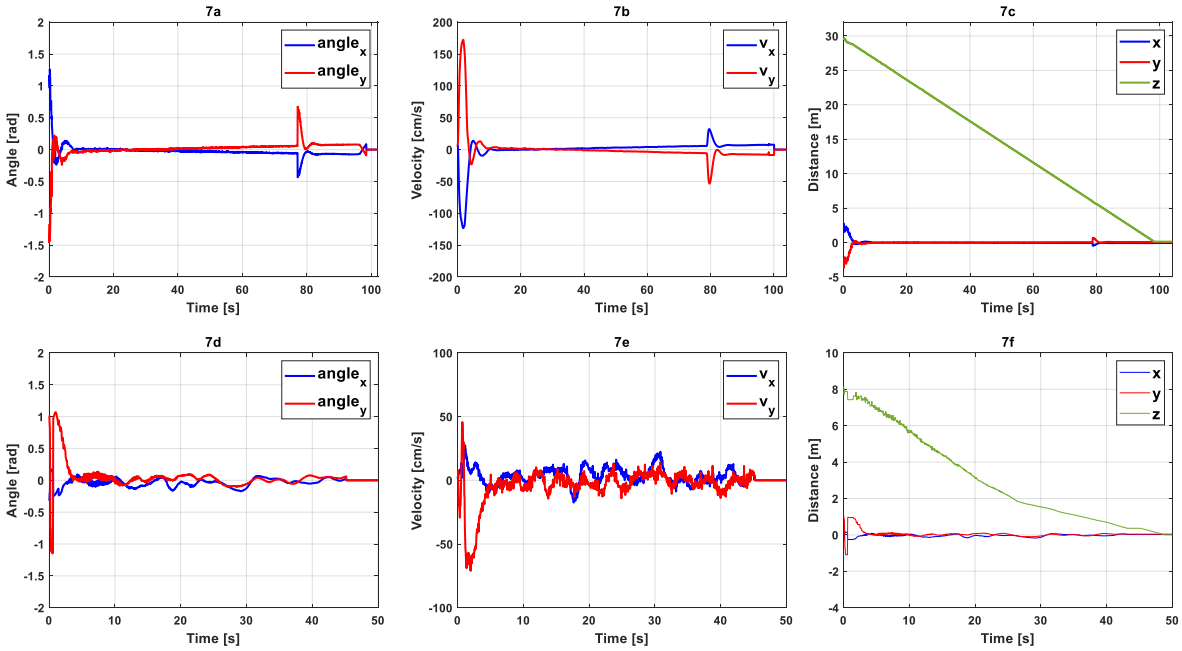


Figure 7. *angle_x, angle_y* angular offset of the target from the center of the image in the simulation ROS/Gazebo (a) vs. in the real-world flight tests (d). Velocity of the drone in the horizontal plane in the simulation ROS/Gazebo (b) vs. in the real-world flight tests (e). Position of the drone with respect to the markers on the ground in the simulation ROS/Gazebo (c) vs. in the real-world flight tests (f).

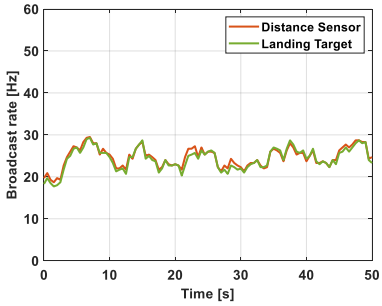


Figure 8. The broadcast rate of two MAVLink messages in real-world flight tests.

The NVIDIA Jetson Nano sent the MAVLink messages *Landing_Target* and *Distance_Sensor* to the Pixhawk 4. The broadcast rate is exactly what was expected and desired (Fig. 8).

Firmware ArduPilot and MAVLink protocol fully support the precision landing feature. Although, during the real-world flight tests, there was a slight oscillation above the landing position, the successful test flights yielded desirable results. To compare the proposed solution to those mentioned in Section 2, Table 3 summarizes the results achieved, showing the main differences between them.

Table 3. Comparison of results.

Source	Accuracy [m]	Maximum altitude [m]	Landing speed [m/s]	Outdoors	Moving target
Our algorithm	0.05 – 0.11	8 – 50*	0.2 – 0.3	Yes	No
GPS – based	1 – 3	∞	0.3 – 2.0	Yes	No
ArduPilot PL with IR-Lock sensor [2]	< 0.3	15	n/a	Yes	Yes**
Araar et al. [3]	0.13	0.8	0.06	No	Yes
Patruno et al. [4]	0.01	n/a	n/a	Yes	No
Chen et al. [5]	n/a	2.5	0.23	No	Yes
Lange et al. [6]	0.03 – 0.23	0.7	n/a	Yes	No

** Depending on the size of the ArUco markers on the ground and the resolution of the camera.*
*** Apply for both the moving target and the stationary one.*

Our approach applies to the Vertical Take-Off and Landing (VTOL) UAVs. These UAVs have the flight controller that must support the ArduPilot firmware and an onboard computer. The real-world flight tests were performed in near-perfect weather conditions: outdoors, averaging wind speeds of 3-5 m/s, in the bright daylight condition to ensure quick target detection. Under comparable circumstances, ArduPilot community trials on landing by GPS or with an IR-Lock sensor [2] were also carried out.

Experiments by Araar et al. [3] performed with the AR.Drone 2.0 quadrotor from Parrot. Chen et al. [5] worked with a T-Lion board, which installs the flight controller, powerful processors and several essential sensors together. These two proposals are carried out in-house without considering environmental factors.

For the experiment of Patruno et al. [4], detecting the geometric properties of the H-shaped mark adopted by a monocular camera is challenging. However, the proposed approach based on a Point-to-Line Distance method is outstanding. They achieved the average RMSE values of 0.0137 m and 1.04° in the position and orientation estimations. The experiment did not mention the altitude at which the drone can land and the achieved landing speed.

Lange et al. [6] also used OpenCV to recognize a landing pattern, a set of concentric circles with a maximum diameter of 45 cm. One benefit of this approach is that the UAV does not need to see the complete marker to recognize the landing platforms. The camera, on the other hand, could only detect the target at less than 70 cm. These two approaches [4], [6] do not mention much the flying conditions.

6 Conclusions and Future Work

This study has essentially completed the project requirements' objectives. To begin with, the hardware system fully meets the requirements for landing from a specific height. The ArUco markers could be detected with the camera sensor employed in this project at heights ranging from 8 to 30 m, depending on the size of the landing platform on the ground. In terms of software architecture, the DroneKit-Python API synchronously compiles MAVLink signals transmitted from the NVIDIA Jetson Nano to the Pixhawk 4, making it simple to grasp even for beginners to start designing a drone program. For this approach of landing target detection, OpenCV utilized in the image processing, is totally responsive and powerful.

The precision landing approach was evaluated using the ROS/Gazebo simulator. The accuracy of this method of landing is very impressive, possibly reaching less than 20 cm. The landing time can be adjusted using the vertical landing speed parameter, directly in the ArduPilot firmware.

Finally, this landing approach employs a single camera sensor. It serves as both a target detector and a rangefinder providing depth parameters from a ground marker. As a result, this is a cost-effective solution that eliminates the need for any depth sensor such as LiDAR or Sonar.

The algorithms in this landing method are not the most optimal but have met the precision landing needs as the requirements. In the future work, the algorithm can be improved by integrating the UAV's state (acceleration, velocity, etc.) in the control loop to improve the flight behavior. It is also planned to

use cascaded ArUco markers instead of using 1 or 2 ArUco markers as the landing platforms. The landing accuracy and speed will both improve as a result of this. Finally, the algorithm should be further developed for scenarios of night vision and less favorable weather conditions.

Acknowledgment

I am extremely grateful to my supervisor, COL. Assoc. Prof. Josef Bajer, Ph.D. for his invaluable academic support and suggestions for the problem-solving in this work. I would like to express my deepest thanks to my consultant MAJ. Eng. Václav Křivánek, Ph.D. for the equipment involved, and for advising me on the camera sensors and the drone landing procedures. I would also like to express my very appreciation to my co-consultant MAJ. Eng. Radek Bystřický, Ph.D. and my enthusiastic colleague Bc. Radek Vala for always directly supporting me in my practical experiments, designing and building the drones, hardware repairing, and monitoring to ensure the safety of the real-world flight tests.

References

[1] X. Pan, D.Q. Ma, L.L. Jin, Z.S. Jiang. *Vision-based approach angle and height estimation for UAV landing*. In Proceedings of the 1st International Congress on Image and Signal Processing, CISP 2008, Sanya, China, 27–30 May 2008; Volume 3, pp. 801–805.

[2] ArduPilot Dev Team. *Precision Landing and Loiter with IR-LOCK*. ArduPilot - Versatile, Trusted, Open [online]. 27.02.2022 [cit. 2022]. Available online: https://ardupilot.org/copter/docs/precision-landing-with-irlock.html (accessed on 17.04.2022)

[3] O. Araar, N. Aouf, & I. Vitanov. *Vision Based Autonomous Landing of Multirotor UAV on Moving Platform*. J Intell Robot Syst 85, 369–384 (2017). https://doi.org/10.1007/s10846-016-0399-z

[4] C. Patruno, M. Nitti, A. Petitti, et al. *A Vision-Based Approach for Unmanned Aerial Vehicle Landing*. J Intell Robot Syst 95, 645–664 (2019). https://doi.org/10.1007/s10846-018-0933-2

[5] X. Chen, S. K. Phang, M. Shan and B. M. Chen, "*System integration of a vision-guided UAV for autonomous landing on moving platform*," 2016 12th IEEE International Conference on Control and Automation (ICCA), 2016, pp. 761-766, doi: 10.1109/ICCA.2016.7505370

[6] S. Lange, N. Sunderhauf and P. Protzel, "A vision based onboard approach for landing and position control of an autonomous multirotor UAV in GPS-denied environments," 2009 International Conference on Advanced Robotics, 2009, pp. 1-6.

[7] T. H. Nguyen, M. Cao, T. -M. Nguyen and L. Xie, "*Post-Mission Autonomous Return and Precision Landing of UAV*," 2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV), 2018, pp. 1747-1752, doi: 10.1109/ICARCV.2018.8581117.

[8] J.P.C. De Souza, A.L.M. Marcato, E.P. de Aguiar, et al. Autonomous Landing of UAV Based on Artificial Neural Network Supervised by Fuzzy Logic. J Control Autom Electr Syst 30, 522–531 (2019). https://doi.org/10.1007/s40313-019-00465-y

[9] Dronecode Project. *Introduction - MAVLink Developer Guide*. MAVLink Developer Guide [online]. [cit. 2022]. Available online: https://mavlink.io/en/ (access 15.03.2022)

[10] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[11] ZIWEN Jiang. *An Autonomous Landing and Charging System for Drones*. Department of Electrical Engineering and Computer Science, MIT, 2019. Master's thesis. Massachusetts Institute of Technology. Thesis supervisor: Prof. Hari Balakrishna.

[12] D. Tang, F. Li, N. Shen and S. Guo, "*UAV attitude and position estimation for vision-based landing*," Proceedings of 2011 International Conference on Electronic & Mechanical Engineering and Information Technology, 2011, pp. 4446-4450, doi: 10.1109/EMEIT.2011.6023131.

[13] Dronecode Project. *Landing Target Protocol*. MAVLink Developer Guide [online]. [cit. 2022]. Available online: https://mavlink.io/en/services/landing_target.html (accessed on 01.05.2022)

Traffic Capture Infrastructure

Bc. Lukáš Hejcman, David Beneš
CESNET a.l.e.
Zikova 4, 160 00 Prague 6, Czech Republic

{hejcman, benesdavid}@cesnet.cz

Abstract. Diverse and high-quality datasets are the cornerstone of any machine learning algorithm development and research. However, creating such datasets in the context of modern high speed, distributed, and privacy sensitive networks can be problematic. This is why we propose a solution in the form of Traffic Capture Infrastructure; a one-stop system solution for creating, processing, and handling large datasets created from network traffic across a large number of network nodes. Furthermore, our system supports extensive user management features to ensure dataset privacy and system integrity. In this paper we present the architecture and working of this system. We present its advantages and possible improvements. Lastly, we prove the value of this system with a number of publications that have used our system for creating their underlying dataset.

Keywords. Traffic, Capture, Infrastructure, Dataset, TCI.

1 Introduction

The quantity and quality of underlying datasets is seen to be one of the main limiting factors in the development of new machine learning algorithms [1]. However, data processing can also be very time consuming and is widely seen as a one of the least enjoyable tasks in data science [2]. Furthermore, the difficulty of preserving data privacy for sensitive datasets is self evident from the recent increase in the interest in the methods of collaborative learning, which are designed to allow training models without sharing their underlying dataset [3].

A number of existing solutions exist for capturing network data across a number of nodes. The main of these is using the IPFIX protocol. However, this approach is not suitable for deep packet inspection of the underlying network traffic. IPFIX data exporters extract information from the underlying network flow, and therefore contain only a subset of information present in the captured network traffic [4]. Contrary to this, PCAP files present a one-to-one copy of the transferred data, which is necessary to conduct low level research of the transferred data, as it contains information overlooked using the network flow approach.

In this paper, we propose a developed system for network dataset creation called Traffic Capture Infrastructure (TCI). This system is designed to be a one stop solution for creating datasets of network traffic at a large scale, including their capture, processing, user management, and more. This system has already been deployed on the infrastructure of CESNET (Czech Education and Scientific NETwork), a developer and operator of national e-infrastructure for science, research, development and education in the Czech Republic. The system has been used successfully as a part of multiple papers, publications, and open-source datasets.

In this paper, we will examine existing solutions in this space (section 2) and the structure and working of our proposed system (section 3). We will further present an overview of possible use-cases

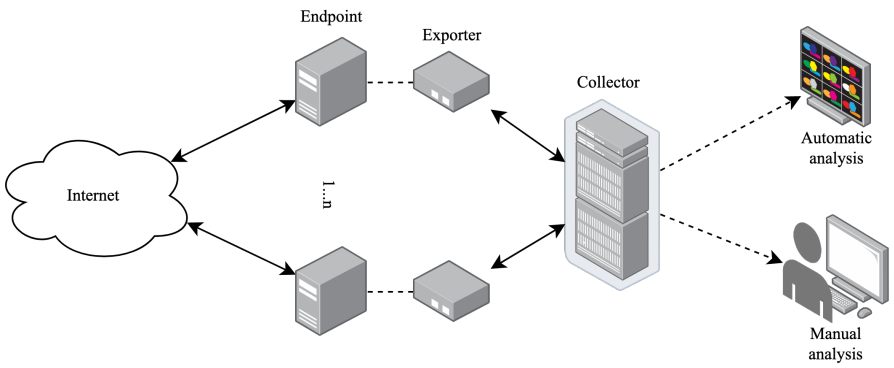


Figure 1: Scheme of a flow capture and processing systems.

(section 4) and publications which were written with the usage of TCI created datasets (section 5). Lastly, we will outline possible improvements to the system (section 6).

2 Related Work

The most common approach for distributed capturing of network traffic are systems based on the IPFIX and NetFlow standards [5]. These systems work on the basis of flows, which represent statistical information about the underlying network communication. As such, network flows are usually not concerned with the contents of transferred packets but only their headers. This makes them suited to deployments on high speed networks, as they are able to monitor transferred traffic without high computational overhead.

These systems work on the basis of exporters and collectors. Exporters are responsible for creating network flows which are then sent to collectors. Each collector can receive data from 1 to n exporters based on the underlying implementation and performance considerations. An example of such a system can be seen in figure 1.

However, since flows usually don't process the contents of packets but only their headers, they contains less information about the underlying transferred data. As such, they are useful for high level overview of the network state rather than low level inspection of the network traffic. They also have limited use in the domain of specific protocol research, which requires methods such as Deep Packet Inspection (DPI) [6].

In the commercial space, tools such as the Flowmon Packet Inspector exist [7]. Unlike the IPFIX suite of software, this tool works on the basis of PCAP files. This tool is however aimed at network administrators and maintainers, as it provides built-in tools for packet and traffic analysis. Our proposed system doesn't include such analysis tools, but they can be integrated into the system using its processing capabilities (see section 3.3.3) or the available REST API (see section 3.3.4).

Furthermore, the Flowmon Packet Inspector isn't aimed at creating and managing large and privacy sensitive datasets, and as such doesn't include a system for setting different user roles and authorizations. Lastly, the Flowmon Packet Inspector is capable of traffic capture on networks with a throughput of up to 100 Gbit per second. However, our system has been deployed on networks with multiple nodes capable of a throughput of upwards of 200 Gbit per second each. We are also making plans for including 400 Gbit cards into our deployment.

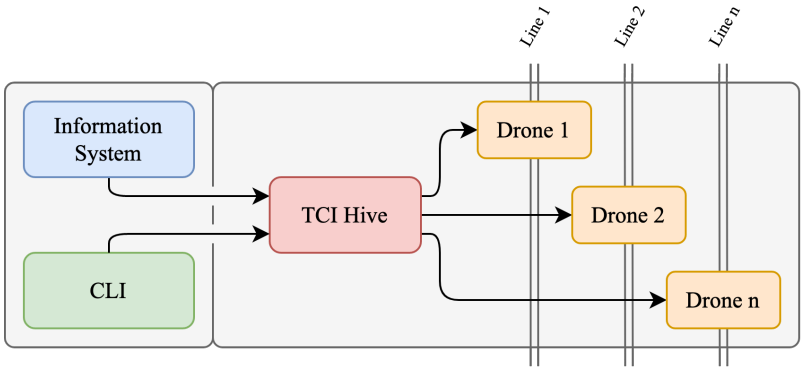


Figure 2: A structural overview of the TCI system.

