NISTIR 8365

A Collaborative Robot Work-Cell Testbed for Industrial Wireless Communications

Yongkang Liu
Mohamed Hany
Karl Montgomery
Richard Candell

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8365

National Institute of
Standards and Technology
U.S. Department of Commerce

NISTIR 8365

A Collaborative Robot Work-Cell Testbed for Industrial Wireless Communications

Yongkang Liu
Mohamed Hany
Karl Montgomery
Richard Candell
Intelligent Systems Division
Engineering Laboratory

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8365

May 2021

U.S. Department of Commerce
Gina M. Raimondo, Secretary

National Institute of Standards and Technology
James K. Olthoff, Performing the Non-Exclusive Functions and Duties of the Under Secretary of Commerce for Standards and Technology & Director, National Institute of Standards and Technology

National Institute of Standards and Technology Interagency or Internal Report 8365
Natl. Inst. Stand. Technol. Interag. Intern. Rep. 8365, 101 pages (May 2021)

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8365

Abstract

A new testbed design for assessing the performance of wireless techniques in support of industrial operations is proposed. For this, a collaborative robot work-cell manufacturing scenario, which serves as the emulated cyber-physical system model in the testbed was developed. Specifically, the information exchanged between networked work-cell components, e.g., programmable logic controllers, robotic arms, and machining tools, was carried out via messages of industrial communication protocols; network connections employed were both Ethernet-based wired links and alternative wireless interfaces. A measurement framework was introduced to collect various system events from both work-cell operations and network traffic; a distributed clock synchronization scheme was developed to ensure the temporal consistency of measurement records at distributed sources. To facilitate offline performance analysis and data exploration, a new data pipeline was adopted that uses the graph database for testbed data management. The testbed has been in use in evaluating industrial wireless performance by the NIST team and industrial partners.

Keywords

Cyber-physical system (CPS); smart manufacturing; testbed; wireless communications.

Table of Contents

List of Tables

List of Figures

1. Introduction

Wireless applications are among key enabling technologies in smart manufacturing innovations that promise flexible deployment capability and mobile support for data transmissions on the factory floor [1]. Emerging wireless techniques, such as recent wireless local area networks (WLAN), also known as Wi-Fi 6, and the fifth-generation (5G) cellular networks, are anticipating to land their killer applications in industrial communication scenarios for which they are planning technical deployment routes. As industrial plants are complex cyber-physical systems (CPS), general wireless communication metrics cannot fully meet the need of comprehensively evaluating wireless impact onto manufacturing activities which are closely related to production efficiency and operation safety. Among the first attempts, standard development organizations (SDO) have proposed multiple classification methods of identifying different industrial communication use cases and their specific performance requirements, e.g., latency and reliability metrics [2]. Measurement sciences that assess wireless techniques in support of manufacturing practices are essential to the success of industrial wireless communications [3].

The National Institute of Standards and Technology (NIST) has been sponsoring industrial wireless performance studies in its long-term research program entitled "Trustworthy Systems, Components, and Data for Smart Manufacturing" [4]. As a key supporting unit in this program, the industrial wireless system (IWS) lab at NIST anticipates low-latency and high-reliability wireless communications would play an important role in manufacturing areas. Wireless links can support real-time process monitoring and automation control, which involve various industrial devices, such as programmable logic controllers (PLCs), field sensors and actuators, and machine tools. In addition, it is anticipated that there will be a mixed network topology with wired and wireless connections that carry standardized and proprietary industrial communication messages. These new features raise challenging issues in the design and implementation of industrial wireless communications. However, limited system models and testing platforms are readily available for engineers to evaluate emerging industrial wireless solutions in real environments.

This report presents our recent measurement activities at NIST in collaboration with industrial partners. A testbed was designed by replicating various data flows in a managed production environment for the purpose of evaluating the impact of industrial communication systems on the factory automation processes. A collaborative robot work-cell was emulated as the CPS model whose service requirements are representative of a typical industrial application. The testbed characterizes deterministic and reliable communication needs between work-cell components in a machine tending application. Measurement methods were utilized to ensure the compliance and consistency of the observed data from distributed monitoring points regarding production states and network traffic. A time synchronization design was used to keep the collected data records aligned in time. Experiments were performed with the Ethernet-based wired configuration and the wireless alternative using WLAN devices. A data workflow based on the graph database (GDB) technique was developed to streamline the processing and analysis of heterogeneous testbed

data in exploring CPS performance factors. Since the testbed design has been partly introduced in our earlier publications [5–7], this report aims to serve as a comprehensive review of the testbed architecture, component module details, related measurement activities, and lessons learned from these implementations. The report's goal is to assist industrial engineers, wireless designers, and CPS researchers when they are designing similar evaluation platforms and dealing with typical industrial wireless system models. Note that cybersecurity discussions of industrial wireless communications are not in the scope of this report. Interested readers can refer to the peer project in the NIST program [8].

The remainder of this report is organized as follows. Section 2 briefly discusses the related work of the testbed project. Section 3 provides an overview of the testbed architecture. Hereafter, design topics are elaborated in the following sections. Production-related modules including the Supervisor, computer numerical control (CNC), and robots are introduced in Section 4; the network design is addressed in Section 5 where both wired and wireless networking issues are discussed; the measurement framework is described in Section 6. The testbed data are stored and managed in a non-relational (NoSQL) database organized by a graph whose model and processing flow are defined in Section 7. We will explore the latency performance of the selected communication links in Section 8. The presented experimental configurations and preliminary analysis results are used to showcase the capability and potentials of this testbed. In the end, Section 9 will summarize the main takeaways and discuss the future work of the testbed.

2. Related Work

Trending efforts on developing evaluation methods and tools for emerging industrial wireless communications can be classified into three major areas: theoretical analysis, simulations, and testbed platforms. Based on field measurements of industrial radio channels and traffic traces, statistical models have been developed to characterize channel propagation features, interference variations, and usage patterns. For example, NIST has performed a series of channel measurement campaigns in different topographical sites to understand the industrial channel characteristics [9]. Considering the complexity of modern manufacturing processes, both network performance and manufacturing productivity need to be captured thoroughly, especially when transmissions are closely associated with process precision and plant safety. Therefore, performance evaluation is still mainly conducted through computer simulations and in testbed platforms with real hardware configurations.

The computer simulation approach has been widely used in studying the performance of networked systems, which can flexibly adjust the studied network scale while tuning configuration parameters in various regular/extreme test cases. Recently, the co-simulation of physical processes and associated network activities has gained the attentions of CPS researchers. For example, a co-simulator design, that combines the production pipeline and wireless network operations into an integrated discrete-time framework, was proposed in [10, 11]. However, the trustworthiness of simulation results to estimate the real system performance largely relies on the model accuracy and the computational complexity which

often point to the opposite directions given the simulator's limits. Therefore, it is critical to design a testbed platform for both validating theoretical models and demonstrating the performance of industrial wireless solutions in realistic operational and environmental configurations, which is a convincing approach to encourage the adoption of emerging techniques.

Table 1. Feature comparison with existing industrial wireless evaluation platforms (reprinted from [7])

	Application Domain	Use Case	System Setup			Data Collection		Evaluation	
			Physical System	Wireless Network	RF Factor	Physical	Network	Data Process	CPS Metrics
This work	Factory Automation	Robotic Work-cell	HW (PLC, robots, 10 - 125 Hz updates)	HW (WLAN IEEE 802.11b/g/n as presented)	Indoor (Lab), managed WLAN interference radios in 2.4 GHz	Device logging, remote access	Ethernet /WLAN sniffer	Pipeline, scripts, graph database	Control delay, system failure
Aminian 2013 [12]	Process Automation	Dual-Tank level control as presented	SW (by Simulink)	SW (Wireless Mesh with IEEE 802.15.4), HIL (tentative)	SW	Simulator logging	Simulator logging	Visual inspection, scripts	Control, I/O
Jecan 2018 [13]	Process Automation	Industrial Wireless Network	No	HW (WirelessHART plus ISA100.11a)	Indoor (Lab)	No	Network manager	scripts	No
Ding 2015 [14]	Process Automation	Wireless Sensor & Actuation	HW (valve control, 1 Hz updates)	HW (WirelessHART, ISA100.11a)	Indoor (factory), managed Zigbee and WLAN in 2.4 GHz	No	Packet sniffer	scripts	No
Liu, Q 2018 [15]	Process Automation	Wireless medical telemetry	HW (operation room surgical monitoring)	HW (WLAN IEEE 802.11b/g/n)	Indoor (multiple rooms separated by walls)	Health database (MySQL)	WLAN signal analyzer, JPerf	scripts	Database requests
Fink 2013[16]	Robotics	Robot teams	HW (mobile AGVs)	HW (IEEE 802.15.4)	Indoor (two office/lab buildings)	Location reported at 5 Hz	Signal strength reported at 5 Hz	scripts	No
Liang 2019 [17]	Factory Automation	AGV, safety	HW (mobile AGVs)	HW (WIA-FA)	Indoor (industrial sites)	Device logging, field measures	Signal analyzer, spectrum analyzer, network analyzer	scripts	Motion distance error
Candell 2015 [18]	Process Automation	Chemical process control	HIL (process simulator, PLC, sensors)	HW (IEEE 802.15.4-TDMA)	Indoor (Lab)	Simulator logging	No	scripts	Process control
Liu, Y. 2016 [19]	Process Automation	Chemical process control	SW (process simulator)	SW (IEEE 802.15.4-TDMA)	SW (PER-SNR curves)	Simulator logging	Simulator logging	scripts	Process control safety

Notes: HW: hardware testbed; SW: software simulation; HIL: hardware-in-the-loop simulation.
"scripts" stands for the data processing that uses specific code/program to treat experiment data in the performance evaluation.

Various industrial scenarios have been considered to address the featured operational and environmental factors in the production site. Table 1 summarizes recent efforts on building testbeds for industrial wireless communications. Details of the comparison can be found in our earlier publication [7]. A work-cell, serving as the atomic production unit in modern manufacturing factories, has been identified as a promising industrial communication scene to observe the impact of communications on factory operations. Besides, new techniques can be introduced into measurements to improve the accuracy of data capture and facilitate data processing and analysis. Among them, time synchronization protocols and database techniques are examples that are playing a very important role. The former,

e.g., Network Time Protocol (NTP)) and Precision Time Protocol (PTP), equips measurement devices with the tuned clocks synchronized to the same time server so that the captured local events can be realigned in a single timeline for the global view of various events and their interactions with each other [20]. The latter, e.g., GDBs, enables advanced data management schemes to organize heterogeneous measurement records and facilitate data exploration to get insights of CPS operations [7]. Some of the design and measurement work has been presented in our earlier publications [5–7]. This report serves as a reference document to provide more details in constructing the testbed and enabling the data workflow for further evaluation efforts.

3. Overview of the Testbed Architecture

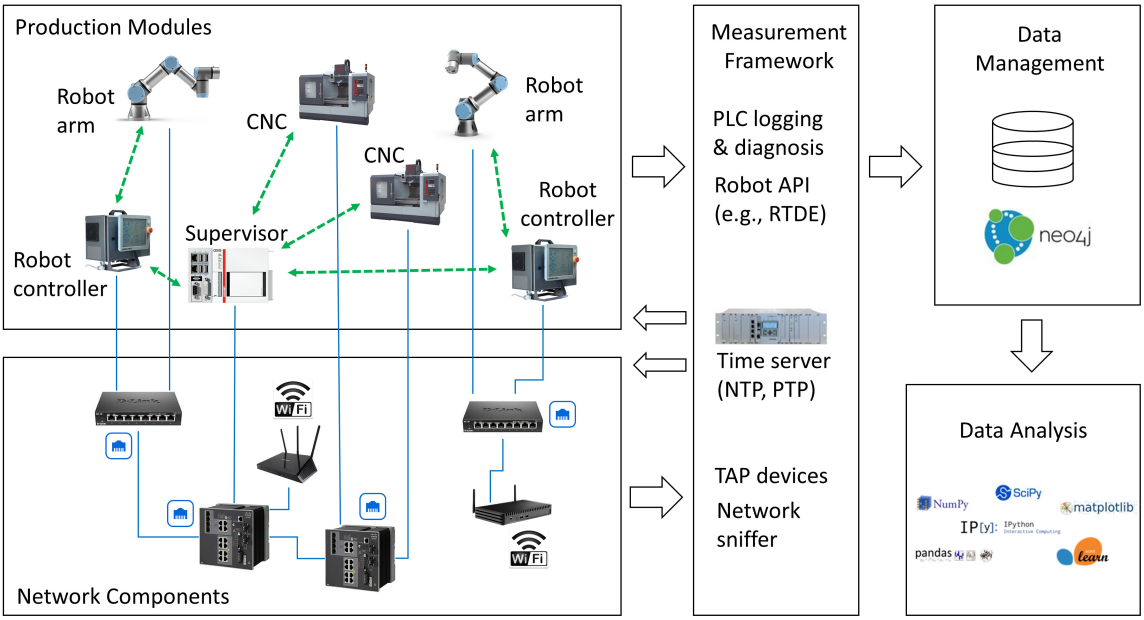


Fig. 1. The NIST industrial wireless testbed architecture

The NIST industrial wireless testbed is a work-cell level evaluation platform which generates typical communication sessions between production modules and enables close examination of transmission performance in communication links using wired or wireless technologies. As shown in Fig. 1, the architecture consists of subsystems including production modules, network components, a measurement framework, and data tools for management and analysis. *Production modules* are components of the work-cell manufacturing system that participate in job-oriented operations, such as order processing, task scheduling and assignment, tooling procedures, and machine tending. *Network components* are networking devices and the communication infrastructure, if there is any, on the factory floor. They are used to connect production modules to each other and enable their communications for information exchange and process control within and beyond work-cell

fences. These two subsystems can form various industrial system use cases in the testbed experiments with corresponding configurations. The *measurement framework* probes into selected objects and connections to collect data, such as manufacturing progress, machine status, link quality, and network health. Rounds of experiment data will be transferred to a centralized *data management* system which manages records from heterogeneous sources and of different formats. We can utilize *data analysis* tools to explore the stored data for insights of the production process, the underlaying network, and their correlation. In the designed architecture, individual subsystems are connected by predefined interfaces and integrated as a complete evaluation process. Design details of each one will be elaborated in the following sections.

4. Production Modules

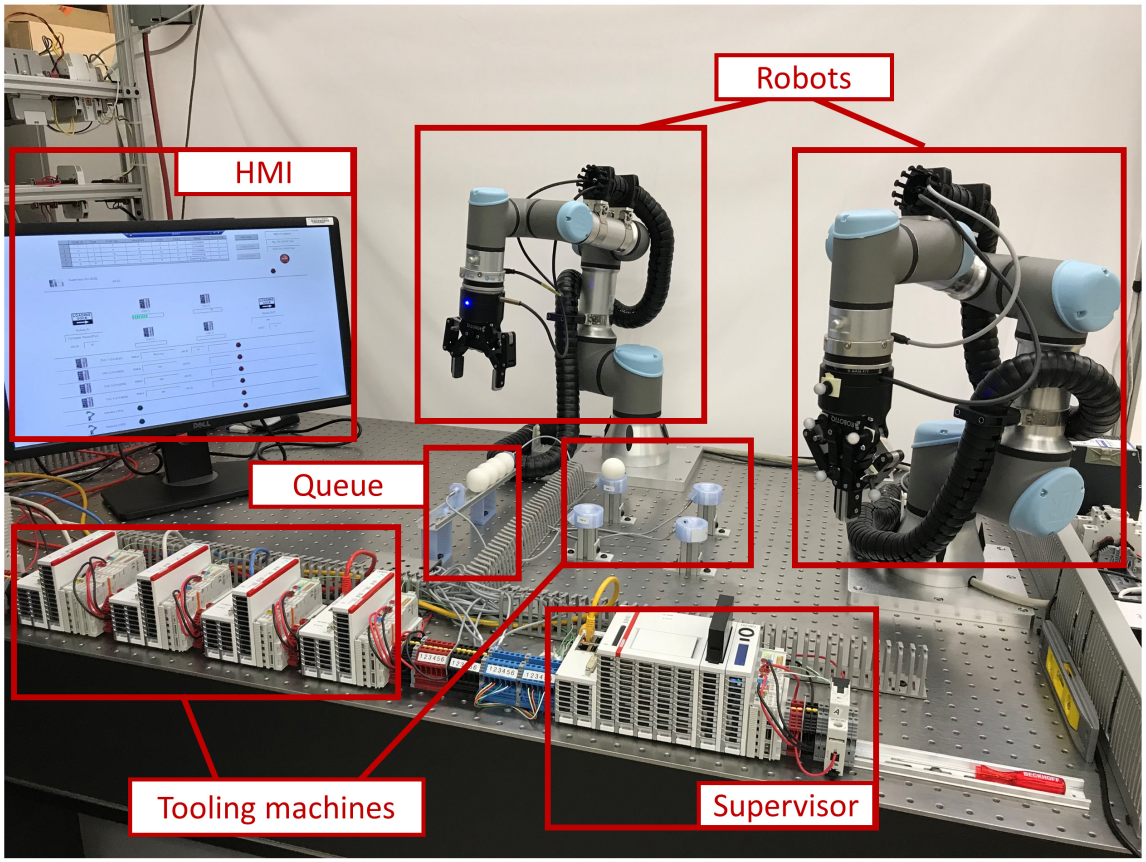


Fig. 2. Product modules in the testbed

In the NIST testbed, production modules are identified as the work-cell supervisor, tooling machines, collaborative robots, job buffers, and human-machine interface (HMI) according to their roles in the production process. Fig. 2 shows a snapshot of production modules in the testbed.

In this section, we first specify the generic production workflow to introduce all the modules. Then, each module will be addressed with its own operation features and design details, such as the components, state machine, and module interface.

4.1 Work-Cell Workflow

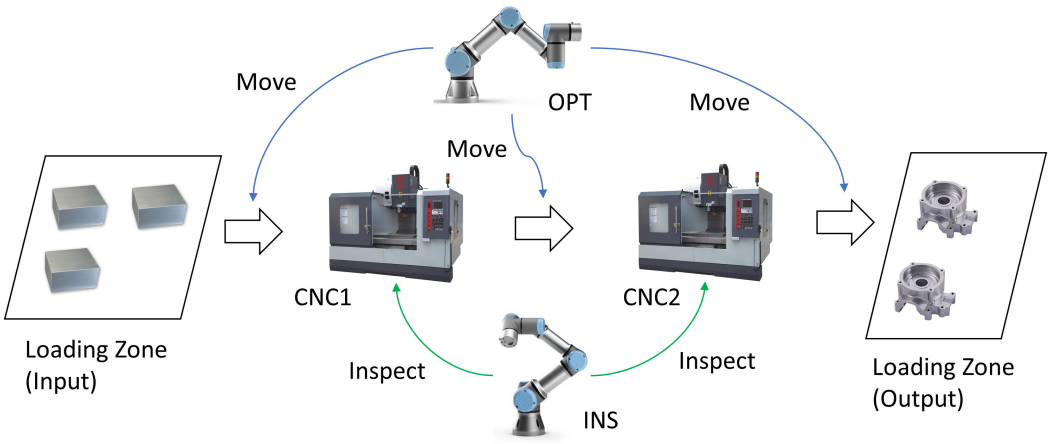


Fig. 3. Illustration of work-cell interactions in a tooling path

Based on available modules, we can develop many different production scenarios in a typical work-cell. In the first release of this report, the testbed is set to reproduce a simple, but typical, work-cell operation with collaborative robots, i.e., the machine tending use case. In the following discussions, we will focus on design issues of realizing this use case in the testbed. Specifically, work orders for the work-cell are submitted through the HMI. The supervisor adds received orders into a queue and processes them in the order assigned by the priority rule. Currently, it adopts the simple "first come, first served" policy. An order contains a batch of parts to be processed and the work plan specifying the tooling path, i.e., how each part moves through the work-cell until it is completed. The manufacturing job on a part can be one or multiple procedures performed in CNC machines, such as drilling, welding, painting, and heat treatment. Each machine is referred to as a job stop in the tooling path. Loading zones, or job buffers, are also treated as independent job stops where parts are temporarily stored waiting for the transfer to next procedure. There are two robots in the work-cell having different jobs in the production: one as the operator (OPT) and another as the inspector (INS). OPT moves parts between job stops; INS checks the part quality after each machine task and reports the inspection result to the supervisor. CNC machines and robots are the job-level actuators. They all rely on the supervisor for instructions of the next move. The supervisor, on its own, handles work orders, schedules operations of machines and robots, and monitors the whole work-cell status. Fig. 3 illustrates interactions in a simple two-stop tooling path for a single part

production. The repeated operations provide ample opportunities to collect statistically significant metrics of both the network and the operation of the work-cell.

4.2 Supervisor

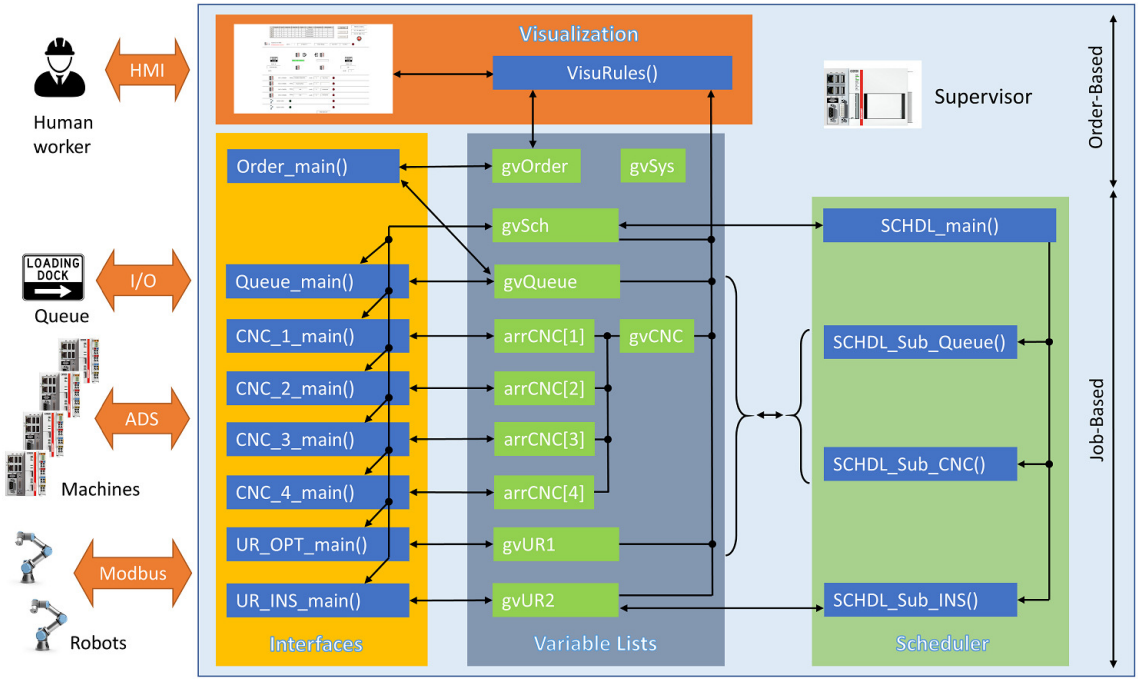


Fig. 4. Function modules in the supervisor PLC. Subsystem functions/programs are in BLUE, and global variables are in GREEN.

In the work-cell, the supervisor plays the role of a "brain". It interacts with upper-level manufacturing managers on behalf of the entire work-cell and manages the equipment, input materials, and output products in its coverage. Along the part's tooling path, as shown in Fig. 3, the supervisor keeps monitoring and manipulating all job stops and robots that are involved in individual procedures following the work plan. To fulfill its role as the central controller in the production, we identify the main functions of the supervisor: production scheduling, interfaces to upper management units or human workers (known as "northbound" interfaces), interfaces to field components of the work-cell (known as "southbound" interfaces), and inventory management.

The supervisor of the testbed is deployed in a Beckhoff PLC of the model CX2020. Accordingly, these functions are implemented in the PLC as shown in Fig. 4. To deal with different work-cell objects, a modular design is used to manage the PLC's memories, computing power, and interfaces. Four major modules defined in the supervisor PLC are the scheduler (SCHDL), global variable lists (GVL), interfaces, and visualization. SCHDL makes actuation decisions based on real-time equipment status and production progress.

When a decision is made, the PLC will generate command messages and send them out through the interfaces to corresponding external actuators. Among *Interfaces*, the northbound connects to upper-level factory entities, such as manufacturing execution systems (MES) and enterprise resource planning (ERP) systems, dealing with incoming orders and status reporting; the southbound, on the contrary, is linked with field devices including machines, robots, and job buffers. The system-wide data sharing between function modules takes place in GVL, e.g., caching machine status and control decisions exchanged between interfaces and scheduler. System variables associated with a specific function or work-cell component are managed in its GVL instance and are named accordingly, e.g., *gvCNC* contains four array objects each of which stores the relevant information of a CNC machine in the work-cell. In addition, the PLC also hosts a human-machine interface (HMI) in the *visualization* module for interactions with human staff.

From the perspective of information processing, functions and data memories in the supervisor can be divided into two planes: order-based and job-based. Order-based functions deal with incoming orders, update of the order status based on real-time production results, and maintaining the inventory. On the other hand, job-based functions mainly work on the associated work-cell components and coordinate their production activities following the schedule. Such a modular design allows the supervisor to easily adapt to the evolving work-cell tasks and utilize state-of-the-art techniques to leverage individual functions.

4.2.1 Scheduler

The supervisor is deployed in a Beckhoff's CX2020 PLC, which runs the TwinCAT 3 real-time engine in a Windows embedded standard (WES) 7 operating system. TwinCAT 3 guarantees determinism of PLC tasks with fixed cycling steps. The cycle time is set to 1 ms in the testbed. The scheduler function is called at the beginning of each cycle so that the scheduler checks system status and makes operation decisions at 1 kHz.

The supervisor implements a simple job scheduler which follows a few rules to work.

- **Rule 1: Single active order**
  At any time, the schedule can treat one and only one order in the work-cell. Once an order is being processed, the other orders should be held until it is completed. When the scheduler is free for picking the next order, the selection criteria include, but not limited to, the order's priority (i.e., the primary factor) and its arrival time in the queue (i.e., the secondary factor). Since multiple parts in an order can be processed at different CNC machines simultaneously, the scheduler has to manage production components and coordinate their actions in the parallel tasks given the work-cell's capacity. For example, if the order indicates a tooling path involving two machines, the scheduler can assign a new part to the machine of Step One after the finished part leaves for Step Two. Both machines can run at the same time working on parts in different procedures for the same order.

- **Rule 2: Single active robot**
  Robots are moving objects in the work-cell. For safety reasons, only one robot is

allowed to move in the work-cell at any time. In other words, the scheduler will lock one robot in its home position before enabling another robot to work. As shown in Fig. 3, the production progress is always related with robot moves, either a part transition between job stops (by OPT) or an inspection (by INS). Specified in this rule, each time the scheduler can only make one robot, either OPT or INS, work. We use the term of *scheduling window* to refer to the moment when the scheduler is free to choose the next move for the robots.

- **Rule 3: In-process parts first**
  In-process parts are those who have started their tooling procedures in work-cell machines. In a scheduling window, in-process parts may have finished the work in current job stops and are waiting for the robot to either perform inspection or transfer to the next stop. Following Rule 2, they should be served first before any new parts. Of note, among all pending requests, intra-stop moves, i.e., inspection, are always prioritized. With multiple part move requests of the same type, the scheduler uses the round robin method to select the first one to serve in the queue.

As shown in Fig. 4, the scheduler's main function, *SCHDL_main*, is called to verify the scheduling window and look for the next move at the beginning of each cycle. Individual member functions will then create instructions to the selected machine and/or robot based on the scheduler's decision and cache them in the corresponding variable lists. In the same cycle, interface functions will check the saved variables and transmit the newly added scheduling information to external devices.

4.2.2 HMI

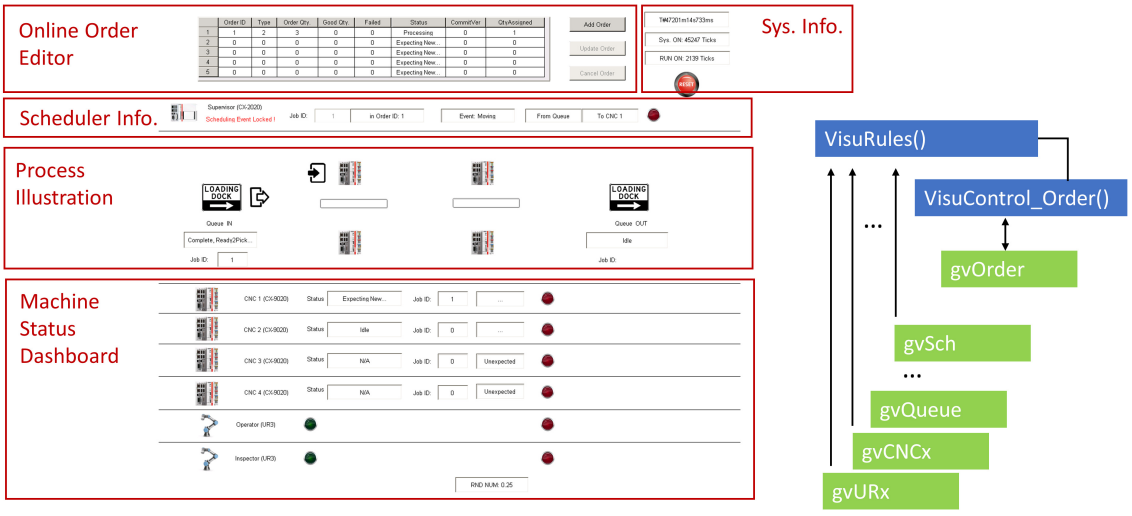


Fig. 5. Work-cell HMI and the control functions

The work-cell HMI server is deployed in the supervisor PLC as shown in Fig. 5. HMI serves as the main northbound interface for external communications. HMI displays the work-cell status collected by the supervisor in multiple windows. Fig. 5 shows the main window during the operation. Transitions to the other HMI windows provide additional configuration options other than the default, e.g., measurement setup. Human workers can remotely monitor and interact with the automated production process in specific windows, such as placing orders and stopping/resetting the production.

HMI has its own PLC task thread in Beckhoff CX2020 whose priority is lower compared to the job scheduler. The main visualization control function, *VisuRules*, is called every 200 ms to refresh the view. It reads the GVL variables, such as *gvSch* for the scheduler, and converts values to strings or Boolean values for respective display objects. It uses an independent function, *VisualControl_Order*, to handle any update in the online order editor of the main window. *VisualControl_Order* interacts with *gvOrder* in GVL to add, remove, and update orders in the work-cell.

HMI can be accessed directly through the supervisor PLC. The PLC runs a customized Windows operating system and has fully functioned computer interfaces, such as a video output and USB ports, HMI can be displayed in a screen connected to the PLC and operated by input devices such as a mouse. Thanks to the Windows remote desktop connection, we can also access the work-cell HMI through a remote host that resides in the testbed's LAN.