3 System overview

The Traffic Capture Infrastructure (TCI) system takes inspiration from the existing network flow systems and is composed of multiple interconnected modules; an overview of which can be seen in figure 2. The traffic capture itself is managed by the TCI hive and the TCI drones, which work similarly to a flow collector and exporter respectively. This backend captures the traffic using 1 to n drones. The captured data is then collected and merged using the TCI hive.

The whole backend system can be controlled and managed using the other distinct part of the TCI system, the Command Line Interface (CLI) or the Information System (IS). The CLI allows for a low level access to the TCI system, which can be used for initial setup, administrator management, or in environments where no further functionality is required from the TCI system. On the other hand, the information system makes TCI a system which can be comfortably deployed and integrated into existing infrastructure in an academic or a corporate environment, where privacy and integrity considerations arise.

The whole TCI system is written in Python using the Flask [8] framework. Flask was selected due to its lightweight size and large number of extension libraries. The system uses SQLAlchemy [9] as an ORM which gives us the flexibility of deploying the system using a large number of available database backends.

What follows is a high level overview of the different system modules.

3.1 Backend

The backend of the system is composed of the hive and 1 to n drones. The job of the hive is managing the communication and the state of the different drones. When a new capture command is sent to the hive, it contacts all the relevant drones and instructs them to capture data with the specified settings. It then controls the state of the drones and collects the captured data from them to be merged. The communication between the drones and the hive happens through a REST API.

The drone receives commands from the hive and executes them. The drone can be configured to contain multiple capture utilities, such as `tcpdump` [10] or the Network Development Kit (NDK) [11] for capturing data on high speed network accelerators. This makes the drone portable as it can be used on any custom device with support for packet capture, such as custom network accelerator cards. The captured data is saved locally on the drone until it is requested by the hive.

Because the TCI system can be deployed on high speed networks, it is important to limit the data that is stored by the drone. For example, saving all the data passing through a 100 Gbit network connection to a PCAP file would easily overwhelm the storage capabilities of the drones. It is therefore possible to

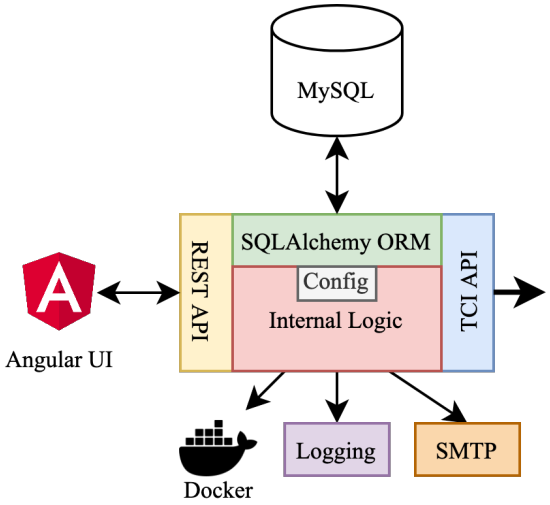


Figure 3: Structure of the TCI Information System.

use a custom Hardware Accelerated Network Interface Card (HANIC) filter [12], which limits the scope of the data that can be captured by the drone.

3.2 Command Line Interface (CLI)

The command line interface presents a very simple access point to the TCI system, and it gives the user full access to the functionality of the TCI backend. As such, it is meant for setup and troubleshooting of the system by administrators, and all users are expected to access the system through the IS. The CLI communicates with the hive using a REST API which supports encryption using TLS 1.3.

3.3 Information system (IS)

As you can see in figure 2, the backend made up of the drones and the hive serves the whims of the Information System or the CLI. As such, all higher level functionality is implemented in the Information System, which makes the TCI system enterprise ready.

The structure of the information system can be seen in figure 3. As can be seen from the figure, the user interacts with the information system using an Angular web interface. This front-end framework was chosen for its stability and cross platform availability. The information system communicates with the TCI hive using the TCI API library, which wraps internal logic for communicating with the hive REST API.

What follows is the description of the most important tasks of the information system.

3.3.1 Traffic capture

The capture of network traffic can be triggered by creating a new capture job. A capture job is a request to the TCI system for capturing network data with certain user-set options. A simplified overview of a capture job flowchart can be seen in figure 4.

The most important attribute of a job is the satisfaction criteria, which determine when the capture of network traffic can conclude. The two criterions are the maximum number of captured data or the maximum capture time. The TCI system keeps track of both of these criteria, and concludes the capture job when either of them is satisfied.

A user must further specify 1 to n capture lines on which this job should be captured. Currently the TCI system doesn't allow concurrent capture of two jobs on the same line, so a job halts until all of the specified lines are free. The amount of data captured is taken to be the sum of data captured across all the nodes. For example, if a user requests a capture of 20 GB on two lines with a throughput of 100 Gbit and 50 Gbit respectively, it is likely that the amount of captured data will be in the ratio of $2 : 1$ across these two lines.

The user can also specify a filter which will be used for filtering the captured traffic. Since the system can be deployed on high speed networks, filtering traffic is important to limit the memory and processing requirements of the capture process.

After creating a capture request, the user can still interact with it by cancelling the request before or during capture. Furthermore, the users can view the progress of the job in realtime.

3.3.2 User management and privileges

All users accessing the TCI system must do so through their user account. This account can be manually created by the administrator or it can be automatically created by the system using LDAP integration. Each user has an assigned authorization level from the authorization level hierarchy. This authorization level contains an ID and the privileges that the users of that level have when working with the TCI system. Administrators can create new authorization levels with custom privileges, which can then be assigned to users of the system. The main authorizations that can be assigned to users are:

Editing users allows users to edit user accounts of all users with a lower authorization level.

Editing authorizations allows users to edit and create authorization levels for all levels under the current user's level.

Viewing the system log can be done only by authorized users as it contains a history of all the user interactions with the system. They could therefore view how other users use the system which would present a break of privacy.

Being a job master gives users the privilege to control other users' jobs. Normally, only the author of a certain capture job can modify it (stop it, cancel it, etc.).

Being a script master allows users to upload custom non-administrator approved scripts to the TCI system for processing the captured traffic. This authorization also allows users to bypass processing the data. All data is processed using an anonymization script by default.

3.3.3 Processing the captured jobs

Captured jobs can be processed using shell scripts after their capture. This gives the TCI system great power, as it allows the users to modify and prepare new datasets automatically without any further actions.

All the processing of the jobs occurs in a closed Docker container with prepared utilities for network traffic management and modification (eg. flow exporters, custom libraries, etc.). This container is not connected to the internet to prevent unauthorized leaking of unprocessed network traffic. Furthermore, the state of the Docker container is not persistent between sessions, so each new job is processed in a clean environment.

For privacy reasons all data is processed by an anonymization script by default, which modifies the IP addresses of the captured traffic. However, this can be bypassed by authorized users (see section 3.3.2) with the script master privilege. The users can upload custom scripts to the system, and assign them to created captured jobs.

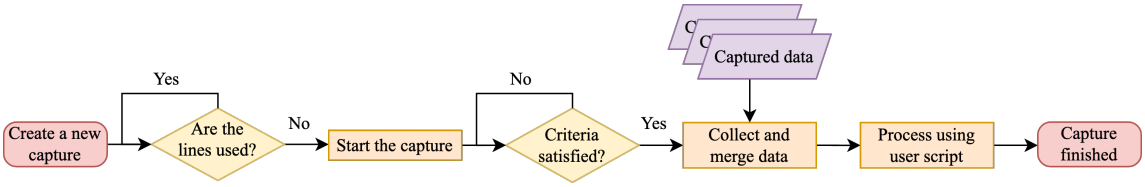


Figure 4: A simplified flowchart of a TCI capture.

3.3.4 Providing a public REST API

The information system provides a public REST API, so that users can interact with the system without accessing the web interface. This API allows the writing of user scripts which automatically start capture jobs based on some triggers. Just as the web interface, this API is secured and requires user login or a user token to access. The existence of this API allows integrating TCI with other tools and systems, and makes it a much more versatile tool.

4 Targeted real use-cases

4.1 Capturing live traffic

The TCI system is mainly aimed at researchers and scientists working in the field of computer networks and network monitoring. The expected usecase from this point of view is creating datasets by capturing live network traffic. If only a specific subset of traffic is required, it can be filtered using the built in filter functionality. This can be comfortably done using the information system UI. The advantage of using TCI compared with existing approaches such as `Wireshark` or `tcpdump` is the automatic and distributed nature of the captured data.

4.2 Creating a dataset using Deep Packet Inspection

Whilst capturing and working with raw traffic can be useful for some scenarios, most usecases require the parsing and processing of these packets using approaches such as Deep Packet Inspection (DPI). Users of the TCI system can use uploaded scripts and the provided Docker environment and its tools to automatically process the captured traffic. Using this approach is seamless for the researcher, as the process of merging and processing the captured traffic happens automatically after starting the capture request.

4.3 Network anomaly research

Due to the the nature of modern large scale networks and the global internet landscape, many computer networks are observing a long term trend in the growing number of anomalies [13]. Due to this threat, the development of accurate anomaly detection systems is paramount. Traffic Capture Infrastructure can be used for capturing ongoing anomalies such as Denial of Service attacks (DoS) for later analysis using the information system REST API and an automatic script. The distributed and low overhead nature of TCI makes it fit for this purpose.

5 Real world deployment and results

The described TCI system has been deployed on the infrastructure of CESNET. This long-term deployment has been in operation successfully for multiple months without any major downtime, and has been

used for creating hundreds of capture jobs. This instance is deployed on multiple nodes including network accelerator cards with a throughput of 200 Gbit. The deployed instance of the TCI system takes full advantage of the developed Information System to enable integration into the CESNET LDAP system for authorizing user access and their privileges within the system. This deployed system has already been used for multiple publications.

Luxemburk and Čejka used this system for capturing a large number of TLS messages, which were used for annotating existing flows with their Server Name Indicator [14]. This dataset was then used to implement and train a classifier which achieved a 97.04% classification accuracy and detected 91.94% of unknown services with a 5% false positive rate.

Luxemburk et al. again used the TCI system for annotating an existing dataset of millions of traffic flows with a packet-level feature set [15]. This dataset was then used for training a machine learning classifier with results that surpass the state-of-the-art solutions, and that has been shown to be viable for production deployment.

Tropková et al. used this system to develop and implement a new network traffic characteristic called Sequence of packet Burst Length and Time (SBLT) [16]. The value of this approach was shown using an implemented classifier with an achieved accuracy of 99%. The used dataset is also publicly available [17].

Smejkal used this system for creating a detailed dataset of SSH connections used for classification and user identity identification [18]. This dataset was then used for creating a classifier of SSH connections with an accuracy of up to 99.78%.

Hulák et al. have used the TCI system when developing a classifier for the classification of network traffic based using traffic features [19]. The TCI system was used for creating a dataset used to annotate an existing dataset of network flows [20]. The created classifier based on decision trees achieves an accuracy upwards of 88%.

The TCI system is also being used by CESNET-CERTS, a computer security incident response team [21], in cooperation with CESNET network operators. This team uses TCI to capture samples of Distributed Denial of Service (DDoS) data in order to increase the accuracy of existing attack mitigation and filtering rules. Along the same lines, Žádník has used traffic samples captured using our system for the improvement of the inference of DDoS mitigation rules [22].

Furthermore, Jeřábek et al. created a public dataset of DNS over HTTPS traffic [23]. This dataset, amongst other datasets being created by the TCI system, is being used for the development of future publications.

6 Future development

The main improvements that are planned for the TCI system are centered around user comfort and system autonomy. The main drawback of the TCI system is currently the fact that users must manually transfer the created and processed dataset to the server where further development will take place. The proposed solution to this problem is to allow users to automatically set a destination for each capture job. After processing, the TCI system will connect to the desired destination using a user specified SSH key and transfer the data.

Another perceived problem is regarding the deployment of the TCI system. From an administrator point of view, it would be useful to include storage management into the TCI system. The administrator could set data usage quotas to different users, who could not capture jobs larger than a certain threshold.

7 Conclusion

In this paper we presented an existing Traffic Capture Infrastructure system. This system was created as a one-stop solution to creating and processing datasets of network traffic for researchers and administrators of computer networks. We presented an overview of the structure and architecture of the system, and explained its main functionality. We gave an overview of the main usecases of the system, and proved its usefulness using a list of publications which used the TCI system during their development.

Acknowledgement

This work was supported by the Ministry of Interior of the Czech Republic (Flow-Based Encrypted Traffic Analysis) under grant number VJ02010024.

References

[1] Alon Halevy, Peter Norvig, and Fernando Pereira. "The Unreasonable Effectiveness of Data". In: *IEEE Intelligent Systems* 24.2 (Mar. 2009), pp. 8–12. ISSN: 1541-1672. DOI: 10.1109/MIS.2009.36. URL: http://ieeexplore.ieee.org/document/4804817/ (visited on 05/17/2022).

[2] Gil Press. *Cleaning Big Data: Most Time-Consuming, Least Enjoyable Data Science Task, Survey Says*. Forbes. URL: https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/ (visited on 05/17/2022).

[3] Balazs Pejo, Qiang Tang, and Gergely Biczok. "Together or Alone: The Price of Privacy in Collaborative Learning". Aug. 24, 2018. DOI: 10.48550/arXiv.1712.00270. arXiv: 1712.00270 [cs]. URL: http://arxiv.org/abs/1712.00270 (visited on 05/26/2022).

[4] Paul Aitken, Benoît Claise, and Brian Trammell. *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information*. Request for Comments RFC 7011. Internet Engineering Task Force, Sept. 2013. 76 pp. DOI: 10.17487/RFC7011. URL: https://datatracker.ietf.org/doc/rfc7011 (visited on 05/26/2022).

[5] Rick Hofstede et al. "Flow Monitoring Explained: From Packet Capture to Data Analysis With NetFlow and IPFIX". In: *IEEE Communications Surveys Tutorials* 16.4 (2014), pp. 2037–2064. ISSN: 1553-877X. DOI: 10.1109/COMST.2014.2321898.

[6] Reham Taher El-Maghraby, Nada Mostafa Abd Elazim, and Ayman M. Bahaa-Eldin. "A Survey on Deep Packet Inspection". In: *2017 12th International Conference on Computer Engineering and Systems (ICCES)*. 2017 12th International Conference on Computer Engineering and Systems (ICCES). Dec. 2017, pp. 188–197. DOI: 10.1109/ICCES.2017.8275301.

[7] Flowmon. *Flowmon Packet Investigator*. URL: https://www.flowmon.com/en/products/software-modules/packet-investigator (visited on 05/28/2022).

[8] *Welcome to Flask — Flask Documentation (2.1.x)*. URL: https://flask.palletsprojects.com/en/2.1.x/ (visited on 05/26/2022).

[9] *SQLAlchemy - The Database Toolkit for Python*. URL: https://www.sqlalchemy.org/ (visited on 05/26/2022).

[10] Harris Guy. *TCPDUMP by The Tcpdump Group*. Version 4.x.y. The Tcpdump Group, May 26, 2022. URL: https://github.com/the-tcpdump-group/tcpdump (visited on 05/26/2022).

[11] Jakub Cabal and Radek Iša. *NDK Minimal Application*. CESNET, May 8, 2022. URL: https://github.com/CESNET/ndk-app-minimal (visited on 05/26/2022).

[12] Lukáš Huták. *Hardware Accelerated Network Interface Card*. CESNET Liberouter. URL: https://www.liberouter.org/technologies/hanic/ (visited on 05/26/2022).

[13] Mohiuddin Ahmed, Abdun Naser Mahmood, and Jiankun Hu. "A Survey of Network Anomaly Detection Techniques". In: *Journal of Network and Computer Applications* 60 (Jan. 2016), pp. 19–31. ISSN: 10848045. DOI: 10.1016/j.jnca.2015.11.016. URL: https://linkinghub.elsevier.com/retrieve/pii/S1084804515002891 (visited on 05/26/2022).

[14] Jan Luxemburk and Tomáš Čejka. "Fine-Grained TLS Services Classification with Reject Option". Feb. 24, 2022. DOI: 10.48550/arXiv.2202.11984. arXiv: 2202.11984 [cs]. URL: http://arxiv.org/abs/2202.11984 (visited on 05/26/2022).

[15] Jan Luxemburk, Karel Hynek, and Tomáš Čejka. "Detection of HTTPS Brute-Force Attacks with Packet-Level Feature Set". In: *2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC)*. 2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC). Jan. 2021, pp. 0114–0122. DOI: 10.1109/CCWC51732.2021.9375998.

[16] Zdena Tropková, Karel Hynek, and Tomáš Čejka. "Novel HTTPS Classifier Driven by Packet Bursts, Flows, and Machine Learning". In: *2021 17th International Conference on Network and Service Management (CNSM)*. 2021 17th International Conference on Network and Service Management (CNSM). 2021, pp. 345–349. DOI: 10.23919/CNSM52442.2021.9615561.

[17] Zdena Tropková, Karel Hynek, and Tomáš Čejka. *Dataset Used for HTTPS Traffic Classification Using Packet Burst Statistics*. June 8, 2021. URL: https://zenodo.org/record/4911551 (visited on 05/26/2022).

[18] Radek Smejkal. "Klasifikace Komunikace SSH Protokolu". České vysoké učení technické v Praze. Vypočetní a informační centrum., June 3, 2021. URL: https://dspace.cvut.cz/handle/10467/94562 (visited on 05/26/2022).

[19] Matej Hulák and Tomáš Čejka. "Classification of Network Traffic Using Traffic Features". In: Prague Embedded Systems Workshop. Prague, 2020. ISBN: 978-80-01-06772-7.

[20] Matej Hulák. "Klasifikace Provozu a Zařízení v Počítačových Sítích Na Základě Toků". České vysoké učení technické v Praze. Vypočetní a informační centrum., Sept. 3, 2020. URL: https://dspace.cvut.cz/handle/10467/90394 (visited on 05/27/2022).