4.2.3 Job Buffers

Job buffers are the temporary storage for parts before or after the production which is comprised of two loading zones in the work-cell, i.e., the input (Queue IN) and output (Queue OUT) buffers. They also serve as the start and end job stops in a tooling path for a single part, respectively. The tooling path can be in turn presented as a sequence of job stops, e.g., $< Queue\ IN > - < CNC1 > - < CNC3 > - < Queue\ OUT >$. In the testbed, both input and output job buffers are implemented as subsystems in the supervisor and managed by the function *Queue_main*. The supervisor detects part arrivals/departures in the buffers with proximity sensors which are connected to the PLC's digital input module, one for each.

4.2.4 Production Tasks

Work-cell operations in the testbed are designed around three types of production tasks: orders, parts, and jobs. Orders are the basic unit of input assignments from upper-level management systems to a work-cell. An order contains quantities of the requested parts and their manufacturing specs. Parts move between different machines following the planned tooling path. A job is always associated with an actuator, i.e., a series of actions are taken by the actuator in treating the part. For example, a CNC machine serves to drill holes in the part, or a robot moves the part between the consecutive job stops. Elements used to describe a job include the commanding body (which assigns it), the execution body (which performs it), the treated part/object, action(s), and the time.

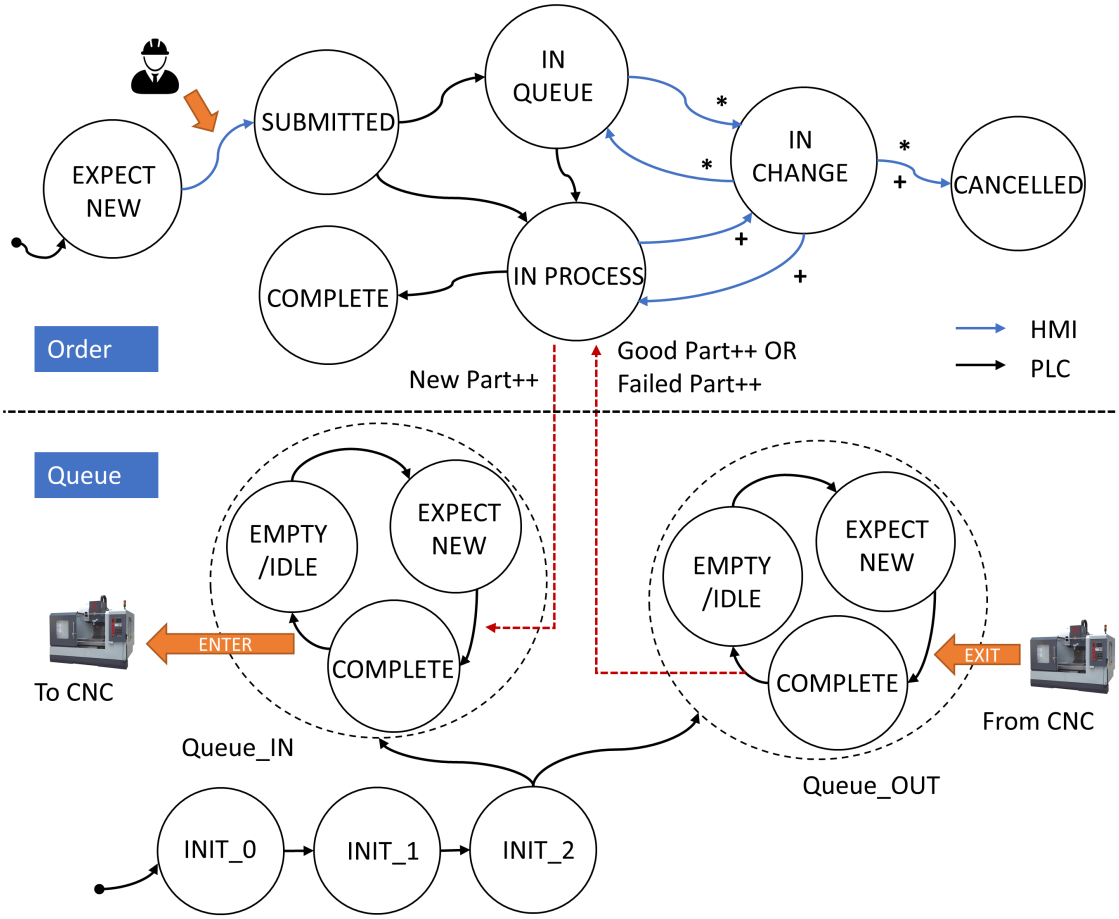


Fig. 6. State machines of orders and queues

In the work-cell, the supervisor is responsible for treating the order-related information. It receives orders of parts, assembles production instructions for them, and coordinates work-cell components in the production. At the supervisor, the order module and the queue module (which manages job buffers) have their own state machines, as shown in Fig. 6. The scheduler interprets an incoming order and identifies the part's tooling path in terms of a planned sequence of jobs at the selected work-cell actuators. Specifically, only the "IN PROCESS" order is loaded by the scheduler, i.e., one active order is in production. Each time when a new part is scheduled for the current order, the scheduler assigns a new job to the input buffer. The part production starts at the input buffer and moves on to the first tooling machine once it is available. The scheduler keeps tracking the part status until it finishes the last tooling task and arrives at the output buffer. Then, based on the inspection results, either a good product or a labeled failure is added into the order's record. Once the good part number is fulfilled, the order will move to the "COMPLETE" state. The queue module will start to process the next pending order by labeling its state to "IN PROCESS".

Interactions between the order module and the queue module, which are indicated as

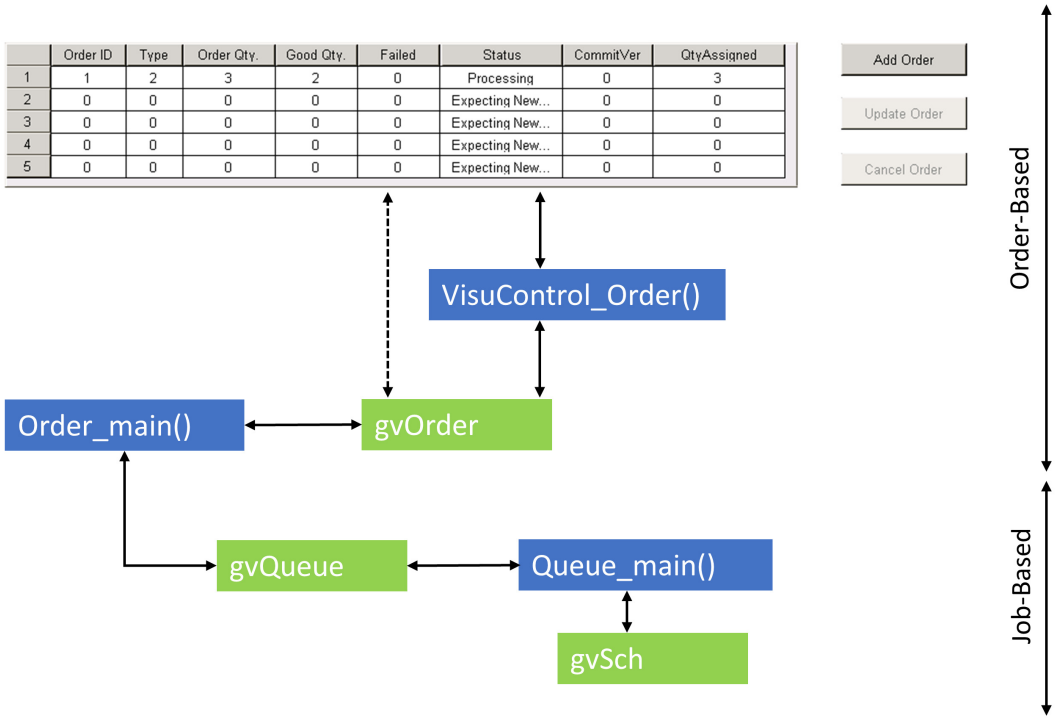


Fig. 7. The online order editor module in HMI

red dashed arrows in Fig. 6, are mainly through the exchange of variables in the GVL *gvQueue* shown in Fig. 7. All the updates in the order module are finally visualized in the HMI online order editor.

4.3 Tooling Machines

We consider four CNC machines as machining tools in the work-cell. Production processes for individual jobs in these machines are programmed and managed by their own controllers. The embedded CNC controllers communicate with the other work-cell members, mainly with the supervisor, for production coordination. Therefore, each CNC machine is emulated by a separate PLC in the testbed which focuses on characterizing its tooling behaviors and communication traffic patterns. Each CNC machine consists of a Beckhoff CX9020 PLC, a part holder, and a proximity sensor. The PLC mimics states of the emulated CNC machine in its tooling cycle and exchanges the instantaneous status and job information with the supervisor. The part holder represents the machine's working zone where the part is treated. The proximity sensor is placed in the holder to monitor the part arrival/departure and is connected to the PLC's digital input module.

The testbed is aimed to evaluate the correlation between data transmissions and work-cell operations which impact the whole system performance. Therefore, the CNC emulator is focused mainly on mimicking the machine's behavior with time dependent and statistical

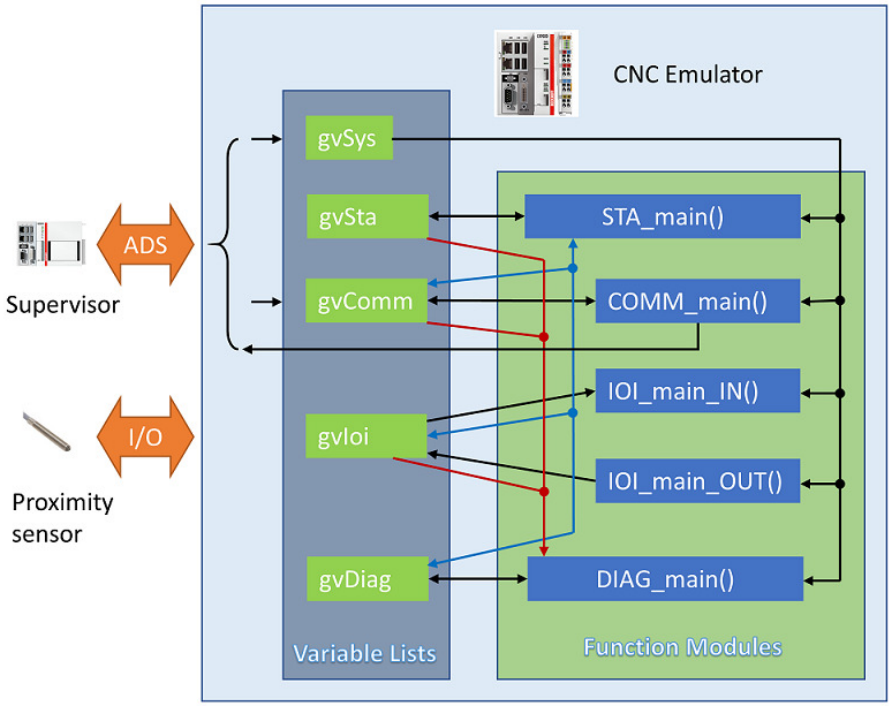


Fig. 8. Architecture of the CNC machine emulator

performance features, such as the production efficiency, error and downtime distributions, and part defects. Meanwhile, the emulators also help shape the work-cell data traffic with their periodic status updates and on-demand messages during the production.

Following the similar modular design in the supervisor, the CNC emulator also defines its function modules and GVL in the implemented PLC as shown in Fig. 8. Specifically, function modules include the state machine (STA), communications (COMM), I/O module interfaces (IOI)[1], and diagnostics (DIAG). The shared system variables between modules are maintained in GVL, e.g., *gvSta* maintains variables related with the state machine and *gvSys* contains the system-wide information such as the machine's identification (ID) and network address.

To fully capture operational and communication activities of a machine tool, the CNC emulator conducts state-dependent operations and communications characterized by the state machine, as shown in Fig. 9. The state machine is serviced in the STA function module, which contains three main states: initialization (INIT), IDLE, and BUSY. Each main state contains a few substates, which characterize further details of operations. INIT along with its substates facilitate the synchronization among distributed nodes whose design will be discussed with more detail in Section 4.5.1. The substates of the "BUSY" state represent a series of machine operations regarding a single job. Using either embedded timers

[1]The IOI functions are further grouped into IOI IN and IOI OUT, respectively.

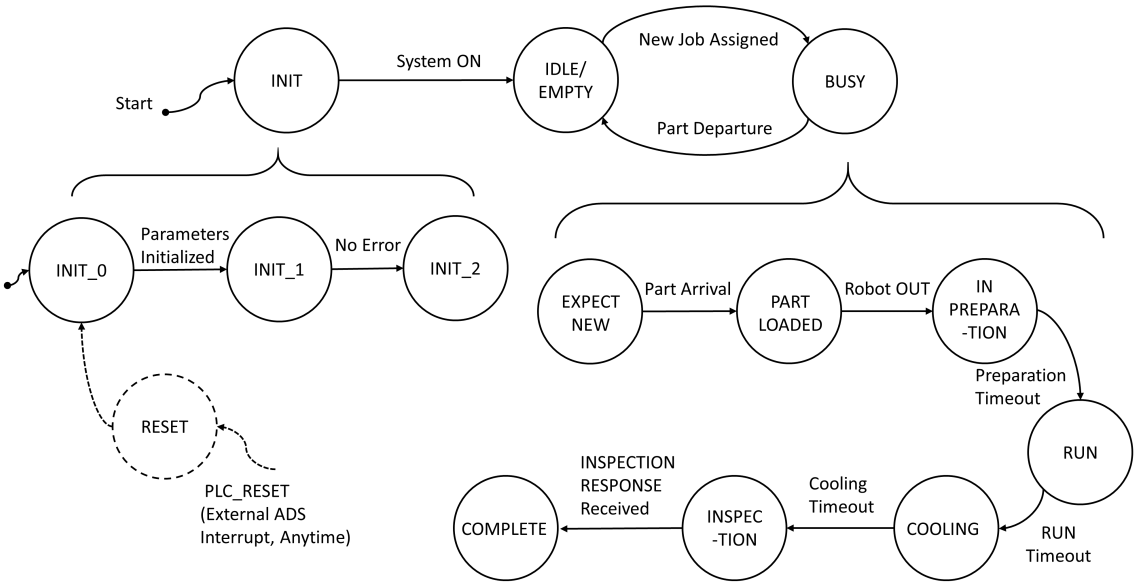


Fig. 9. CNC state machine

or external communication messages, the emulator triggers state transitions. The dwelling time in each (sub-)state can be either timed according to the machine's specification, e.g., the approximate G-code execution time and material removal rate, or determined by external events that trigger state transitions, e.g., a notification message. Randomness can also be introduced based on statistical machine/production models. Examples of randomness components in the models include: 1) the time of a tooling procedure; 2) time varying energy consumption in different states, e.g., power variations in material-drilling processes; 3) tool life estimation; 4) part defect rate; 5) measurement drift between calibrations; and 6) safety related events, e.g., unexpected interrupts due to object intrusion. Using empirical models and measurement data, we can model the above performance metrics statistically and regenerate the state-related traffic for the studied machine.

Therefore, the machines emulated in the testbed can be programmed to highlight the details of real practices to study the network impact on the work-cell performance. Process variables are modeled in the testbed by focusing on different topics such as 1) the production (task) efficiency, e.g., the execution time, material removal rate, energy consumption, and part defect rate; 2) asset health, e.g., the tool life time, failure probability, and downtime schedules for calibration and maintenance; and 3) work-cell collaboration, e.g., the clock drift, coordination precision, and safety. Besides checking the network support on routine data transmissions as scheduled, the testbed is particularly useful for testing the network performance in extreme cases with rare occurrences, e.g., the shutdown due to overheating or power surges. The machine emulator can produce the traffic in different use cases following the code, such as the recovery from unexpected overload events or in emergency

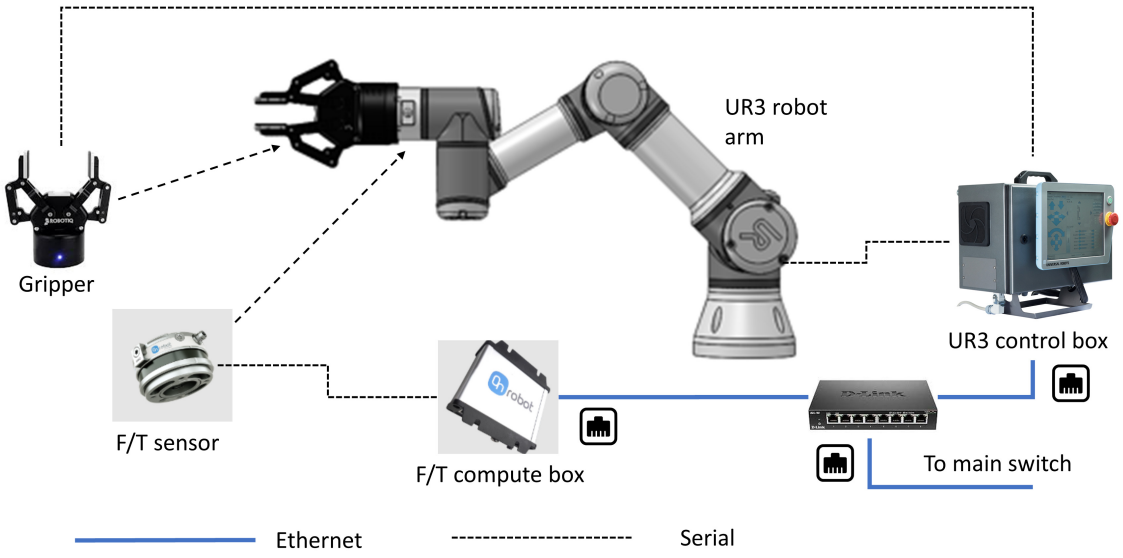


Fig. 10. A UR3 collaborative robot system

cases, and can repeat the case in multiple runs for conclusive experiments or comparative studies.

The quality of "product" in each job stop is also virtually rendered in the testbed. Each part quality in a machining process is randomly determined following a statistical model that mimics the typical defect rate in a real machine. The result is made in the inspection phase which allows the supervisor to schedule the next move accordingly. According to the study of the quality and quantity relationship in production systems [21, 22], part failures are associated with both independent and dependent factors. Independent failures usually follow a Bernoulli distribution with the uncertainty of temporal independence. On the other hand, dependent types of failures, which are often referred to as "persistent" or "systematic" ones, are those caused by tool failures, such as the broken drill or clog in the painting tube. In such cases, the failure of product is highly related with the asset failure rate. Since both types of failures are intuitively decoupled, the testbed carries product failures as well as the ones in assets to emulate occurrences of exceptions across time. Delivery delay or loss in communication links also affect operation performance and safety measures.

4.4 Robots

The testbed hosts two Universal Robots UR3 CB-series robots that perform machine tending in the work-cell. As shown in Fig. 10, each UR3 system consists of a 6 degree-of-freedom (6 DoF) robot arm, a controller in the control box, and extended UR capabilities (URCaps) which are accessories including a gripper and a 6-axis force torque (F/T) sensor. The F/T sensor is further divided into two parts: the sensor unit attached to the end effector and a compute box connected to the UR3 control box. The UR3 provides a graphical user interface (GUI), called Polyscope, which serves as the main programming interface in the

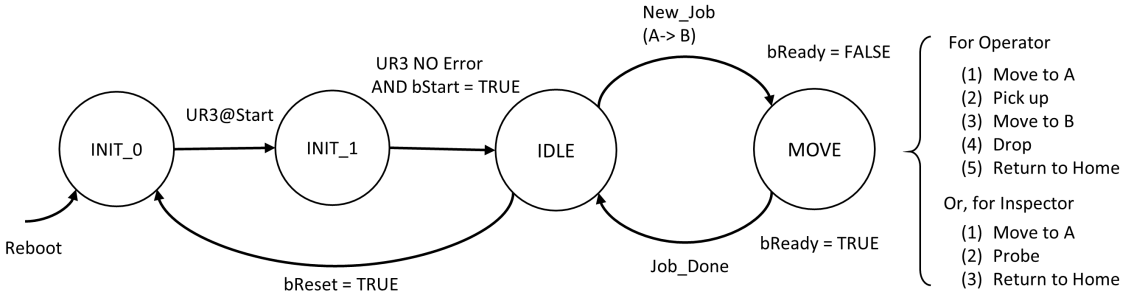


Fig. 11. The state machine of the UR3 robot in the work-cell

teach pendant. In Polyscope, users can input actuation commands and configure waypoints using programming wizards. In addition, users can also use the URScript programming language to assemble scripts for the robot control. In the control box, the UR3 controller interprets user input to plan the trajectory of the robot arm and dispatches set points to microcontrollers of individual joints at 125 Hz. At the same rate, each microcontroller manipulates the corresponding joint with the calculated force and torque.

The controller also acts as the robot's communication portal to the work-cell. Robots receive actuation commands from the supervisor and report their status back. In the work-cell, the OPT's job of transiting parts can be translated into a sequence of actuation commands in the UR3 script: (1) moving to waypoint A (from the Home position), (2) picking up the part at A, (3) moving to waypoint B, (4) unloading the part at B, and (5) returning to the Home position. Setting profiles of actions, such as arm positions at individual waypoints, are stored in the controller and loaded into the program once it starts. Similarly, INS's actions in a job include (1) moving to waypoint A (from the Home position), (2) probing the seated part at A and generating an encoded inspection result, and (3) returning to the Home position.

To interplay with the supervisor/scheduler and machines in the work-cell, the robot's state machine is also divided into two phases: initialization and operation, as shown in Fig. 11. During initialization, the robot walks through *INIT*_0 and *INIT*_1 following instructions from the supervisor. In *INIT*_0, it resets all internal parameters and returns to its Home position out of the working space; in *INIT*_1, the robot exchanges its status with the supervisor and gets ready for any new operation. In the normal operations, the robot switches states between IDLE and MOVE while reporting the supervisor about its real-time status through the network.

The UR3 performs a series of status checks at the beginning of its control cycle and takes actions correspondingly given the state machine. Fig. 12 illustrates the major steps and branches in a typical control cycle.

4.5 Coordination between Work-Cell Modules

Since work-cell components are collaboratively working in the production, the testbed implements multiple approaches to coordinate these distributed nodes.

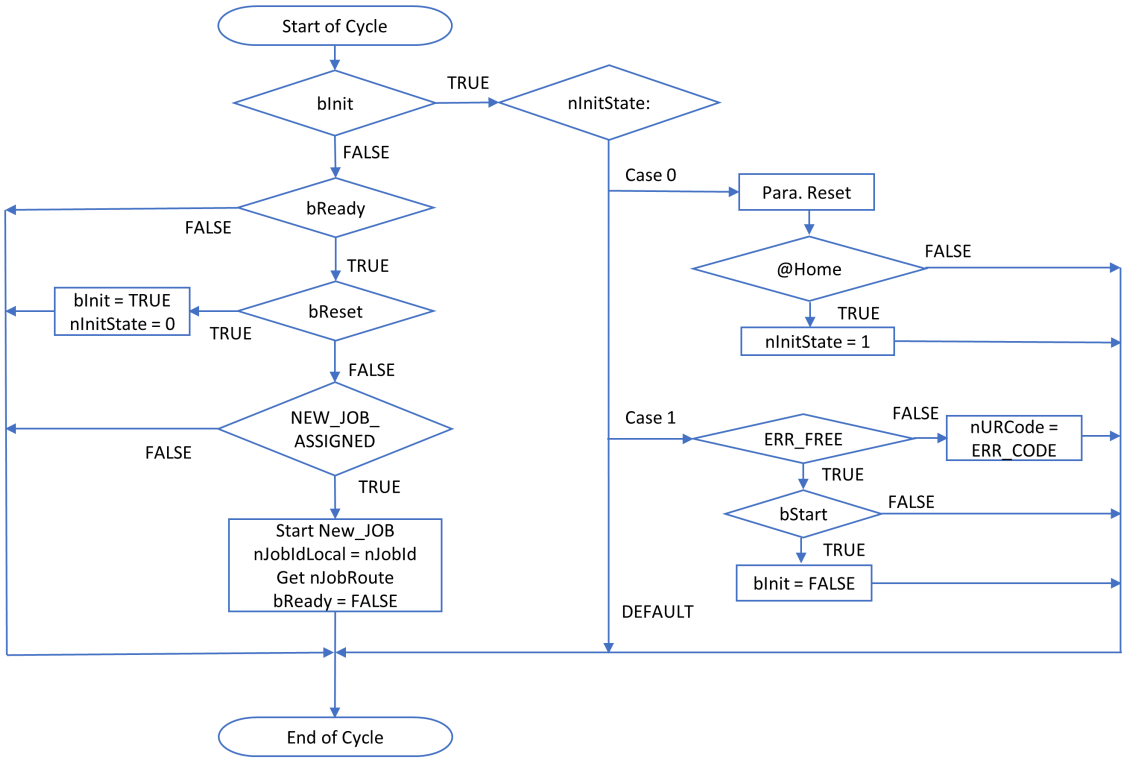


Fig. 12. The flow diagram in a UR3 robot cycle

4.5.1 Coordination in Initialization

At the beginning of the testbed experiment, after powering on all components, a phased initialization process is performed where three steps are taken sequentially:

- **INIT_0** Parameter initialization/reset,
- **INIT_1** Logic error check and confirmation, and
- **INIT_2** Loading ready-to-go state.

The scheduler in the supervisor provisions the whole initialization process over the work-cell. Fig. 13 illustrates such a process within the supervisor and other remote production modules. Red dashed lines refer to state switches triggered by messages. Specifically, the scheduler triggers its own state transitions right after it receives notifications from the other sub-modules in the supervisor that acknowledge the completion of designated procedures in the current phase. As shown in Fig. 13, the triggering events are denoted by blue curved arrows. The scheduler will then dispatch signals in the supervisor to trigger transitions in the internal sub-modules (shown in red curved arrows). For production modules in the field and their corresponding interfaces in the supervisor, messages carrying status information of the counterpart can serve to trigger internal state transitions. For example, the CNC x interface needs the remote CNC x machine to confirm the reception of restart command before it switches from *INIT_0* to *INIT_1*. In another case, remote modules wait for the interfaces to set a start flag before it completes *INIT_2* and starts normal operation. In both cases, messages are denoted by green curved arrows in the illustrated process which serve as a necessary condition in state transitions. In *INIT_2*, remote modules first enter the idling mode getting ready for optional commands from the supervisor. Then the supervisor's sub-modules enter the ready mode coordinated by the scheduler. Once the entire testbed is ready, the production starts from the first order placed into the order module.

Thanks to the introduced initialization process, a "soft" online reset scheme is implemented in the testbed. Any time when clicking the "reset" button in the HMI, we can gracefully stop the ongoing experiment and reset the whole work-cell. Once reset is flagged, the scheduler can detect it at the beginning in the next PLC cycle (in 1 ms or less). It dumps all remaining orders in the queue, clears working status, and restarts from *INIT_0*. As shown in Fig. 14, starting from the scheduler, a system-wide reset will first expand from the scheduler to all internal modules in the supervisor, then reach out to individual work-cell modules through the established interfaces.

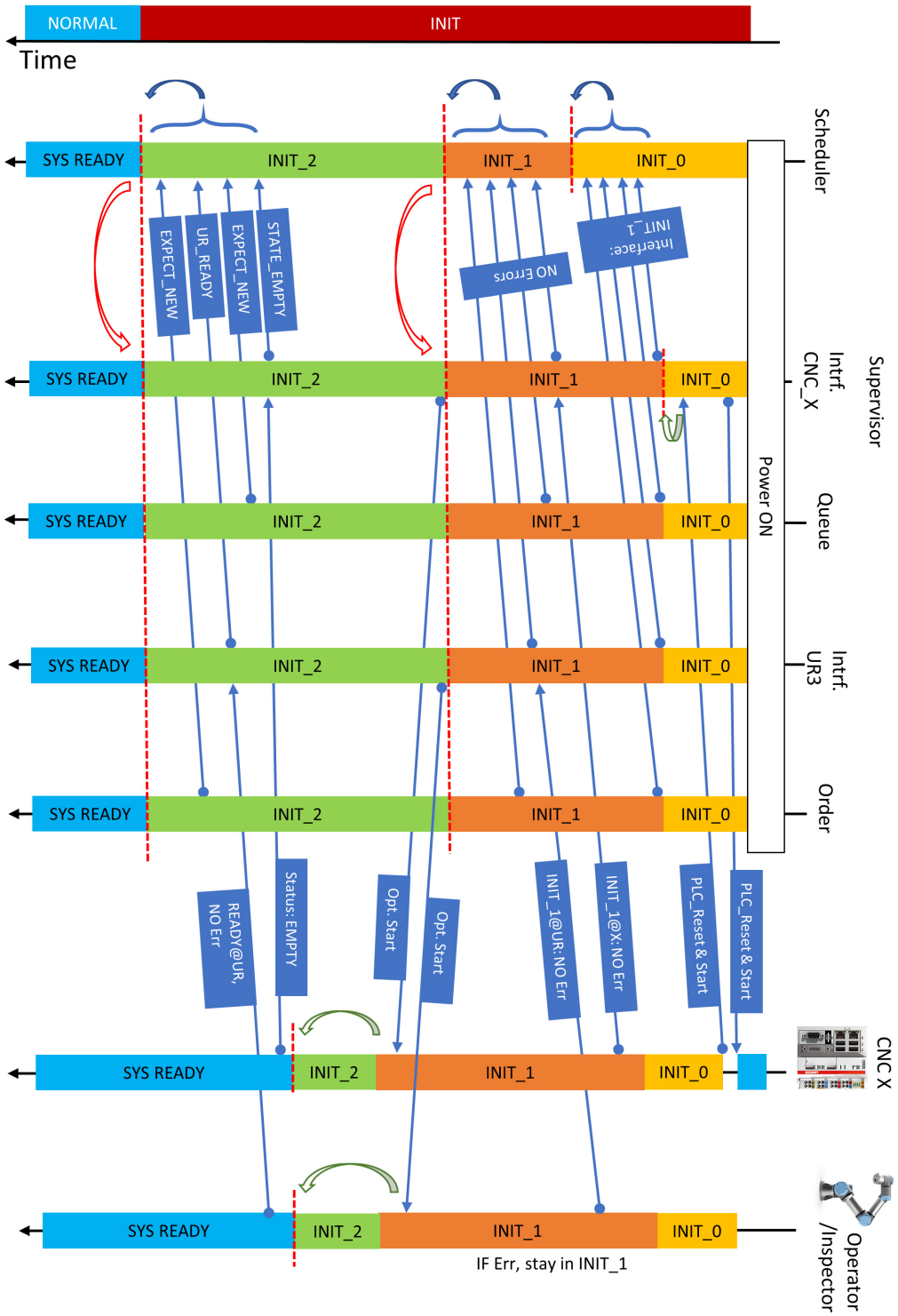


Fig. 13. Timeline of initialization steps.