[21] CESNET a.l.e. *CESNET-CERTS*. URL: https://csirt.cesnet.cz/ (visited on 05/28/2022).

[22] Martin Žádník. "Towards Inference of DDoS Mitigation Rules". In: IEEE/IFIP Network Operations and Management Symposium. Budapest, 2022.

[23] Kamil Jeřábek et al. *DNS over HTTPS - Real World Dataset*. Zenodo, Feb. 3, 2022. DOI: 10.5281/zenodo.5956044. URL: https://zenodo.org/record/5956044 (visited on 05/26/2022).

Classification of network traffic

Matej Hulák, Tomáš Čejka
FIT Czech Technical University in Prague
Thákurova 9, Prague 6

matej.hulak@cvut.cz, cejkato2@fit.cvut.cz

Abstract. This paper describes the context of existing approaches to real-time network flow classification and focuses on the contributions of bachelor and master thesis of the author. The paper also proposes several research questions that are planned for the future Ph.D. study.

Keywords. classification, network traffic analysis, machine learning, NEMEA, network flows

1 Introduction

Computer networks are getting bigger and more complicated every day. Communication over the internet is gradually becoming one of the basic people's needs. Popularity of smart homes, phones, watches, fridges and in general smart devices, increased network usage of many households and this phenomenon has become known as the Internet of Things (IoT). Furthermore, the recent pandemic situation drastically intervened. Many people started using computer networks to connect to their jobs remotely on a daily basis or moved other daily activities, like shopping and entertainment, to the internet. These factors accelerated network's growth and importance. Unfortunately, this growth also caused an increased number of cyber attacks, as it is discussed by Monteith et al. in [1].

To counteract this, security of devices and networks must be researched. Pahi et al. [2] claim one of the essential requirements of network security is the knowledge and understanding of the current situation on the network. Network monitoring systems and processes play an essential role in maintaining visibility into network traffic and overall situational awareness. Moreover, identification and classification of network traffic are useful and core components in the whole ecosystem of tools for network security.

2 Classification Techniques

There are many approaches to network traffic classification known in practice or in literature. One of the most basic classification methods is based on port numbers of a transport protocol. It uses well-known registrations of port numbers by the most popular applications that are maintained and assigned by IANA [3]. Based on the port number, we can assume what service and protocol is running on an unknown device. Therefore, the network protocol can also be recognized based on port number. However, this method has its drawbacks — some of the protocols use multiple ports and some ports are used by multiple protocols. In addition, an application can usually be configured to use an arbitrary port number. If someone wants to hide

a malicious connection, he can easily change the port number to some non-standard port, which will lead to false classification.

Another widely used method of classification is based on deep packet inspection. Every packet contains protocol headers. These headers are used to navigate packet through network and to assure packet integrity. Every used protocol will add its header to the packet. These headers can be extracted during packet analysis, which will determine used application protocol. This method is very accurate and reliable when the communication is not encrypted. One significant downside of this method is the necessity to capture and analyze the whole packet, including user data, thus method uses significant CPU time and memory. As it is discussed in [4], packet analysis in high-speed network is not favourable, and methods based on IP flows (aggregated information about sequences of packets represented by flows) are preferred, as they are more scalable.

IP flows are used by many modern monitoring systems, especially ones targeting high-speed networks. One of them is the NEMEA system [5]. Our objective was the research and analysis of solutions and implementation of a classification module for the NEMEA system, capable process and classify flows in real-time. During our research, we were able to create two classification modules which are presented in next sections.

The latest approaches to network analysis and classification are based on machine learning (ML) technology [6][7]. The main idea is to exploit statistical information from packets and flows (or eventually bytes of the packets) and reveal characteristic patterns as classification based on port number or packet data has its limitations [8].

3 Statistical classifier

Classification of network traffic relies on provided IP flow data. Different flow exporters provide different levels of information. IP Flow exporters, which are used to generate IP Flow data, provide at least basic information presented in Table 1. Additionally, there are numerous extensions which add additional detailed information on the network traffic, such as application layer (L7) headers from the monitored packets.

Table 1: Basic flow information, provided by ipfixprobe [9]

Data Type	Field Name	Description
ipaddr	SRC_IP	Source address of a flow
ipaddr	DST_IP	Destination address of a flow
uint16	SRC_PORT	Source transport-layer port
uint16	DST_PORT	Destination transport-layer port
uint8	PROTOCOL	L4 protocol (TCP, UDP, ICMP, etc.)
uint32	PACKETS	Number of packets in a flow or in an interval
uint64	BYTES	Number of bytes in a flow or in an interval
time	TIME_FIRST	Timestamp of the first packet of a flow
time	TIME_LAST	Timestamp of the last packet of a flow
uint8	TCP_FLAGS	TCP flags of a flow (logical OR over TCP flags field of all packets)

One of the possible approaches to classification is the usage of basic IP flow information and statistical heuristics. This approach was elaborated and tested in the bachelor thesis [10]. The

main goal of this thesis was to create real-time network classification tools based on basic flow data information only.

The thesis presented an application of confidential intervals derived using the created training datasets. In total, nine classification models were created and tested: http, https, imap, pop3, smtp, ssh, telnet, dns, ntp. Every classification model held pre-calculated median vicinities of relevant features and most occurred values of categorical values. The module used five features: duration, bytes count, packets count and L4 protocol. During the evaluation, every feature of the investigated flow record was compared with the median vicinities of every model. Assigning of a resulting class was performed using a designed scoring system. The model with the highest score was selected as the result of the classification process.

The evaluation was performed using the 10-cross validation [11]. Results are presented in Table 2.

Table 2: Results of classification module [10]

Label	Flow count	True positive [12]	Unclassified	False positive [12]	Accuracy [12]
http	16 131	16 059	72	0	99,55 %
https	14 697	14 173	523	1	96,43 %
imap	21 725	20 478	849	398	94,26 %
pop3	20 121	17 429	1854	838	86,62 %
smtp	33 020	29 105	2952	936	88,22 %
ssh	33 281	29 842	444	2995	89,67 %
telnet	39 471	36 043	1519	1909	91,29 %
dns	61 772	59 257	745	1770	95,93 %
ntp	2 750 865	2 569 889	180 976	0	93,42 %
				Average accuracy	92,82 %

The classification module achieved good results and was deployed on a laboratory network for long time reliability test, which was successful.

3.1 ML-based classification

Nowadays, a completely different approach to classification that can exploit larger sets of traffic features is represented by machine learning algorithms. Machine learning methods are very helpful in recognizing relevant features and dependencies and relations between them. Possibilities of network traffic classification using machine learning methods were reviewed in the master thesis [13]. The primary goal of this work was to evaluate different machine learning methods on the data sets which contained different levels of information.

The first part of the research focused on the evaluation of 3 distinct data sets:

- UniFlow - basic unidirectional flow (NetFlow v5)
- BiFlow - basic bidirectional flow (NetFlow v5)
- Extended BiFlow - BiFlow enriched by PSTATS

Data sets for training and evaluation were captured on network CESNET2[1] and were annotated by TShark application. Every data set contains 360-thousand total and 40-thousand flows of each label (http, https, imap, pop3, smtp, ssh, telnet, dns, ntp). Unnecessary features and

[1]National research and education network infrastructure in the Czech Republic.

port numbers were removed. Every data set contains different levels of information, differences are explained in the list above.

Extension PSTATS contains information about the first packets of IP-flow (default: 30). The extension gathers four types of information: bytes count, timestamp, direction of packet and TCP flags. This information can be very useful, as we are gaining more information about the first part of the connection, which can be crucial, as many protocols start communication in a different manner. Machine learning methods cannot process these features directly as they are formed by arrays. For feature extraction, we used Feature Exploration Toolkit (FET)[2], which extracted statistical values (median, min, max). Comparison of data sets was performed using scikit-learn library, which implements many classification models and evaluation functions. For comparison, we used method of decision trees. This method is very comprehensible and can be easily interpreted[15]. For the evaluation, we used 10-cross validation [11]. The results of the evaluation on different types of data sets are presented in Table 3.

Table 3: Data sets comparison.

Data set	Accuracy [12]	Precision [12]	f1-score [12]
UniFlow	84,7 %	84,7 %	84,7 %
BiFlow	90,6 %	90,7 %	90,7 %
Extended BiFlow	95,8 %	95,9 %	95,8 %

Results of this experiment proved the expected additional value of the data sets with a higher level of information, which led to achieving better results with a minimal time penalty.

The second part of the experiment was focused on classification methods comparison. We selected nine methods from the scikit-learn library based on the recommendations from documentation, and we performed experiments on the dataset Extended BiFlow. The size of the dataset was reduced to 27-thousand flows due to time complexity, and only 3-cross validation was used. The results are presented in Table 4.

Table 4: Classification methods comparison.

Classification method	Accuracy [12]
DT	91,7 %
KNN	80,8 %
Extra-tree	93,9 %
Random forest	93,8 %
Ada Boost(DT)	93,8 %
Gradient	91,5 %
Naive Bayes	22,0 %
MLP	67,1 %
Linear SVM	60,9 %

We recognized Extra-tree method and Extended BiFlow dataset as the most effective, and we proceeded to implement classification module which will use these techniques.

[2]FET was developed as a support tool in the thesis by Uhříček [14]

3.2 Developed Classification Module

In the master thesis [13], a new classification module has been developed based on the experiments. It is divided into two programs: the first one is used to create classification model and provides graphic interface and overview of dataset and its proprieties; the second program is used for real-time classification and is designed to run without user control, and implementation is focused on time efficiency. The trained ML model is saved in a dedicated file and is loaded during the start. This feature allows to distribute the created model among several devices.

The classification module was tested using Extended BiFlow dataset and achieved similar results as the classifier in the experiments. According to the premature test, the module is capable classify 55-thousands flow per second. Results are presented in Table 5.

Table 5: Classification module test results

Classifier	Accuracy [12]	Precision [12]	f1-score [12]
Classification module(Extra-tree)	94,8 %	95,2 %	94,8 %

Classification module is currently deployed in the laboratory network at FIT CTU, and it is currently continuously tested.

4 Research Questions

This work is the first stepping stone to the further experiments and research in the future Ph.D. thesis. In the doctoral study, we are planning to continue our work in this area and deal with the following research questions:

1. Is it possible to estimate approximate performance of the machine learning algorithm based on the given training and evaluation dataset?
2. Is it possible to predict what machine learning algorithm will provide sufficient performance and what computation resources it will require?
3. Is it possible to find a machine learning model based on simple algorithms to replace more complex ones in order to save required resources?

5 Conclusion

During this research, we were able to create and test two classification modules and perform extensive experiments with machine learning methods. Differences between different classification modules are currently being evaluated.

This paper briefly summarized existing approaches to network traffic classification and described the evaluated methods from bachelor and master theses by the author. This activity is the base of the future Ph.D. research on network traffic classification.

5.1 Future work

During our research, we recognized that many works [16][17], including ours, were focused on finding the best machine learning classification method for some application. In our work and many others, method of brute force was used to find the most effective and suitable method. We recognized this behaviour as not ideal, as classifiers are usually tested only in default settings. In addition, in some cases, its needed to deploy machine learning classifier as fast as possible and

searching adequate method by brute force is not possible. In general, the success of machine learning deployment heavily relies on human experts, who manually select appropriate ML architecture and their hyperparameters [18]. This topic caught our interest, and in the future, we would like to create a method or guideline for choosing an adequate classification method based on data set properties and the purpose of the classifier.

Acknowledgment

This work was supported by the Grant Agency of the CTU in Prague, grant No. SGS20/210/OHK3/3T/18 funded by the Ministry of Education Youth and Sports of the Czech Republic.

References

[1] Monteith, S.; Bauer, M.; Alda, M.; aj.: Increasing cybercrime since the pandemic: Concerns for psychiatry. Current psychiatry reports, ročník 23, č. 4, 2021: s. 1–9.

[2] Pahi, T.; Leitner, M.; Skopik, F.: Analysis and Assessment of Situational Awareness Models for National Cyber Security Centers. In ICISSP, 2017, s. 334–345.

[3] Cotton, M.; Eggert, L.; Touch, D. J. D.; aj.: Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry. RFC 6335, Srpen 2011, doi:10.17487/RFC6335. Dostupné z: https://rfc-editor.org/rfc/rfc6335.txt

[4] Hofstede, R.; Čeleda, P.; Trammell, B.; aj.: Flow Monitoring Explained: From Packet Capture to Data Analysis With NetFlow and IPFIX. IEEE Communications Surveys Tutorials, ročník 16, č. 4, 2014: s. 2037–2064.

[5] Cejka, T.; Bartos, V.; Svepes, M.; aj.: NEMEA: a framework for network traffic analysis. In 2016 12th International Conference on Network and Service Management (CNSM), IEEE, 2016, s. 195–201.

[6] Williams, N.; Zander, S.; Armitage, G.: A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification. ACM SIGCOMM Computer Communication Review, ročník 36, č. 5, 2006: s. 5–16.

[7] Soysal, M.; Schmidt, E. G.: Machine learning algorithms for accurate flow-based network traffic classification: Evaluation and comparison. Performance Evaluation, ročník 67, č. 6, 2010: s. 451–467.

[8] Nguyen, T. T.; Armitage, G.: A survey of techniques for internet traffic classification using machine learning. IEEE communications surveys & tutorials, ročník 10, č. 4, 2008: s. 56–76.

[9] CESNET: IPFIX flow exporter [online]. GitHub. [vid. 2021-08-09]. Dostupné z: https://github.com/CESNET/ipfixprobe

[10] Hulák, M.: Klasifikace provozu a zařízení v počítačových sítích na základě toků. B.S. thesis, České vysoké učení technické v Praze. Vypočetní a informační centrum., 2020.

[11] Nature Inspired Technologies Group: Testování modelů a jejich výsledků [online]. ČVUT. [vid. 2020-05-01]. Dostupné z: https://cw.fel.cvut.cz/old/$_{m}edia/courses/a6m33dvz/03-testovanimodelu.pdf$

[12] Raschka, S.: An overview of general performance metrics of binary classifier systems. arXiv preprint arXiv:1410.5330, 2014.

[13] Hulák, M.: Klasifikácia sieťovej premávky pomocou strojového učenia. Diplomová práce, České vysoké učení technické v Praze. Vypočetní a informační centrum., 2022.

[14] Uhříček, D.: Detekce IoT malware v počítačovỳch sítích. Diplomová práce, České vysoké učení technické v Praze. Vypočetní a informační centrum., 2021.

[15] Arrieta, A. B.; Díaz-Rodríguez, N.; Del Ser, J.; aj.: Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. Information fusion, ročník 58, 2020: s. 82–115.

[16] Laštovička, M.; Dufka, A.; Komárková, J.: Machine learning fingerprinting methods in cyber security domain: Which one to use? In 2018 14th International Wireless Communications & Mobile Computing Conference (IWCMC), IEEE, 2018, s. 542–547.

[17] Rauber, T. W.; da Silva Loca, A. L.; de Assis Boldt, F.; aj.: An experimental methodology to evaluate machine learning methods for fault diagnosis based on vibration signals. Expert Systems with Applications, ročník 167, 2021: str. 114022.

[18] Hutter, F.; Kotthoff, L.; Vanschoren, J.: Automated machine learning: methods, systems, challenges. Springer Nature, 2019.

Detection of Cryptomining in High-speed Networks

Richard Plný, Karel Hynek
FIT CTU in Prague CESNET a.l.e.
Thákurova 9 Prague 6 Zikova 4, Prague 6

plnyrich@fit.cvut.cz, Karel.Hynek@fit.cvut.cz

Abstract. This paper addresses cryptomining from the security perspective with an emphasis on abusive mining. It explores the possibility of detecting cryptominers in high-speed computer networks using a flow-based monitoring approach. Based on the analysis of mining communication, we proposed detection method, which can be deployed on high-speed networks. The proposed solution was implemented as a group of NEMEA modules. Moreover, it was deployed and evaluated on the national network CESNET2 operated by CESNET.

Keywords. Cryptoming, Detection, High-speed Networks, Abusive Mining

1 Introduction

Cryptocurrencies have become widely popular since the Bitcoin's origins in 2008. People invest and also trade many different cryptocurrencies. But then, someone has to verify these transactions and add a new block to the cryptocurrency's Blockchain. This process is called *mining* — adding new blocks of verified transactions in exchange for a reward. Mining is done by many individuals (miners) who are trying to be the first one because only the first one will get a reward. This process is therefore very competitive.

Miners can increase the chance of being the first one by increasing their computational resources — more work they do, the higher chance they have to succeed. And this is often done by using computers of others, which can be against policy [1]. Another often way is to abuse the computational resources of others by *cryptomalware* [2, 3, 4].

Cryptomalware, mining malware, or illicit cryptomining refers to mining carried out by criminals using resources stolen from their victims [5]. It is a way to use more devices, increase your overall hash rate and the chance for a reward. Especially Monero became very popular when it comes to mining malware since it is known to be hard to trace [6]. Nearly 5% of Monero coins (with a value of almost $40 million at that time) in circulation in 2018 were mined by malware [7]. Specialists from McAffee [8] reported that "coin miner malware" grew more than 4000% in the year 2018.

Pastrana and Suarez-Tangil spent 12 years on their work [5] analyzing mining malware. They described two possible types of mining malware — *browser-based* and *binary-based* cryptomining. Browser-based cryptomining, also called *cryptojacking*, uses scripts to run in web pages (typically JavaScript). Mining process starts when a user visits a web page. Binary-based cryptomining uses malware to infect a machine connected to the Internet and then runs a binary that handles the mining process.

Mining malware also targets corporate networks [9]. Furthermore, UN Security Council (UNSC) found a malware mining Monero that sent mined coins to the servers located at Kim Il Sung University in Pyongyang. The Republic of Korea Financial Security Institute attributed a similar cryptojacking attack on a South Korean company [10].