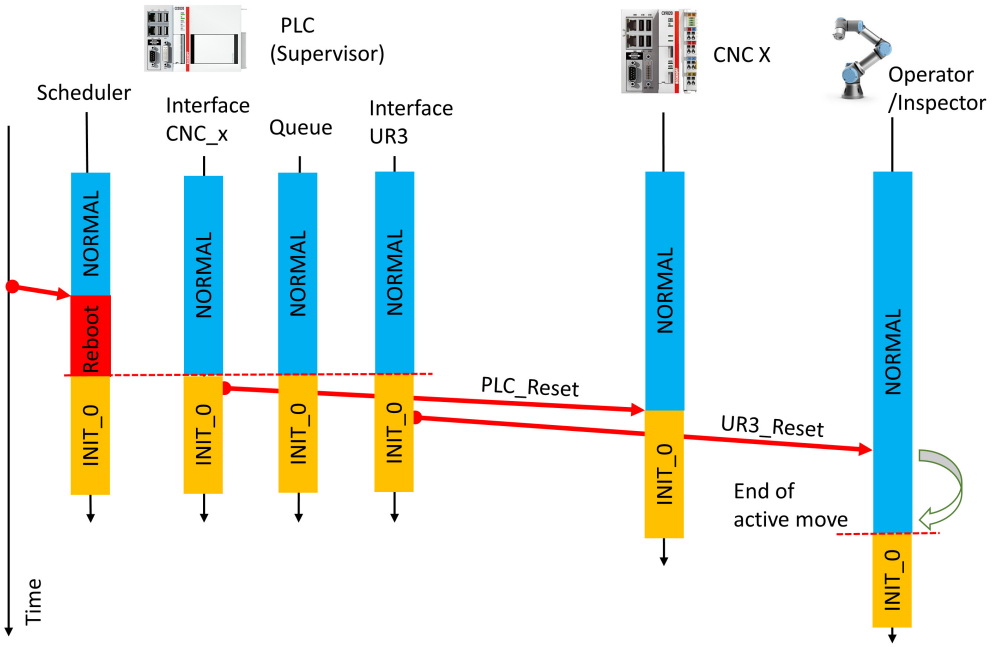


Fig. 14. Reset coordination between work-cell modules

4.5.2 Coordination in Job Operations

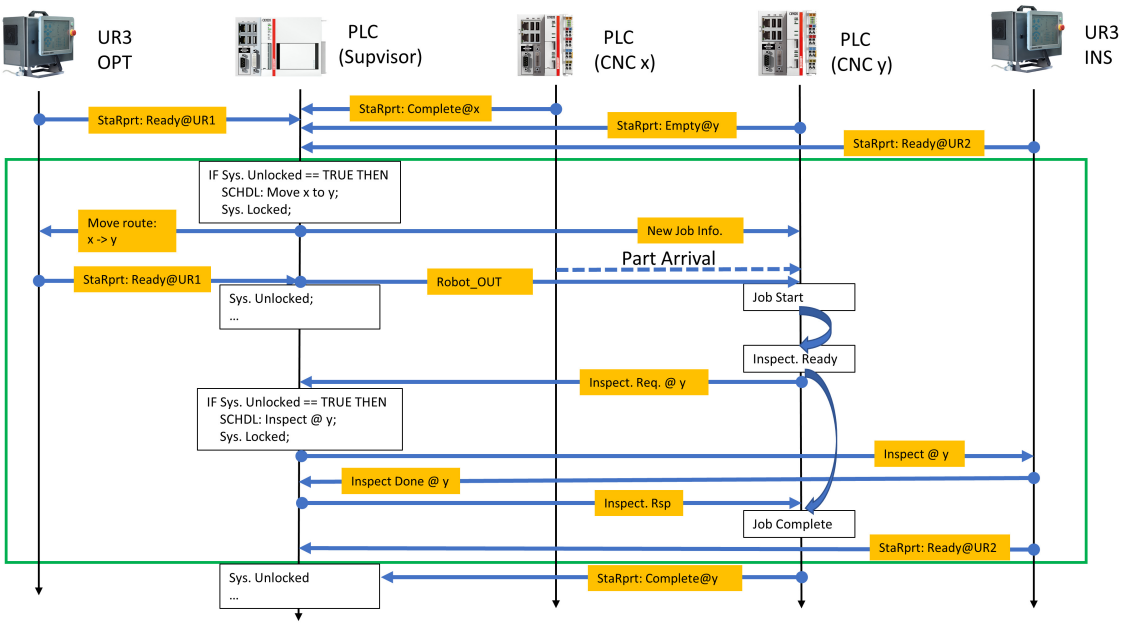


Fig. 15. Timeline of the coordination in an intermediate tooling procedure

Multiple production modules are involved in the tooling path. Recall the workflow as

introduced in Section 4.1, to complete the part processing in a single machine, it needs the supervisor, both robots, and the machine to work together. The coordination between actuators becomes necessary to fulfill production operations and guarantee the safety. Fig. 15 illustrates the timeline of such coordination in a machining procedure.

Testbed operations are designed as a fully automated process, which needs no physical contact with parts or actuators by human staff in the production. Workers are anticipated to manage the work-cell through the HMI remotely. Possible collision risks are only related with part-oriented interactions between robots and running machines, e.g., when OPT unexpectedly approaches to a part that is being treated in a fast running CNC. Without well-established safety rules for work-cell operations, such incidents may interrupt the ongoing production, or even worse, cause asset damages that brings the work-cell down. Therefore, we have introduced multiple safety approaches in the job coordination to eliminate risks and protect the assets. First, the supervisor implements a safety flag in its scheduler to indicate whether an active robot is moving in the work-cell. It complies with the scheduling Rule 2, i.e., at most one robot is actively operating. Once the flag is set, the locked scheduler would not assign a new job to another robot so that collisions between robots are avoided. Second, the moving robot keeps notifying the machine(s) that may be affected by its maneuvers in the active job trajectory. The machine(s) on watch would not start to process the part until the robot returns to its Home position, such that it leaves a safe space for tooling operations. As shown in Fig. 15, the "Robot OUT" message clears such watch after the acting robot finishes its job. In addition, a logic check is performed at the robot on the waypoint instruction sent from the scheduler, which prevents the robot from operating some out-of-date instructions due to transmission failures. Preliminary experimental results indicate that the introduced approach supports collision-free operations in the testbed through very light coordination, as low as 1 Hz, in the supervisor-robot link.

In the work-cell, there are two main interaction scenarios, part transition and inspection, both of which involve multiple machines with different jobs in the respective processes. Fig. 16 and Fig. 17 illustrate these two cases, respectively.

The main interactions between work-cell (sub-)modules in a part transition scenario are listed as follow,

- **[A (START)]** CNC x just finishes the part processing and changes its status to "COMPLETE";
- **[B]** CNC x updates its status to the supervisor via routine status report in its ADS connection to the supervisor;
  - **[B1]** The COMM module in CNC x loads current status and formulates a status report triggered by self timer;
  - **[B2]** The COMM module sends the status report to the supervisor (ADS Write);
- **[C]** The scheduler in the supervisor receives the status and updates it in the machine status register;

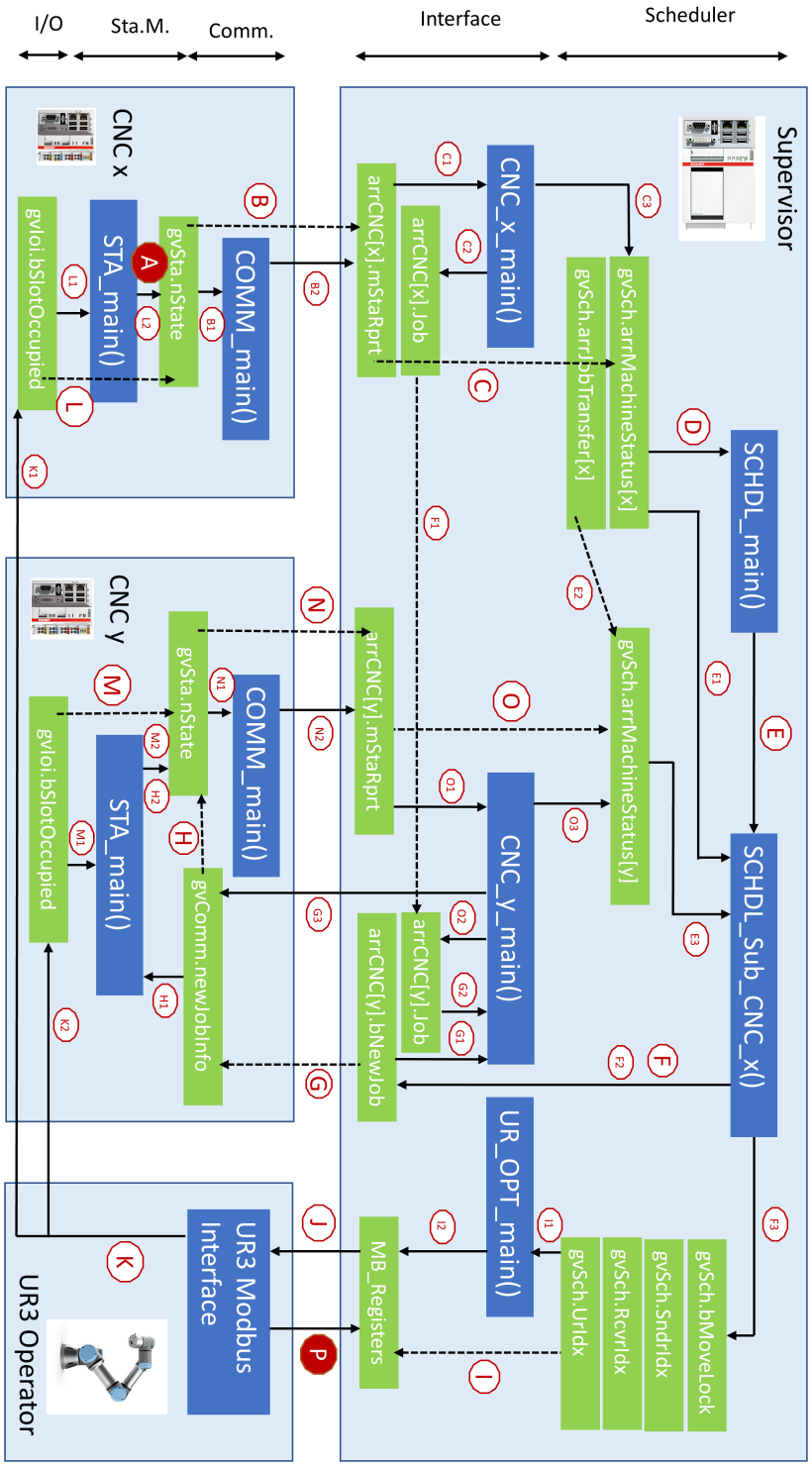


Fig. 16. Module interactions in a part transition case

  - **[C1]** CNC x interface at the supervisor loads the status report;
  - **[C2]** CNC x interface updates the job status;
  - **[C3]** CNC x interface submits the status to the scheduler by writing into the machine status register;

- **[D]** The scheduler loads the CNC machine status to make scheduling decisions;
- **[E]** The scheduler assigns the job of moving the part from CNC x following the scheduling algorithm (which is independent from the architecture but includes at least the following information);
  - **[E1]** The scheduler loads CNC x's status;
  - **[E2]** The scheduler looks for the next hop station from x based on the job type;
  - **[E3]** The scheduler loads the status at the next hop station, i.e., CNC y;
- **[F]** The scheduler makes the moving scheduling if CNC y is in the "IDLE/EMPTY" state;
  - **[F1]** The job information is copied from CNC x's interface to CNC y's interface;
  - **[F2]** The scheduler sets the new job notification to CNC y (followed by Step G);
  - **[F3]** The scheduler writes the scheduling information (followed by Step I);
- **[G]** The new job information is shared with the remote CNC y;
  - **[G1]** The CNC y's interface at the supervisor detects a new job assigned by the scheduler;
  - **[G2]** The CNC y's interface at the supervisor loads the job information;
  - **[G3]** The CNC y's interface sends the new job information to CNC y via ADS;
- **[H]** The new job notification updates the machine's status at CNC y;
  - **[H1]** CNC y's state machine detects the arrival of new job description;
  - **[H2]** CNC y's state machine switch to "EXPECT NEW" waiting for the incoming part;
- **[I]** The Supervisor's scheduling decision is transformed into the Modbus commands for the UR3 operator;
  - **[I1]** The Operator's interface at the supervisor detects the new assignment;
  - **[I2]** The Operator's interface writes the scheduling information into the shared Modbus registers that are read/written by the Operator;

- **[J]** The Operator obtains the assigned job through Modbus Register Read;
- **[K]** The Operator follows the job assignment to move the part from CNC x to CNC y;
  - **[K1]** The Operator removes the part from CNC x (followed by Step L);
  - **[K2]** The Operator places the part into CNC y (followed by Step M);
- **[L]** The part departs from CNC x and triggers the state transition at CNC x;
  - **[L1]** The state machine at CNC x detects the departure of the part;
  - **[L2]** The state machine at CNC x updates the state as "IDLE/EMPTY";
- **[M]** The part arrives at CNC y and triggers the state transition at CNC y;
  - **[M1]** The state machine at CNC y detects the arrival of the part;
  - **[M2]** The state machine at CNC y updates the state as "PART LOADED";
- **[N]** The new state at CNC y is reported to the supervisor via routine status report;
  - **[N1]** The communication module at CNC y loads current state and formulates the report message;
  - **[N2]** The communication module sends the report message to the supervisor via ADS Write;
- **[O]** The CNC y's interface at the supervisor updates the state of CNC y at the scheduler (similar as Step C);
  - **[O1]** The interface at the supervisor loads the newly received status report;
  - **[O2]** The interface writes the new state into the Job description;
  - **[O3]** The interface updates CNC y's state at the scheduler;
- **[P (END)]** The UR3 OPT returns to its Home position and notifies the supervisor.

Similarly, the main interactions between work-cell (sub-)modules in a part inspection scenario are listed as follow,

- **[A (START)]** A part being processed at CNC x requests the inspection;
- **[B]** COMM of CNC x sends an inspection request via ADS;
- **[C]** CNC_x_interface at the supervisor detects the new inspection request;
- **[D]** CNC_x_interface at the supervisor sets the flag (ON) in the Inspection Request variable;

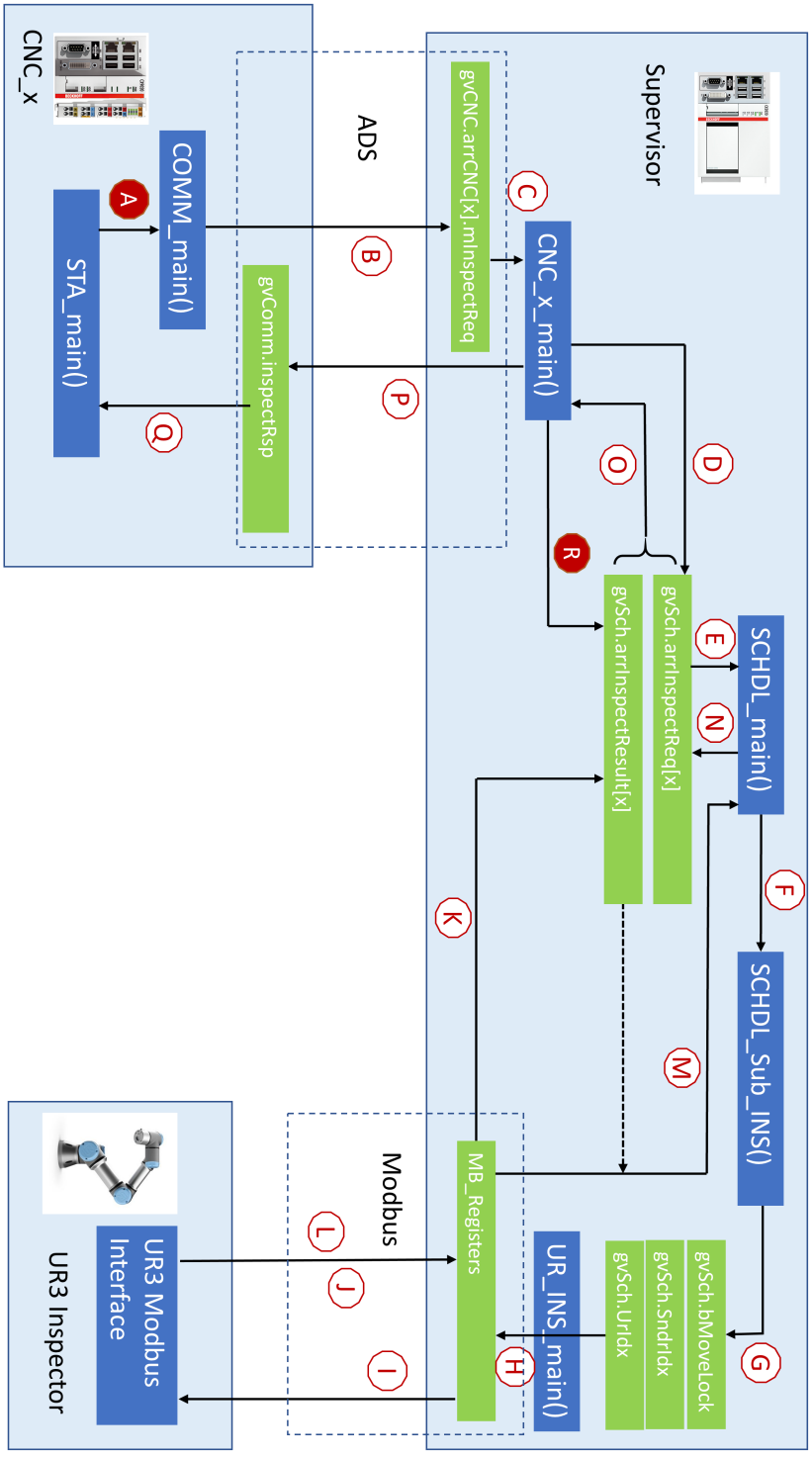


Fig. 17. Module interactions in a part inspection case

- **[E]** The scheduler of the supervisor collects CNC inspection requests;
- **[F]** The scheduler picks the inspection CNC target by calling *SCHDL_Sub_INS*;
- **[G]** *SCHDL_Sub_INS* assigns an inspection task to the UR3 INS robot;
- **[H]** The INS interface at the supervisor writes the task into Modbus registers;
- **[I]** INS retrieves commands via Modbus and executes them;
- **[J]** INS writes inspection results back into Modbus registers;
- **[K]** The inspection result is updated from Modbus registers;
- **[L]** INS finishes the task, returns its Home position, and notifies the supervisor;
- **[M]** The scheduler at the supervisor waits until both Step K and L are completed;
- **[N]** The scheduler at the supervisor removes the flag in the Inspection Request (OFF) and waits for the next call;
- **[O]** CNC_x_interface at the supervisor waits until Step K and N are done;
- **[P]** CNC_x_interface at the supervisor sends an inspection response via ADS back to CNC x;
- **[Q]** CNC x processes the received Inspection Response;
- **[R (END)]** CNC_x_interface at the supervisor clears the finished result for the next call.

5. Network Components

To fulfill the coordination between work-cell modules, a communication network is deployed in the testbed that links distributed modules together and enables communications with multiple purposes. Besides process variable updates and control commands transmissions in the emulated production, the network also enables data transmissions for the other testbed functions, such as distributed clock synchronization and measurement data collection. Therefore, the testbed network needs to carry multiple heterogeneous traffic streams. In this section, we focus on explaining network components implemented in the testbed to serve these applications. First, communication traffic needs are identified in the links; then, the network architecture comprised of wired Ethernet links is illustrated; finally, the extension to wireless links is discussed.

5.1 Communications in Work-Cell Applications

As discussed earlier in Section 4 and illustrated in Fig. 15, communications in the work-cell is centered around the supervisor, which manages the production process and coordinates various modules. In the network topology, the supervisor also acts as the information hub and gateway for operational data flows, both internal and external ones. Messages within and beyond the work-cell are formalized by different communication protocols according to individual applications including, but not limited to, order handling, part status tracking, machine diagnostics, and safety alerts. We will first examine these communication messages and their requirements on the links.

5.1.1 Remote HMI Operations

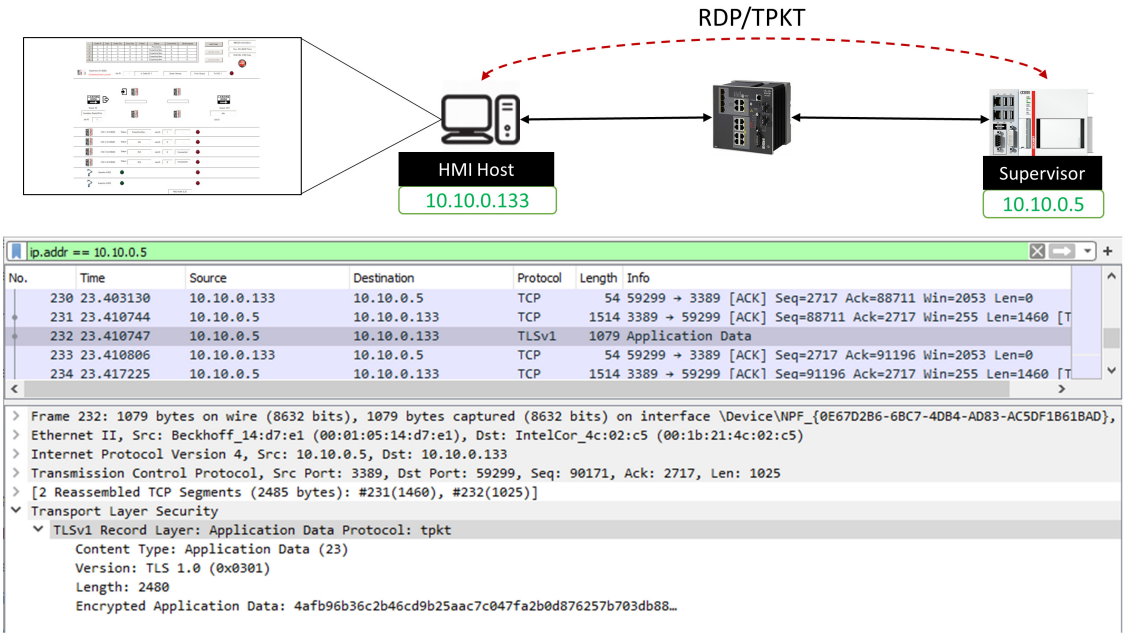


Fig. 18. Communications for remote HMI operations and packet samples captured by WireShark

The main configuration and monitoring tasks in the testbed are through the HMI. When the work-cell HMI is operated remotely, a Windows remote desktop connection is established between the host Windows machine and the supervisor PLC as shown in Fig. 18.[2] Windows manages such a connection using a proprietary application protocol for Terminal Server services, which is known as Remote Desktop Protocol (RDP). RDP uses the ISO Transport Service on top of the TCP (TPKT) protocol in the transport layer. TPKT uses the particular TCP port 3389 for RDP applications. The RDP traffic presents an on-demand

[2]Messages are captured at a test access point (TAP) device in the Ethernet link and recorded by WireShark. The basic structure of Ethernet packets is presented in Appendix A.

transmission pattern, i.e., RDP messages are transmitted in the link only when the HMI content needs to be updated in the host's remote desktop application. For example, if the remote desktop in the host is minimized, no RDP messages are transmitted in this case. The traffic load also largely depends on dynamics of the displayed desktop content, such as how fast it changes and the proportion of dynamic areas compared to the whole view. In the testbed, the active HMI session exhibits an intermittent and light traffic pattern with an average throughput less than 100 kbps.[3]

5.1.2 Supervisor-CNC Machines Interactions

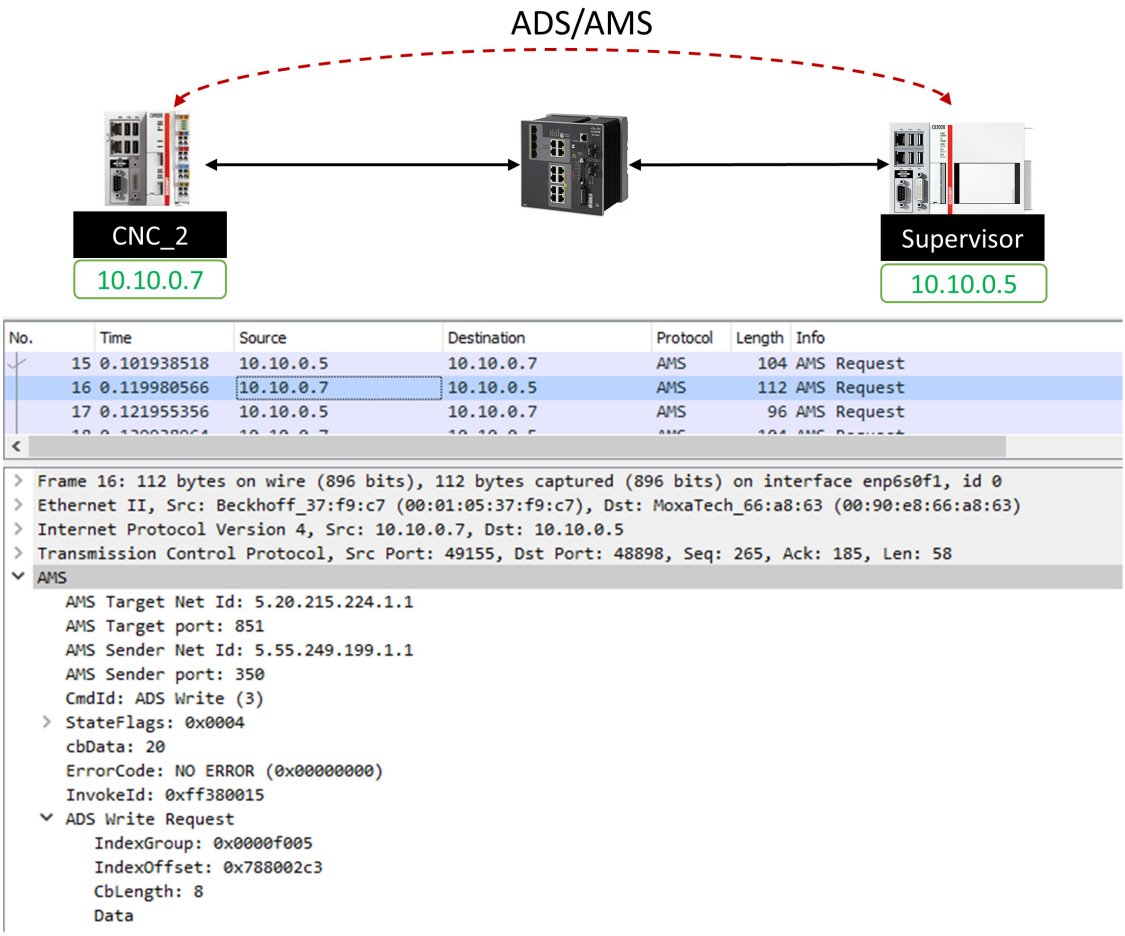


Fig. 19. The supervisor-CNC Communications and packet samples captured by WireShark

[3]The throughput was calculated during normal operations when the remote HMI main view stayed on the host desktop and was shown in full screen . In some cases, e.g., when the remote desktop switches between different HMI views or between HMI and the other Windows views, it may cause a larger instant RDP traffic up to 1 Mbps.

In the coordination process as discussed in Section 4.5, the supervisor needs to maintain continuous communications with CNC machines for tracking production efficiency and device health. Various message types are identified in multiple concurrent active sessions in the links. First, the supervisor routinely receives the status update from the CNC machine so that it can estimate the load at each single job stop, which is reflected in its scheduling decision. Second, the supervisor also routinely updates the work-cell status to field machines so that individual production modules have the real-time status of their partners that are used to coordinate their action. Furthermore, there are state-based on-demand communications, such as the inspection request-response conversions in the links.

In the testbed, the data exchanged between the supervisor (i.e., a Beckhoff PLC) and CNC machine controllers (i.e., also Beckhoff PLCs) are formatted as Automation Device Specification (ADS) commands, which are defined in the proprietary communication protocol for Beckhoff's TwinCAT devices known as Automation Message Specification (AMS). ADS/AMS is a medium-independent protocol. As shown in Fig. 19, ADS/AMS messages are carried in transmission control protocol/Internet protocol (TCP/IP) packets. The PLC runtime uses the reserved TCP port 851 as the identifier. Details about ADS/AMS are presented in Appendix A. The PLC uses the EtherCAT protocol to manage internal data flows with its terminal modules, such as motion control and I/O modules, which is outside the scope of this testbed design.

5.1.3 Supervisor-Robots Interactions

Similar to communications in the supervisor-CNC link, the supervisor also exchanges information with the UR3 robots for status update and job instructions, as shown in Fig. 20. Modbus is used in the links from the supervisor to UR3 control boxes of OPT and INS, respectively. Modbus allows data exchange between heterogeneous industrial appliances through the shared registers. In Modbus link, the supervisor serves as the Modbus server, which maintains the shared data in its memory. Through Modbus queries, the client can visit the remote data at particular register addresses. In the testbed, UR3 robots are the clients who send the write/read commands to the supervisor for updating robot status and obtaining instructions for the following moves. Modbus messages are identified by the TCP port 502. Basic information of Modbus communications is also provided in Appendix A.

5.1.4 Robotic Subsystem Communications

Robots are usually equipped with peripherals to better perform in different tasks. In this testbed, every UR3 robot has one gripper attached to its end effector along with a F/T sensor. The UR3 control box communicates with peripherals through TCP/IP socket communications. In the testbed, the F/T compute box is connected to the UR3 controller in the same subsystem switch as shown in Fig. 21. Every 8 ms, the controller updates the compute box once with UR3 system status which is formatted in a TCP package containing 139 values in a 1108 byte payload. TCP messages are sent through the UR3's TCP port for

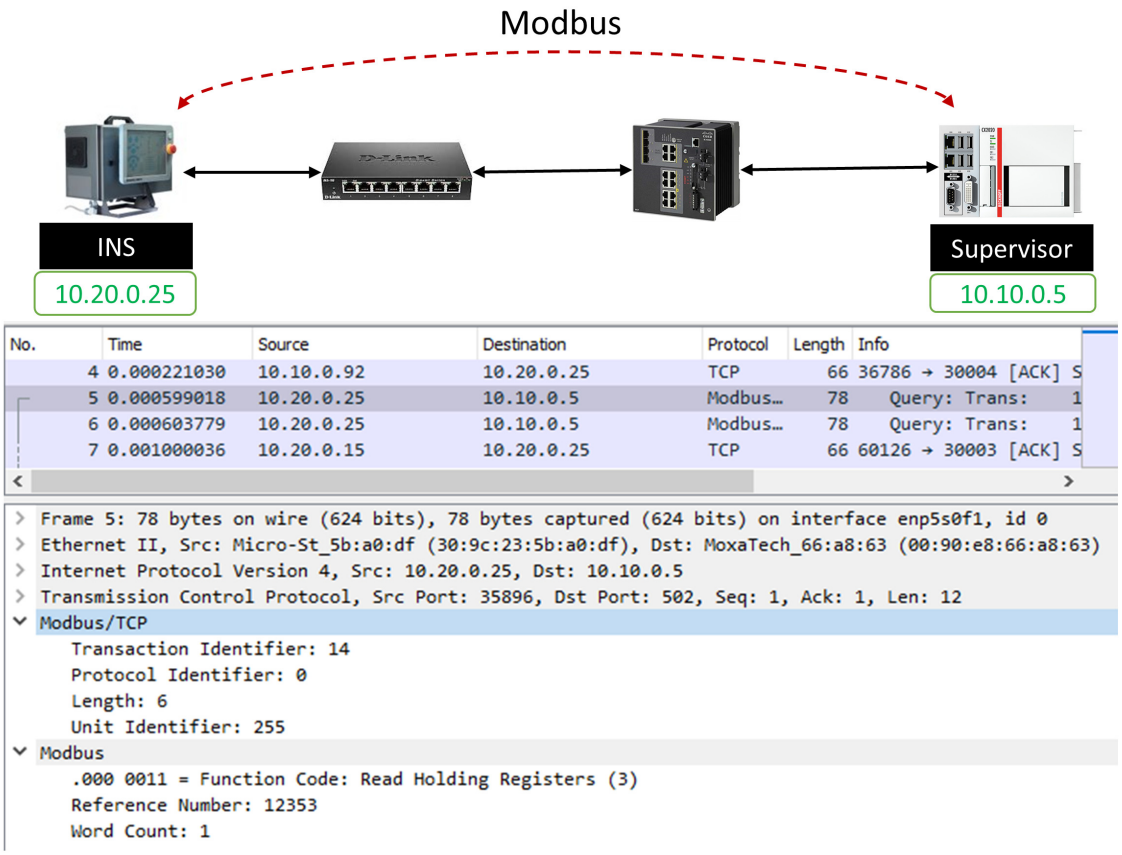


Fig. 20. The supervisor-robot communications and packet samples captured by WireShark

real-time traffic (30003). The F/T compute box software also updates the controller with the real-time F/T sensor values at 125 Hz.[4]

5.1.5 A Summary of Communication Traffic

Table 2 summarizes the emulated data flows in the testbed.

5.2 Networking Architecture

In support of work-cell communications, the testbed first implements fully wired connections between production modules. Ethernet enables inter-node communications in a wired medium. First, each production module in our testbed has one or more on board RJ-45 slots along with built-in Ethernet-based local area network (LAN) adapters (see Appendix D for network interfaces of individual devices). Second, the identified communication messages

[4]Current F/T firmware only supports an error-free connection, such as in an Ethernet link, with the control box. Missed controller updates will result in the link disconnection, which calls a system exception and pauses the robot operation. It requires a manual reset of the compute box.

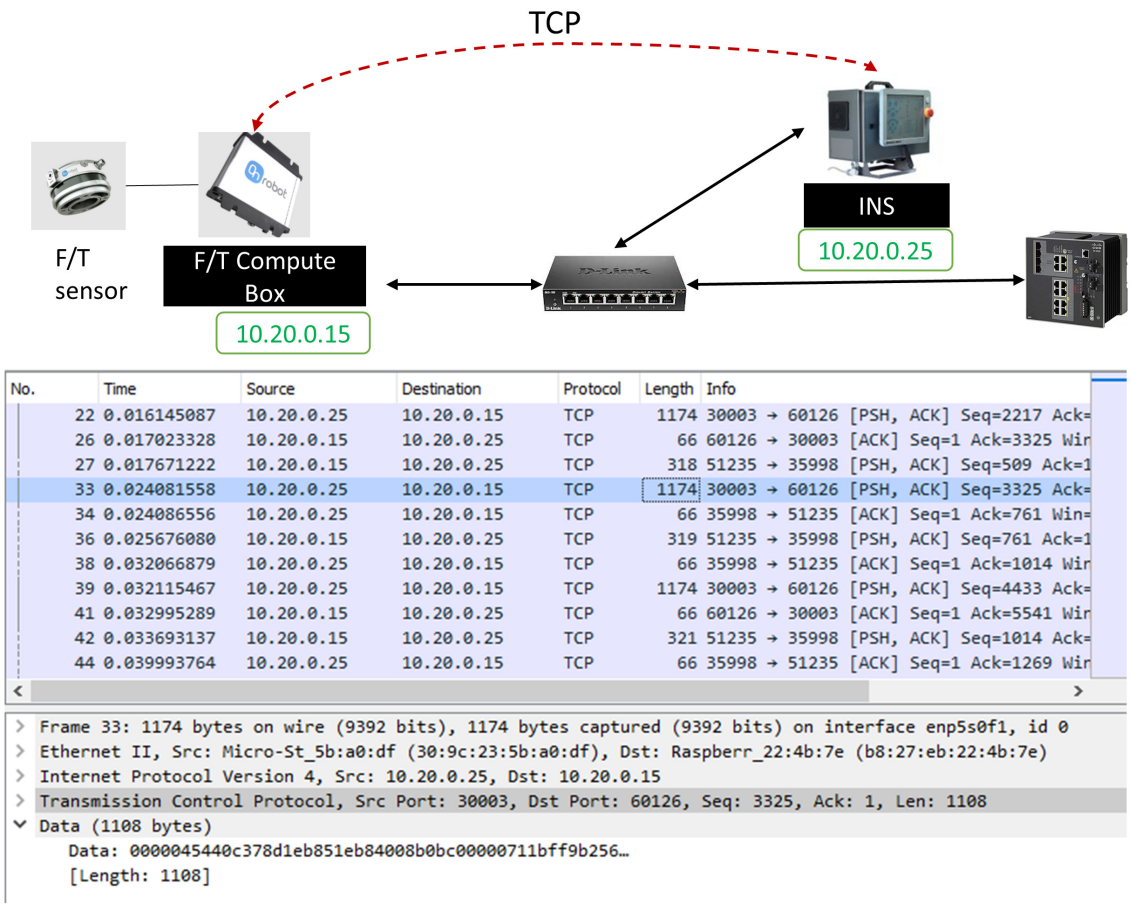


Fig. 21. Communications between the robot controller and the F/T sensor with packet samples captured by WireShark

in Section 5.1 are originally designed for the carrier frame structure in terms of TCP/IP-Ethernet packets (see Appendix A for packet frame structures).

Fig. 22 illustrates the wired networking architecture. The network backbone is formed by a bundle of two Cisco IE-4000 industrial Ethernet switches. Each IE-4000 switch has 12 gigabit Ethernet ports that provide managed Layer 2 switching functions as well as advanced features such as the support of time synchronization (e.g., NTP and PTPv2) and time sensitive networking (TSN). One gigabit port in each switch is reserved for bridging switches in trunk mode. Therefore, there are in total 22 gigabit ports available for work-cell communications and the other testbed communication functions.

All production modules, which are also testbed network nodes, use one of their on-board Ethernet ports for work-cell production communications. The supervisor PLC and CNC PLCs are directly plugged into the Cisco switch bundle through Ethernet cables. The UR3 robots have their own sub-network switches to interconnect parts in each robotic subsystem. Two D-Link unmanaged switches are deployed for robot sub-networks, one for

Table 2. Specifications of sampled data flows between work-cell components

Link	Data	Update Rate	Size * (Bytes)	Protocol
Supervisor -CNC	Status report	1 Hz - 100 Hz	10s	ADS
	Safety	100+ Hz	10s	ADS
	Inspection request/response	On-demand	10s	ADS
PLC -Peripheral	Motion control	1000 Hz	A few	ADS
Supervisor -Robot	Actuation	1 Hz - 50 Hz	A few	Modbus
	Safety	125 Hz	A few	Modbus
Robot -Peripheral	6 axis F/T sensor	100 Hz - 500 Hz	100s - 1000	TCP/IP
Supervisor -External	HMI	10 Hz - 50 Hz	100s - 1000s	ADS
	IoT	>1 Hz	10s - 100s	MQTT

* *Note*: The size is referred to as the application data size.

each. Each D-Link switch has 8 gigabit Ethernet ports and is linked with the backbone.

The testbed uses IPv4 addresses to manage communication devices. The gateway is 10.10.0.1 and the subnet mask is 255.0.0.0. For individual appliances that have direct connections to the backbone, they are assigned with IP addresses in 10.10.0.x. For robot sub-networks, INS subsystem has IP addresses in 10.20.0.x while OPT components have IP addresses in 10.30.0.x. Specific IP addresses are illustrated in the network diagrams of Appendix E.

Additional network connections are also planned in the testbed that provide measurement data links and other complimentary features, such as time synchronization services. These supportive connections will be introduced in the measurement framework of Section 6.

5.3 Wireless Extension

Wireless links are added into the testbed network to verify the capability of wireless technologies in support of mission-critical industrial communications. Upon the first release of this report, WLAN connections have been implemented in the testbed where one or more selected Ethernet connections are replaced. Since industrial appliances are originally designed for Ethernet networks, there is no built-in wireless adapter available. We introduce Intel's Next Unit of Computing (NUC) devices as Ethernet-WLAN adapters that trans-

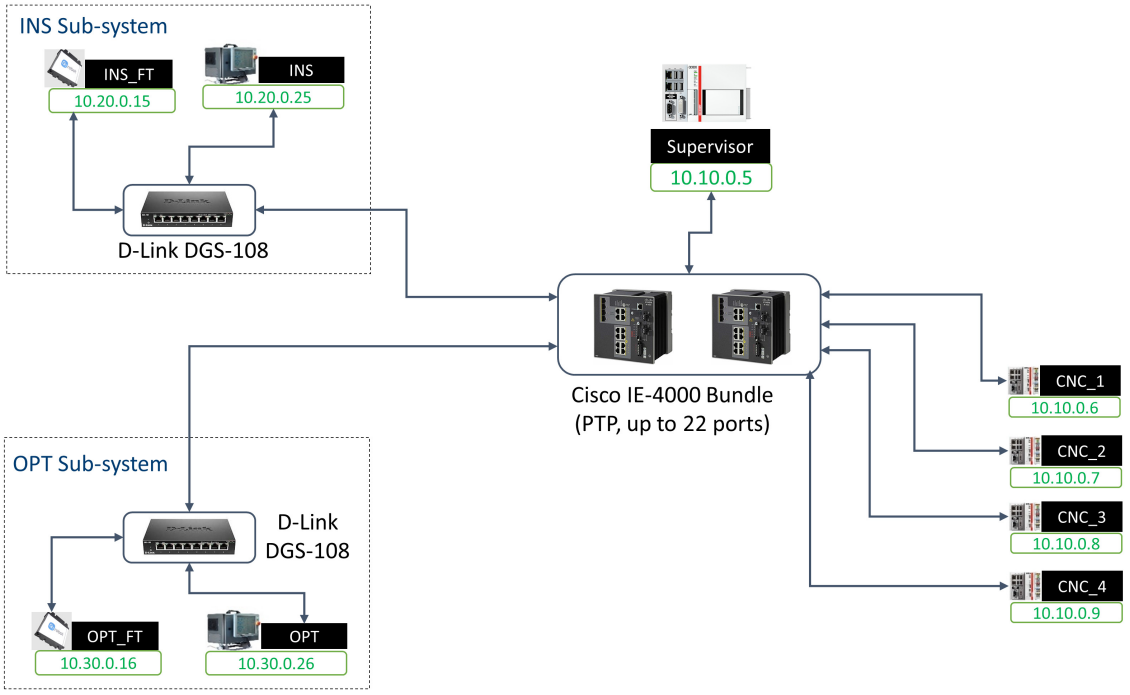


Fig. 22. Work-cell network architecture using full wired Ethernet connections

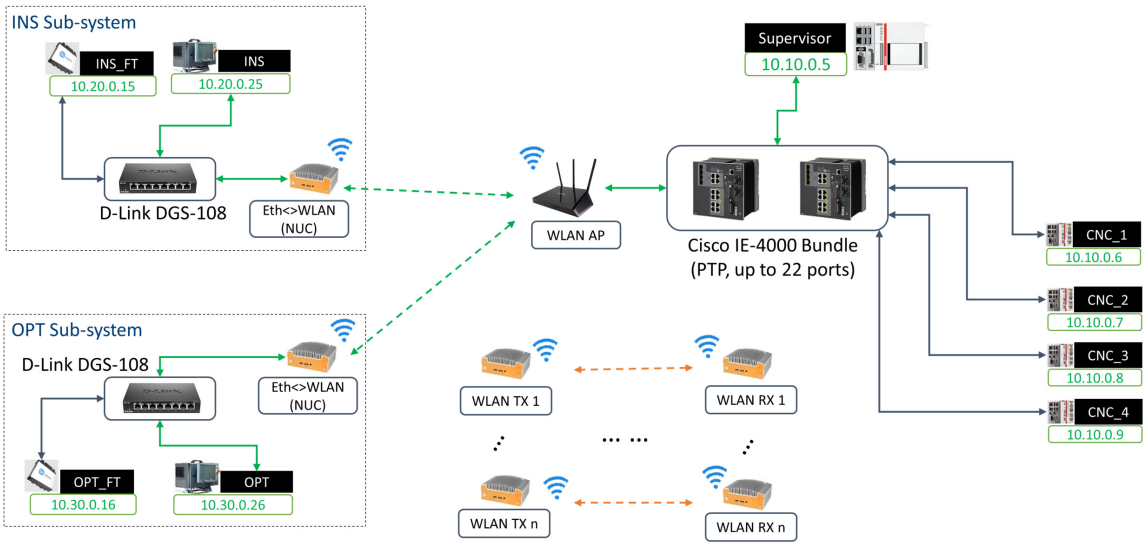


Fig. 23. Wireless extension to the work-cell network architecture

late industrial messages between Ethernet frames and WLAN packets. Fig. 23 illustrates the introduced wireless links in the work-cell network. In this use case, UR3 subsystems communicate with the rest of work-cell through WLAN links. Specifically, each UR3 sub-network is equipped with an Intel NUC serving as the Ethernet-WLAN adapter. A Netgear

AC1900 WLAN router (R7000) is used as the WLAN access point (AP) and connected to the switch bundle. WLAN connections between NUCs and the AP enable UR3 controllers and the supervisor to coordinate the OPT and INS operations, respectively. Wireless transmissions comply with IEEE 802.11b/g/n protocols. WLAN radio modules negotiate to determine the working mode based on the instant channel quality. One or more background traffic generators can also be deployed in the working WLAN channels that emulate the interference in real industrial sites. These traffic generators are performed by extra pairs of Intel NUCs in the testbed that run iPerf scripts to create user datagram protocol (UDP) unicast packets as background stress data.

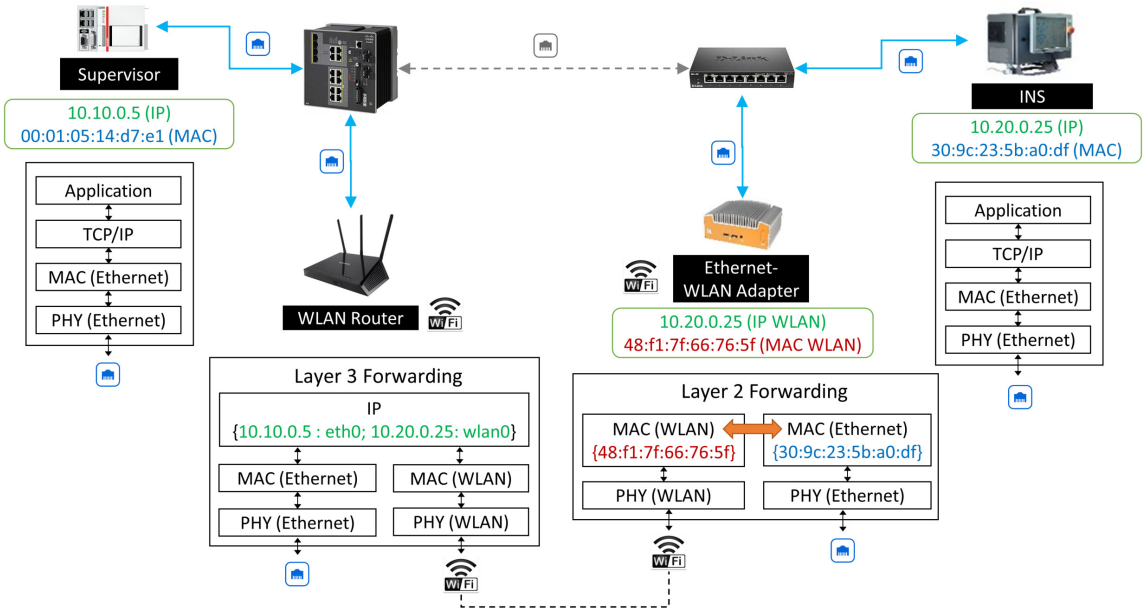


Fig. 24. Illustration of Layer 2 forwarding through the Ethernet-WLAN adapter

NUC by its nature is a barebone computer running Linux operating system (OS), e.g., Ubuntu 14.04 used in the testbed. Equipped with network interfaces including Ethernet and IEEE 802.11b/g/n adapters, NUCs can be programmed to work in different modes to meet the service requirements of different roles in the testbed. Generally, NUCs are designed to work in the following roles: Ethernet-WLAN adapter, wireless sniffer, and background traffic generator/sink.

When working as an Ethernet-WLAN adapter, it converts the received Ethernet frames to WLAN packets, and vice versa, which is called the Layer 2 forwarding. Compared to the Layer 3 forwarding at a router which handles IP packets, the NUC first resumes the in-bound medium access control (MAC) protocol data unit (PDU) (i.e., the IP packet(s) encapsulated in a MAC frame), repacks it into another MAC frame format, and sends in the new medium as the out-bound data. Fig. 24 illustrates how the NUC serves as the UR3 inspector's Ethernet-WLAN adapter in its communication with the supervisor.

Since such conversion is performed by a NUC program in software, it is a type of "soft" forwarding approach. The entire forwarding process is transparent to end users as the end-to-end connection is IP-based, whose information is intact in the lower layer process. The only change in the network is that the NUC adapter is treated as an alias of the served host along the path. In the case as shown in Fig. 24, all en route devices (i.e., the supervisor, the Cisco switch bundle, and the WLAN router) address the NUC as the counterpart to the supervisor in supervisor-INS communications. For instance, the supervisor uses the NUC's MAC address, not the INS's one, in the MAC destination field of a message destined for INS. Accordingly, the NUC needs to replace it by the INS's MAC address in the forwarded packet.

As a wireless sniffer, the NUC uses the **iw** command, i.e., a new command-line interface (CLI) configuration utility for wireless devices, to create a virtual interface, associate it to the physical WLAN interface, and set it in the monitor mode. Once specifying the working WLAN channel for the virtual interface, we can run TShark or Tcpdump to sniff all WLAN packets in the channel.

In experiment scenarios considering concurrent WLAN transmissions as the co-channel interference, extra NUCs can be paired up as background traffic sources and sinks using Iperf, an Internet network test tool, to generate managed background traffic [23].

6. Measurement Framework

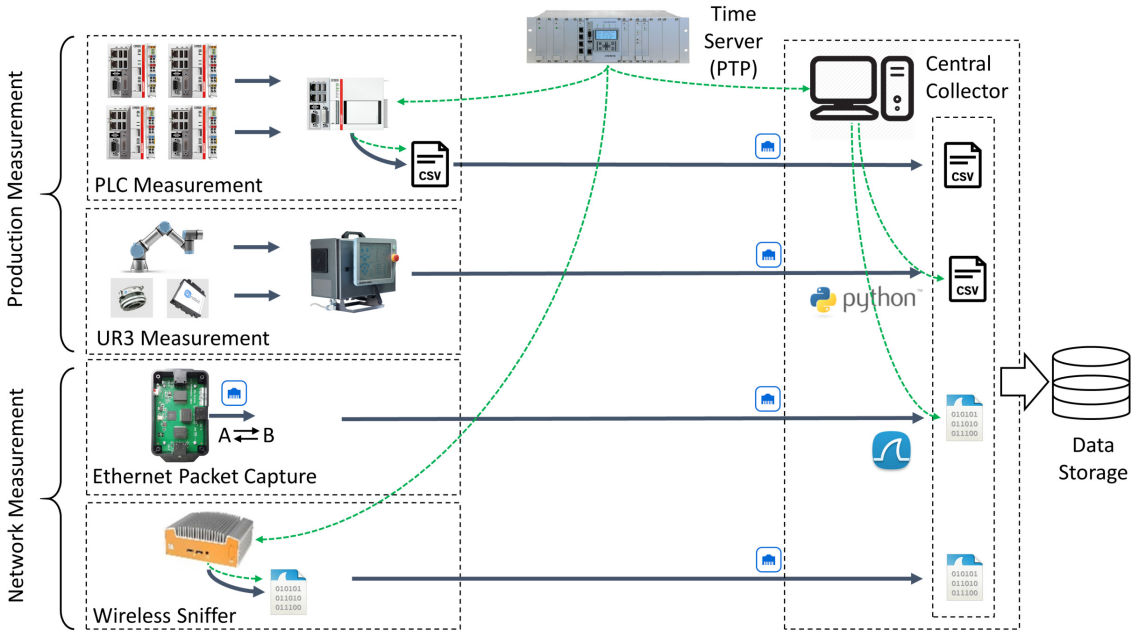


Fig. 25. The testbed measurement framework

Measurements are taken in various points of interest to obtain data showing features of

the emulated production processes and network operations as well as their relationships. We have introduced types of measurement apparatus in the testbed and developed a centralized framework parallel to work-cell operations. As shown in Fig. 25, the developed measurement framework provides complimentary functions and features that enhance data collection in the testbed and feed further analysis with trustworthy data. In this section, we will systematically review individual measurement components. Specifically, data types in measurements are enumerated and their properties are explained; tools implemented in the framework for obtaining corresponding measurement data are introduced whose collection flows are shown as blue arrows in Fig. 25; time synchronization over measurement probes is also discussed that unifies timed observations from distributed data records, which is presented as green dashed arrows in the figure.

6.1 Measurement Data Collection

We have identified data classes by their sources, i.e., production modules and network components, which are labeled as operational and network-related, respectively. Accordingly, measurement tools and collection procedures are developed for individual data sets.

Testbed data are managed in the central collector which is a desktop computer running Ubuntu 18.04 and equipped with one 1 terabyte (TB) solid-state drive (SSD) for data storage, as shown in Fig. 25. The collector has ten gigabit Ethernet ports that can be connected to testbed apparatus for data collection in the work-cell. Details about the full network diagram are illustrated in Appendix E. Local measurement results, such as the ones collected at the supervisor PLC and wireless sniffer, are transferred to the collector after individual experiments. Meanwhile, real-time measurement data, such as the UR3 real-time data exchange (RTDE) data and network packet captures, are directly recorded at the central collector who assigns independent measurement links and runtime threads for individual collection programs. Data is saved with timestamps for post data analysis, and data files are indexed with the unique name regarding each experiment.

6.1.1 Operational Data Collection

Operational data refer to the recorded process variables and control commands saved during the operation of production modules. Data from the supervisor, CNC machines, and UR3 robots belong to this class. For individual modules, state-related variables are the primary data source, which contains rich behavioral information for modeling the digital copy of state machines and analyzing the operation performance in different situations.

As discussed in Section 4, the supervisor watches over the entire work-cell's production. Therefore, the first measurement point is set in the supervisor as shown in Fig. 25. Data from the supervisor indicate how the scheduler manipulates individual machinery operations based on its local view of work-cell status. All GVL variables used by PLC programs can be routinely reported to the collector as measurement data. Currently, we focus on testbed data that may affect the scheduler's decision and coordination performance.

Since status information is reported in a routine manner through the network, network connections may affect the supervisor's response to any state change in the production process, which is the main analysis goal in this work.

Optional data at individual CNC machines and robots can also be subscribed that reflect the local view of machine status. By comparing the initiation time of a production command and its real actuation time, it implies the control delay which consists of the processing time as well as the delay in communication links, which provides the insight of network impact on production performance. For CNC machines, since emulated CNC controllers are also implemented in Beckhoff's PLCs which share the same program architecture as the supervisor PLC, their GVL variables can be subscribed in measurements. Details of measuring PLC data are explained in Appendix B.1. PLC data are recorded at 125 Hz and stored locally in the PLC's hard drive. Each round of experiment data is saved in a single CSV file.

For UR3 robots, internal program variables and URCaps' information are accessible through the UR's RTDE interface. RTDE outputs a comprehensive list of robot status variables such as position, velocity, acceleration, current, voltage, and other critical metrics. For example, in trajectory planning, both actual and target setpoints of position variables are collectible. Since RTDE data showcases internal operation status within robot subsystems, which are closely related with collaborative robot applications, such as work-cell safety, we set another set of measurement points at UR3s. RTDE measurement is configurable through an extensible markup language (XML) file where it is flexible to subscribe or remove variables in the measurement output. We refer the interested readers to check the supported robot state parameters in the RTDE guide [24]. RTDE provides periodic updates at 125 Hz through the TCP port 30004. The central collector retrieves real-time RTDE readings through TCP socket connections with control boxes of OPT and INS, respectively. Python scripts are developed here for RTDE data collection outputting results in CSV files.

6.1.2 Network Traffic Captures

Network traffic data are communication messages that are captured in links between work-cell appliances. Data types have been identified in the earlier Section 5.1. In the testbed, we use network test access point (TAP) devices to passively capture Ethernet messages transmitted in the monitored links. As shown in Fig. 26, a simple TAP is usually equipped with three Ethernet ports. Port A and B are internally bridged, which enables through traffic to pass so that the monitored link does not break. During packet forwarding between A and B, the TAP makes one copy of each through packet and send it to the third port (A $\rightleftarrows$ B) which is connected to a data collector. If the TAP has more than one A $\rightleftarrows$ B port on board, it generates multiple copies of through traffic, one for each.

Fig. 27 indicates locations of TAP devices that we have employed to monitor work-cell links carrying critical production messages. The central collector (not shown in this figure) reaches out to each deployed TAP device through a direct gigabit Ethernet link so that

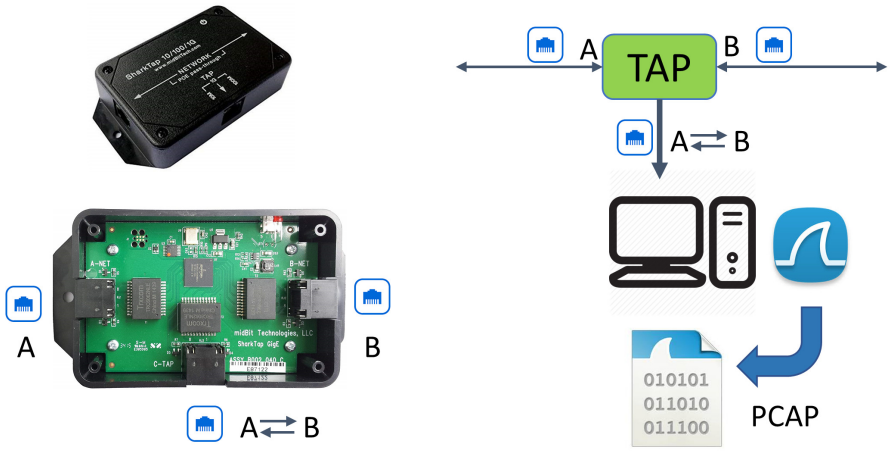


Fig. 26. Network traffic test access point (TAP) device

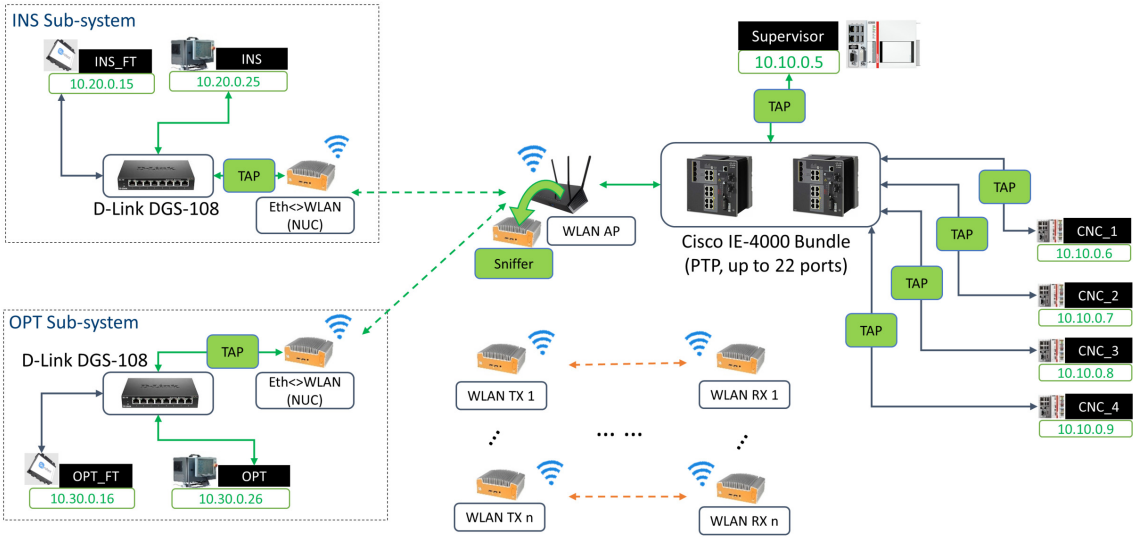


Fig. 27. Network traffic probes in the testbed network

the real-time traffic collection does not affect network load of work-cell communications or introduce bandwidth competition between TAP links. We refer the interested readers to check more details of link connections in the full network diagram of Appendix E. The collector collects link data by using the network protocol analyzer, e.g., TShark/WireShark. Each TAP link is monitored by an independent TShark thread whose data are saved in a separate packet capture (PCAP) file.