Due to the increasing prevalence of abusing mining, detection of cryptomining in a computer network is viable for maintaining security. It can help security personnel reveal potentially abused or infected machines. Moreover, companies and others can save financial resources by securing abused machines since cryptomining consumes a lot of electricity [11]. Several previous attempts provided accurate detection methods, which were mainly based on Machine Learning (ML) [12, 13], but unfortunately not suitable for usage in high-speed networks.

In this work, we propose a novel cryptomining detector, which can be deployed in high-speed 100 Gbps backbone network lines and potentially protect millions of users. The proposed detector uses so-called weak-indication architecture, which aggregates multiple heterogeneous detectors into single, robust, and more precise information with a low false-positive rate.

The paper is organized as follows: Sec. 2 describes existing related works. Sec. 3 provides neccesarry theoretical background about cryptomining. Sec. 4 describes the proposed detection design. Sec. 5 provides evaluation methodology and its results. Finally, Sec. 6 concludes the paper.

2 Related Work

The study performed by Jingqiang et al. [14] focused on the detection of browser-based "silent miners". Their method uses a sandbox for loading a page and then analyzes the website's resources to detect Javascript (JS) miners. A similar approach was also studied by Kharraz et al. [15]; in their work, they inspected compilation time, JS engine execution time, garbage collection, and other statistics were used as features for ML models. The best model (SVM) had an above 95% true positive rate. Compared to them, we do not have access to detailed browser-based statistics and perform detection from network communication.

The network-level detection was studied by Sweden et al. [16]. They proposed Mining Detection and Prevention System (MDPS) based on the man-in-the-middle proxy. Their proposal used URL blocklists, detection of mining code, and VirusTotal API for further URL investigation. However, since they required proxy, their approach cannot be deployed onto 100 Gbps network lines.

Muñoz et al. [13] presented a machine learning method that is able to detect cryptocurrency miners using NetFlow/IPFIX network measurements. The presented method does not need to inspect packets' payload but still achieves similar accuracy as the DPI-based techniques. Captured traffic was analyzed and it was determined that miner flows are long duration and have a small number of transferred packets. Moreover, a server typically sends 20 times more data than a client. ML models were trained on several features based on these characteristics and the best model was Naive Bayes which achieved an average accuracy of 96.3%. However, only around 700 flows representing the miner traffic were used; therefore more thorough evaluation would be needed for actual deployment..

Approach discussed by Žádník et al. [12] combines passive detection and a secondary verification of false positives by active probing. A feature vector is created from flow data and an ML-based detector decides if flow looks like the miner's communication. Many false positives can occur at this stage due to the heuristic nature. Active probing is then used to verify if a server where the client is connecting is really part of a mining pool structure. Several packet traces of miners connecting to well-known mining pools were analyzed in order to select features for ML. During this analysis, it was discovered that miners' traffic has the following characteristics:

1. Mutual communication between a miner and a mining server often lasts for several hours
2. Packets are generally small, often in the range from 40 to 120 bytes
3. Most flows are observed with TCP ACK and PUSH flags set

4. The destination port is either a well-known port of a different service or not well-known but definitely lower than the source port

5. Flows are generally long-lasting, often exported before its end due to an active timeout

6. Communication is not disrupted, i.e., most flows do not contain the RST flag

However, it was determined that this design has performance issues, primarily because of the active probing. Therefore, it is not suitable for deployment in high-speed networks as well.

3 Theoretical Background

Ghimire et al. [17] described miners as "individuals who secure cryptocurrency's network". Mining is described as "the process of adding transaction records to cryptocurrency's public ledger of past transactions or Blockchain" [17]. This process involves solving a puzzle — a hard mathematical problem. Mining of a new block is rewarded by obtaining coins of the cryptocurrency (either new coins or transaction fees). Thus, a miner uses electrical power in exchange for a reward. Two types of mining were described by Tarman [18] – *solo* and *pooled*.

Tarman [18] described solo mining as "an attempt to confirm blocks of transactions on the Blockchain alone, as an individual miner". Miner will get block rewards and transaction fees all by himself —- large payments within longer intervals. Miners who are mining solo have to communicate directly with the cryptocurrency's network. Unfortunately, there is a high chance that solo mining will not produce any reward [18].

The other type of mining described in [18] is pooled mining. Miners connect to a mining pool and share resources in order to mine blocks more often. Rewards for mined blocks are split between miners, providing smaller but steady payments. This allows the miner to get a reward even if he is not the one who generated the new block.

Based on Bitcoin Developer Guides [19], the mathematical problem that miners must solve in the case of Bitcoin is to find a hash of a nonce (number used only once) and a block header. For this hash to be valid and accepted, it has to be below the target threshold, meaning that this hash has to start with a certain number of leading zeroes (based on the difficulty).

3.0.1 Mining protocols

Mining protocols are used by miners and mining pools for mutual communication. The most frequently used mining protocol is Stratum (Stratum V1), which described in [20]. It was introduced in 2012 and was firstly implemented on Bitcoin.cz Mining Pool (called Slush Pool nowadays) [20]. Stratum uses plain TCP sockets where packet payloads are JSON messages (based on JSON RPC 2.0[1]) with "\n" at the end. There are three message types — request, response, and notification.

Stratum network protocol specification [21] defines two formats of messages — request and response. Every RPC request has the following fields:

- ID — integer, string or null
- method — unicode string
- parameters — list of parameters

Moreover, requests can be of two types. The first one is part of the RPC standard, this type expects a response. The other type of request is called *notification* and does not expect a response. These two types

[1] `www.jsonrpc.org/specification`

can be distinguished by the value of id, notification has id set to null. Stratum response has the following fields:

- ID — same ID as in request (for pairing request-response)
- result — any JSON encoded result
- error — null or list (error code, error message)

4 Our Approach

We thoroughly analyzed cryptominers' traffic captured on the CESNET2 network to develop an efficient detection method. Based on this analysis and articles with previous detection attempts [12, 13], we made the following conclusions about cryptominers' traffic:

1. Traffic is long-lasting, typically lasts for several hours
2. Low amount of data are transferred (packers are usually small)
3. Longer intervals between packets in a flow than in other traffic
4. Packets have usually TCP PUSH flag set
5. Significant amount of traffic is still unencrypted

We decided to use multiple data sources to increase accuracy and design a robust detection method. It is based on the ensemble Machine Learning, where outputs of multiple models are combined together. We propose a heterogeneous ensemble classifier (Meta Classifier), which can detect flows, generated by a miner communicating with a mining pool. The high-level scheme of the Meta Classifier is shown in the figure 1.

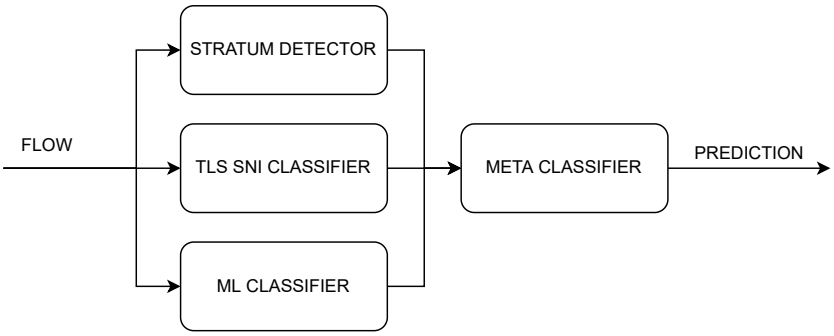


Figure 1: High-level scheme of the Meta Classifier

4.0.1 ML Classifier

ML Classifier uses an ML model to predict the probability of a flow representing communication between a miner and a mining pool. We experimented with several models, such as AdaBoost (with underlying Decision Tree), Decision Tree, Random Forest, Logistic Regression, and k-nearest neighbors. Random Forest achieved the best results based on accuracy and F1-score and was therefore selected. The output of this classifier is the probability, which is then processed by the Meta Classifier.

4.0.2 Stratum Detector

Our analysis showed that almost 80% of the captured cryptominers' traffic bears unencrypted data. Since ipfixprobe[2] supports extraction of the first several bytes, we decided to use these bytes as one of the data sources for possible detection. Specifically, Stratum Detector is designed to look for the Stratum mining protocol. We experimented with 100 bytes of the first packet of the flow sent from client to server and the first packet sent from server to client. A total of 200 bytes are used for detection, 100 bytes of the first packet from each direction.

4.0.3 TLS SNI Classifier

TLS SNI Classifier is designed to detect suspicious domain names in the Server Name Indication (SNI) extension of the TLS protocol. It detects two groups of keywords, which can be easily adjusted for the needs of different networks.

During our research of the current cryptocurrency situation, we noticed that domain names of the mining pools usually contain a short name of a cryptocurrency. Mining pools can have more mining servers, and a domain name can contain the name of the mined cryptocurrency. Some examples are **xmr**-eu1.nanopool.org and **eth**-us-west.flexpool.io. Therefore, the first group of keywords contains a list of short names of cryptocurrencies. We kept this list short in our experiments, but it can be easily extended. Moreover, TLS SNI Classifier transforms each entry of this list into four enhanced patterns to lower the number of false positives:

- -$NAME
- $NAME-
- .$NAME
- $NAME.

Our analysis also showed that many domain names of mining pools contain some words indicating the mining process. The second group therefore contains such keywords — *pool*, *mine*, *mining*. This list can be again easily extended. Some examples are ether**mine**.org or bee**pool**.com.

TLS SNI Classifier performs detection of two groups of keywords and, based on the detection results, assigns a TLS SNI score, which the Meta Classifier then further uses.

4.0.4 Meta Classifier

ML Classifier, Stratum Detector, and TLS SNI Classifier together form a group of support classifiers. As mentioned above, the Meta Classifier processes and combines outputs of these support classifiers (shown in the figure 2) to provide more robust predictions.

Since detection of a mining protocol is a strong indicator all by itself, it is processed as the first one. If Stratum was detected, a flow is marked as a miner right away, and other support classifiers are not invoked.

Then, features for ML are calculated, and the ML Classifier is used to get the probability of a flow being a miner. If the TLS SNI is not present, the probability is compared to the ML threshold to make the final prediction.

If a TLS SNI value is present, the TLS SNI score is obtained via the TLS SNI Classifier. This score is combined with the probability from the ML Classifier by the Dempster-Shafer Theory (DST) [22, 23]. The DST allows us to express a belief for each data source and then combine them via Dempster's Rule

[2] www.github.com/CESNET/ipfixprobe

of Combination. The resulting probability is then compared with the DST threshold to make the final prediction.

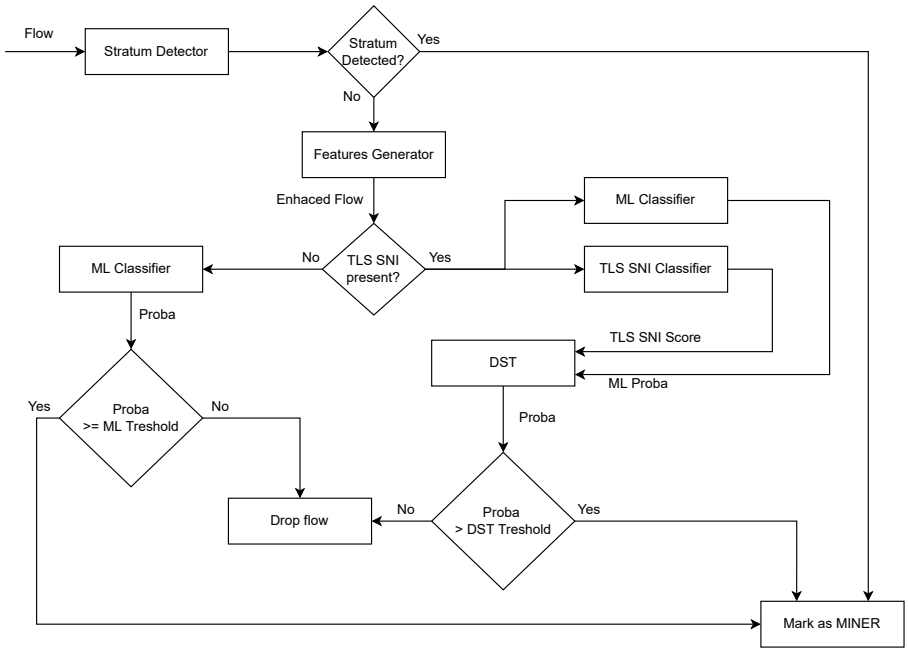


Figure 2: Detailed scheme of the Meta Classifier

5 Results

We implemented our detection method as a NEMEA module and evaluated this module on our datasets, created by capturing real-world traffic on the CESNET2 network. The capturing was performed between December 2021 and March 2022. The recognition of cryptominers was done by using the list of known mining pools, which were updated on a daily basis.

The created datasets contain 693 237 flows representing communication of cryptominers and 1 331 666 flows representing the other types of communication (Internet browsing, video streaming…). Our detector achieved the accuracy of $\mathbf{95.88\%}$, exact numbers are shown in the figure 3. The DST and ML thresholds were set to be stricter to lower the false-positive rate, which increased the number of false negatives resulting in smaller accuracy. However, the low false-positive rate is essential for the deployment onto large infrastructure, which processes hundreds of thousands of flows per second. Even a false positive rate of $\sim$1% can thus generate hundreds of false alarms per second, which is unusable for security personnel.

The speed of our detector was evaluated on the computer with AMD Ryzen 7 3700X 8-Core Processor, 3.59 GHz, and 16 GB RAM. The detector was able to process up to $\mathbf{41\,500}$ flows per second. Moreover, it was already deployed on the national network CESNET2 in the Czech Republic, and no performance issues have been discovered so far.

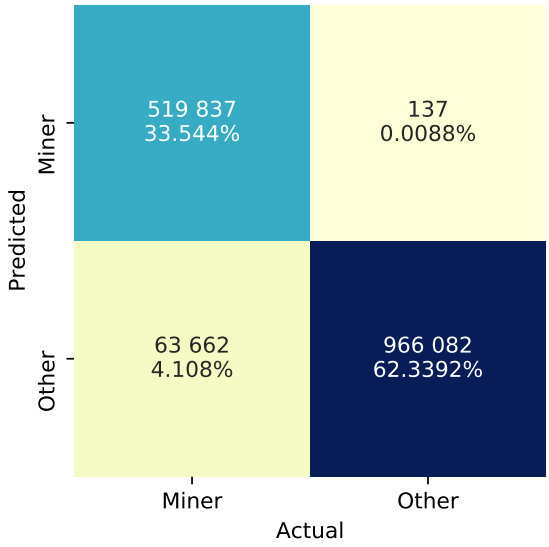


Figure 3: Confusion matrix of the Meta Classifier

6 Conclusion

Cryptomalware poses a severe security threat, and thus it is essential to be able to detect cryptominers. This paper proposed an efficient detection method deployable in high-speed networks with high accuracy. The proposed method was implemented as a NEMEA module and deployed on the national CESNET2 network, proving that it is efficient enough to operate on such a network. Moreover, the method produces a minimum of false positives, which is essential for preventing flooding security teams and network operators with false alerts.

Acknowledgment

This research was funded by the Ministry of Interior of the Czech Republic, grant No. VJ02010024: Flow-Based Encrypted Traffic Analysis, and by the Grant Agency of the CTU in Prague, grant No. SGS20/210/OHK3/3T/18.

References

1. BACIU, Paula. *Czech Prime Minister Accuses Pirate Party of Mining Bitcoin* [online]. 2018 [visited on 2022-04-06]. Available from: `https://bitcoinist.com/prime-minister-accuses-czech-pirate-party-of-mining-bitcoin-so-what/`.
2. CIMPANU, Catalin. *Malvertising Campaign Mines Cryptocurrency Right in Your Browser* [online]. 2017 [visited on 2022-04-06]. Available from: `https://www.malwarebytes.com/malvertising`.
3. HRUSKA, Joel. *Browser-Based Mining Malware Found on Pirate Bay, Other Sites* [online]. 2017 [visited on 2022-04-06]. Available from: `https://www.extremetech.com/internet/255971-browser-based-cryptocurrency-malware-appears-online-pirate-bay`.

4. VUIJSJE, Eliana. *Cryptocurrency Malvertising Campaign Hijacks Users' Browsers* [online]. [N.d.] [visited on 2022-04-06]. Available from: https://www.geoedge.com/cryptocurrency-malvertising-campaign-hijacks-users-browsers/.

5. PASTRANA, Sergio; SUAREZ-TANGIL, Guillermo. A First Look at the Crypto-Mining Malware Ecosystem: A Decade of Unrestricted Wealth. In: *Proceedings of the Internet Measurement Conference*. Amsterdam, Netherlands: Association for Computing Machinery, 2019, pp. 73–86. IMC '19. ISBN 9781450369480. Available from DOI: 10.1145/3355369.3355576.

6. HAYWARD, Andrew. *What Are Privacy Coins? Monero, Zcash, and Dash Explained* [online]. 2021 [visited on 2022-03-14]. Available from: https://decrypt.co/resources/what-are-privacy-coins-monero-zcash-and-dash-explained.

7. KHATRI, Yogita. *Crypto Mining Malware Has Netted Nearly 5% of All Monero, Says Research* [online]. 2019 [visited on 2022-03-14]. Available from: https://www.coindesk.com/markets/2019/01/10/crypto-mining-malware-has-netted-nearly-5-of-all-monero-says-research/.

8. MCAFFEE. *McAfee Labs Threats Report: December 2018* [online]. 2018 [visited on 2022-03-14]. Available from: https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-dec-2018.pdf.

9. NELSON, Jason. *This Monero Malware Is Targeting Enterprise Networks* [online]. 2021 [visited on 2022-03-14]. Available from: https://decrypt.co/87485/this-monero-malware-is-targeting-enterprise-networks.

10. SPEZZA, Gianluca. *North Korea and the Role of Science Diplomacy* [online]. 2022 [visited on 2022-03-14]. Available from: https://isdp.eu/content/uploads/2022/01/North-Korea-and-the-Role-of-Science-Diplomacy-24.01.2022.pdf.

11. GONZALEZ, Oscar. *Here's how much electricity it takes to mine Bitcoin and why people are worried* [online]. [N.d.] [visited on 2022-05-27]. Available from: https://www.cnet.com/personal-finance/crypto/heres-how-much-electricity-it-takes-to-mine-bitcoin-and-why-people-are-worried.

12. VESELÝ, Vladimír; ŽÁDNÍK, Martin. How to detect cryptocurrency miners? By traffic forensics! *Digital Investigation*. 2019, vol. 31, p. 100884. ISSN 1742-2876. Available from DOI: https://doi.org/10.1016/j.diin.2019.08.002.

13. MUÑOZ, Jordi Zayuelas i; SUÁREZ-VARELA, José; BARLET-ROS, Pere. Detecting cryptocurrency miners with NetFlow/IPFIX network measurements. In: *2019 IEEE International Symposium on Measurements Networking (M N)*. 2019, pp. 1–6. Available from DOI: 10.1109/IWMN.2019.8804995.

14. LIU, Jingqiang; ZHAO, Zihao; CUI, Xiang; WANG, Zhi; LIU, Qixu. A Novel Approach for Detecting Browser-Based Silent Miner. In: *2018 IEEE Third International Conference on Data Science in Cyberspace (DSC)*. 2018, pp. 490–497. Available from DOI: 10.1109/DSC.2018.00079.

15. KHARRAZ, Amin; MA, Zane; MURLEY, Paul; LEVER, Charles; MASON, Joshua; MILLER, Andrew; BORISOV, Nikita; ANTONAKAKIS, Manos; BAILEY, Michael. Outguard: Detecting In-Browser Covert Cryptocurrency Mining in the Wild. In: *The World Wide Web Conference*. San Francisco, CA, USA: Association for Computing Machinery, 2019, pp. 840–852. WWW '19. ISBN 9781450366748. Available from DOI: 10.1145/3308558.3313665.

16. SWEDAN, AbedAlqader; KHUFFASH, Ahmad N.; OTHMAN, Othman; AWAD, Ahmed. Detection and Prevention of Malicious Cryptocurrency Mining on Internet-Connected Devices. In: *Proceedings of the 2nd International Conference on Future Networks and Distributed Systems*. Amman, Jordan: Association for Computing Machinery, 2018. ICFNDS '18. ISBN 9781450364287. Available from DOI: 10.1145/3231053.3231076.

17. GHIMIRE, Suman; SELVARAJ, Henry. A Survey on Bitcoin Cryptocurrency and its Mining. In: *2018 26th International Conference on Systems Engineering (ICSEng)*. 2018, pp. 1–6. Available from DOI: 10.1109/ICSENG.2018.8638208.

18. TARMAN, Marko. *What is solo mining and how does it work?* [Online]. 2022 [visited on 2022-04-04]. Available from: https://www.nicehash.com/blog/post/what-is-solo-mining-and-how-it-works.

19. BITCOIN PROJECT. *Bitcoin Developer* [online]. [N.d.] [visited on 2022-04-04]. Available from: https://developer.bitcoin.org/devguide/mining.html.

20. SLUSHPOOL. *Stratum V1* [online] [visited on 2021-11-18]. Available from: https://braiins.com/stratum-v1.

21. STRATUM PLATFORM. *Stratum — network protocol specification: draft* [online]. 2011-12 [visited on 2022-03-13]. Available from: https://docs.google.com/document/d/17zHy1SUlhgtCMbypO8cHgpWH73V5iUQKk_0rWvMqSNs/edit.

22. DEMPSTER, A. P. Upper and Lower Probabilities Induced by a Multivalued Mapping. *The Annals of Mathematical Statistics*. 1967, vol. 38, no. 2, pp. 325–339. Available from DOI: 10.1214/aoms/1177698950.

23. SHAFER, Glenn. *A Mathematical Theory of Evidence*. Princeton University Press, 2021. ISBN 9780691214696. Available from DOI: doi:10.1515/9780691214696.

Vision of Active Learning Framework Approach to Network Traffic Analysis Research

Jaroslav Pešek **Dominik Soukup**
FIT CTU in Prague
Thákurova 9, Prague, Czech Republic

Tomáš Čejka
CESNET
Zikova 4, Prague, Czech Republic

pesekja8@fit.cvut.cz, soukudom@fit.cvut.cz cejkat@cesnet.cz

Abstract. Current research in the network security domain intensively uses machine learning (ML) and artificial intelligence to automate processes and reveal hidden patterns in data. These technologies, however, require lots of training datasets with ideally high quality. Additionally, network infrastructures continuously evolve and thus network traffic dynamically changes in time as well. There is an urgent need to adapt machine learning models, update datasets with the latest samples of annotated network traffic and retrain the models regularly to sustain feasible performance. Active Learning Framework (ALF) directly targets these demands and aims to provide a modular platform for scientific experiments and deployment in practice as well as to support research activities regarding quality of datasets. This paper particularly describes ALF software and proposes its possible use cases in research and practice domains.

Keywords. network traffic analysis, machine learning, active learning framework, ALF.

1 Introduction

Modern approaches to network traffic analysis and classification are being improved and the current research usually deploys machine learning technology to automate, speed up and improve the data processing. Especially in the networking area, there are large amount of data that must be processed and with the coming era of the traffic encryption it is very hard to do the analysis manually or to develop deterministic human-crafted algorithms.

Despite many benefits of ML technology, it requires large datasets with ground truth (also referred as annotated datasets) and the quality of ML models depends on the content and information richness of the datasets. It is impossible to train a good model using a bad or incomplete dataset. Also, as the network traffic changes over time, the existing datasets can become obsoleted and, as a result, models are getting less efficient without retraining using an updated dataset.

Traditional way of ML model development starts with gathering an annotated dataset (either publicly available or self-made) and follows by tuning parameters and hyperparameters of a ML algorithm. Usually, researchers train multiple algorithms in parallel to find the best one with optimal performance. Even though the process of training is or can be automated, every new version of the model is usually governed by some human operator or researcher, who controls the process. Generally we target to maintain the quality of ML models and avoid many false positive (or false negative) results.

This paper briefly introduces the topic of active learning that has been published by the researchers in the past. This introduction is needed to explain a new Active Learning Framework (ALF) that was inspired by the published related works. ALF has been designed and developed by the authors to address the application of the active learning principle on the online stream of IP flow data from the network traffic. The aim is to automatically generate new annotated datasets of enough size containing the flow data. Such datasets are exploited by ALF to train and continuously update ML models for network traffic classification and security threats detection. Finally, the paper lists some research areas related to ALF and active learning that will be studied in the near future.

2 Active Learning Principle

Active learning was presented by Shahraki et al. in [1] and the main idea is to repeatedly update ML model using a feedback. This principle is well known from the area of artificial intelligence and is commonly used, for example, as a reinforcement learning. For active learning, there is an entity called annotator, who provides ground truth to annotate unlabeled data, i.e., assign the correct label. This way, it is possible to construct a comprehensive labeled dataset that contains all known samples.

However, active learning was applied on a stream of data much earlier in 2013 by Žliobaitė et al. in [2]. The authors studied three strategies to handle data drift in the stream data processing by active learning. The aim was to cover data drifts when the changes are expected. Even though this work is not focused on network data, the ideas were inspiration to establish our research on application active learning principle on network security and network monitoring issues.

3 Active Learning Framework

In the modern monitoring system infrastructure, there are monitoring probes that observe network traffic in form of packets, process the packets and export aggregate information in form of IP flows. IP flows represent conversations that are transferred via network, thus they contain flow keys (IP addresses, ports and protocol) and statistical information (such as total number of transferred bytes and packets). There are also enhanced probes that are able to extract and compute more advanced features such as statistics of packet sequences within the connections, distributions of packet sizes and timing etc.

IP flow records are sent from the probes into the flow collector, which is usually a central machine. The collector is designed to store and process the data; usually it also performs network security operations such as intrusion detection and traffic classification. The information from flow collectors is useful for network security teams to detect network security incidents and to maintain situational awareness. Flow collector is an ideal component in the monitoring pipeline to deploy active learning technology.

Active Learning Framework (ALF) is a software implementation[1] based on active learning principle that provides necessary building blocks for developing and deploying ML models. It aims to offload standard tasks related to the capturing flow records and parsing them. The context of ALF is schematically visualized in Fig. 1. The input of ALF are network flows that are being processed by internally running ML model. When an instance of ALF is deployed, the user provides initial dataset (D_0) and annotator. D_0 is used to train an initial ML model.

[1] Active Learning Framework has been developed as a master thesis by Jaroslav Pešek, the thesis was submitted in 2022.

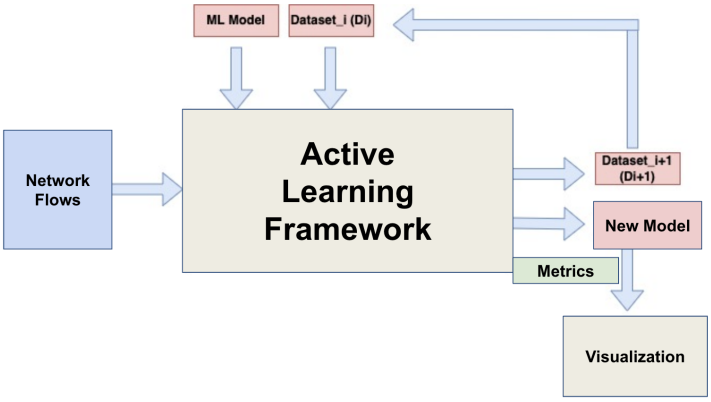


Figure 1: Overview of the Active Learning Framework principle

As the ALF instance is running, ML processes the incoming data and its results are compared with the ground truth provided by annotator. This process allows for updating dataset and thus retraining the model.

Naturally, having a perfect annotator seems like an ideal solution to the whole classification task. However, in practice the annotator is not available for all cases and cannot be deployed to process the whole traffic. For example, annotator based on information from end-points cannot be used in the network environment with devices out of control. Use of decryption to annotate the traffic is not feasible at large scale. Sometimes, the annotation process, i.e., retrieval the ground truth, is computational or time consuming and in such cases, we require a simpler (probably less efficient) machine learning model that can classify unknown traffic and approximate the results of the missing annotator. Also, the idea is to train the machine learning model using annotator (ALF) in some controled environment and, afterwards, deploy and update classification models in the production.

The building blocks of ALF are shown in Fig. 2. The framework contains Input Load, Preprocess, Annotate, Query Strategy, Evaluate, and Postprocess modules that are ready for use in an ALF instance. Additionally, each of the provided part of the pipeline can be overrided by a developer/researcher to adapt the behavior of the system.

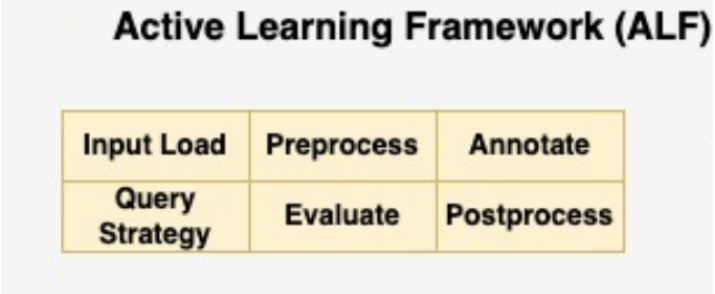


Figure 2: Modular architecture of ALF

4 Use Cases under Elaboration

ALF framework has already been experimentally deployed for pilot testing on a flow collector in CESNET2 network infrastructure[2]. This pilot deployment improved debugging and revealed new requirements and feature requests. Also, the deployed system under comprehensive monitoring (and visualizing the monitored statistics) explored several interesting observations and helped to define future research areas

The first ALF instance that was deployed is related to the DNS over HTTPS (DoH) research of our team. Having a list of known DoH resolvers (i.e., providers of DoH service) and also using an active scan, it is possible to check whether a connection is expected to be a DoH. This process of checking creates an annotator required for ALF. This way, there is a training of a machine learning model that is constantly evolving to detect DoH communication.

The second use case is targeted on Tor communication. Tor is a tool for annonymization in the internet and is useful for many situations to increase privacy. However, this technology is also being misused to cloak malicious activities. In this case, we also have an annotator that is too slow to be used as a single method. As a result, only a subset of traffic is checked by the annotator and the rest traffic is classified by the internally improved machine learning model.

4.1 Collection of datasets

ALF is useful in situations, when there is no available initial dataset. Simultaneously, ALF instance is able to run even in such configuration with empty dataset and annotator works as a generator of the initial dataset inside ALF. This way, it is possible to generate annotated datasets automatically from the real network traffic and to evolve machine learning models.

5 Research Areas

This section lists some ideas of the future research that can be accelerated by prototyping in ALF.

Continuous Evaluation of Datasets from Real Traffic ALF is ready to process online stream of flow records, update datasets and machine learning for theoretically infinite time. This way datasets might grow to impractically large sizes that are not feasible for training. Therefore, ALF is ready for the future research of evaluation strategies and methods that can be automatically applied on datasets to asses their quality.

Detection of obsoleted state of the trained machine learning model Another research potential lies in the detection of state when the machine learning model must be retrained due to data drift or distribution drift in the network traffic. This area is useful to avoid unnecessary computation for model training that is no better than the current one. On the other hand, timely detection of an obsoleted machine learning model can improve overall performance of the deployed security system containing the machine learning model.

Research in sample replacement strategies in datasets To limit the maximal size of a dataset, ALF is capable of replacement of the traffic samples from the dataset by the modern ones. However, research of optimal replacement strategies is essential. Removing essential information from the dataset can lead to disrupting performance of the model and decrease of the dataset quality.

[2]CESNET2 is a national research and education network infrastructure in the Czech Republic

Research in datasets optimization and dataset merging Replacement strategy is related to the optimization of the dataset. Sometimes, there are large datasets that contain too many redundant or similar samples. The larger the dataset, the more time consuming is the training of machine learning models. Therefore, it is beneficial to remove redundant information and decrease size of the dataset. Unfortunately, the removal must be performed carefully to avoid information loss that would prevent training successful machine learning models.

5.1 Conclusion

Active learning principle is known to the academic community for many years, however, its application on network traffic analysis appeared quite recently. Network monitoring and traffic analysis (e.g., for network security purposes) are crucial areas that can benefit from machine learning technology. Moreover, active learning approach allows for continuous updates of both datasets and machine learning models.

Therefore, we have developed a new Active Learning Framework (ALF) to address the needs related to machine learning application in the network traffic analysis domain. ALF is designed to process online stream of extended flow data (which are commonly used in practice to monitor large network infrastructures), and to evolve datasets and machine learning models using an annotator and real online traffic. The reason for ALF use over the annotator itself lies in the performance and computational / time complexity of the annotating (i.e., labeling process). Therefore, ALF is meant to train machine learning models using available information by the annotator, so that such models can be deployed in the other network environments where no annotator can be used.

The paper also lists several research areas that are related to active learning and more specifically to ALF. We believe, the current version of ALF will help to accelerate the research activities in areas like quality of dataset assessment in practice, research of replacement strategies and dataset optimization and so on. The introduced research areas will be studied in the near future as a part of the dissertation thesis by Dominik Soukup.

Acknowledgment

This work was supported by the Ministry of Interior of the Czech Republic (Flow-Based Encrypted Traffic Analysis) under grant number VJ02010024, and also by the Grant Agency of the CTU in Prague, grant No. SGS20/210/OHK3/3T/18 funded by the Ministry of Education Youth and Sports of the Czech Republic.

References

[1] A. Shahraki, M. Abbasi, A. Taherkordi, and A. D. Jurcut, "Active learning for network traffic classification: A technical study," *IEEE Transactions on Cognitive Communications and Networking*, vol. 8, no. 1, pp. 422–439, 2022.

[2] I. Žliobaitė, A. Bifet, B. Pfahringer, and G. Holmes, "Active learning with drifting streaming data," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 1, pp. 27–39, 2014.

Dataset Quality Assessment in Autonomous Networks with Permutation Testing

Katarzyna Wasielewska[1], Dominik Soukup[2], Tomáš Čejka[3], and José Camacho[4]

[1]Dep. of Signal Theory, Telematics and Comm., CITIC University of Granada Granada, Spain
[2]FIT CTU in Prague, Thakurova 9, Prague 6
[3]CESNET a.l.e., Zikova 4 Prague, Czech Republic

k.wasielewska@ugr.es, soukudom@fit.cvut.cz, cejkat@cesnet.cz, josecamacho@ugr.es

Keywords. dataset quality assessment, network dataset, permutation testing, classification

Abstract

The development of autonomous or self-driving networks is one of the main challenges faced by the telecommunication industry. Future networks are expected to realise a number of tasks, including network optimization and failure recovery, with minimal human supervision. In this context, the network community (manufacturers, operators, researchers, etc.) is looking at Machine Learning (ML) methods with high expectations. However, ML models can only be as good as the datasets they are trained on, which means that autonomous networks also require a sound autonomous procedure to assess, and if possible improve, dataset quality. Although the application of ML techniques in communication networks is ample in the literature, analyzing the quality of the network datasets seems an ignored problem. This paper presents work in progress on the application of permutation testing to assess the quality of network datasets. We illustrate our approach on a number of simple synthetic datasets with pre-established quality and then we demonstrate its application in a publicly available network datasets.