TAP devices only serve to capture Ethernet packets in wired connections. To further obtain wireless transmission quality information, we have also employed wireless sniffing tools in the measurement. Fig. 27 illustrates a wireless data collection case where an Intel NUC is deployed as a wireless sniffer co-located with the AP. Working in the monitor

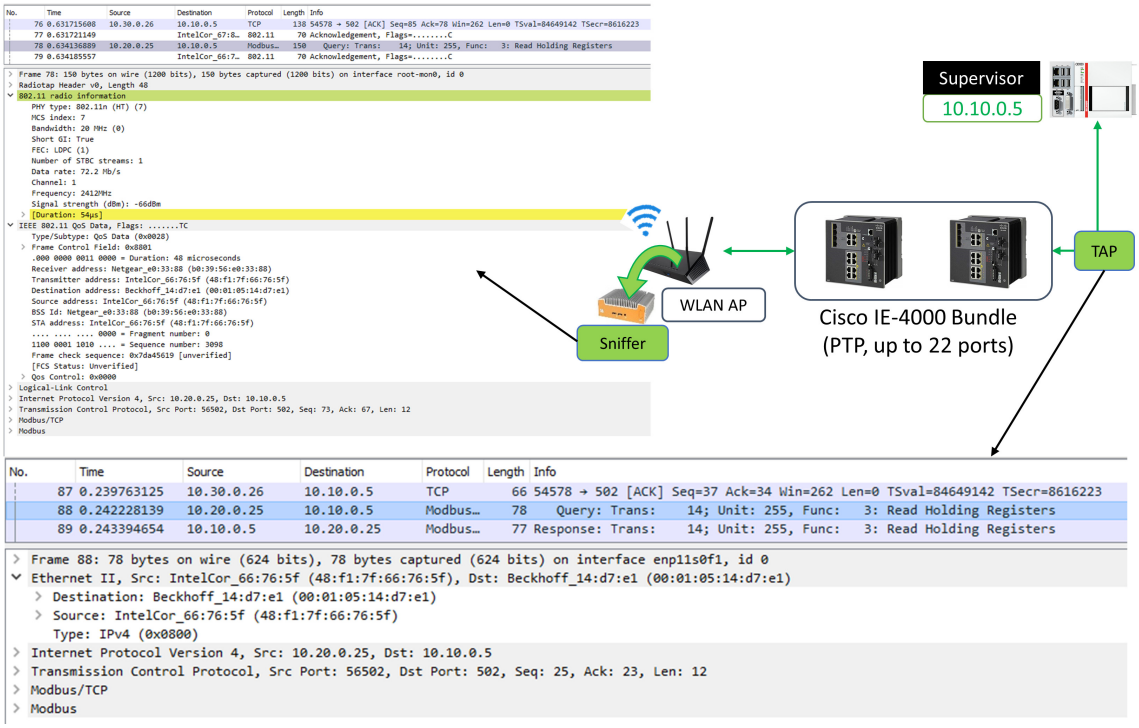


Fig. 28. Packet header samples from TAP and wireless sniffer captures

mode, the NUC's built-in wireless adapter is tuned to the AP's working channel to passively record any wireless packets sent to it. Only packet captures addressed to the co-located AP are saved in the PCAP file. The sniffer shares a common antenna with the AP so that the captured WLAN header information can indicate the channel quality experienced at the AP's receiver in UR3-AP links. Fig. 28 illustrates the captured copies of a Modbus request message recorded at the wireless sniffer (the AP is co-located with) and the TAP next to the supervisor. The wireless packet capture provides detailed information regarding transmission settings and channel quality.

6.2 Time Synchronization

Data timestamps are rendered at individual machines, which are synchronized with a central time server. Their formats along with the enabling tools are summarized in Table 3.

Since measurements are performed at distributed nodes and saved in multiple formats, timing information becomes a critical reference, in some cases being the sole one, to rebuild the process timeline in post data analysis. Therefore, the accuracy of distributed clocks at measurement devices determines the data quality because they are in charge of rendering record timestamps. We have introduced time synchronization approaches to guarantee the trustworthy timing information contained in measurement data.

Table 3. Measurement data timestamp settings via PTP

Data	Recorder	Time Writer	Reference Time	Format	Time Zone, Daylight Sav.	Ordinary clock (OC)	PTPv2
PLC (CSV)	CX2020	Meas. POU	PLC Task Clock	YYYY-mm-dd -HH:MM:SS .xxxxxx (μs)	Yes, Yes	EL6688	L3 E2E
RTDE (CSV)	Central Collector	Python	Linux Sys. Time	mm/dd/YYYY, HH:MM:SS, xxxxxx (μs)	Yes, Yes	NIC onboard	LinuxPTP
TAP (PCAP)	Central Collector	TShark	Linux Sys. Time	Epoch time (sec, ns-resol.)	N/A, N/A	NIC onboard	LinuxPTP
Wireless Sniffer (PCAP)	NUC	TShark	Linux Sys. Time	Epoch time (sec, ns-resol.)	N/A, N/A	NIC onboard	LinuxPTP

6.2.1 Precision Time Protocol

As discussed earlier in Section 6.1, physical events are captured periodically with recording steps of milliseconds or longer. For those data, sub-millisecond level clock synchronization mechanisms, e.g., using the NTP, would be adequate to capture state transitions in system behavioral studies. However, since our study in the testbed is anticipated to investigate network events which may last only a few microseconds, it is necessary to have a more accurate time resolution in data to correctly identify various measured events in time with clear boundaries of their start and stop moments. Such a precise time measurement is essential to unveil the correlation between different data sets, especially the ones rendered from heterogeneous observers. The PTP provides the sub-microsecond level accuracy for the networked distributed clocks and widely used in timing sensitive networks. In the testbed, we have introduced PTP to synchronize measurement devices' clocks.

The testbed deploys the IEEE 1588-2008 (PTPv2) standard which defines PTP, a method to precisely synchronize computers and devices over a local area network (LAN). A Meinberg M900 PTP time server plays as the grand master clock. Individual data collectors, as introduced in Section 6.1, work as ordinary clocks (OC) in PTP which routinely update their local time based on synchronization signaling sent by the grand master. OC and the grand master are connected through the Cisco IE-4000 switch bundle which also supports PTP. These switches serve as boundary clocks (BC) which relay synchronization signaling messages as well as playing the role of intermediate time reference in the end-to-end path. These PTP appliances are connected in a tree-like topology with the root at the grand master.

Different PTP-capable hardware/software tools are deployed at OC for rendering timestamps of measurement data. For collectors running Linux, e.g., the central collector and wireless sniffer, the PTP-capable Ethernet network interface cards (NIC) serve as OC at local machines. A software tool, LinuxPTP, is used to perform the process of tuning the local system time to the reference time of the grand master [25, Chapter 23 Configuring

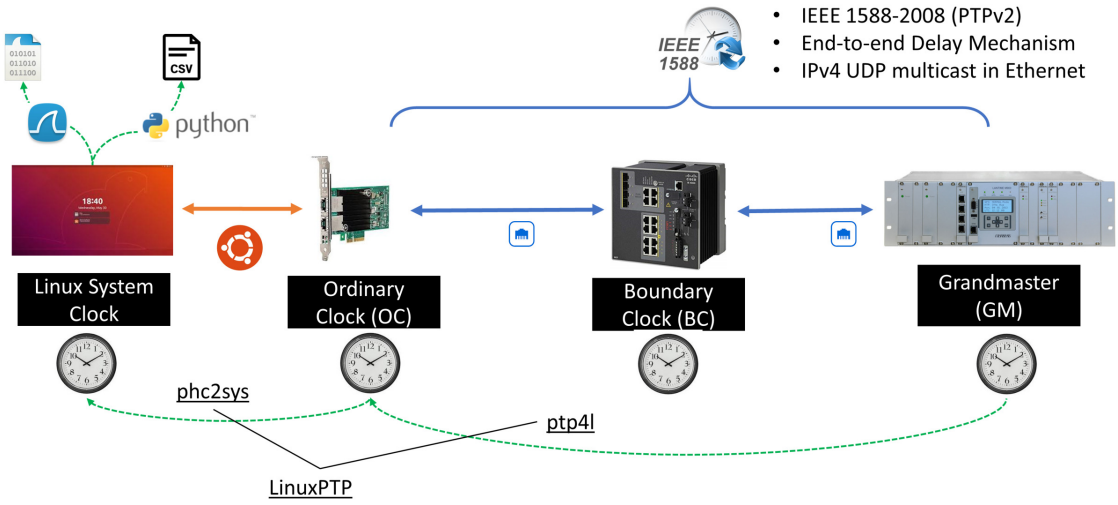


Fig. 29. PTP time synchronization using LinuxPTP

PTP using PTP4L]. Fig. 29 illustrates the synchronization process at the central controller, which runs a Ubuntu 18.04 OS. LinuxPTP contains two programs: *ptp4l* and *phc2sys*. The *ptp4l* program is in charge of synchronizing the PTP hardware clock (PHC) on the NIC, i.e., the OC, to the grand master using an end-to-end delay mechanism to measure the clock offset. Once OC is synchronized, the *phc2sys* program will then adjust the Linux system clock according to the PHC. The synchronization is performed in a repeated manner to ensure the clock offset is always managed. At the central collector, Python scripts use the *datetime* library to obtain the system time at the moment of writing RTDE data. For network captures from TAP devices, the recording time is stamped given the system clock reading when TShark detects the received packet in the sniffing port.

For PLC, an add-on hardware module, Beckhoff's EL6688 IEEE 1588 module, is used to enable the PTP synchronization in a similar way. The EL6688 module serves as the OC in the synchronization path to the grand master while providing the reference time to the connected TwinCAT devices, such as the PLC, through internal EtherCAT connections. The OC routinely measures the time offset between its clock and the PLC's system clock, which is also called the task clock (TC), and outputs the offset reading to the PLC program. At the moment when a PLC record is cached in the program, its timestamp is calculated by adjusting the current TC time based on the latest offset to the OC. A detailed explanation of PLC's PTP synchronization is presented in Appendix B.2.

6.2.2 Timestamp Formats

Besides implementing PTP to ensure the clock accuracy, we have further surveyed various collection tools in the testbed to verify specific timestamp formats of individual data so that the timing information can be correctly used in the analysis. The testbed is located in NIST's Gaithersburg campus where Eastern Time is used. Given specific dates in a year,

Eastern Standard Time (EST) and Eastern Daylight Time (EDT) are used appropriately. EST(UTC-5:00) is 5 hours behind the Coordinated Universal Time (UTC); EDT (UTC-4:00) has a four hour delay from UTC. The Meinberg M900 time server utilizes the UTC time which is neutral to time zones.

Table 3 illustrates timestamp formats used in individual data files. Linux machines, i.e., the central collector and wireless sniffer NUC, maintain their system clocks that are synchronized with the global time server through PTP. For network traffic data, TShark records in the epoch time which indicates the number of seconds that have elapsed since January 1, 1970 of the UTC time. Therefore, timing information in network traffic data is immune to the setting of time zone and daylight saving time. However, the local time settings should be verified before treating production measurement data. RTDE data contain timestamps made by Python scripts at the central collector. Using the *datetime* library, the code outputs the local time in Eastern Time with awareness of daylight saving time. The Linux kernel takes care of the time zone setting, e.g., enabling/disabling daylight saving time, once it is configured in the system. On the contrary, for PLC records, such settings need to be manually verified and transferred to the PLC program as configuration variables, including leap seconds, time zone, and daylight saving time. In the *Meas_Main* module, the PLC program calls the time function to obtain the timestamp shown in local time. Appendix B.2 provides further information about rendering PTP synced timestamps in PLC data.

7. Data Management

A GDB is used to manage data collected from testbed measurements of both network traffic and physical operations. GDBs, as NoSQL databases, don't specify any predefined data structure or rules to enforce a fixed structure. Instead, GDBs treat individual data records as distributed node entities in a random graph and identify relationships as database components that link different nodes together. This feature allows GDBs to catch varying states and dynamics in a complex system and gradually improve data management along with better understanding of the system. In this section, we introduce graph components developed for the testbed and the data processing flow that transforms measurement results to graph entities.

7.1 Graph Database

The testbed GDB is built in the Neo4j desktop application. Neo4j is a native graph database which manages data and their relationships in an index-free environment. Its official query language, Cypher, is declarative, i.e., focused on what data to retrieve from the database instead of how to form queries to obtain the needed results.

7.1.1 Graph Data Model

In a GDB, the data model, which can be roughly analog to the "schema" of relational databases, illustrates how data records are organized and stored in a graph. However, un-

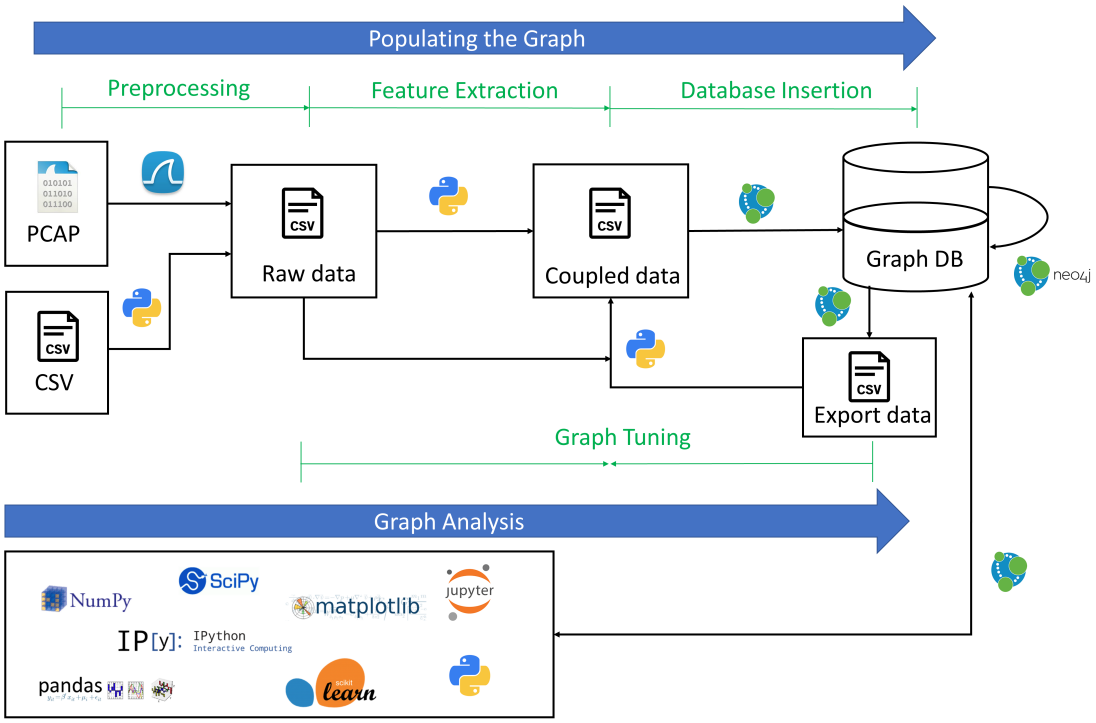


Fig. 30. Data processing flow from factory work-cell to database

like a fixed schema, the data model of GDBs has more flexibility of depicting diverse data types, content, and connections between different entities whose structure and property profile can update and evolve with more data and/or better observation. A data model contains different node types with specific properties in the graph and various relationships between them. We first identify requirements of such a data model and build a graph containing nodes and relationships that mainly exhibit information regarding networked industrial devices in a factory work-cell [6]. We further populate the previous work-cell data graph by introducing additional node types characterizing physical actions that are newly captured. Accordingly, we update the relationships, such as associating individual quality-of-service (QoS) report data made by the wireless sniffer, with the packets captured at the collocated receiver. The updated data model provides a comprehensive view of production operations, information flows, and wireless channel variations in the testbed, which facilitates further analysis work.

As shown in Fig. 31, an example is illustrated here that summarizes nodes, relationships, and their key properties used in the GDB.

7.1.2 Building Blocks

The graph is comprised of two classes of building blocks: nodes and relationships. The former represents testbed objects, e.g., work-cell modules and network components, and

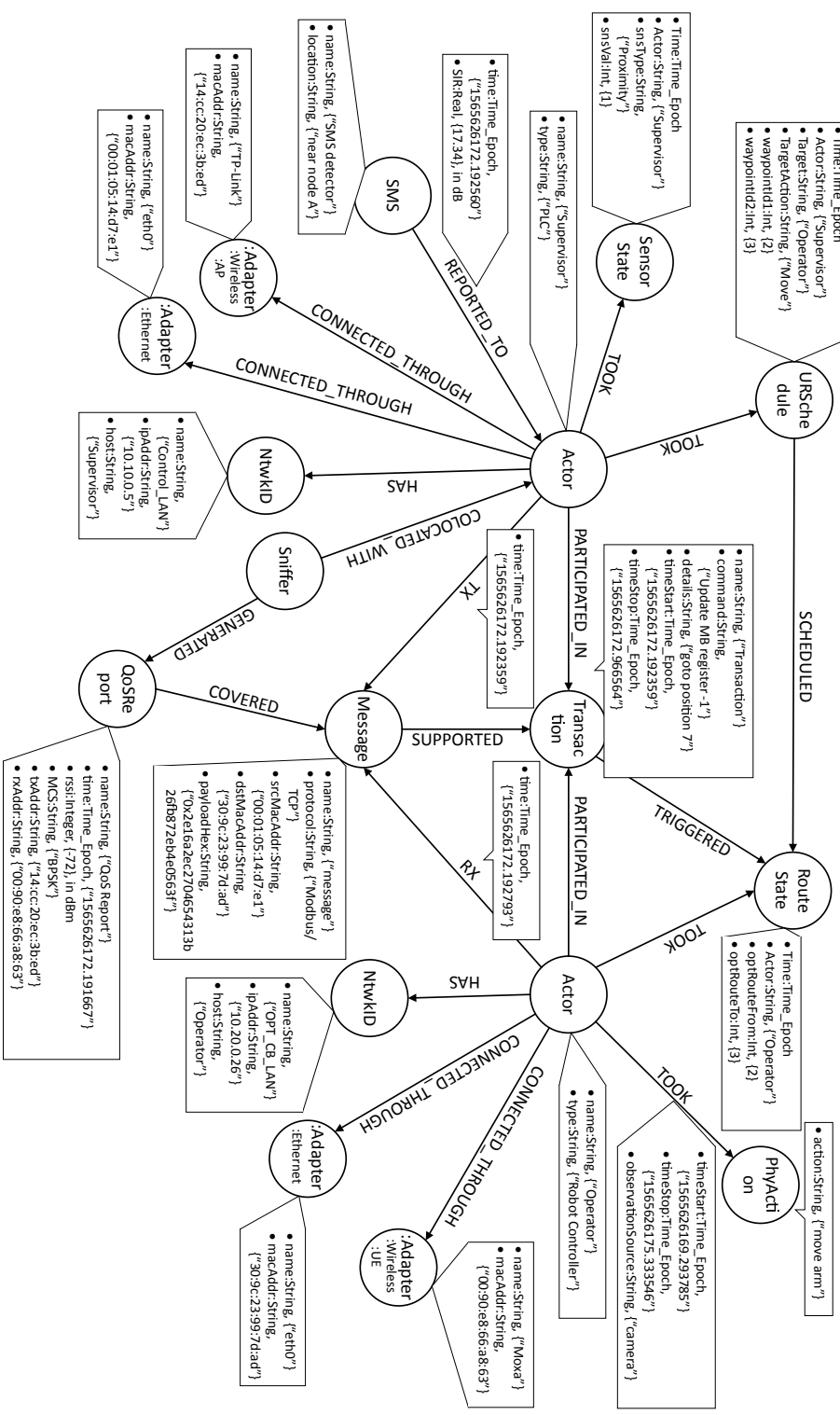


Fig. 31. The data model of the graph database used for each operational run of the testbed.

events that are captured in the production process and network operations, e.g., machine states and communication messages; the latter is denoted as edges that link two node instances and record their relationships in terms of actions or properties. We will enumerate nodes and relationships defined for the GDB in the following parts. A thorough review of their properties can be found in Appendix C.1.

Node Design

To effectively depict testbed operations in the measurement, we define a series of node types in the graph. For different purposes, nodes can be used to identify testbed components, their states, and messages that create varying snapshots of the testbed for further analysis. They can be found in two main classes depending on what type of objects the node represents.

The class of *static* nodes covers testbed setup profiles, which contain testbed components, network interfaces, and their settings. These entities are normally predetermined or collected in the initialization of each measurement. They usually remain constant in each round of measurements.

Actor A physical component within the factory work-cell such as a robot, PLC, or other networked item.

NtwkID A network address item for an actor such as an Internet Protocol (IP) address.

SMS A spectrum monitoring service (SMS) observes and records significant spectral events within the work-cell and may report those events to actors within the work-cell.

Sniffer Measurement device that records all transmissions conducted over the wireless medium and includes the wireless header information for each wireless transmission detected.

Adapter Device that serves to connect an actor to a network (adapters are divided into sub-categories depending on the type of interface to a network).

Adapter:Ethernet A subcategory of adapter representing an Ethernet interface.

Adapter:Wireless A subcategory of adapter representing a wireless interface.

Adapter:Wireless:AP A subcategory of adapter representing a wireless access point interface.

Adapter:Wireless:UE A subcategory of adapter representing a wireless user equipment interface.

The class of *dynamic* nodes in the graph captures various system events such as machine status reports, network traffic, and information flows in the testbed. These nodes are dynamically added into the graph whose quantities and properties are determined by the real-time data in the measurement.

Transaction A complete information exchange between two or more actors (multiple actors may participate in a transaction).

Message A network transmission event that occurs between two actors (messages are essentially packet transmissions captured at the transport layer; multiple messages support a transaction).

QoSReport Quality of service report of a message (not all messages have a QoS report).

Physical Action (PhyAction) A physical occurrence within the factory work-cell associated with Actors through multiple time-based relationships.

PhyAction:URSchedule A subcategory of PhyAction representing a schedule decision made by the supervisor PLC for a robot

PhyAction:SensorState A subcategory of PhyAction representing a real-time reading of the proximity sensor state in a CNC machine

PhyAction:RouteState A subcategory of PhyAction representing a real-time reading of the action route in a robot

Graph Relationships

A relationship in the graph denotes an action taken to associate two nodes, either homogeneous or heterogeneous ones, which shows their connections in the topology, timeline, or affiliation. We identify the following relationships in the testbed.

PARTICIPATED_IN Actors will participate in transactions. A transaction exists for each logical set of messages between actors, such as the setting of a Modbus register or the sending of a command to a robot. Therefore, actors will participate in many transactions, and multiple actors may participate in a single transaction.

SUPPORTED Messages (i.e., packets between actors) are associated with transactions through the SUPPORTED relationship. Depending on the protocol and the quality of the channel, a single transaction could have one or many messages connected through this relationship.

TX/RX An actor may either transmit (TX) or receive (RX) a message. Both the TX and RX relationships contain a timestamp in the format of an epoch time which is a floating point number in seconds since January 1, 1970, with a resolution of microseconds.

TOOK When an actor performs a physical action, a TOOK relationship is created between the actor and the physical action node. This relationship contains start and stop time properties as well as the source of the observation such as a networked camera.

REPORTED_TO An SMS may be a passive or active listener within a work-cell. When an SMS operates as an active listener, spectral reports from the SMS may be sent to an actor such that the actor can respond intelligently to the spectral event. Reports from an SMS to an actor are captured within this relationship.

COVERED A wireless sniffer keeps monitoring the working wireless channel(s) and extracts the real-time link QoS information from the sniffed wireless packets, such as the received signal strength indicator (RSSI). A COVERED relationship links the QoSReport node with the concurrent Message node received at the same spot. Not all Message nodes have such a relationship with QoSReport which depends on the availability of the sniffer collocated with the receiver and any wireless sniffer data reported during the transmission.

Other relationships shown in Fig. 31 but not explained above are considered self-explanatory.

Closer Examination

The graph data model is designed in a way where nodes and relationships are centered around Actors. Actors have dual roles in the work-cell operations. In the factory system, Actors participate in the production operations. In the example of Fig. 31, two Actor nodes are presented. In this case, Actor "Supervisor" is the supervisory controller, and Actor "Operator" is a robot arm. The Supervisor schedules the production, collects the other Actors' states, and hosts supportive services, such as SMS. The Operator follows the instructions of the Supervisor and moves parts between work stations. Meanwhile, Actors also act as communication nodes which exchange messages between each other through various network interfaces. In Fig. 31, Actors participate in a transaction, which, in this example, is a Modbus/TCP exchange. The transaction itself is associated with one or more messages (i.e., packets). Each message associated with a transaction manifests itself as a node in the graph. Multiple message nodes will exist for each transaction. Additionally, QoS reports may be associated with each actor node through a collocated sniffer node.

Dynamic event nodes in the measurement, i.e., physical actions, network messages, information transactions, and QoSReport records, have timestamps representing "measurement time" of the recorded events. Once a new event occurs, a proper relationship would be added between the actor and the physical/network event node. All timestamps are accurately synchronized to the grand-master clock.

7.2 Data Importing Pipeline

A multi-stage workflow is deployed to feed the graph with instances of nodes, relationships, and their properties that are extracted from measurement data, as shown in Fig. 30. In the data set, network data are captured from distributed probes in the selected links and stored in PCAP files, while operational data that come from different PLC and robot controllers are stored in comma separated value (CSV) files. The whole data process contains

four steps including preprocessing, feature extraction, database insertion, and graph tuning. Such conversion from raw measurement data to a ready-to-go graph has been automated by scripts deployed in the central collector, which maintains data repositories and deploys the Neo4j desktop application. Functions and operation features in individual steps are discussed next.

7.2.1 Data Preprocessing

Data preprocessing is the beginning of the whole process. In this stage, measurement data from various sources are verified, cleaned, and formatted in a way to facilitate the following processing and interpretation. As fore-mentioned, measurement data contains records collected from heterogeneous modules/devices in the testbed, which may utilize different data types, sampling rates, time and metric resolution, as well as file formats. For example, different machines may store records' timing information in various timestamp formats depending on local clock preference. In the process, we have unified the data set's time format, i.e., using the time epoch with the microsecond resolution. Besides, we have also deployed packet filters to reduce the size of output data for the following steps, e.g., removing uncorrelated packet captures. For example, when treating wireless sniffer data in experiments with interference links, we have managed to reduce the size of sniffer data, which was originally of gigabytes, to only a few megabytes with only signaling handshakes in the studied links.

7.2.2 Feature Extraction

Feature extraction refers to the process of extracting relevant information from measurement data to prepare the data for insertion into the database. Nodes and relationships are defined by a set of features that share common views. We have developed Bash and Python scripts that pick the desired features to produce CSV files that are ready for insertion into the Neo4j database. In this step, a Bash script runs TShark to extract fields of protocol headers in packet captures and save the field information into CSV files. Each row in these CSV files will lead to one Message node instance being created which is one packet copy at the sender or receiver. A Python script was also used to detect physical action changes and label those state switching moments, which were associated with communication messages.

7.2.3 Graph Insertion

We load the prepared data into the Neo4j GDB using bulk importing, which can create multiple new nodes and/or relationships by reading a CSV file once. Neo4j uses Cypher to construct GDB queries that import data. As the output of feature extraction, each CSV line contains the information for creating one new node entity and/or pairing two nodes in a new relationship. Properties of new entities can be assigned explicitly by the column values of records or inferred from predetermined rules such as some fixed combination of

nodes and edges in the graph. Multiple types of nodes can be created from the same data file using one common node template in which each node type has its own subgroup of properties. For example, Modbus and ADS packets use the same Message node structure in our graph to manage the common transmission information such as IP addresses and TCP session identification. Meanwhile, each of these Messages maintains its own application layer header information in the node properties, e.g., Modbus register addresses and ADS function codes.

7.2.4 Graph Tuning

Graph tuning refers to any additional modification in the graph after CSV data has been imported. This step treats a number of cases where the unexplored raw data is coupled with the imported one to extend/improve the graph. First, in the additive insertion cases, i.e., when new data is to be added, this step links the newly added nodes to the existing ones and creates necessary relationships between them. For example, time series data often uses this method to link consecutive event nodes in the recorded process. Second, there are cases when further insights of system properties can be obtained by querying existing database records which lead to new nodes and relationships. For example, Transaction nodes are built upon Message nodes who participate in the same application transactions; Message nodes themselves are also the summary of packet data, i.e., packet copies at link transceivers. Third, it can serve as one input to the feature extraction step by providing existing graph information for purposes such as coupling data records. For example, coupling QoS reports and Messages in their observation windows used to be an extremely time-consuming process. On one hand, each Message raw data, i.e., the transmitter or receiver copy, contains only half of the transmission time window information. On the other hand, the Cypher query takes a long time to find all eligible relationships as Neo4j would generate a huge Cartesian product when treating the large sample set. We solved this issue by obtaining qualified Message nodes and feeding them into feature extraction where a more efficient Python script finds all Message-QoS Report pairs, and later, presents them in the graph as new COVERED relationships.

The above four steps can perform multiple iterations to treat data and refine the graph according to the data complexity and requirements. To further demonstrate such a process in treating experiment data, we enumerate various techniques used in the testbed and show them in Appendix C.

8. Data Analysis & Graph Exploration

8.1 Graph Database Schema

Once the Neo4j database is populated with testbed data, we can apply Cypher queries to explore the constructed graph and extract information for assessing the work-cell performance and visualizing selected network and operational events within the work-cell. More advance data science tools, such as Python's Pandas and Scikit-learn packages, can also

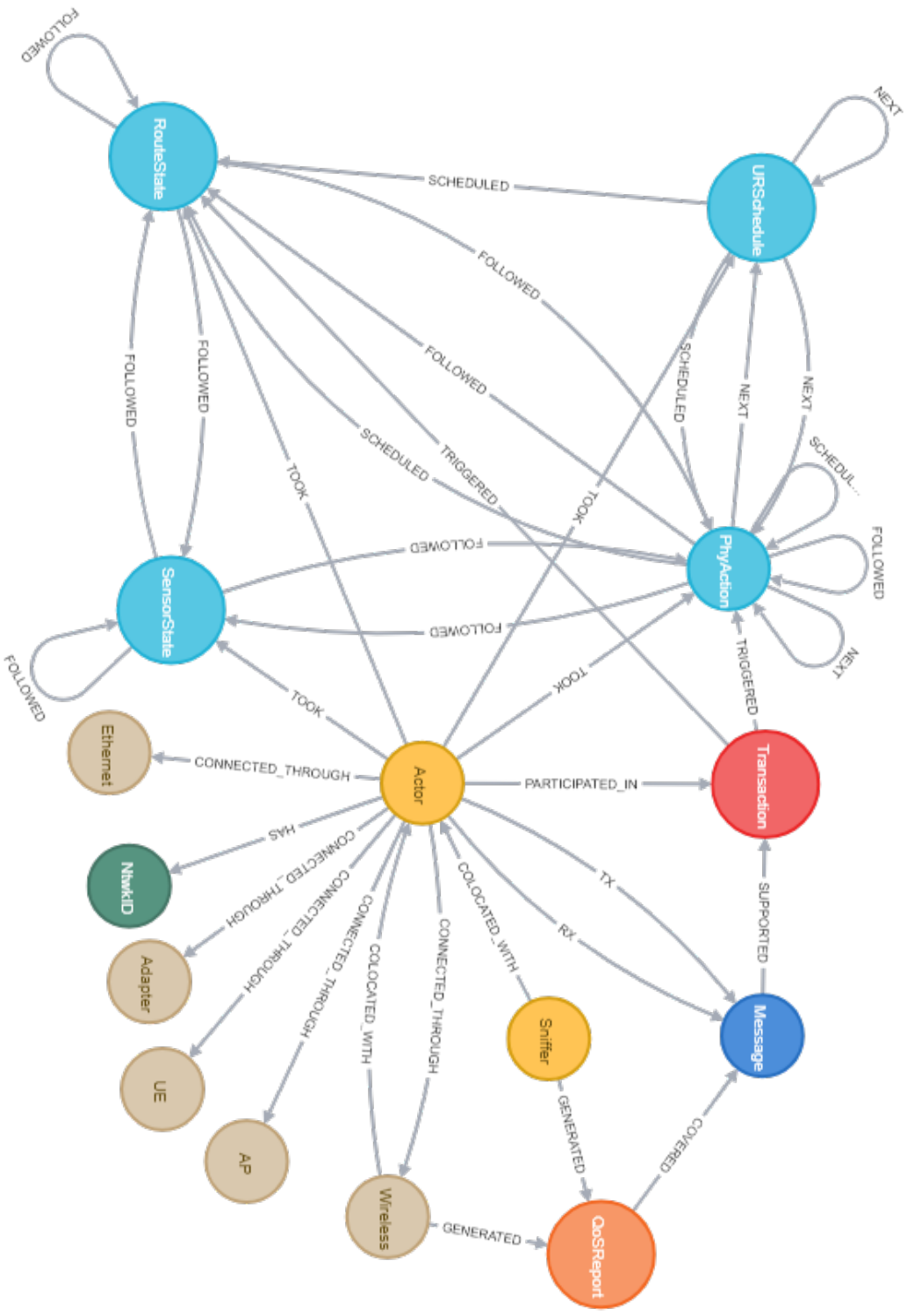


Fig. 32. Realized schema of the graph database fully populated after capturing network and operational data from the industrial wireless testbed.

be introduced using Python-Neo4j interfaces, details of which can be referred to in Appendix C.2.4.

Fig. 32 shows the graph schema as a quick check of the Neo4j graph elements (as proposed in Fig. 31) using the query command, call $db.schema.visualization()$.

8.2 Experiment Configurations

Section 4 and 5 have introduced general configurations on the production modules and network components, respectively. Additional configuration settings are provided here for testbed experiments of evaluating the work-cell performance under different network conditions.

During each run of an experimental scenario, the production of 20 parts was emulated. Each part went through a directed tooling path, i.e., $< Queue\ IN > - < CNC1 > - < CNC2 > - < CNC3 > - < CNC4 > - < Queue\ OUT >$, which resulted in around 12 minutes of network activity.

We performed four different experimental cases with respect to the communications network settings, namely,

1. Wired baseline: All links are connected using Ethernet cables to act as a benchmark for performance comparison (see Fig. 22 for the wired architecture and Fig. 64 for the measurement diagram);

2. Wireless scenarios: Two wireless links are used to connect the robot controllers and the wireless AP that is connected to all the other actors in the testbed (see Fig. 23 for the wireless architecture and Fig. 65 for the measurement diagram). The wireless nodes are Intel NUC devices with IEEE 802.11b/g/n interfaces.

   (a) Wireless baseline: The robots' traffic is the only traffic transferred over the wireless network;

   (b) Wireless communications with a single interference pair: One additional pair of Intel NUC devices forms an interference link in which 2500 packets per second (pps) wireless traffic is generated in the same working WLAN channel as the robot links. The interference link packets have the size of 1000 Bytes.

   (c) Wireless communications with two interference pairs: Two additional pairs of Intel NUC devices form two interference links each of which generates 1250 packets per second (pps) wireless traffic in the same working WLAN channel as the robot links. The interference link packets have the size of 1000 Bytes.

As shown in the measurement diagrams of Fig. 64 and Fig. 65, we collected network capture data from TAP devices in the testbed. In wireless scenarios, we also collected wireless packet captures at the wireless sniffer to describe the WLAN channel condition. Data from the supervisor was also collected which included the system states and the supervisory commands. In addition, data from the robots was used to describe the physical actions taken.

8.3 Preliminary Results

In this subsection, we use the extracted data from the GDB to study the impact of the wireless communications on the physical action performance. The results have been partially reported in our earlier publications [5], [6], and [7]. The data is being prepared for public access, e.g., the data presented in [6] is available in [26]. We focus our analysis on the URSchedule and RouteState progress over time where URSchedule is the dynamic node to represent a physical action decision at the supervisor and RouteState is the dynamic node to represent a physical action command received by one of the robots where the command parameters are stored at the robot registers. Note here that the transaction between a robot controller and the supervisor is initiated by a request message from the robot controller and terminated by correctly receiving a response message from the supervisor to the robot controller as well.

The supervisor makes decisions based on available information about the testbed. Once it makes a decision, it is reflected on the value of the URSchedule. We define the supervisor processing time as the time from the instant the decision is taken to the instant when the wireless transaction is initiated to request a new physical action and it is denoted by T_{Sup}. Then, the transaction latency is the total time spent by all the wireless packets corresponding to an action such that it is the time between the instant, at which the wireless transaction is initiated by the robot controller to require a new action until the data arrives from the supervisor at the intended robot controller. The wireless transaction latency is denoted by T_{W}. The robot processing time is the time between the instant the wireless data is received by the robot controller to the instant when the required action is updated in the RouteState register indicating the physical action starts. The robot processing time is denoted by T_{Rob}. The total physical action time, which represents the time needed for a supervisor command to be reflected at the corresponding robot, is denoted by T_{Act} and evaluated through

$$T_{\text{Act}} = T_{\text{Sup}} + T_{\text{W}} + T_{\text{Rob}}. \tag{1}$$

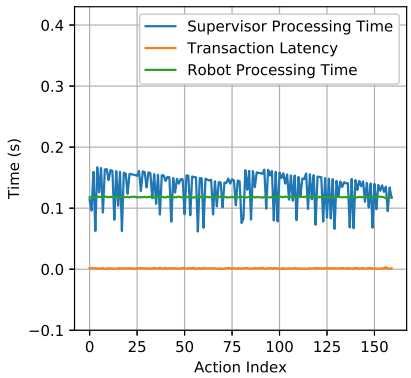


Fig. 33. Wired baseline physical action time

In Fig.33-37, we present the values of the three components of the total physical action time for each run of the testbed. The horizontal axis represents the action index for all the

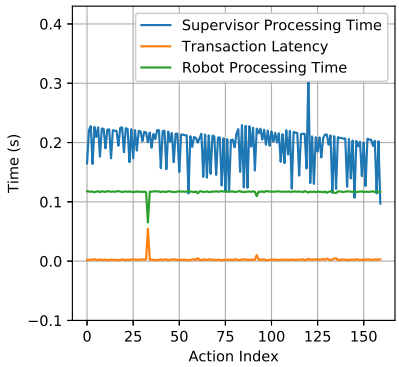


Fig. 34. Wireless baseline physical action time

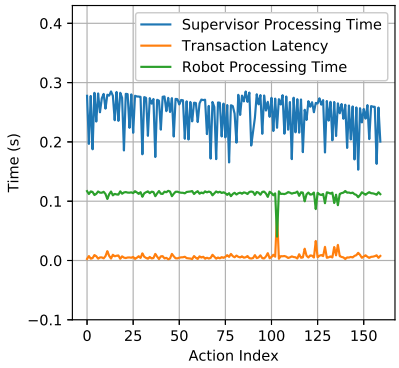


Fig. 35. Wireless with 2500 pps traffic physical action time (run 1)

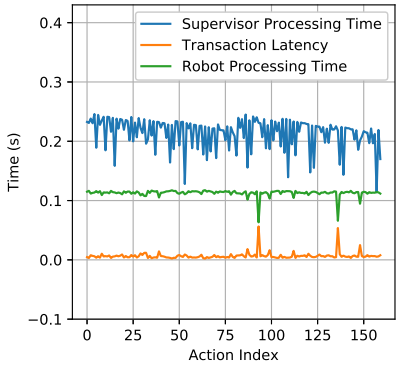


Fig. 36. Wireless with 2500 pps traffic physical action time (run 2)

operator and inspector actions, while the corresponding time components are shown in the vertical figure axis.

We present the normalized histograms of the transaction latency and the total physical action time in Figs. 38 and 39, respectively. Interested readers can refer to [7] for further discussions on the key factors to these metrics.

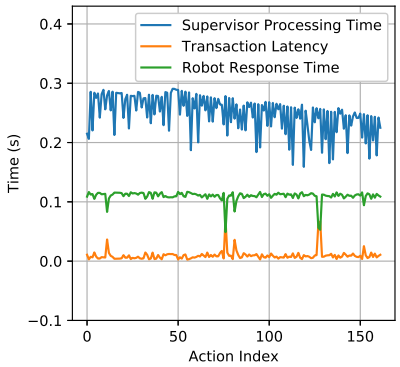
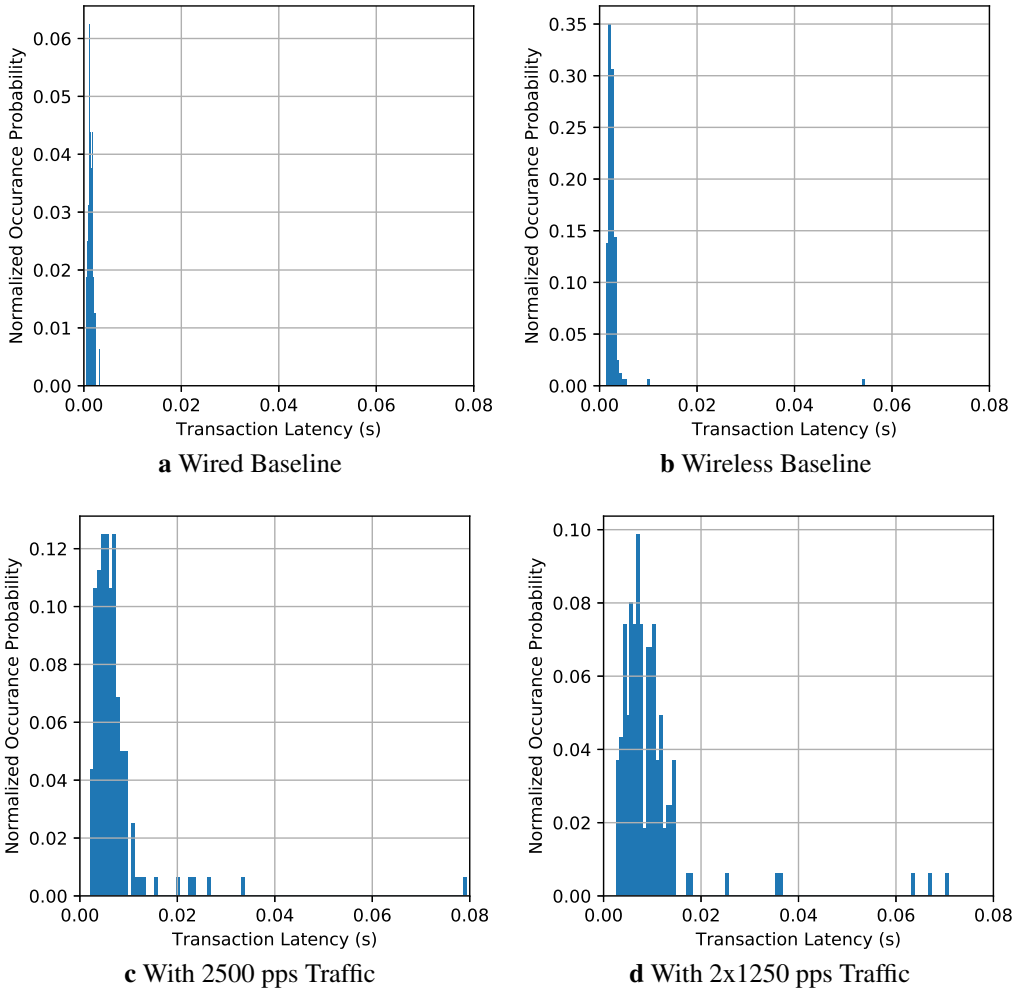


Fig. 37. Wireless with 2x1250 packets/s traffic physical action time



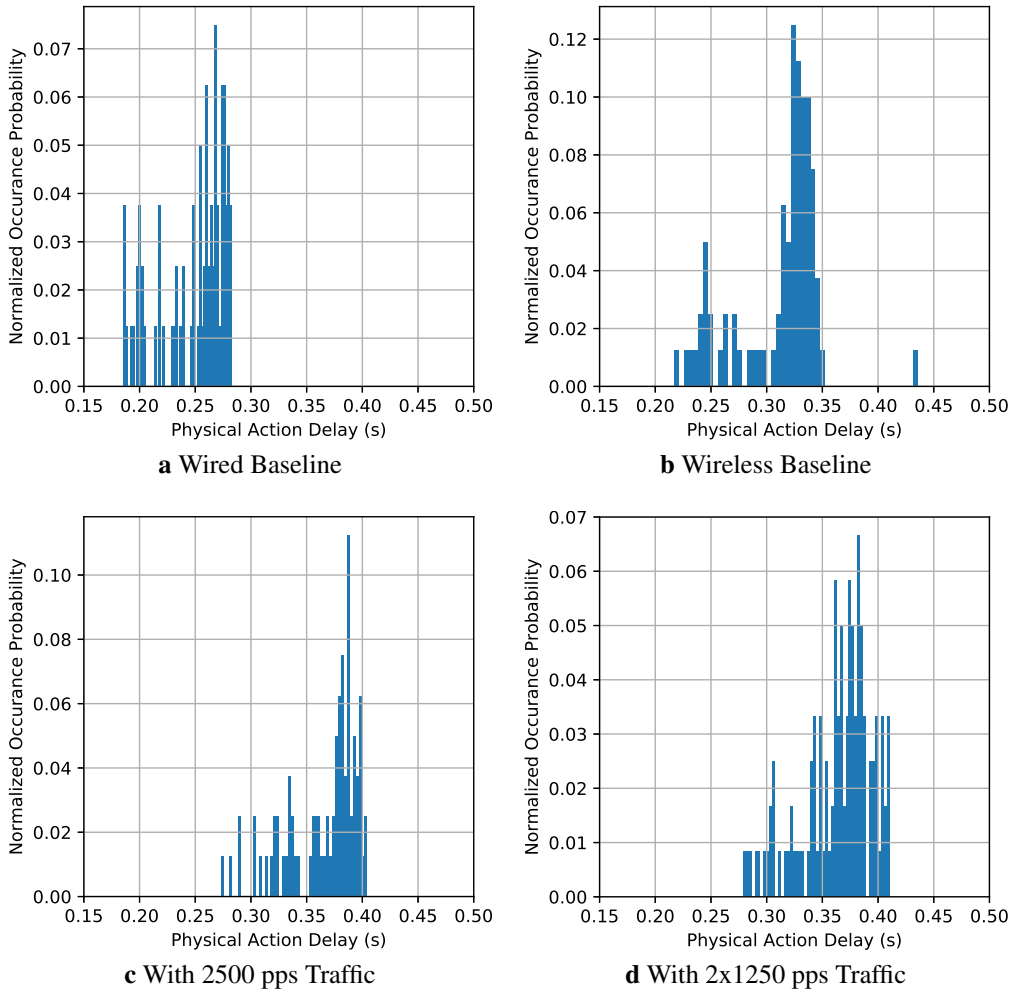
a Wired Baseline

b Wireless Baseline

c With 2500 pps Traffic

d With 2x1250 pps Traffic

Fig. 38. Histograms of Transaction Latency for Various Experimental Scenarios



a Wired Baseline

b Wireless Baseline

c With 2500 pps Traffic

d With 2x1250 pps Traffic

Fig. 39. Histograms of Physical Action Time for Various Experimental Scenarios

9. Conclusion

We have presented a new testbed design idea in the study of wireless techniques in support of industrial operations. The proposed collaborative robot work-cell scenario represents a wide spectrum of manufacturing activities with data transmission needs to coordinate their work. The measurement approach and data processing pipeline based on the graph database provide flexibility in exploring data across the CPS domains. The testbed has been used in our research work in collaboration with industrial partners to validate and showcase the capability of recent WLAN products in serving industrial applications. More measurement events are planned for this testbed to verify emerging industrial wireless techniques, such as wireless TSN and the next generation WLAN solutions. Further progress will be reported in future releases.

Acknowledgments

The authors would like to thank Jing Geng at the University of Maryland for helping us to develop Bash scripts of using TShark in dissecting packet captures. We would like to also thank our industrial partners, Dave Cavalcanti and Susruth Sudhakaran from Intel Labs, for their contributions in providing the Intel NUC devices. We would like to thank Timothy Zimmerman, Ya-Shian Li-Baboud, CheeYee Tang, Frederick M. Proctor, and Keith Stouffer at NIST, for their continuous support and valuable suggestions on the testbed development.

References

[1] Liu Y, Kashef M, Lee KB, Benmohamed L, Candell R (2019) Wireless network design for emerging IIoT applications: Reference framework and use cases. *Proceedings of the IEEE* 107(6):1166–1192.

[2] Montgomery K, Candell R, Liu Y, Hany M (2019) Wireless User Requirements for the Factory Work-cell. The National Institute of Standards and Technology (NIST, Technical report. https://doi.org/10.6028/NIST.AMS.300-8. URL https://www.nist.gov/publications/wireless-user-requirements-factory-workcell

[3] Candell R, et al. (2018) Guide to industrial wireless systems deployments. National Institute of Standards and Technology Gaithersburg, MD, Technical report. https://doi.org/10.6028/NIST.AMS.300-4. URL http://nvlpubs.nist.gov/nistpubs/ams/NIST.AMS.300-4.pdf

[4] NIST (2018) Trustworthy systems, components, and data for smart manufacturing program, https://www.nist.gov/programs-projects/trustworthy-systems-components-and-data-smart-manufacturing-program. Accessed: 2020-09-30.

[5] Liu Y, Candell R, Kashef M, Montgomery K (2019) A collaborative work cell testbed for industrial wireless communications — the baseline design. *2019 IEEE 28th International Symposium on Industrial Electronics (ISIE)*, pp 1315–1321. https://doi.org/10.1109/ISIE.2019.8781524

[6] Candell R, Kashef M, Liu Y, Montgomery K, Foufou S (2020) A Graph Database Approach to Wireless IIoT Workcell Performance Evaluation. *2020 IEEE International Conference on Industrial Technology (ICIT)* (IEEE), pp 251–258. https://doi.org/10.1109/ICIT45562.2020.9067199. URL https://ieeexplore.ieee.org/document/9067199/

[7] Kashef M, Liu Y, Montgomery K, Candell R (2020) Wireless cyber-physical systems performance evaluation through a graph database approach. *Journal of Computing and Information Science in Engineering* :1–19,URL https://doi.org/10.1115/1.4048205.

[8] NIST (2018) Cybersecurity for smart manufacturing systems, https://www.nist.gov/

programs-projects/cybersecurity-smart-manufacturing-systems. Accessed: 2020-09-30.

[9] Candell R, et al. (2017) Industrial wireless systems radio propagation measurements. the National Institute of Standards and Technology (NIST, Technical report. https://doi.org/10.6028/nist.tn.1951

[10] Li H, et al. (2019) Design space exploration for wireless-integrated factory automation systems. *2019 15th IEEE International Workshop on Factory Communication Systems (WFCS)*, pp 1–8.

[11] Geng J, et al. (2020) Integrating field measurements into a model-based simulator for industrial communication networks. *2020 16th IEEE International Conference on Factory Communication Systems (WFCS)*, pp 1–8.

[12] Aminian B, Araujo J, Johansson M, Johansson KH GISOO: A virtual testbed for wireless cyber-physical systems. *IECON 2013 - 39th Annual Conference of the IEEE Industrial Electronics Society* (IEEE), pp 5588–5593. https://doi.org/10.1109/IECON.2013.6700049

[13] Jecan E, Pop C, Padrah Z, Ratiu O, Puschita E A dual-standard solution for industrial Wireless Sensor Network deployment: Experimental testbed and performance evaluation. *2018 14th IEEE International Workshop on Factory Communication Systems (WFCS)* (IEEE), pp 1–9. https://doi.org/10.1109/WFCS.2018.8402360

[14] Ding Y, et al. Experimental investigation of the packet loss rate of wireless industrial networks in real industrial environments. *2015 IEEE International Conference on Information and Automation* (IEEE), pp 1048–1053. https://doi.org/10.1109/ICInfA.2015.7279441

[15] Liu Q, et al. (2018) Design and Evaluation of a Real Time Physiological Signals Acquisition System Implemented in Multi-Operating Rooms for Anesthesia. *Journal of Medical Systems* 42(8):148. https://doi.org/10.1007/s10916-018-0999-1

[16] Fink J, Ribeiro A, Kumar V (2013) Robust Control of Mobility and Communications in Autonomous Robot Teams. *IEEE Access* 1:290–309. https://doi.org/10.1109/ACCESS.2013.2262013

[17] Liang W, et al. (2019) WIA-FA and Its Applications to Digital Factory: A Wireless Network Solution for Factory Automation. *Proceedings of the IEEE* 107(6):1053–1073. https://doi.org/10.1109/JPROC.2019.2897627

[18] Candell R (2015) A Research Framework for Industrial Wireless Deployments. *Proceedings of 2015 ISA Instrumentation Symposium* URL https://www.researchgate.net/publication/285396923_A_Research_Framework_for_Industrial_Wireless_Deployments.

[19] Liu Y, Candell R, Lee K, Moayeri N (2016) A simulation framework for industrial wireless networks and process control systems. *2016 IEEE World Conference on Factory Communication Systems (WFCS)* (IEEE), pp 1–11. https://doi.org/10.1109/WFCS.2016.7496495

[20] Technical Committee (2008) IEEE Std 1588-2008, IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems.

Society https://doi.org/10.1109/IEEESTD.2008.4579760

[21] Schick IC, Gershwin SB, Kim J (2005) Quality/Quantity Modeling and Analysis of Production Lines Subject to Uncertainty, Phase I, Final Report. Massachusetts Institute of Technology USA, Technical report. Also available as http://web.mit.edu/manuf-sys/www/oldcell1/papers/GM_PhaseI_FinalReport-2005.pdf. Accessed: 10/01/2020.

[22] Kim J, Gershwin SB (2005) Integrated quality and quantity modeling of a production line. *OR Spectrum* 27(2-3):287–314.

[23] iPerf (2019) iPerf2 - A tool that measures network performance of TCP/UDP, https://sourceforge.net/projects/iperf2/. Accessed: 2020-10-14.

[24] Universal Robots (2020) Real-time data exchange (RTDE) guide, https://www.universal-robots.com/articles/ur/real-time-data-exchange-rtde-guide/. Accessed: 2020-09-30.

[25] Red Hat (2017) Deployment, Configuration and Administration of Red Hat Enterprise Linux 6. Red Hat USA, Technical report. Also available as https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/deployment_guide/index. Accessed: 10/01/2020.

[26] Montgomery K (2020) Measurement and Processed Data From A Graph Database Approach to Wireless IIoT Work-cell Performance Evaluation, https://data.nist.gov/od/id/mds2-2242. Accessed: 2020-09-30.

[27] Acromag (2020) Introduction to MODBUS TCP/IP. Acromag USA, Technical report. Also available as https://www.acromag.com/wp-content/uploads/2019/08/White-Paper-Introduction-to-ModbusTCP_765B-.pdf. Accessed: 10/01/2020.

[28] Lyon GF (2009) *Nmap Network Scanning [Online Edition]* (Nmap Project), . Available as https://nmap.org/book/toc.html. Accessed: 10/01/2020.

[29] Beckhoff (2020) TwinCAT ADS/AMS - Specification. Beckhoff Information System USA, Technical report. Also available as https://infosys.beckhoff.com/english.php?content=../content/1033/tcadsamsspec/html/tcadsamsspec_adscmds.htm&id=16054250480117790 71. Accessed: 10/01/2020.

[30] Beckhoff (2018) IEEE 1588 external synchronization interface (EL6688). Beckhoff Information System USA, Technical report. Also available as https://www.beckhoff.com/english.asp?ethercat/el6688.htm. Accessed: 10/01/2020.

[31] WireShark (2020) tshark - Dump and analyze network traffic. wireshark.org USA, Technical report. Also available as https://www.wireshark.org/docs/man-pages/tshark.html. Accessed: 10/01/2020.

[32] Universal Robots (2020) UR CB3, https://www.universal-robots.com/cb3/. Accessed: 2020-09-30.

[33] Robotiq (2020) 2F-85 and 2F-140 Grippers, https://robotiq.com/products/2f85-140-adaptive-robot-gripper. Accessed: 2020-09-30.

[34] OnRobot (2020) 6 axis Force Torque Sensor, https://onrobot.com/en/products/hex-6-axis-force-torque-sensor. Accessed: 2020-09-30.

[35] Beckhoff (2020) CX2020 Basic CPU Module, https://www.beckhoff.com/english.

asp?embedded_pc/cx2020.htm. Accessed: 2020-09-30.
[36] Beckhoff (2020) CX9020 Basic CPU Module, https://www.beckhoff.com/english.asp?embedded_pc/cx9020.htm. Accessed: 2020-09-30.
[37] Shuttle (2020) Entry-level Redefined: Fanless 1 liter PC with Intel Gemini Lake (DL10J), http://global.shuttle.com/products/productsSpec?productId=2281. Accessed: 2020-09-30.
[38] Meinberg (2020) LANTIME M900/PTP, https://www.meinbergglobal.com/english/products/modular-3u-ieee-1588-grandmaster-clock.htm. Accessed: 2020-09-30.

Appendices

A. Communication Messages in the Testbed

In this appendix, we provide the basic knowledge for understanding various network packet formats that have been used in the testbed measurements and analysis. Protocol header information and field values are verified using WireShark packet captures from the testbed network links.

A.1 Basic Packet Format

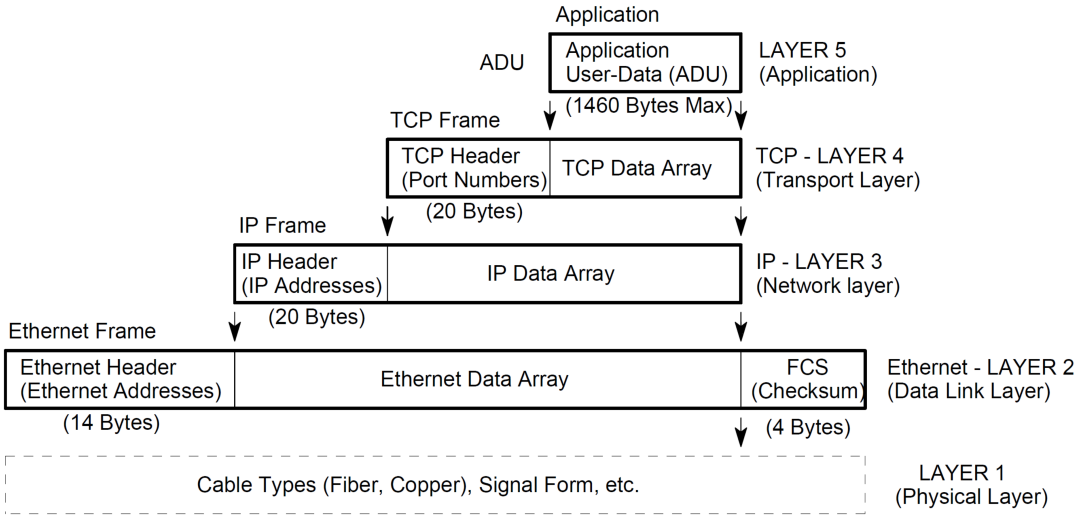


Fig. 40. Construction of a TCP/IP-Ethernet data packet (reprinted from [27])

Application messages refer to data exchanged between networked testbed components for work-cell coordination and other testbed functions. The control information regarding individual protocol layers is usually concatenated into the packet's headers and sent along with data in the medium, such as Ethernet links or wireless channels. Following the OSI's 7-layer model, the analysis on testbed messages uses the information of following layers including application, transport, network, MAC, and PHY.

Communication protocols such as ADS, Modbus, RDP, and intra-robot messages use TCP in the transport layer and IP in the network layer. These TCP/IP packets are originally designed to carry application data in Ethernet links. Most of network components in the testbed are equipped with one or more Ethernet ports as the main network interface to the wired network. Fig. 40 illustrates the layered packet format and header fields of individual

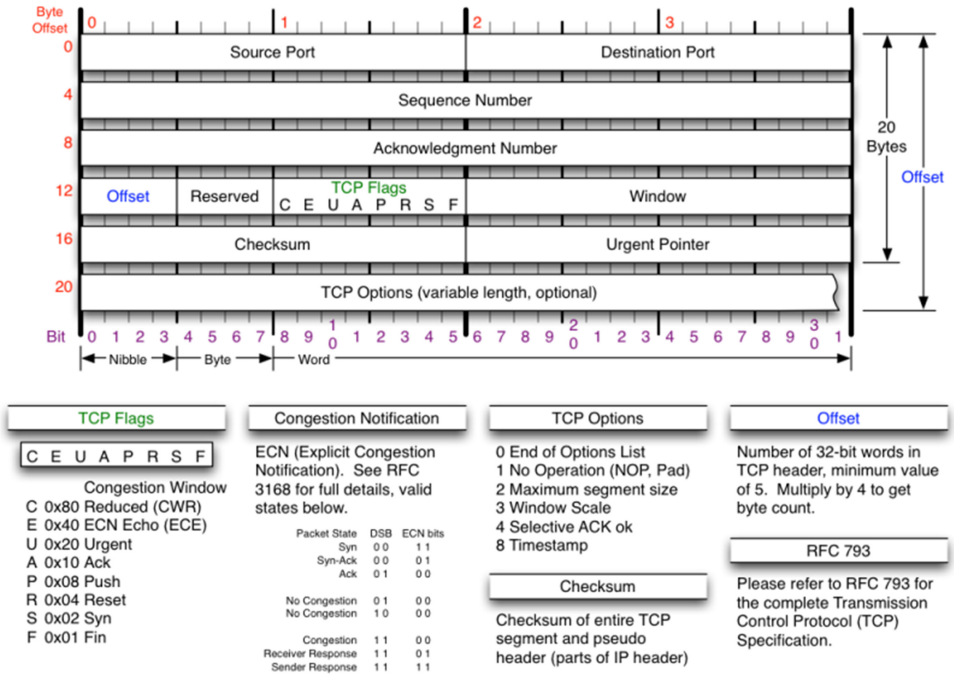


Fig. 41. TCP header (reprinted from [28, TCP/IP Reference])

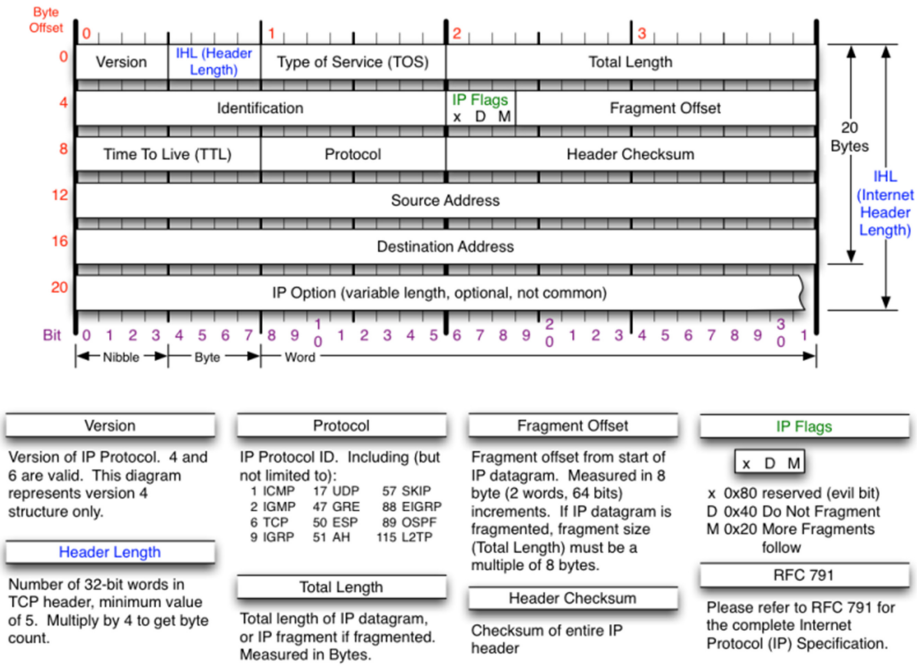


Fig. 42. IP header (reprinted from [28, TCP/IP Reference])

layers in a typical TCP/IP data packet through Ethernet. If such packets transmit wirelessly, the wireless protocol header will replace the Ethernet header. The upper layer meta information and message data remain intact. Fig. 41 and Fig. 42 illustrate details of TCP and IP header information, respectively.