2 Introduction

This paper introduces a novel approach for the application of permutation testing to assess the quality of network datasets. Preliminary results show a promising performance of the proposed method in both synthetic and real data and for tasks like anomaly detection and classification. The rest of the paper is as follows: Section II introduces the proposed permutation approach for dataset quality assessment, Section III presents the results of the experiments and Section IV draws the conclusions.

Autonomous networks need efficient, fast and effective algorithms, which improve their performance, optimization, and security [1]. With machine learning (ML), autonomous networks can radically transform networks to become intelligent, self-servicing, and problem-resistant. However, ML algorithms use historical data as input, and their performance is upper bounded by the quality of the dataset. Low-quality datasets can lead to bad performance of ML models deployed in a production environment, or to suspicious results based on irrelevant features in data. Unfortunately, many research papers mention the lack of high-quality network datasets [2, 3].

What does the quality of the dataset mean? In general, a dataset is of high quality when it meets the requirements for its intended use. If we have high requirements for ML algorithms, we must provide them with high-quality datasets. How to assess the quality of the dataset? Unfortunately, this problem has been overlooked in literature, even though the assessment of the dataset quality is the key in big data analysis and modeling [4].

Researchers and practitioners are doing their best to improve the performance of ML models, but they focus more on the quality of the model (trying to get the best performance possible) than on the quality of the dataset. This is because improving the dataset quality is a much more challenging task and takes more time than optimizing the model's hyperparameters. Dataset cleaning typically involves dealing with data inconsistencies, duplicates, outliers or missing values, but also class imbalance, class overlapping, noise in data, incorrect labeling, and even a dataset size problem. However, the overall characteristics of the dataset, such as accuracy, precision, variety, uniqueness, separability or completeness, remain difficult to assess [5], while it is these statistical dimensions that describe the quality of the data. In addition to errors and problems, the dataset may contain irrelevant, insufficient, unreliable or even biased information that prevents ML algorithms from learning correct patterns. Therefore, assessing the quality of a dataset is a real challenge, especially in the area of autonomous networks that are expected to operate with minimal human intervention.

This paper introduces a novel approach for the application of permutation testing to assess the quality of network datasets. Our results show a promising performance of the proposed method in both synthetic and real data and for tasks like anomaly detection and classification.

3 Proposed Method

Permutation testing [6] is an uncomplicated and intuitive form of non-parametric inferential statistics. Thus, it is used for the computation of probabilities and p-values that are specifically tailored to the dataset at hand, without any assumption about the null distribution. This method can be used to assess the extent to which the model quality measures (e.g., classification accuracy errors, correlation coefficients, etc.) are the result of chance.

The permutation testing is a resampling approach. Take for instance a dataset (X, y), where X is a data block (i.e., the observations) and y is a set of labels. The dataset has N samples, i.e., pairs of X and y. We can test whether the correlation between these variables is significantly high by creating a large number (say one hundred) of permuted instances of X where the order of the observations in (only) the first variable is randomly shuffled. This would produce one hundred new correlation coefficients, one per permuted dataset, representing a null distribution of the correlation coefficients we can expect in a dataset like this one. If our true correlation is above, say, 99 of the 100 resamples, then we can say it is statistically significantly high with a p-value ≤ 0.01.

Let M is the value of the performance metric calculated from the original dataset and M^* is the value of the performance metric computed after permutation. The p-value can be defined as follows [7]:

$$\text{p-value} = \frac{\text{No. of } (M^* \geq M) + 1}{\text{Total no. of } M^* + 1} \tag{1}$$

We use the p-value as a measure of the statistical significance of the results. The smaller p-value, the higher quality of the dataset. We set the significance level to 0.01. A p-value ≤ 0.01 means that we have indications that our supposition that the results were obtained by chance was unfounded. Conversely, the p-value > 0.01 means that we can suppose that there is no strong relationship between $\mathbf{X}$ and original labels, and classification may not be correct. However, it is worth noting that p-value does not measure the probability that the dataset is random. The p-value is a statement about the dataset against a hypothetical assumption (as defined above). In other words, the p-value is the probability that the statistic M^* is greater than or equal to the M when the above hypothesis is true.

Using the pool of classifiers, we obtain a pool of performance measures for a given dataset. In order to check if the performance is significant, we apply the permutation test to different percentages of observations to evaluate the performance loss (if any) whenever some data is permuted. This allows us to assess the significance not only of the entire dataset, but also of parts of it. The percentage of labels depends on the size of the dataset. Big datasets require a lower percentage because we need to check the sensitivity of the algorithm to a permutation of a small number of labels. We permuted a maximum of 50% of the labels.

Dataset quality assessment is based on the interpretation of performance results from all ML models. If all models are performing better on the permuted labels than on the original labels, then the quality of the original dataset is *bad*. In such case, practically the entire p-value table will be populated with values greater than the significance level. On the other hand, for originally *good* datasets, at least one of the models will get worse performance metric with the increasing percentage of permuted labels. Ideally, we will have statistical significance for all percentages in at least one model.

The results depend on the capabilities of the selected ML algorithms: from simple (linear) to complex (non-linear) classifiers. It is possible that a good-quality dataset cannot be classified using even the most modern ML methods. However, in such case, the dataset has little practical use.

4 Conclusion

This paper presents work in progress to develop a methodology to evaluate the quality of a dataset based on a pool of classifiers and permutation testing.

The proposed solution shows clear advantages. With the help of permutation testing, we can evaluate the correctness, consistency, and separability of the labeling. Testing and evalution of our approach was done on synthetic and real datasets to demonstrate the value and explain outcomes of our method. The results of our extensive research with both real and simulated datasets show how useful this method can be in assessing the quality of the dataset. In future work we would like to investigate more metrics to asses dataset quality.

Paper origin

This paper has been accepted and presented at the conference AnNet 2022.

Acknowledgment

This work is partially funded by the European Union's Horizon 2020 research, innovation programme under the Marie Skłodowska-Curie grant agreement No 893146, by the Ministry of Interior of the Czech Republic (Flow-Based Encrypted Traffic Analysis) under grant number VJ02010024, and also by the Grant Agency of the CTU in Prague, grant No. SGS20/210/OHK3/3T/18 funded by the MEYS of the Czech Republic The authors would like to thank Szymon Wojciechowski for his support on the Weles tool.

References

[1] P. Kalmbach, J. Zerwas, P. Babarczi, A. Blenk, W. Kellerer, and S. Schmid, "Empowering self-driving networks," in *Proceedings of the afternoon workshop on self-driving networks*, 2018, pp. 8–14.

[2] I. Sarker, A. S. M. Kayes, S. Badsha, H. Alqahtani, P. Watters, and A. Ng, "Cybersecurity data science: an overview from machine learning perspective," *Journal of Big Data*, vol. 7, 07 2020.

[3] L. Caviglione, M. Choraś, I. Corona, A. Janicki, W. Mazurczyk, M. Pawlicki, and K. Wasielewska, "Tight arms race: Overview of current malware threats and trends in their detection," *IEEE Access*, vol. 9, pp. 5371–5396, 2021.

[4] F. A. Batarseh, L. Freeman, and C. Huang, "A survey on artificial intelligence assurance," *Journal of Big Data*, vol. 8, no. 1, 2021.

[5] V. Gudivada, A. Apon, and J. Ding, "Data quality considerations for big data and machine learning: Going beyond data cleaning and transformations," *International Journal on Advances in Software*, vol. 10, pp. 1–20, 07 2017.

[6] F. Pesarin and L. Salmaso, "A review and some new results on permutation testing for multivariate problems," *Statistics and Computing*, vol. 22, no. 2, pp. 639–646, 2012.

[7] M. J. Anderson, *Permutational Multivariate Analysis of Variance (PERMANOVA)*. John Wiley Sons, Ltd, 2017, pp. 1–15. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118445112.stat07841

LAURA: LoRa enAbled secURe off-grid messAging

Dries Martin[1], Jethro Pans[1], Md Masoom Rabbani[2], Jo Vliegen[2], and Nele Mentens[2,3]

[1] KU Leuven & University of Hasselt, Belgium
{dries.martin,jethro.pans}@student.uhasselt.be
[2] ES&S, imec-COSIC, ESAT, KU Leuven, Belgium
{mdmasoom.rabbani,jo.vliegen,nele.mentens}@kuleuven.be
[3] LIACS, Leiden University, The Netherlands
n.mentens@liacs.leidenuniv.nl

Abstract. Smartphones and the subsequent emergence of app-based culture have infiltrated every aspect of modern life. Without a doubt, technological improvements enrich our daily lives, but they also introduce the risk of a single point of failure. The adoption of a dedicated communication channel is essential since disconnecting LTE communication will render a large portion of society inaccessible. To address this issue a new communication channel is necessary. Further, the majority of people communicate via mobile phones, thus the new communication channel should be compatible with them. A customized internet could be set up using Wi-Fi, but it has a range limitation that would necessitate a large number of routers, making it a costly alternative. However, LoRa can solve this issue as it is a long-range and low power communication protocol that only requires a few devices. Further, to secure the communication, a lightweight cryptographic technique needs to be employed over LoRa communication.
To provide security over LoRa communication, we proposed "LAURA: LoRa enAbled secURe off-grid messAging". LAURA exploits Elliptic Curve Cryptography (ECC) and Speck to address this challenge. ECC performs as a key generation algorithm. Using the Diffie-Hellman protocol, the public keys can be exchanged and used to calculate one common symmetric key. Speck performs as an encryption algorithm that uses the secret key to encrypt and decrypt the data. We also validate LAURA through a Proof-of-concept (PoC) implementation. The results show that LAURA is lightweight and can be employed by any user.

Keywords: Elliptic Curve Cryptography, LoRa, Off-grid communication, Speck

1 Introduction

Wireless communication over LTE has become a standard in most parts of the civilized world. However, LTE towers are often not installed in less densely populated regions due to the low amount of users and thus the very small chance

to earn back the installation and operation costs. Thus, a low-cost, long-range and low power alternative is required to provide connectivity to those users [2]. In addition, LTE towers are also vulnerable to the powers of nature as they are bulky and installed at a fixed location. This makes them unreliable for situations such as rescue operations in case of a natural disaster. This necessitates the alternative to be small, mobile and optionally battery powered [15]. Furthermore, authoritarian governments can exploit LTE networks by blocking communication over LTE. Since, these authorities have the ability to shut down the grid. They can also intercept data passing through LTE towers, resulting in catastrophic consequences for communication participants. In addition, the location of the communication participants can be determined when communicating over LTE. As a result, the alternative should be off-grid, encrypted and the transmission power should be configurable [3].

Wi-Fi is a viable alternative because it is compatible with every smartphone. However, it is short-range, requiring a large number of routers to facilitate communication across long distances. This makes this alternative too expensive [13]. However, another interesting alternative is the communication over LoRa (long range) modulation. It is a low cost, low power and long-range protocol which is proved by the LoRaWAN distance world record of 766 km with a power usage of only 25 mW [7]. LoRa modulation transceivers are also small, mobile and can be battery powered because of their low power usage. This makes off-grid communication possible [11]. However, messages sent over LoRa modulation are not encrypted by default. Nevertheless, this can easily be solved with a custom security protocol so that messages cannot be intercepted. Finally, LoRa modulation transceivers can be connected with a Bluetooth transceiver to make it usable with a mobile phone. This makes communication over LoRa modulation the ideal alternative to the transmission of short messages over common communication channels like LTE.

To address these issues, we propose "LAURA: LoRa enAbled secURe off-grid messAging". The purpose of this research is to let devices securely communicate with each other over LoRa modulation.

Contribution

1. **Lightweight cryptographic implementation**

   We employ lightweight cryptography for LAURA that can also be used for the encryption of other communication protocols between low-end embedded devices.

2. **PoC implementation**

   We use ESP32s and LoRa HATs that are very affordable and easy to find. This makes it possible to set up a node by virtually anyone. These nodes are low-power and low-cost. Thus, the achieved PoC is secure and accessible to anyone.

2 Related work

A LoRa messaging system for isolated communities has already been developed [2]. It is proven that LoRa is a feasible alternative to traditional LTE networks in remote villages. However, security is not taken into account for the messaging system. A very reliable off-grid network can also be formed by using LoRa nodes for rescue operations [15]. However, security was not taken into account either. This research expands on existing work with a security protocol to create LAURA. Moreover, authors in [4], [5] proposed a lightweight, secure communication protocol for low-end embedded devices that operate in Routing Protocol for Low-Power and Lossy Networks (RPL). However, these proposed solutions are short-range and energy hungry. In contrast, LAURA is long-range and low-powered thus making it a suitable candidate to use for low-end embedded devices for secure communication.

3 Background

3.1 LoRa

LoRa is a spread spectrum modulation technique developed by Semtech [11] which works at the physical layer of the OSI model [6]. It is based on the chirp spread spectrum (CSS) technology thus a LoRa modulated transmission is robust against disturbances and can be received across great distances [14]. Additionally, the low power usage makes it ideal for battery powered applications.

3.2 Cryptographic primitives

Elliptic Curve Cryptography

Elliptic Curve Cryptography (ECC) is a form of Public-key cryptography and is based on ECDLP (Elliptic Curve Discrete Logarithmic Problem). This research uses the ”micro-ecc”-library for Arduino to execute the ECDH (Elliptic Curve Diffie-Hellman) key exchange. Using the Diffie-Hellman protocol, the transmitter sends its public key towards the receiver. The receiver answers by sending its public key. By multiplying the public key of the opposite party and its own private key, both parties can generate a common secret key. This common secret key can be used to encrypt and decrypt a message [9].

Speck

Speck is a lightweight block cipher algorithm which uses a symmetric key to encrypt and decrypt a message. It is created to maintain a decent level of security despite working with constrained devices. Table 1 shows an overview of the cipher block size in comparison with the key size. In this work, ECC generates a key with a size of 128 bits. Thus, the encrypted block cipher contains 128 bits [1].

If the message is larger than 128 bits, it is required to use a block cipher mode. In this research, Cipher Block Chaining (CBC) is used. The first cipher block starts with an Initialization Vector (IV) to encrypt the first 128 bits. The next cipher block uses the previous cipher block as IV [10].

Table 1: Cipher block sizes and corresponding key sizes [1]

block size	key sizes
32	64
48	72, 96
64	96, 128
96	96, 144
128	128, 192, 256

The aim for an algorithm such as Speck is to create an encryption which works well and is simple to code. Due to its simplicity, Speck is an outstanding alternative to consider when implementing a secure lightweight algorithm [1].

4 System model

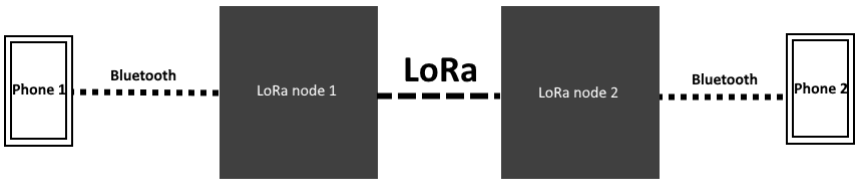


Fig. 1: The system model.

As shown in Fig. 1 the system model consists of 2 mobile phones which need to be able to communicate with each other securely. As most mobile phones have Bluetooth built-in, Bluetooth is chosen as the protocol to connect to the LoRa nodes. The LoRa nodes are responsible for the encryption, decryption and forwarding of the messages over LoRa.

5 Adversary model

The following adversarial actions are considered for LAURA:

1. Adversary can find the address of the transmitter and receiver of a message and uses their addresses to intercept a certain message.
2. Adversary is able to solve the Diffie-Hellman problem and is capable of computing the shared secret key.
3. When the same secret key is used to encrypt multiple messages, the adversary can use those encrypted messages to discover the used secret key.

6 Our Proposal: LAURA

In this section we explain the PoC implementation details for LAURA. The implementation consists of two nodes to which a smartphone is connected over Bluetooth. This allows the sending and receiving of messages over LAURA.

6.1 PoC components

For the Proof of Concept, we used the following component.

Waveshare SX1268 LoRa HAT

We employed Waveshare SX1268 LoRa HAT to connect the boards over LoRa. This module contains an SX1268 chip. This chip acts as a LoRa transceiver, controlled with UART. The module can be used to send packets in a frequency range from 410.125 MHz to 493.125 MHz. The range of this module is 5 km on a sunny day in an open area. It contains two selection jumpers. First, the UART selection jumpers contain three options: control through USB to UART, control through Raspberry Pi and access Raspberry Pi through USB to UART. Second, the LoRa mode selection jumpers contain four options determined by the placement of these two jumpers: Set both M0 and M1 to short to use transmission mode, set only M0 to short for configuration mode, set only M1 to short for Wake On Radio (WOR) mode and set both M0 and M1 to open for deep sleep mode [16].

6.2 PoC implementation

Each LoRa node in LAURA consists of an ESP32 which is connected to a Waveshare SX1268 LoRa HAT. As shown in Fig. 2 the GPIO pin 16 of the ESP32 is connected to the Raspberry Pi GPIO RX pin of the LoRa HAT and GPIO pin 17 of the ESP32 is connected to the Raspberry Pi GPIO TX pin of the LoRa HAT. The UART2 pins of the ESP32 are thus used for the UART communication between both modules. This is done to avoid the use of the same UART ports by multiple different services. The 5V and GND pins of the ESP32 are also connected to the 5V and GND pins of the LoRa HAT to power the LoRa HAT. This structure allows the communication over LoRa between two LoRa nodes.

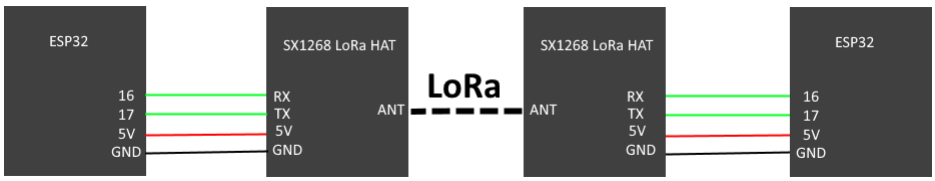


Fig. 2: A schematic of the Proof Of Concept.

The code consists of two different modes which are configuration mode and transceiver mode. All the properties of the LoRa HAT (e.g., frequency, spreading factor, bit rate, bandwidth) are configurable by changing the defined static variables. To apply the configuration, the code has to be re-uploaded to the ESP32 while the static defined variable ConfigActive is set to 1, $M0$ is shorted and $M1$ is opened. Afterwards, the code has to be reuploaded again with the static defined variable ConfigActive set to 0 and $M0$ and $M1$ shorted to enter transceiver mode. The UART selection jumpers are always in mode B which means that the Raspberry Pi headers are used for the UART communication.

In transceiver mode, the Bluetooth service on the ESP32 is started to which a mobile can be connected via the "Serial Bluetooth Terminal" app. Next to this, a private and public key are also created by the micro-ECC library. The initialization vector of the Speck library is also set to a specific 16-byte array.

6.3 Cryptographic details

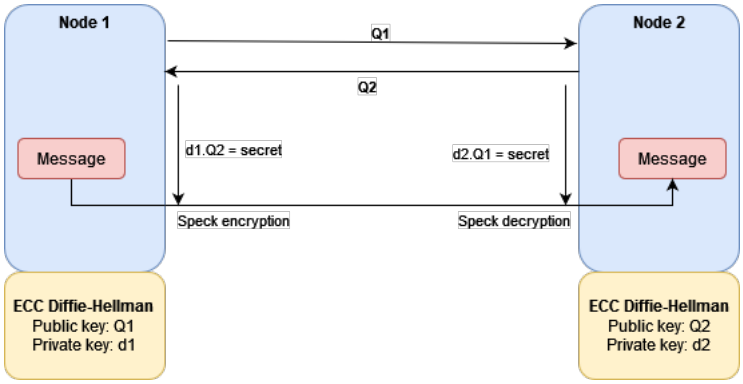


Fig. 3: A visualisation of the security protocol.

When a message is transmitted via the app over Bluetooth, it is first stored in an array on the ESP32. The security procedure is then launched. As shown in Fig 3, first, Node 1 sends its 64-byte public key (Q1) to Node 2. Then Node 2 replies with sending its own public key (Q2) to Node 1. When both nodes have received each others public key (Q1 and Q2), they both create a 32-byte shared secret key by multiplying their respective private key (d1 and d2) and the public key (Q1 and Q2) of the other node via a point multiplication (i.e., d1*Q2 = d2*Q1). This shared secret is then set as the 128-bit key which is used by Speck. Thereafter, the stored message is encrypted by Speck and sent by Node 1 to the Node 2 which then decrypts this message with Speck. Finally, the decrypted message is sent over Bluetooth to the app of Node 2.

6.4 Results

In this section, we present the results based on the PoC implementation of LAURA.

We connected two LoRa nodes using LAURA and made a small conversation between Node 1 and Node 2. To send and receive messages on the phone over Bluetooth, we used the Serial Bluetooth Terminal app (available on the Google Play Store). The app is able to show timestamps of the messages. The timings of a sent message can be calculated using those timestamps.

We summarize the time taken to send a message over LAURA between two nodes. We send the message "Hi Dries!" three times for our experiments. As shown in the Table 2, LAURA takes $\approx$ 2.4 *to* 2.5 seconds to send and receive messages between two nodes.

Table 2: Overview of measured times to send the message "Hi Dries!"

	Test 1	Test 2	Test 3
Transmitter sends key	1"183	1"137	1"103
Receiver sends key	0"273	0"266	0"269
Encryption	0"001	0"001	0"001
Send message	1"049	1"056	1"083
Decryption	0"007	0"004	0"034
Total	2"513	2"464	2"490

7 Security Analysis

In this section we discuss the security properties of LAURA based on the security assumptions in Section 5.

1. When an adversary intercepts a sent message, securing the message can mitigate this threat. The implementation of ECC and Speck on a lightweight device in LAURA is relatively secure. The ECC Diffie-Hellman algorithm generates a 128-bit secret key. With a 128-bit symmetric key the Speck algorithm encrypts the data in blocks of 128 bits using CBC. This step prevents an adversary to decrypt an intercepted message.
2. When an adversary solves the Diffie-Hellman problem, one mitigation could be to use a hashing algorithm to compress the public key at both the transmitter and receiver. ECC can compress the public key so it can still be used to generate the secret key, however the adversary cannot use it [12]. The implemented micro-ecc library contains a function to compress its generated keys.
3. A frequent renewal of the secret keys by repeating the ECC Diffie-Hellman algorithm reduces the threat of adversaries that are capable of obtaining a secret key. When a previous secret key is obtained, it will be ineffective in

the next information exchange that is encrypted with a freshly generated secret key.

8 Discussion

Choice of cryptographic primitives. The use of cryptographic algorithms on a small microcontroller could be questioned. However, the key generation with micro-ecc and the encryption with Speck have a small code size and low memory usage. The results show it only takes approximately one millisecond to encrypt a message of at least 128 bits. The decryption only takes $\approx$ 15 milliseconds. This proves a successful implementation of lightweight and secure algorithms to protect the data.

Flexibility. LoRa HAT modules were used as transceivers to transmit over LoRa modulation. However, these LoRa HAT modules can also be replaced by other transceivers to support other modulation techniques. Moreover, smartphones have been used for the PoC but these can also be replaced by other low-power devices which support Bluetooth or serial communication over USB. Finally, the ESP32s can be put in deep sleep mode to reduce the power consumption even further. The LoRa HAT module can then be connected to a pin which is used as wake up pin to still receive messages at any time. Another pin can then also be used to re-enable the Bluetooth server to transmit messages.

Limited message size. The transmission length of the LoRa HAT is limited to 240 bytes. This means that even longer messages can be sent in one transmission than when using SMS (Short Message Service) which is limited to 160 characters per transmission. It also has a buffer of 1000 bytes so a 1000-byte message can be split in multiple transmissions [16, 8]. However, the standard size of the TX and RX buffer in the used library "HardwareSerial" is 64 bytes which means that only 64 bytes can be sent and received at the same time. In addition, the first 16 bytes are lost because of the encryption and the last 2 bytes of each message are used for the carriage return and line feed command. This leaves 46 bytes for the actual message. Although, the RX and TX buffers can eventually be expanded based on the available RAM of the used ESP32s for each node.

9 Conclusions & Future Work

In this work we propose LAURA, a LoRa enabled secure off-grid messaging protocol. LAURA is capable of exchanging messages at a relatively fast rate on a low-power microcontroller. These achieved results have been tested with a small LoRa antenna and a low-power ESP board. Based on the preliminary results we can argue that LAURA is lightweight, low-power and secure.

As future work, we plan to explore the use of a stronger LoRa module, to create low power networks for remote regions that lack proper network coverage. A network of gateways could also be added in order to further extend the range.

10 Acknowledgement

This work is supported by the ESCALATE project, funded by FWO and SNSF (G0E0719N), and by Cybersecurity Initiative Flanders (VR20192203).

References

1. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The simon and speck lightweight block ciphers. In: Proceedings - Design Automation Conference DAC '15. pp. 1–6 (2015)
2. cardenas, A.M., Pinto, M.K.N., Pietrosemoli, E., Zennaro, M., Rainone, M., Manzoni, P.: A lora enabled sustainable messaging system for isolated communities. In: Proceedings of the 4th EAI International Conference on Smart Objects and Technologies for Social Good. p. 118–123 (2018). https://doi.org/10.1145/3284869.3284888, https://doi.org/10.1145/3284869.3284888
3. Chlosta, M., Rupprecht, D., Holz, T., Pöpper, C.: Lte security disabled: misconfiguration in commercial networks. In: WiSec '19: 12th ACM Conference on Security and Privacy in Wireless and Mobile Networks. pp. 261–266 (2019)
4. Conti, M., Kaliyar, P., Rabbani, M.M., Ranise, S.: Split: A secure and scalable rpl routing protocol for internet of things. In: 2018 14th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob). pp. 1–8 (2018). https://doi.org/10.1109/WiMOB.2018.8589115
5. Conti, M., Kaliyar, P., Rabbani, M.M., Ranise, S.: Attestation-enabled secure and scalable routing protocol for iot networks. Ad Hoc Networks **98**, 102054 (2020). https://doi.org/https://doi.org/10.1016/j.adhoc.2019.102054, https://www.sciencedirect.com/science/article/pii/S1570870519304470
6. Davoli, L., Pagliari, E., Ferrari, G.: Hybrid lora-ieee 802.11s opportunistic mesh networking for flexible uav swarming. Drones **5**, 26 (04 2021). https://doi.org/10.3390/drones5020026
7. LoRa Alliance: LoRaWAN® distance world record broken, twice. 766 km (476 miles) using 25mW transmission power (2017), [Online]. Available: https://lora-alliance.org/lorawan-news/lorawanr-distance-world-record-broken-twice-766-km-476-miles-using-25mw-transmission/. [Opened on May 14th, 2022]
8. Los Angeles Times: Why text messages are limited to 160 characters (2009), [Online]. Available: https://latimesblogs.latimes.com/technology/2009/05/invented-text-messaging.html. [Opened on May 14th, 2022]
9. Mehibel, N., Hamadouche, M.: A new approach of elliptic curve diffie-hellman key exchange. In: 2017 5th International Conference on Electrical Engineering - Boumerdes (ICEE-B). pp. 1–6. IEEE (2017)
10. Prajapati, P., Chaudhari, K.: Kbc: Multiple key generation using key block chaining. Procedia Computer Science **167**, 1960–1969 (2020)
11. Semtech: LoRa (PHY), [Online]. Available: https://www.semtech.com/lora/what-is-lora. [Opened on February 21th, 2022]
12. Svetlin Nakov: Elliptic Curve Cryptography (ECC), [Online]. Available: https://cryptobook.nakov.com/asymmetric-key-ciphers/elliptic-curve-cryptography-ecc#ecc-keys. [Opened on May 25th, 2022]

13. Tahir, M.N., Katz, M., Javed, Z.: Poster: Connected vehicles using short-range (wi-fi & ieee 802.11p) and long-range cellular networks (lte & 5g). In: 2021 IEEE 29th International Conference on Network Protocols (ICNP). pp. 1–2 (2021)
14. The Things Network: What are LoRa and LoRaWAN?, [Online]. Available: https://www.thethingsnetwork.org/docs/lorawan/what-is-lorawan/. [Opened on February 21th, 2022]
15. Vithayathil, A., Mohamed, A., Jacob, A., Raju, C.M., Jacob, J.: Lora based wireless network for disaster rescue operations. In: International Conference on Advanced Computer Control, ICACC. pp. 1–7 (2021)
16. Waveshare: SX1268 433M LoRa HAT, [Online]. Available: https://www.waveshare.com/wiki/SX1268_433M_LoRa_HAT. [Opened on May 13th, 2022]

Automated Annotation of Network Traffic with Data from Web Browser

Jan Kala[1] and Dominik Soukup[2]

[1]Brno University of Technology, Bozetechova 2/1, Brno
[2]FIT CTU in Prague, Thakurova 9, Prague 6

xkalaj01@stud.fit.vutbr.cz, soukudom@fit.cvut.cz

Abstract. Encrypted traffic classification requires Machine Learning (ML) algorithms and a large amount of data to learn patterns and classify network communication without decrypting it. For the learning stage of ML models, we need a reliable and trusted dataset that delivers the ground truth for the whole classification. However, building a dataset is a very complicated and time-consuming task that stops ML to be used in the production environment of target networks. The aim of this work is to address this topic for network flow annotation using web traffic data. This paper introduces to problematics of network IP flow monitoring, analysis and classification. This problem is demonstrated on HTTP and HTTPS protocols. Moreover, this work describes a technique of data collection from web browsers and their pairing with traffic flows to create a reliable annotated dataset automatically.

Keywords. Network traffic monitoring, network traffic analysis, web traffic annotation, HTTP, HTTPS, TLS, ipfixprobe, browser extension.

1 Introduction

Network traffic analysis become an important feature in many networks over the last decades to provide visibility and security. During the last years, this task is more and more complicated since most of the traffic is encrypted, the amount of devices is bigger and the heterogeneity of traffic is growing [1]. Due to these reasons, traditional methods based on Application Layer (L7) information do not work properly anymore [2]. ML is a very promising technique to address network traffic analysis challenges since it can find advanced patterns in collected data. Therefore, researchers are focused on enhancing components of network flow to provide more information for ML algorithms.

However, many ML models for network traffic classification are based on annotated datasets that are used for training. This brings a big challenge since it is very complicated to create annotated and trusted dataset. Annotating datasets is a very time consuming and often manual task. Moreover, this manual approach can add mislabels and verification of correct labels is almost impossible.

The aim of this work is to address the challenges of annotation HTTP and HTTPS traffic even in encrypted and tunnelled traffic. We propose a web browser plugin that gathers information directly from the user's web browser using the available Chromium [3] API (Application Programming Interface). Gathered data is paired with network flows to create respective annotations for training datasets for the ML model.

Section 2 describes available related work regarding this topic. Section 3 propose the complete architecture of our solution. Section 4 presents achieved results, use cases and tests. Finally, the conclusion and future work is provided in Section 5.

1.1 Problem description

Web traffic is represented by HTTP protocol that does not provide any encryption. To improve the security of HTTP, TLS protocol is applied to encrypt and protect L7 data. The combination of HTTP and TLS protocols is called Hypertext Transfer Protocol Secure (HTTPS). Using this approach no random observer on the network can no longer see the content of communication between end endpoints.

On the other hand, HTTPS complicates the analysis of L7 data in network communication and proper classification. This work addresses this challenge by intercepting communication from the web browser before the data is encrypted via TLS. Using available API, we can access the data of each request and see the response of those requests in real-time. Another challenge identified during this work is the pairing of web browser data and network IP flows. There is an absence of sufficient information from the network environment, that would help us with pairing the HTTP communications with the network flows. If possible we use Server Name Indication (SNI) field to uniquely pair network flow and web requests. In case SNI is missing, we use time-based pairing that is not unique in case the user has parallel communication at once.

In conclusion, this work brings an annotation system, that overcomes the problem of annotation encrypted HTTPS data by accessing the web-browser data. Further, we collect HTTP request/response data at the user's device and associate these data with correct network flows.

2 Related work

Based on our best knowledge, there is no other publicly existing tool, that would access data from a web browser for the purpose of network flow annotation. Moreover, we are not aware of a tool displaying more detailed network information about each request.

The most similar tool that helps to annotate network traffic is OSQuery[1]. It provides data about the whole operating system in a local database. This database can be easily accessed using SQL-like queries to gather info about the operation system. For example, we can query information from network flows to add information regarding the source application and process in an operating system. Our approach is more focused on web communication. We can add more relevant and detailed information about HTTP/S network communication from the web browser plugin. On the other hand, both tools can be used simultaneously and provide more details for annotation.

Aceto et al. [4] propose MIRAGE architecture as ground-truth create from mobile devices. This architecture describes how to gather pcap and strace logs files to create a reliable dataset. Similarly, as above, MIRAGE is focused on system-level services and their data. Web browser data is not considered in this solution.

Baste et al. [5] suggests annotation of network traffic based on passive probes (*iBox*) in the network. These probes investigate each packet using Deep Packet Inspection (DPI) to create required annotations. This approach works mainly on unencrypted traffic, nevertheless, we assume annotation of encrypted traffic.

Vekshin et al. [6] prepared a solid dataset of DoH traffic from live traffic using a block list and network proxy. The final dataset had 1,128,904 annotated IP flows with around 33,000 of them labelled as DoH. Using this dataset authors received an even high F1 score of 99,9%. Network traffic annotation using network proxies and block lists is a valid solution, however, it is applicable only for some protocols. IP blocklist can be used only if we can identify our traffic based on IP addresses. Network proxy can provide more detailed visibility but it bypasses the end-to-end encryption for investigation. The main focus of our approach is to avoid any active influence of network traffic communication. Our web browser plugin is activated on an end-user device to inspect HTTP/S communication before encryption. Due to this, the deployment is easy and the plugin can be controlled per user basis.

[1] https://osquery.io

3 Architecture

The proposed architecture is divided into two parts that differ in the environment that they run in. The architecture is depicted in Fig. 1. The first environment is the user's device, where the web traffic is collected and assigned to the appropriate network flow. The second part resides on the network and is a part of the monitoring probe, where it annotates captured flows using collected data. In this paper, we decided to use the existing implementation of CESNET's *IPFIXprobe* network probe as a base for our annotation module. The rest of this section contains the description of three main parts of the annotation system. This system can even run in standalone mode without the need for the network annotation module.

The first part of the annotation system is the Browser Extension, that's the main task is to redirect browser provided data to the intermediate processes for pairing. This could seem like an extensive task with many different browsers on the market, but actually, the majority of the browsers share the Chromium browser extension API created by Google. This enables us to come up with one browser extension, that could be used across a wide scale of browsers without the need for specific support for each one of them.

The second part of the annotation system also resides on the user's device and its main task is to monitor incoming/outgoing network traffic. The proposed Interface Monitor will seek messages that announce the start or the end of the connection used for HTTP communication transfer. This will provide us with information from the network environment, that will be used for pairing with HTTP communications.

The third and last part of the system is the Joiner & Dispatcher process. There will happen conjunction of the data from the browser and network interface. These data will be available not just in real-time but also retrospectively to widen the usage of the system.