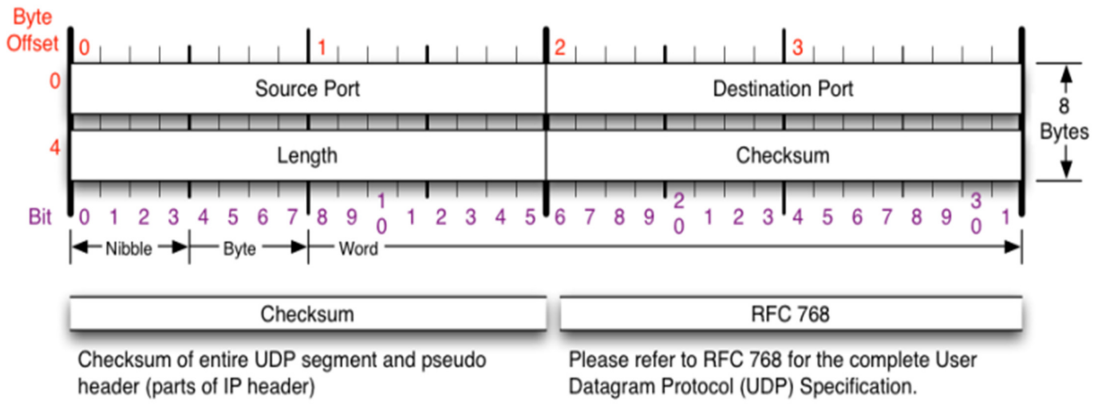


Fig. 43. UDP header (reprinted from [28, TCP/IP Reference])

UDP/IP packets also carry a part of the testbed traffic such as time synchronization messages in both NTP and PTPv2 protocols. Similar to the TCP-based packet format, UDP-based packets contain the UDP header information, as shown in Fig. 43, in the transport layer.

A.2 ADS/AMS

Beckhoff's TwinCAT devices, such as PLCs including both CX2020 and CX9020 and their peripheral terminal modules (if available), use a proprietary communication protocol to exchange data and control messages with each other. Specifically, their messages are formatted as predetermined industrial functions and command codes, named ADS commands. A unique ID called AMS Net ID is used to identify each ADS node in the TwinCAT network. By default, the AMS Net ID is defined as "$a.b.c.d.e.f$" where "$a.b.c.d$" is the device's IPv4 address and the last two segments "$e.f$" are usually set as "1.1". If the ADS node has multiple AMS interfaces or multiple ADS nodes belong to a subnet, the extra segments can help specify the hierarchy of TwinCAT network related with the industrial system architecture. In Beckhoff's terminology, ADS refers to the physical aspect of TwinCAT devices, while AMS is for the network side. In this project, the above two terms are used interchangeably. The implemented TwinCAT devices are often referred to as ADS/AMS nodes/devices.

Fig. 44 illustrates components in an AMS message that carries ADS command(s). Specifically, atop TCP, an AMS message contains three parts: AMS/TCP header, AMS header, and ADS data/command(s). In an AMS header, the ADS/AMS connection information is provided along with an ADS command ID pointing to the contained data section. Table 4 provides the list of available ADS commands. For ADS commands that involve

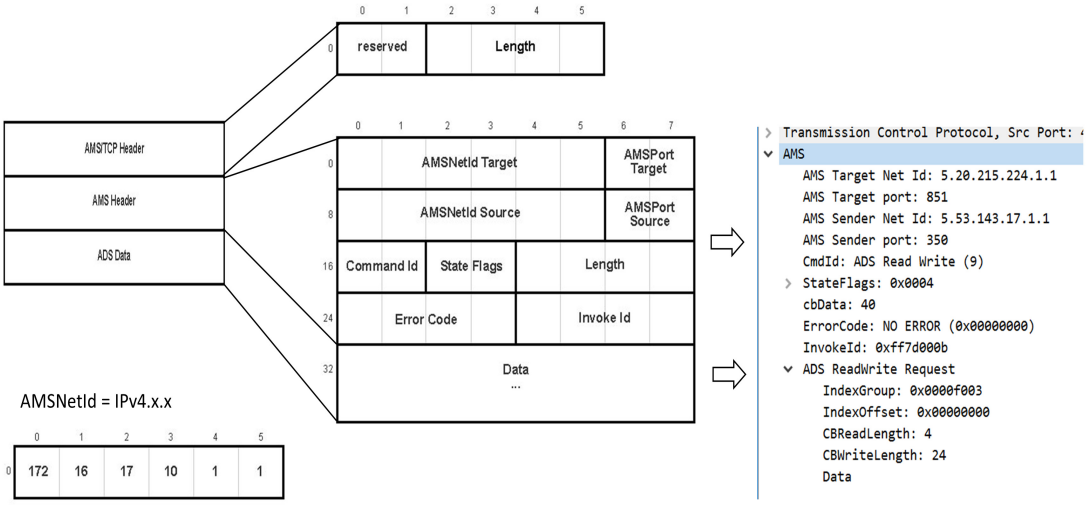
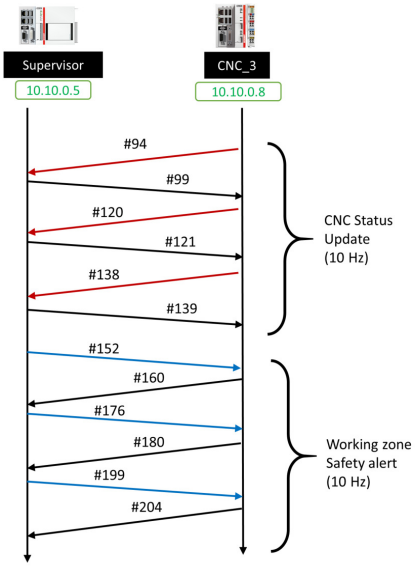


Fig. 44. ADS packet frame (partially reprinted from [28, TCP/IP Reference])

Table 4. ADS Commands (reprinted from [29])

Command ID	Command	Description
0x0000	N/A	N/A
0x0001	ADS Read Device Info	Reads the name and the version number of the ADS device.
0x0002	ADS Read	With ADS Read, data can be read from an ADS device.
0x0003	ADS Write	With ADS Write, data can be written to an ADS device.
0x0004	ADS Read State	Reads the ADS status and the device status of an ADS device.
0x0005	ADS Write Control	Changes the ADS status and the device status of an ADS device.
0x0006	ADS Add Device Notification	A notification is created in an ADS device.
0x0007	ADS Delete Device Notification	It is created before the defined notification is deleted in an ADS device.
0x0008	ADS Device Notification	Data will carry forward independently from an ADS device to a Client
0x0009	ADS Read Write	With ADS Read Write, data will be written to an ADS device. Additionally, data can be read from the ADS device.

process variable operations, e.g., read or write, the target data address of the device is required. The ADS/AMS protocol allows PLC applications to use the variable name to obtain the handle of a remote register so that the device can transmit process information without the knowledge of memory allocation in remote nodes. This feature facilitates the modular design that is focused on nodal functions in the process and encourages individual devices to manage their local memory in a distributed way, which agrees with the dynamics and diversity of industrial applications. In our testbed, we also manage ADS/AMS communications in such a way.




```
STRUCT mStaRrpt {
    _nStaID: UINT; # 16b
    _nState: UINT; # 16b
    _nCode: UINT; # 16b
    _bQueueOn: BOOL; # 8b
}
```

Pkt ID	State Flag	Invoke ID	ADS Data	Index Group	Index Offset
#94	ADS Req.	0xff7d000b	GET_SYMHANDLE_BYNAME	0x0000F003	0x00000000
#99	ADS Resp.	0xff7d000b	No Error		
#120	ADS Req.	0xff7d000c	RD/WR_SYMVAL_BYNAME	0x0000F005	0x58800039
#121	ADS Resp.	0xff7d000c	No Error		
#138	ADS Req.	0xff7d000d	RELEASE_SYMHANDLE	0x0000F006	0x58800039
#139	ADS Resp.	0xff7d000d	No Error		

Pkt ID	State Flag	Invoke ID	ADS Data	Index Group	Index Offset
#152	ADS Req.	0xff650012	GET_SYMHANDLE_BYNAME	0x0000F003	0x00000000
#160	ADS Resp.	0xff650012	No Error		
#176	ADS Req.	0xff650013	RD/WR_SYMVAL_BYNAME	0x0000F005	0x340000a1
#180	ADS Resp.	0xff650013	No Error		
#199	ADS Req.	0xff650014	RELEASE_SYMHANDLE	0x0000F006	0x340000a1
#204	ADS Resp.	0xff650014	No Error		

Fig. 45. Handshakes of ADS message transactions in a WireShark trace file

In Fig. 45, two examples of ADS/AMS transactions between testbed PLCs are demonstrated. In a complete transaction, three rounds of handshakes are performed: 1) getting the symbol handle in the remote device by the name, 2) operating the remote symbol using the returned handle, and 3) releasing the handle. In each round, a request-response conversation forms a two-way communication where the request and response are paired with a unique 32-bit Invoke ID in their AMS headers. In the first example, a CNC status report is initiated by the CNC PLC. In the status report, an array of machine status is updated at the supervisor as a data structure known to both devices. In the second example, the supervisor sets a one-bit flag in the CNC to notify the clearance of safety alert regarding a moving robot in the CNC's working space.

We refer the interested readers to the online Beckhoff Information System for further details of ADS/AMS communications [29].

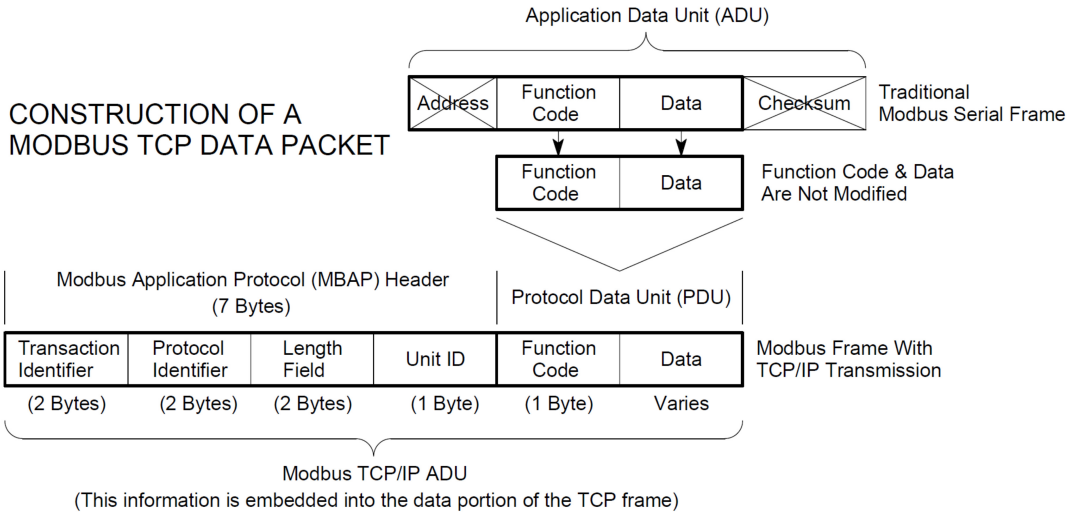


Fig. 46. Construction of a Modbus TCP data packet (reprinted from [27])

A.3 Modbus

The supervisor communicates with robots through Modbus, which allows the data exchange between heterogeneous industrial appliances in the shared registers at the supervisor. UR3 robots use Modbus TCP in their communications. The Modbus TCP protocol modifies the original Modbus serial frame and adds new header information atop TCP. As shown in Fig. 46, a Modbus TCP packet includes a Modbus Application Protocol (MBAP) header and the Protocol Data Unit (PDU), which contains the tailored Modbus data.

For different register types in Modbus, process variables are accessed through different register address groups as shown in Fig. 47. In this testbed, we only consider digital registers each of which stores a 16-bit numerical data (binary or decimal) referenced in the 4xxxx segment. Accordingly, the available operations in the supported registers can be found in Fig. 48.

Fig. 49 and Fig. 50 illustrate examples of formatting Modbus function messages for reading and writing the registers, respectively. The source and destination port numbers of the TCP header indicate whether or not the TCP payload contains a MBAP request/response: If the TCP source port is 502, one or more Modbus responses are contained; if the TCP destination port is 502, one or more Modbus requests/requries are contained. Accesses to different Modbus registers can be managed in parallel Modbus transactions, even along the same TCP connection. The 32-bit transaction identifier is used in the MBAP header for transaction pairing between requests and responses. For the query of reading holding registers, the response returns the requested register values. For the query of writing a value into the register, the response just repeats the content in the request to confirm

Reference	Description
0xxxx	Read/Write Discrete Outputs or Coils. A 0x reference address is used to drive output data to a digital output channel.
1xxxx	Read Discrete Inputs. The ON/OFF status of a 1x reference address is controlled by the corresponding digital input channel.
3xxxx	Read Input Registers. A 3x reference register contains a 16-bit number received from an external source—e.g. an analog signal.
4xxxx	Read/Write Output or Holding Registers. A 4x register is used to store 16-bits of numerical data (binary or decimal), or to send the data from the CPU to an output channel.

Fig. 47. Modbus register addresses (reprinted from [27])

CODE	FUNCTION	REFERENCE
01 (01H)	Read Coil (Output) Status	0xxxx
03 (03H)	Read Holding Registers	4xxxx
04 (04H)	Read Input Registers	3xxxx
05 (05H)	Force Single Coil (Output)	0xxxx
06 (06H)	Preset Single Register	4xxxx
15 (0FH)	Force Multiple Coils (Outputs)	0xxxx
16 (10H)	Preset Multiple Registers	4xxxx
17 (11H)	Report Slave ID	*Hidden*

Fig. 48. Modbus function codes (reprinted from [27])

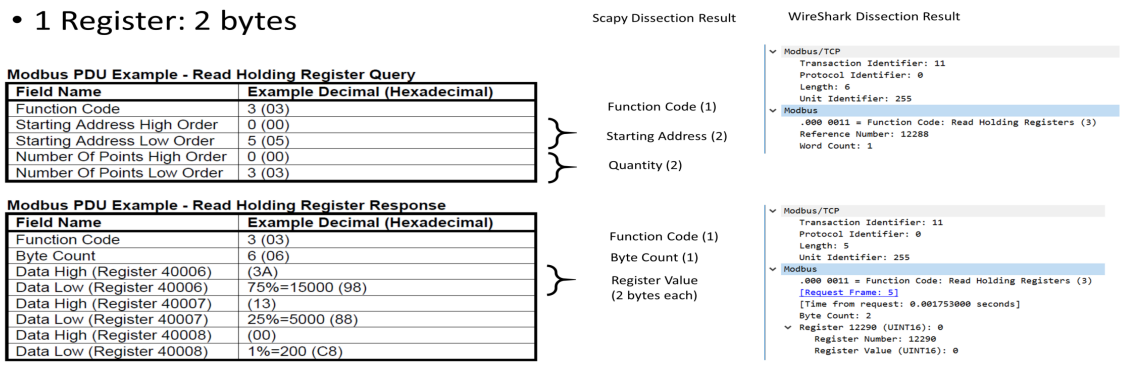


Fig. 49. Modbus function of reading holding registers and capture samples (partially reprinted from [27])

the reception. In the WireShark, we can dissect the transmitted Modbus data and obtain the

Modbus PDU Example - Preset Holding Register Query and Response

Field Name	Example Decimal (Hexadecimal)
Function Code	6 (06)
Register Address High Order	0 (00)
Register Address Low Order	1 (01)
Preset Data High Order	0 (00)
Preset Data Low Order	2 (02)

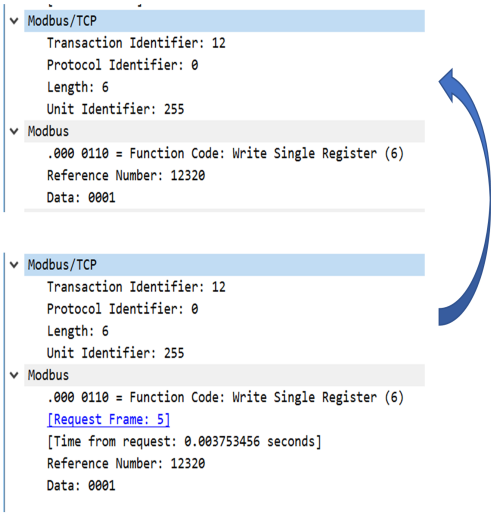


Fig. 50. Modbus function of presetting single register and capture samples (partially reprinted from [27])

useful field information.

We refer the interested readers to the introduction to Modbus TCP/IP for further details of Modbus TCP communications [27].

B. Beckhoff PLC Development

B.1 PLC Data Collection

Using TwinCAT function libraries, we have developed a routine data collection approach in Beckhoff's PLC programs. In a PLC project, programs are organized as objects, known as Program Organization Units (POUs), which include programs, functions, and function blocks. Each POU has its own internal variables that are defined at the file beginning. Meanwhile, for variables that are shared by multiple POUs, they can be defined and managed in one or separate GVL files which are visible globally in the project. Technically, all global and local variables can be routinely saved into files for measurement purposes. Although it is feasible to record variables every PLC cycle (up to 1 kHz), it still depends on multiple factors that determine the real update rate, such as subscribed data size, cache capacity, and file open/write speed. In the testbed measurement, the update rate can be selected within a fixed frequency set with the maximum rate at 125 Hz.

B.1.1 Start a PLC Measurement

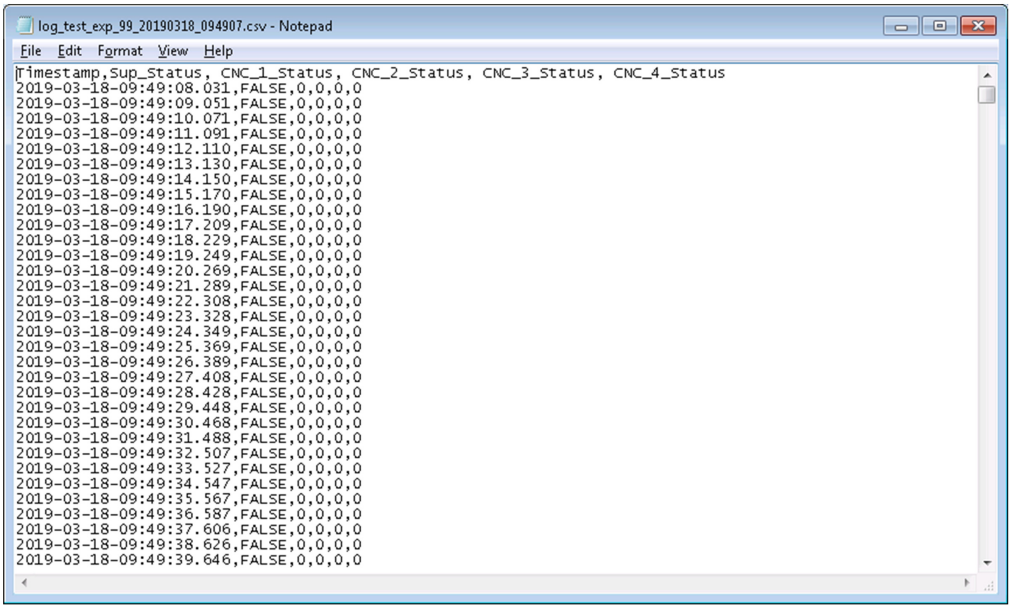


Fig. 51. PLC measurement data in CSV

PLC measurement data are saved as records in CSV format, as shown in Fig. 51. Except the header line on the top, each following row of data contains a list of instant values of the subscribed PLC variables which are separated by comma. The first column is reserved for the record time which is retrieved from the PLC's clock that is synchronized with the testbed's time server via PTPv2. The timing accuracy is on the millisecond level to comply

with the PLC cycling step (as short as 1 ms). Timestamps are later used in analysis to align events from different sources for reconstructing the measured process in the timeline.

```
<plc_meas_config>
    <meas_init>
        <_sFilePrefix>plc_</_sFilePrefix>
        <_bFileNameTimed>true</_bFileNameTimed>
        <_bFeatureInName>true</_bFeatureInName>
        <_sFeatureInName>125hz_</_sFeatureInName>
        <_nLogFreq>125</_nLogFreq>
    </meas_init>
    <meas_log_sub>
        <_bSysTime>true</_bSysTime>
        <_bSupStat>false</_bSupStat>
        <_bCncStat>false</_bCncStat>
        <_bSupSchUrIdx>true</_bSupSchUrIdx>
        <_bSnsStat>true</_bSnsStat>
    </meas_log_sub>
</plc_meas_config>
```

Fig. 52. PLC measurement configuration in XML

Measurement configurations are submitted to the PLC through an XML file as shown in Fig. 52. In the file, configuration settings are divided into two main sections: initialization and variable subscription options. The former, labeled in the $< meas\_init >$ tag, provides global configuration options, such as file naming rules (e.g., indicating start date/time) and data update rate; in the latter, we can enable/disable the collection of available variables as listed in the $< meas\_log\_sub >$ tag. The settings are read by the PLC, stored in the corresponding data structures of the *gvMeas* GVL, and used by the measurement function *Meas_Main* in data collection.

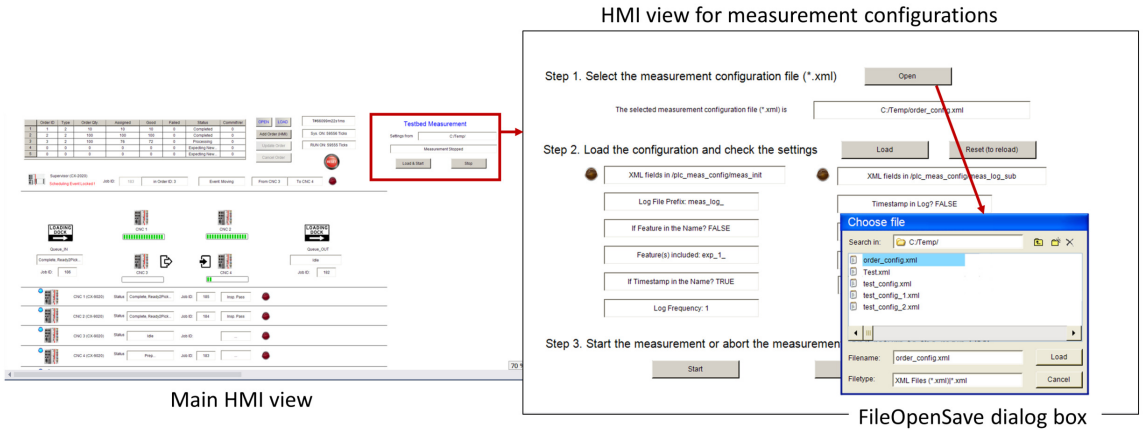


Fig. 53. Load PLC measurement configuration in HMI

We start a new measurement from loading the prepared configuration file. As shown

in Fig. 53, HMI's main view provides a simple user interface for measurement control. By clicking the "Load & Start" button, HMI will switch to the view for measurement configuration. There are three steps here.

- **Step 1. Select the measurement configuration file**

  Click the "Open" button to show the file selection dialog box; Select the configuration file (*.xml) and click "Load";

- **Step 2. Load the configuration**

  Click the "Load" button. Check the loaded configuration settings shown in the view. If the configuration is not as expected, modify the XML file and repeat Step 1 with the right file. If any error indicator is ON (Errors in reading XML fields), double check XML tags and file format. Reload the correct file starting from Step 1, or click "Reset" and then "Load" to reload the opened file after modifications.

- **Step 3. Start the measurement**

  If everything looks good in the loaded configuration, click the "Start" button to start the measurement and return to the main HMI view. The measurement setup can be aborted anytime by clicking the "Abort" button to cancel and return to the main HMI view. The measurement status shows the current log file name and location in PLC.

In the main HMI view, the ongoing measurement is shown with its status. It can be stopped by clicking the "Stop" button. Repeat the "open-load-start" operations to start a new measurement.

B.1.2 Adding New PLC Variables in the Subscription List

As shown in Fig. 52, all available PLC variables that can be subscribed in data collection are listed as logging options in the $< meas\_log\_sub >$ tag of XML configuration. We can enable or disable individual variables in subscription by setting "true" or "false" values, respectively. For variables that have not been listed, we need to first add them into the list visible to measurement. Before showing how to add a new measurable variable, let us first review how measurements are implemented in the PLC.

In the PLC program, all measurement control variables are stored in the *gvMeas* GVL. Through interactive operations in HMI, visualization events in measurement-related views, such as mouse clicks, will load the configuration information from XML into *gvMeas*. Once the measurement gets started, PLC will activate data collection based on *gvMeas* settings. The *Meas_Main* function is in charge of main data collection tasks which routinely retrieve the subscribed variable values and write them into the CSV file.

In order to add a new variable into the subscription list, we need to following three steps that prepare PLC for the new measurement option.

- **Step 1. Locate the to-be-subscribed variable(s) in the Supervisor, usually defined in GVLs**

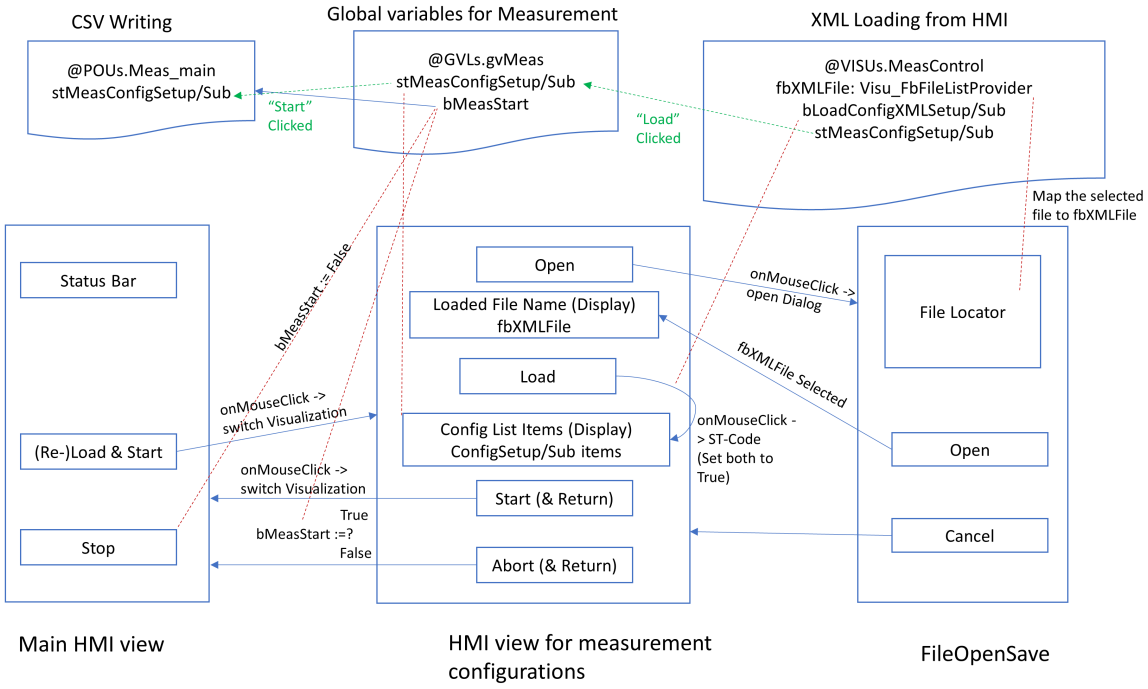


Fig. 54. The design of PLC measurement

- **Step 2. Register the variable(s) in PLC (hard coded)**
  - **Step 2-1. Add selection indicator(s)** In the $DUTs/MEAS\_LOG\_SUB$ structure, add BOOL indicators, one for each newly added variable, e.g., $\_bSupDevTemper$ with default FALSE.
  - **Step 2-2. (Optional) Visualize new indicator(s) in HMI** To check if the newly subscribed variables are loaded in the runtime, add text field(s) in the measurement configuration view of HMI controlled by $VISUs/Visualization\_MeasConfig$. The text variables (STRING) are mapped from the corresponding variable elements that are newly added into the structure object $gvMeas.stMeasLogSub$.
  - **Step 2-3. Add variable(s) in the measurement function** In $POUs/Meas\_Main$, update the generation code regarding the header line and data lines with the newly added variable information. Note that CNC variables are usually organized in vectors each of which may contain four observable data (from CNC 1–4) or more (as STRUCT). In the CSV file, such variables occupy multiple columns in the header line and data lines as counted by elements.
  - **Step 2-4. Compile and build the PLC project** Rebuild the project and load it to the PLC.
- **Step 3. Subscribe the data in measurement**

– **Step 3-1. Update logging option in XML** In the XML configuration file, in the $< plc\_meas\_config >$ element, add new logging items, e.g., $< \_bSupDevTemper >$ $true < /\_bSupDevTemper >$, where the $true$ values means enabling the collection of this value in the CSV file. Save the XML file.

– **Step 3-2. Update measurement settings in the runtime** Stop the ongoing measurement (if there is any, the measurement status is shown in the main HMI view). Load the update XML file, check the status on display (option, available with Step 2-2), and start the new measurement.