3.1 Browser Extension

The source of the data used for final annotation. Annotation data comes from *webRequest* API from Google's browser extension API. In the mentioned *webRequest* API, we are provided with events that are invoked during different stages of the request's lifecycle. For example `onBeforeSendHeaders` event is triggered before the every extension is provided with request's headers to edit or `onCompleted` event is triggered when response to request is fully transferred. In each of these events, we can access a different subset of data from the browser, based on the event description. We will be the most interested in two specific events: `onSendHeaders` and `onResponseStarted`.

The `onSendHeaders` event is triggered before the request is sent out by the browser. In this stage, we are presented with the request ID, headers, origin of the request and other data provided according to `webRequest` API. We will collect all the available data and temporarily save them into the associative array inside the browser extension with the request ID as the key, which will be used later.

The `onResponseStarted` event is triggered when the first byte of the response is received. There we have certainty that the connection is established and not closed yet, which will help us during the pairing process. Preceding events can contain redirection of the request or authentication phases and later events can be triggered after the connection is closed already. Provided data contains request ID, which maps the response to the appropriate request, and that is the reason for saving the request data into the associative array earlier. In this stage, we can even find the IP address and the port of the server, that sent the response, which is a piece of beneficial information for the pairing. The collected bundle of data is then sent to the Joiner.

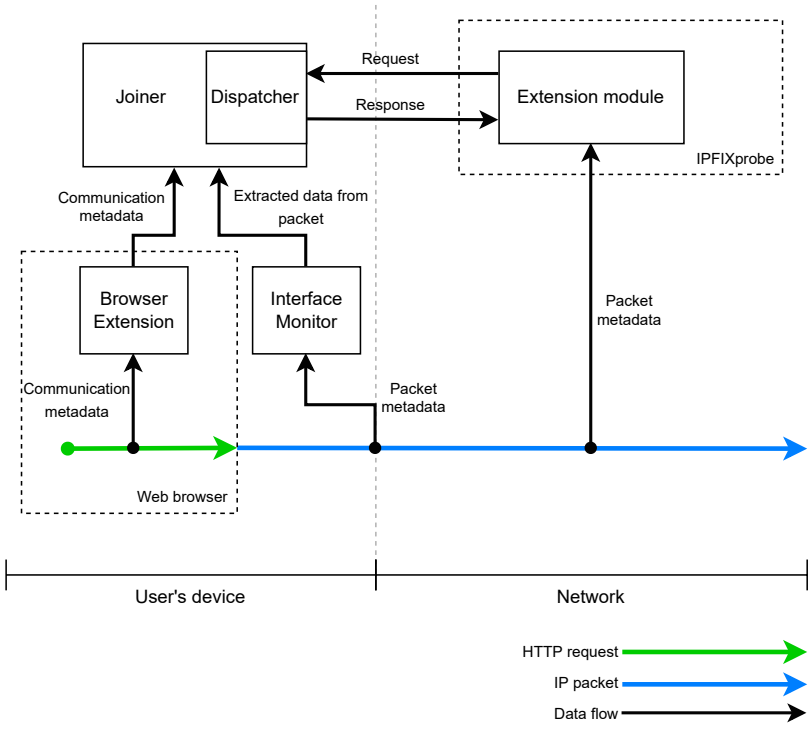


Figure 1: The architecture of the proposed annotation system.

3.2 Interface Monitor

Provider of the network information from the local interface and more specifically about these types of captured packets:

- TLS ClientHello
- HTTP GET or POST request
- TCP Fin packet
- TCP Rst packet

The choice to capture these packets comes from observation of the network traffic. When observing HTTP communication, we can see the start by capturing HTTP GET or POST request that contains all the data we need for pairing (including additional headers). In the case of TLS, we can observe the start of the connection by capturing the ClientHello packet, which contains an indicator SNI which will be also used for pairing. Both of the protocols share the TCP layer, so we can observe the end of the communication by capturing either TCP Fin or TCP Rst packets.

The reason to capture these packets is, that we need to monitor established TLS and HTTP connections to correctly assign appropriate HTTP data. This comes from the observation, that browsers can re-use connections for multiple HTTP requests. In the case of HTTP/1.1, we could easily determine when the connection was re-used, based on the presence of the `Connection: keep-alive` header in request headers [7], but the situation gets more complicated when dealing with HTTP/2 connections,

where not only requests can re-use already established connections, but multiple connections can be created to the same server at the same time [8]. Because of this, we will be only able to create 1:N pairing, where single HTTP communication will be assigned multiple connections, that were active at the time of response transfer.

3.3 Joiner & Dispatcher

These two components share the same process because they both access common storage, where Joiner will save results of the pairing process and Dispatcher will provide these data outside of the annotation system. Access to this storage must be mutually exclusive to ensure that Dispatcher provides complete data.

3.4 Joiner

Sources of the data to join are Browser Extension and Interface Monitor. Before we dive deep into the joining method, we need to first synchronise these two data streams. If we would provide data only retrospectively, no synchronisation would be needed as we could just sort the events based on timestamps and process them. But as the annotation system pledges to provide them with real-time, we need to assure that we won't process incoming messages in an incorrect order. We will use basic synchronisation based on timestamps with buffering of messages from Interface Monitor. When the Joiner receives a message from Interface Monitor, it will save it into the queue for later processing. When the Joiner receives a message from the Browser extension, it will first compare its timestamp with the front of the queue message, then processes the message with a lower timestamp and repeats this until the message from the Browser extension is processed.

After the events are synchronised, we get to the pairing. The first step of pairing is creating a local pool of active network connections, that mirrors the same pool that is being held inside of the browser. We will call this pool Active Connection Pool. The pool entries will be created based on announced packet observations from Interface Monitor and used for pairing with HTTP communications. The data inside the pool are stored in this form:

```
IP address and port of the remote server
  ⌞ Hostname
      ⌞ Local IP address and port
```

Entries into this pool are added when HTTP GET/POST or TLS ClientHello message arrives, in which case we add all the visualized structure (or assign to the keys, that already exist). If the Joiner receives a message from Interface Monitor, in which he announces observation of TCP Fin or TCP Rst message, the Joiner can react in two ways. In the case of the TCP Rst message, entry with the same IPs and ports as the TCP Rst message has will be deleted immediately. In the case of the TCP Fin message, we need to monitor two occurrences of these messages (we don't take into account the acknowledgements) and each one of the must come from the different side of the connection. If both sides send a TCP Fin message, we can safely remove the connection from the Active Connection Pool.

When a message from Browser Extension arrives, lookup inside if the pool will be made using the IP address and port of the remote server as a first key and Hostname as a second key. This lookup will give us a list of currently active connections, that could be used for HTTP message transfer. The result of this lookup is saved into the list for Dispatcher's access and

3.5 Dispatcher

The component is used to read the data from the annotation system and provide them to the outside processes. The idea is to provide all the collected data for one specific network flow identified so the request must contain 5 pieces of information for successful dispatch:

1. Source IP address
2. Source port
3. Destination IP address
4. Destination port
5. Activity timestamp

We can recognise the first 4 information as identification of network flow but added is the ”Activity timestamp”, which should mark any point of activity of the network flow. This measure is needed to recognise two flows, that could have the same parameters but happen at different times.

In case of the incoming request, the Dispatcher will look into the list of observed connections and compare them with the request. Each connection in this list has a link to the HTTP communications, that were transferred using this connection. This doesn't necessarily mean that the connection was the only one used for this transfer. We still need to take into account other possible connections. Linked HTTP data to the entry, that fits the requested parameters are then sent out as a response to the request.

4 Evaluation

The proposed annotation system has been implemented and went through two phases of tests. The first phase was the investigation if the presence of the system on the user's machine has any bigger impact on the time of the whole website to load. If there would be any, the usage of the proposed system wouldn't be possible, as the behaviour of the user on the machine with running annotation system could be different from his normal behaviour. After multiple measurements of the load time of the website that uses HTTP or HTTPS for its transfer with or without the annotation system running, we observed that the difference in speed was 0.3% slower for HTTP traffic and 1.4% slower for HTTPS traffic when annotation system was running. If we take into account the impacts of the network environment for this measuring, we can say that usage of the annotation system has no significant impact on the load times of the web pages.

The second test was the measurement of the usability of the proposed annotation. This measure is highly individual, meaning that we cannot accurately measure the total impact on all the traffic that could be observed globally as this annotation depends on the web traffic generated by the individual user's device. But we can create categories of the web traffic and measure how many network flows can be annotated using the proposed annotation system.

Individual categories have been picked according to the sociodemographic research of the STEM/-MARK company in June 2017 [9]. The bundle of websites for each category has been picked according to the most visited websites in the Czech Republic and worldwide. In Fig. 2, we can see the measuring results of one of the categories, which is ”Online search and news”. We can notice, that during the test we observed more than 13,000 HTTP requests, from which we could successfully pair roughly 70% of the requests with network flow information at the end of the test. This paired subset then equals close to 60% of total network flows captured by the network probe, which was monitoring traffic for this one specific device. From the results of this test, we can assume, that the usage rate is high enough to use this type of annotation for this type of web traffic.

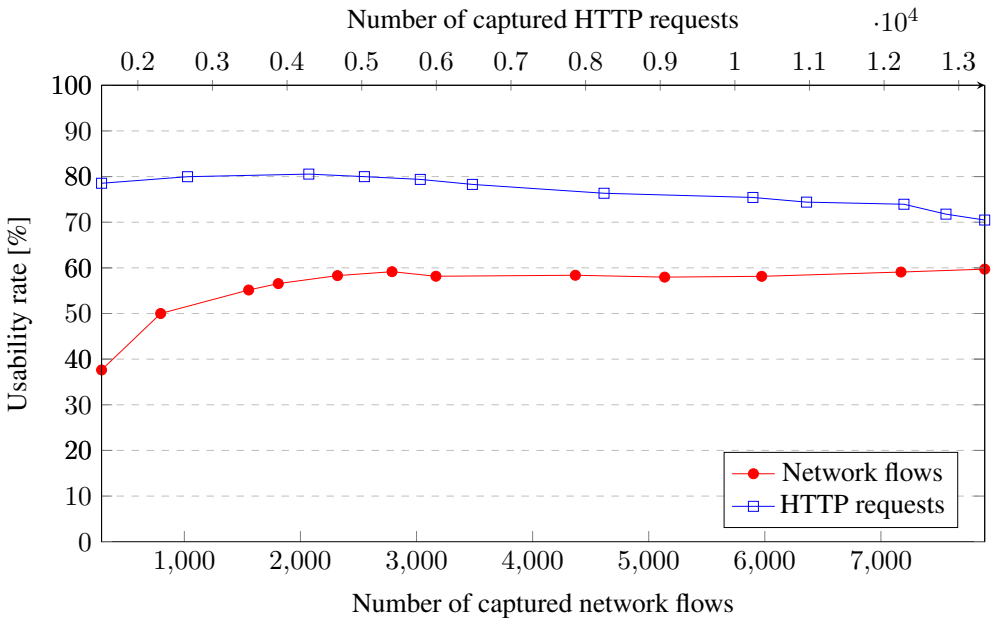


Figure 2: Results of the usability measure for category ”Online search and news”

5 Conclusion

Implemented system has been run and tested on the UNIX-based operation systems, so the main improvement is the support of the widest spread OS, which is Microsoft Windows. This would be a huge improvement in terms of compatibility.
During the testing, some drawbacks of the annotation system were found, that could improve the usability rate. One of the major improvements can be the added support for the QUIC protocol, as this has not been covered in the initial proposal of this work. a significant amount of requests were observed using this protocol for its transfer during the testing. Another improvement can be extending the rules for Interface Monitor, to improve the Active Connection Pool functionality and to more accurately monitor the ends of the connections.

The main goal of this paper was to propose an automated annotation system that would provide information about the part of the web traffic, that couldn't be assigned with any extra information because of the presence of encryption. As no other tool exists, that would be doing the same task, this proposal opens a new path for other users of the Chromium browser extension API for the purpose of network flow annotation.

Acknowledgment

This research was funded by the Ministry of Interior of the Czech Republic, grant No. VJ02010024: Flow-Based Encrypted Traffic Analysis and also by the Grant Agency of the CTU in Prague, grant No. SGS20/210/OHK3/3T/18 funded by the MEYS of the Czech Republic

References

[1] B. Anderson and D. McGrew, "Machine learning for encrypted malware traffic classification: Accounting for noisy labels and non-stationarity," ser. KDD '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3097983.3098163

[2] T. Cejka, V. Bartos, M. Svepes, Z. Rosa, and H. Kubatova, "Nemea: A framework for network traffic analysis," in *2016 12th International Conference on Network and Service Management (CNSM)*, 2016, pp. 195–201.

[3] A. Boodman, "Extensions status: On the runway, getting ready for take-off," 09 2009, [Online; Visited 20.05.2022]. [Online]. Available: https://blog.chromium.org/2009/09/extensions-status-on-runway-getting.html

[4] G. Aceto, D. Ciuonzo, A. Montieri, V. Persico, and A. Pescapé, "Mirage: Mobile-app traffic capture and ground-truth creation," in *2019 4th International Conference on Computing, Communications and Security (ICCCS)*, 2019, pp. 1–8.

[5] A. Baste, D. Chu, D. Joseph, and G. Porter, "Network management through packet annotation," 2005.

[6] D. Vekshin, K. Hynek, and T. Cejka, "Doh insight: Detecting dns over https by machine learning," in *Proceedings of the 15th International Conference on Availability, Reliability and Security*, ser. ARES '20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3407023.3409192

[7] R. Fielding and J. Reschke, "Hypertext transfer protocol (HTTP/1.1): Authentication," 2014.

[8] M. Belshe, R. Peon, and M. Thomson, "Hypertext transfer protocol version 2 (HTTP/2)," 2015.

[9] STEM/MARK and Gemius, "SPIR NetMonitor, Výzkum sociodemografie návštěvníků internetu v České republice," 06 2017. [Online]. Available: https://www.spir.cz/sites/default/files/verejne-vystupy/2017_06_TOTAL_PC.pdf

Enhancing Reactive Ad Hoc Routing Protocols with Trust

Yelena Trofimova, Pavel Tvrdík
Department of Computer Systems, Faculty of Information Technology, Czech Technical University in Prague
Thakurova 9, 160 00 Prague 6, Czech Republic

yelena.trofimova@fit.cvut.cz

Keywords. Ad hoc network, trust, security, reactive routing, route discovery, packet delivery ratio, route discovery delay.

Abstract

In wireless ad hoc networks, security and communication challenges are frequently addressed by deploying a trust mechanism. Many approaches for evaluating trust of ad hoc network nodes have been proposed, including ours that uses neural networks [1]. We proposed to use packet delivery ratios as input to the neural network. This article presents a new method, called TARA (Trust-Aware Reactive Ad Hoc routing), to incorporate node trusts into reactive ad hoc routing protocols. The novelty of the TARA method is that it does not require changes to the routing protocol itself. Instead, it influences the routing choice from outside by delaying the route request messages of untrusted nodes. The method's performance was evaluated on the use case of sensor nodes sending data to a sink node. The experiments showed that the method improves the packet delivery ratio in the network by about 70%. Performance analysis of the TARA method provided recommendations for its application in a particular ad hoc network.

Paper origin

This paper has been accepted and published in MDPI *Future Internet* journal vol. 14, Section *Internet of Things* on 15 January 2022 [2].

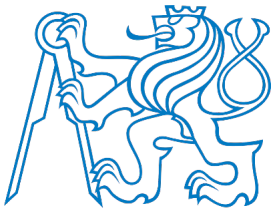
References

[1] Trofimova, Y.; Moucha, A.M.; Tvrdik, P. Application of neural networks for decision making and evaluation of trust in ad-hoc networks. In Proceedings of the 2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC), Valencia, Spain, 26–30 June 2017; pp. 371–377. [CrossRef]

[2] Trofimova, Y.; Tvrdik, P. Enhancing Reactive Ad Hoc Routing Protocols with Trust. In Future Internet, Vol. 14, no. 1: 28. MDPI, Basel, Switzerland, 15 January 2022. [CrossRef]

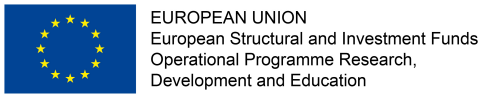
Sponsors

Czech Technical University in Prague



The conference has been sponsored by the CTU project SVK 55/22/F8.

Research Center for Informatics



ASICentrum

ASICentrum, established in 1992 in Prague is a design center of EM Microelectronic and a competence center of ETA, belonging to the Swatch Group. EM Microelectronic is one of the most innovative IC providers. It developed and manufactured the smallest and the lowest power consuming Bluetooth chip on the market, the top performing optical sensors for optical office as well as gaming mice and it was the first to release the award-winning world-first dual-frequency NFC + RAIN RFID emlecho.

CESNET

CESNET is an association of universities of the Czech Republic and the Czech Academy of Sciences. It operates and develops the national e-infrastructure for science, research and education which encompasses a computer network, computational grids, data storage and collaborative environment. It offers a rich set of services to connected organizations.

Daiteq

Digiteq automotive

EATON

Eaton's mission is to improve the quality of life and the environment through the use of power management technologies and services. We provide sustainable solutions that help our customers effectively manage electrical, hydraulic and mechanical power – more safely, more efficiently and more reliably.

METIO Software

Metio Software is a software development company that develops various kinds of software projects.

STMicroelectronics

STMicroelectronics is a world leader in providing the semiconductor solutions that make a positive contribution to people's lives, today and into the future. ST is a global semiconductor company with net revenues of US$ 8.35 billion in 2017. Offering one of the industry's broadest product portfolios, ST serves customers across the spectrum of electronics applications with innovative semiconductor solutions for Smart Driving and the Internet of Things. By getting more from technology to get more from life, ST stands for life.augmented.

Partners

IEEE Student Branch at Czech Technical University in Prague

IEEE Young Professionals

Computer (C) Society Chapter of the Czechoslovakia Section of IEEE