B.2 PLC Time Synchronization

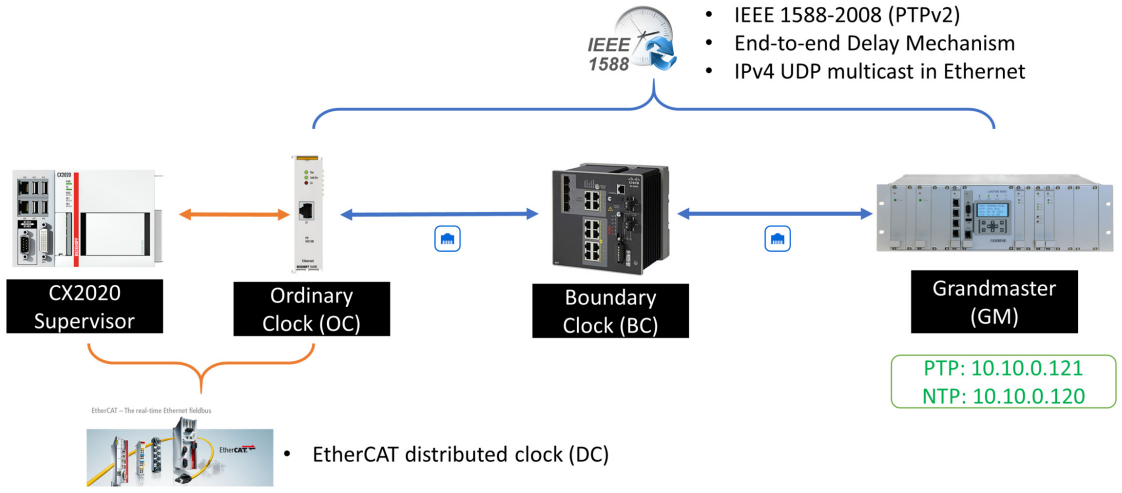


Fig. 55. Time synchronization between PLC and time server through PTPv2

Beckhoff's TwinCAT devices support PTP through an add-on module, the EL6688 IEEE 1588 module. This terminal unit can filter PTP synchronization messages received at the built-in Ethernet port, synchronize its hardware clock to the selected PTP time server, and serve as an OC for providing the synced reference time to PLC applications. Fig. 55 illustrates the end-to-end time synchronization path from the time server to the PLC task cycle. Within the TwinCAT system, different modules exchange timing information through EtherCAT links.

In the network, the synchronization is performed using the end-to-end delay mechanism to calculate the OC's offset from the grand master. Fig. 56 illustrates a round of handshakes that enables the OC to routinely update its clock offset to the selected time server.

The synced OC at EL6688 is also called an EtherCAT Distributed Clock (DC) in Beckhoff systems. DC's offset to the (external) time server clock, $DcToExtTimeOffset$, is in nanosecond. Serving as the local time reference for the PLC, EL6688 also has to keep measuring the time offset between DC and the PLC's task clock (TC) and maintain it as

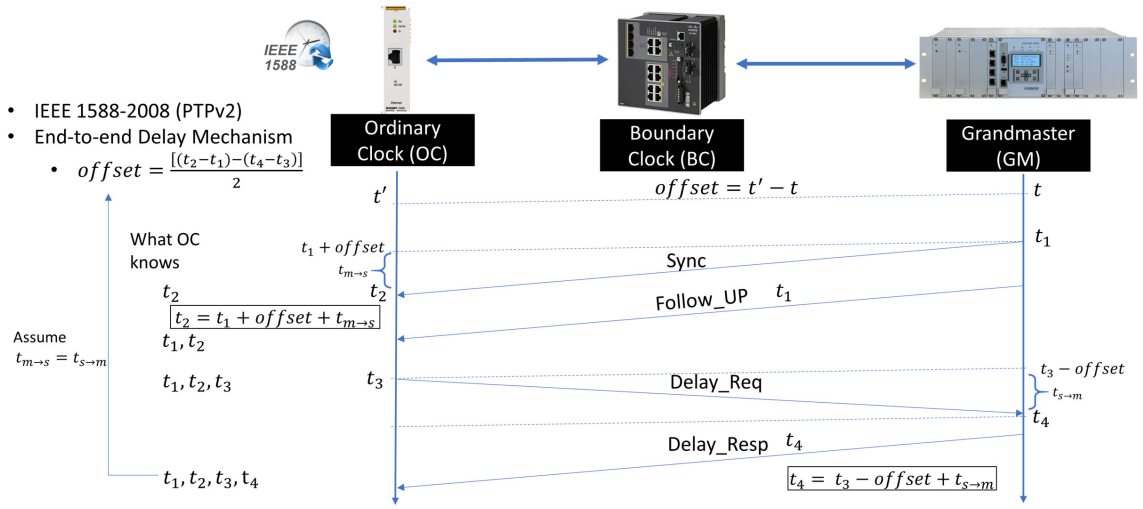


Fig. 56. PTP end-to-end delay mechanism

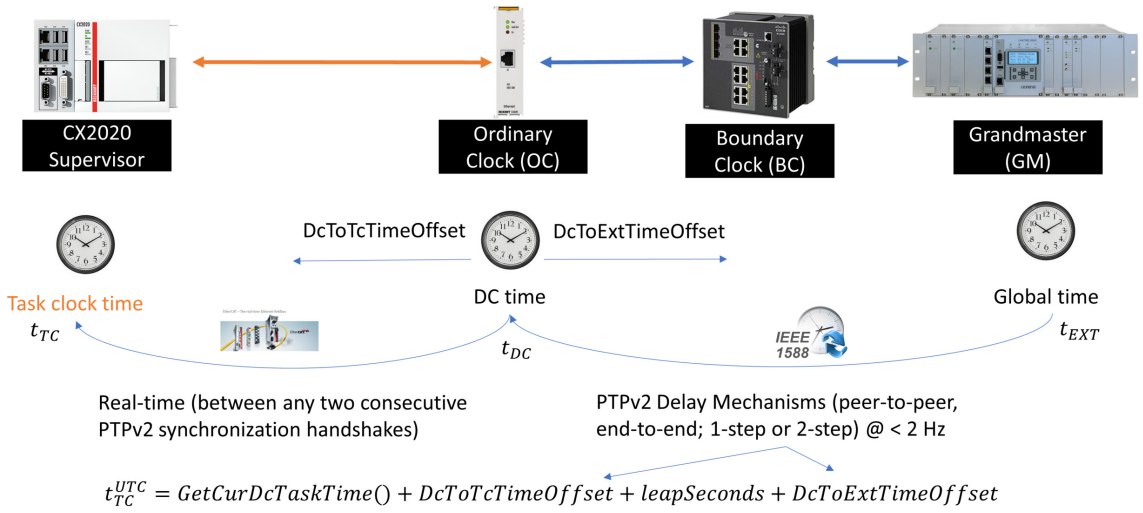


Fig. 57. Distributed clock adjustment in PTP time synchronization

another output variable $DcToTcTimeOffset$ in nanosecond. Fig. 57 illustrates how the PLC uses the PTP-synced clock time. When an application calls for the current TC time, e.g., rendering a timestamp for the measurement record, it is calculated in Eq.(2) [30].

$$t_{TC}^{UTC} = GetCurDcTaskTime() + DcToTcTimeOffset + leapSeconds + DcToExtTimeOffset \quad (2)$$

where the $GetCurDcTaskTime$ component is the instant time reading from the system clock at the PLC, the two offset values, i.e., $DcToTcTimeOffset$ and $DcToExtTimeOffset$, are

read from the PLC's interface to DC. Note that TwinCAT devices, including EL6688, work internally with the temps atomique international (ATI) time. To obtain the UTC time, it requires a further conversion by adding "leap seconds". In November 2019, there were 37 "leap seconds" [30].

In the saved CSV data, PLC writes the local time information at the beginning of each record row, which also applies the time zone and daylight saving time adjustments to the calculated UTC time.

C. Translating Testbed Data to Graph Database

In Section 7.2, the data importing pipeline has been introduced. This appendix documents more details of individual procedures in pipeline implementation. First, properties of graph elements, i.e., nodes and relationships, are summarized in tables for reference in further data analysis of the graph.

C.1 Graph Nodes, Relationships, and Their Properties

Nodes, relationships, and their properties used by the testbed GDB are summarized in the following tables. Each property comes with the name and corresponding variable type. As raw data are stored in CSV files as strings, it refers to these tables in converting CSV records to correct variable types, e.g., integer or float, when creating property instances.

C.1.1 Nodes

Table 5. Nodes and their property tags in the graph for static objects

Node Label	Property	Type	Value	Note
(:Actor)	name	String	{"Supervisor", "CNC-x", "Operator", "Inspector"}	
	type	String	{"PLC", "Robot Controller"}	
	model	String	{"CX2020", "CX9020", "UR3"}	
(:NtwkID)	name	String	e.g., "CNC-1_LAN"	
	ipAddr	String	"a.b.c.d" where a/b/c/d in [0, 255]	IPv4
	host	String	the hosting Actor's name	
(:Adapter [:Wireless :AP/:UE])	protocol	String	{"Ethernet", "802.11"}	
	name	String	Port name, e.g., "eth0"	
	macAddr	String	"aa:bb:cc:dd:ee:ff"	
	host	String	Only for Ethernet NIC, e.g., "Supervisor"	Optional
(:Sniffer)	type	String	{"wireless"}	
	protocol	String	{"802.11"}	
	name	String	Sniffing port name, e.g., "WLS1"	
	macAddr	String	"aa:bb:cc:dd:ee:ff"	

As introduced in Section 7.1, a number of static graph nodes are defined in the graph to represent testbed components and their settings in experiments. Table 5 summarizes these node types and their properties used in the graph. Specifically, static nodes include Actor, NtwkID, Adapter (and subcategorical nodes), and Sniffer, which are first created in the graph and serve as anchor points for later added nodes.

- **Actor** Actor nodes are testbed components which are networked to take industrial operations. In the testbed, Actor nodes include the supervisor PLC, CNC controllers,

and robotic controllers. In the measurement, we collect network traffic from TAP devices that are collocated with these Actors.

- **NtwkID** An Actor can also be associated with one or more NtwkID nodes, which contain the network configuration information about the linked Actor. They are also created in the initiation phase and linked with the served Actor nodes.
- **Adaptor** To connect to the testbed network, each Actor employs one or more adapters. An adapter is a network interface through an Ethernet or wireless link. It can have additional labels to further specify its role in the network. For example, a Wi-Fi router can be represented by a node with the label (:Adapter:Wireless:AP); for Wi-Fi clients in the network, we can use the label (:Adapter:Wireless:UE). Besides, (:Adapter:Ethernet) is used to label the embedded Ethernet interfaces of industrial appliances.
- **Sniffer** Sniffer is a type of device that captures packets in the select medium for online/offline analysis of communication traffic. Currently we use an Intel NUC as the wireless sniffer to monitor the link quality in the selected 2.4 GHz WLAN channel during the testbed operation.

Dynamic nodes denote testbed records that capture physical process status and network traffic in the work-cell. In current GDB, such nodes include messages, transactions, and physical actions. Table 6 summarizes graph nodes that represent communication messages and associated properties. Each Message node is created based on a single network packet transmitted between industrial devices in the wired/wireless communication link. Packets in the testbed are mainly of the TCP/IP type. Message properties include both the embedded information in PCAP records and necessary meta data to establish relationships.

Transaction nodes refer to operational transactions between industrial devices to coordinate them in work-cell operations. They can be viewed as the abstract of real network packets in the application layer. Message nodes and Transaction nodes have the $n : n$ relationship, i.e., one packet may support multiple transactions (e.g., with aggregated Modbus messages), or one transaction involves multiple packets (e.g., ADS/AMS handshakes).

Table 6. (:Message) nodes and their property tags in the graph

Field	Property	Type	Value	Note
MAC	macSrc	String	Source address, “aa:bb:cc:dd:ee:ff”	
	macDst	String	Destination address, “aa:bb:cc:dd:ee:ff”	
IP	ipSrc	String	“a.b.c.d” where a/b/c/d in [0, 255]	IPv4
	ipDst	String	“a.b.c.d” where a/b/c/d in [0, 255]	IPv4
	ipLen	Integer	IP packet length (including IP header)	Bytes
	ipChksum	String	“0x0000abcd”	Optional
TCP	tcpPrtSrc	Integer	TCP source port number	
	tcpPrtDst	Integer	TCP destination port number	
	tcpSeqNum	String	Relative sequence number (in a PCAP)	Optional
	tcpAckNum	String	Relative acknowledgement number	Optional
	tcpChksum	String	“0x0000abcd”	Optional
Appl.	adsCmdId	Integer	applicable when proto=“ADS”	Optional
	adsInvokeId	String	“0xabcd1234”, applicable when proto=“ADS”	Optional
	adsIndexGroup	String	“0x0000abcd”, applicable when proto=“ADS”	Optional
	adsIndexOffset	String	“0xabcd1234”, applicable when proto=“ADS”	Optional
	adsMatchCode	String	Calculated based on selected ADS features	Optional
	mbTransId	Integer	applicable when proto=”Modbus”	Optional
	mbFuncCode	Integer	applicable when proto=”Modbus”	Optional
	mbRefNum	Integer	Modbus register number in decimal	Optional
	mbRegVal	String	Value in the Modbus register number	Optional
	mbMatchCode	String	Calculated based on selected Modbus features	Optional
Meta	msgId	Integer	unique in a measurement set, encoded as a 15-digit integer as “1nnttrrxxxxxxxx”.	Global
	copyType	String	By the record source, in {“tx”, “rx”}	
	proto	String	{“Ether”, “TCP”, “UDP”, “ADS”, “Modbus”}	
	transRole	String	The role in a transaction, in {“request”, “response”, “sole”}	
	rsp2MsgId	Integer	the global msgId of the request message which is responded to, 15-digit	only for response
	txTime	Float	the epoch time of Message captured at TX	
	rxTime	Float	the epoch time of Message captured at RX	
	rxGlobalMsgId	Integer	the global msgId of the RX copy, 15-digit	only for TX copy
	msgCopy	Integer	for multiple Application messages in the same TCP payload, copies share the same field contents of TCP and lower layers	
	cmdType	String	Industrial application command type	
	cmdDetail	String	Industrial application command detail	
	tcpRetxMsgId	Integer	the global msgId of the Message indicated by “tcp_ana_rto_frame” field in PCAP	
	tcpDuplicate -AckMsgId	Integer	the global msgId of the Message indicated by “tcp_ana_duplicate_ack_frame” field in PCAP	

Table 7. (:Transaction) nodes and their property tags in the graph

Property	Type	Value	Note
proto	String	{“Ether”, “TCP”, “UDP”, “ADS”, “Modbus”}	
reqTxId	Integer	the msgId of the init Message (Request) recorded at TX	
rspRxId	Integer	the 15-digit global Message ID copied from rxGlobalMsgId of the closing Message (Response)	Optional
timeStart	Float	the txTime of the init Message (Request) node	
timeStop	Float	the rxTime of the closing Message (Response)	Optional
status	String	{“Open”(init), “Closed”}, switched to “Closed” once all relationships are created	
cmdType	String	Industrial application command type	
cmdDetail	String	Industrial application command detail	
transCopy	String	copied from msgCopy of the Request Message, used in Modbus transactions	Optional
mbTransId	Integer	copied from mbTransId of the Request, used in Modbus transactions	Optional

Table 7 summarizes node properties regarding a Transaction node. Each Transaction serves a specific industrial action purpose in processing monitoring and control. Transactions in the testbed include ADS/AMS messages (between the supervisor and CNCs) and Modbus messages (between the supervisor and robotic controllers).

Table 8. Physical action nodes and their property tags in the graph

Node	Property	Type	Value	Note
(:PhyAction)	action	String	description to the action	
	timeStart	Float	The action start time in the epoch time	
	timeStop	Float	The action stop time in the epoch time	
(:URSchedule)	actor	String	the Actor's name, i.e., "Supervisor"	
	stateId	Integer	unique type index associated with the Actor (in case an Actor may have multiple states to collect)	
	actionId	String	unique in the measurement set; "3"+actorId(2)+stateId(1)+index(8)	14-digit
	time	Float	The epoch time when the state begins	
	target	String	The name of the scheduled Actor, e.g., "Inspector"	
	targetAction	String	The action assigned to the target, e.g., "inspect"	
	waypointId1	Integer	The index of job stop 1. For OPT, it is the part pick-up location; for INS, it is the inspection location.	
	waypointId2	Integer	The index of job stop 2. For OPT, it is the part drop-off location; for INS, it repeats the set value of waypointId1.	
(:SensorState)	actor	String	the host actor's name	
	stateId	Integer	unique type index associated with the Actor (in case an Actor may have multiple states to collect)	
	sensorType	String	The sensor type of the state source, e.g., "Proximity" of the part holder in an CNC	
	actionId	String	unique in the measurement set; "3"+actorId(2)+stateId(1)+index(8)	14-digit
	time	Float	The epoch time when the state begins	
	value	Integer	the state value of the sensor output	
(:RouteState)	actor	String	The actor robot who is assigned with this route instruction	By the scheduler
	actionId	Integer	unique in the measurement set; "3"+actorId(2)+stateId(1)+index(8)	14-digit
	time	Float	The epoch time when the route begins	
	optRoute From	Integer	Job stop index from which the part is moved by robot	Optional, OPT only
	optRoute To	Integer	Job stop index to which the part is moved by robot	Optional, OPT only
	inspRoute	Integer	Job stop index at which the part is inspected by robot	Optional, INS only

Table 8 summarizes physical action (PhyAction) nodes used in the graph database, which are built upon production measurement data. A PhyAction node refers to a record of measured industrial action in work-cell operations. These actions are recorded by external monitors, such as UR3's RTDE interfaces and PLC's recording, other than network sniffers. Since PhyAction nodes are only linked with Actors, we can load PhyAction nodes into the graph in parallel with network-related Message and Transaction nodes.

Table 9. QoSReport nodes and their property tags in the graph

Node	Property	Type	Value	Note
(:QoSReport)	time	Float	Packet reception time in epoch	
	dataRate	Integer	WLAN data rate in Mbps	Radiotap Header
	rssi	Integer	WLAN signal strength indicator, in dBm	Radiotap Header
	channel	Integer	WLAN channel index, [1, 11]	Radiotap Header
	txAddr	String	MAC address of the packet sender	
	rxAddr	String	MAC address of the packet receiver	
	rprtId	Integer	unique in a measurement set, encoded as "2"+SnifferId(2)+index(8)	11-digit
	anchorId	Integer	The pairing index between a Message and a QoSReport, unique in the measurement set	To build [:COVERED] relationship
	msgId	Integer	The global msgId of the Message that this QoSReport covers	To build [:COVERED] relationship
	msgCopy Idx	Integer	The index of the covered Message in all Messages which are covered by this QoSReport node	To build [:COVERED] relationship
	msgIpSrc	String	The source IP of the covered Message	To build [:COVERED] relationship

Table 9 summarizes properties used in the QoSReport node which are obtained from the Radiotap header data of the sniffer's output.

C.1.2 Relationships

Table 10 summarizes relationships between nodes in the graph. They are created based on conditions that links two nodes given their properties. Note that relationships are directional.

Table 10. Relationships and their creation conditions in the graph

Relationship	Connect Node	To Node	Condition(s)	Note
TX	(a:Actor)	(m:Message)	m.ipSrc=n.ipAddr where (a)-[:HAS]->(n:NtwkID)	
RX	(a:Actor)	(m:Message)	m.ipDst=n.ipAddr where (a)-[:HAS]->(n:NtwkID)	
SUPPORTED	(m:Message)	(t:Transaction)	(t) is created from (m)	m.transRole= 'Request'
			m.rsp2MsgId=t.reqTxId	m.transRole= 'Response'
SCHEDULED	(u:URSchedule)	(r:RouteState)	u.actionId AND r.actionId are coupled in timeline	Performed separately for OPT and INS
TOOK	(a:Actor)	(p:PhyAction)	a.name=p.actor	Also apply to (:URSchedule); (:RouteState); (:SensorState)
FOLLOWED	(a:PhyAction)	(b:PhyAction)	a.actionId=b.actionId-1	Also apply to (:URSchedule); (:RouteState); (:SensorState)
PARTICIPATED_IN	(a:Actor)	(t:Transaction)	(a)-[:TX/RX]->(m:Message) -[:SUPPORTED]->(t)	
CONNECTED_THROUGH	(a:Actor)	(n:Adapter :Ethernet)	a.name=b.host	For built-in Ethernet interface(s)
		(n:Adapter :Wireless:AP /:UE)	a.name AND b.macAddr	Specified in experiments
NEXT	(a:PhyAction)	(b:PhyAction)	a.actionId AND b.actionId	Call Neo4j APOC's node link function
COVERED	(q:QoSReport)	(m:Message)	m.msgCopy=q.msgCopyIdx AND m.msgId=q.msgId	
TRIGGERED	(t:Transaction)	(r:RouteState)	t.reqTxId AND r.actionId are coupled in timeline	Modbus commands, performed separately for OPT and INS
HAS	(a:Actor)	(n:NtwkID)	a.name=n.host	Static
GENERATED	(w:Wireless)	(q:QoSReport)	(w) ->(q)	w/ a single Sniffer
COLOCATED_WITH	(w:Wireless)	(a:Actor)	a.name AND w.name	Specified in experiments

C.2 Data Importing Flow

C.2.1 Overview

To streamline data processing steps, a software package is developed. Fig. 58 illustrates its main function modules and highlights the data pipeline along which measurement data are converted into graph elements by the graph data model addressed in Section 7.1.1. The code is mainly written in Python 3.6.6, which also employs Neo4j Desktop 3.5.6 as the GDB application. Using the Neo4j bolt driver, the Python script can visit the Neo4j database with the embedded Cypher query statements. In addition, Bash scripts use TShark's functions

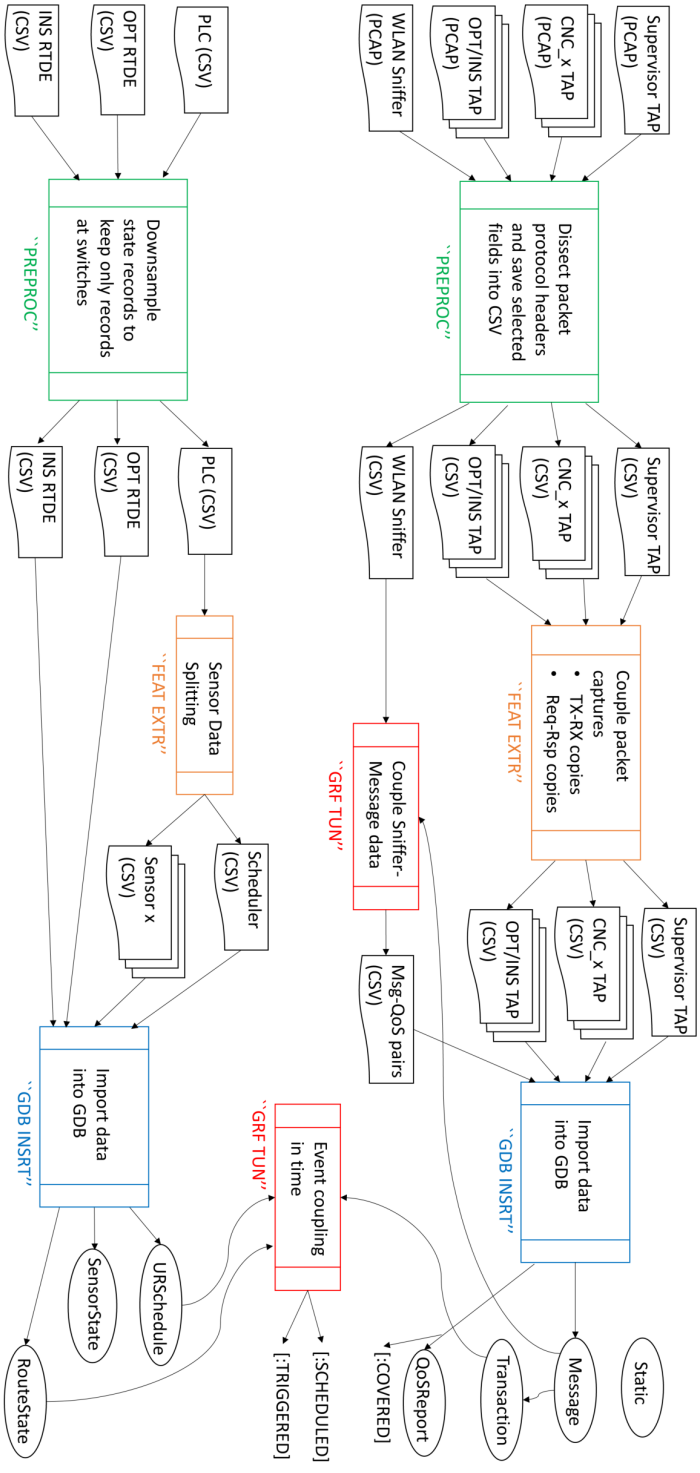


Fig. 58. Implementing the pipeline of importing testbed data into GDB

to dissect network packet captures for header information of different network protocols.

As shown in Fig. 58, network traffic data and production modules are treated according to their own data features. In the preprocessing (PREPROC) stage, network captures in PCAP files are first processed by a packet dissection tool to extract multiple fields from the encapsulated packet headers and save them into separate columns in CSV files. In this tool, Bash scripts use TShark's protocol libraries to resume header fields from packet bytes. These fields will later serve as either properties of packet-oriented graph nodes, such as Messages and Transactions, or pairing conditions to create relationships showing the communication history. In the next stage, i.e., feature extraction (FEAT EXTR), separate packet copies will be paired according to interconnections in transmissions, such as two copies captured on both ends of a link. These pairing information will also be saved as additional meta data in CSV files and used to build the graph. Next, in graph database insertion (GDB INSRT), rows of CSV files will be imported into the GDB to create nodes and relationships. However, it is not the end of the building process. The graph tuning (GRF TUN) stage takes further actions to create more relationships to link nodes together regarding their explicit and implicit connections in the network. Such operations may run solely within GDB, however, it is more common to utilize intermediate graph information combined with raw data to unveil more insight relationships.

In a similar way, OT data collected from work-cell modules, i.e., PLC and robot data, are treated in parallel. For example, in PREPROC, OT data are first downsampled to filter out constant variable readings and keep records indicating state transitions. Such operations are purposed for cleaning raw data and reducing computing load of following steps. In GRF TUN, different node events are assorted by time so that delays in communication networks and control processes can be studied in analysis.

C.2.2 Packet Dissection

To treat the packet capture data collected by WireShark instances in the PREPROC stage, we utilize the command-line tool called TShark to dissect captured packets in the raw data files of PCAP and access the packet header information [31]. The content of selected packet header fields will be extracted and saved as CSV data where each row represents the information of one packet capture while columns store the meta and header fields. CSV files are exported to the next step for further process before importing the data into the graph.

C.2.3 Coupling Packet Captures in Communication Links

In the FEAT EXTR stage, the main task to create relationships between graph nodes. Finding a relationship relies on the engineering knowledge of work-cell operations and network flows to couple discrete event records. Such techniques are widely used in treating our testbed data. As an example, we will showcase how to pair distributed packet copies as well as defining transactions that trigger industrial control actions.

The coupling can be classified into two different types. Type I is to couple packet records in CSV files based on their relationships in link transmissions and communication protocols; Type II is to link the external add-on data with the existing graph.

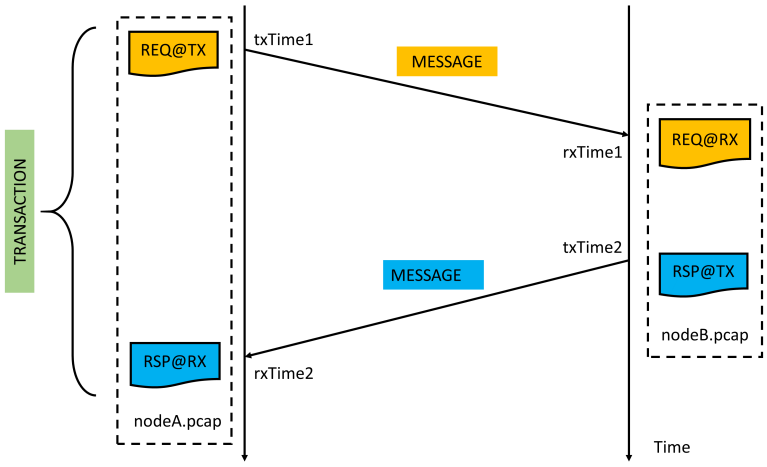


Fig. 59. Timeline illustrations of multiple network captures in a control command transaction

As shown in Fig. 59, we can find the copies of packet captures related with a message transmission in the capture files at both transceivers. As a transaction includes the two-way message exchange, then we have to find four packet captures in order to create two Message nodes and one Transaction node in the graph. The timestamp embedded in each capture record is the main reference information used in the coupling process.

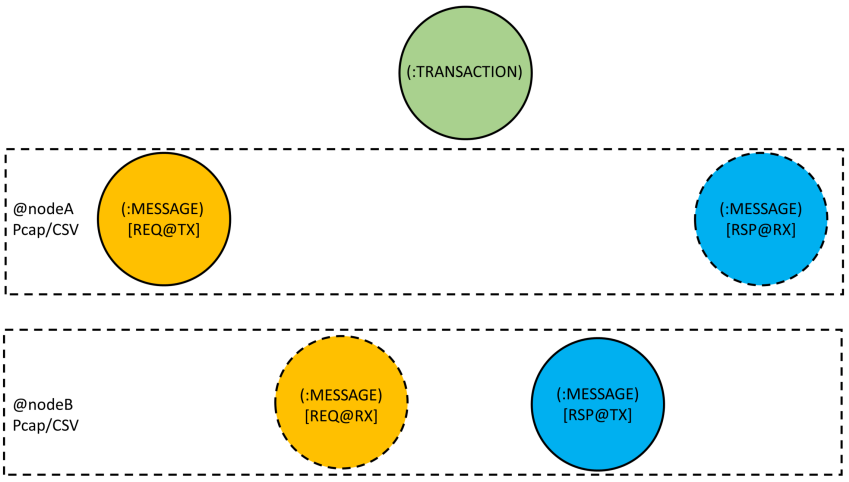


Fig. 60. Graph nodes in a control command transaction

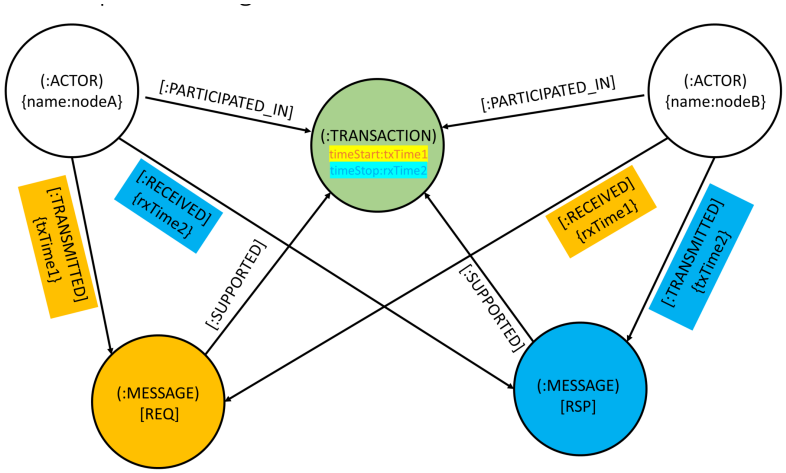


Fig. 61. Graph nodes and the established relationships in a control transaction

Fig. 60 illustrates the graph nodes that are rendered based on the packet captures. Four Message nodes will be created directly from the PCAP data. Accordingly, Message properties will be set by corresponding CSV columns. Note that a Message node is designed to contain the properties related with both the "TX" and "RX" copies of a real message. We will combine the Message copies of the same message, e.g., REQ@TX and REQ@RX, to leave only one node, i.e., REQ@TX. The nodes shown in the dashed circles will be removed from the final graph to reduce the size.

Fig. 61 shows the result of the coupling processing related with a control transaction. Further details at individual steps can be found in Fig. 62.

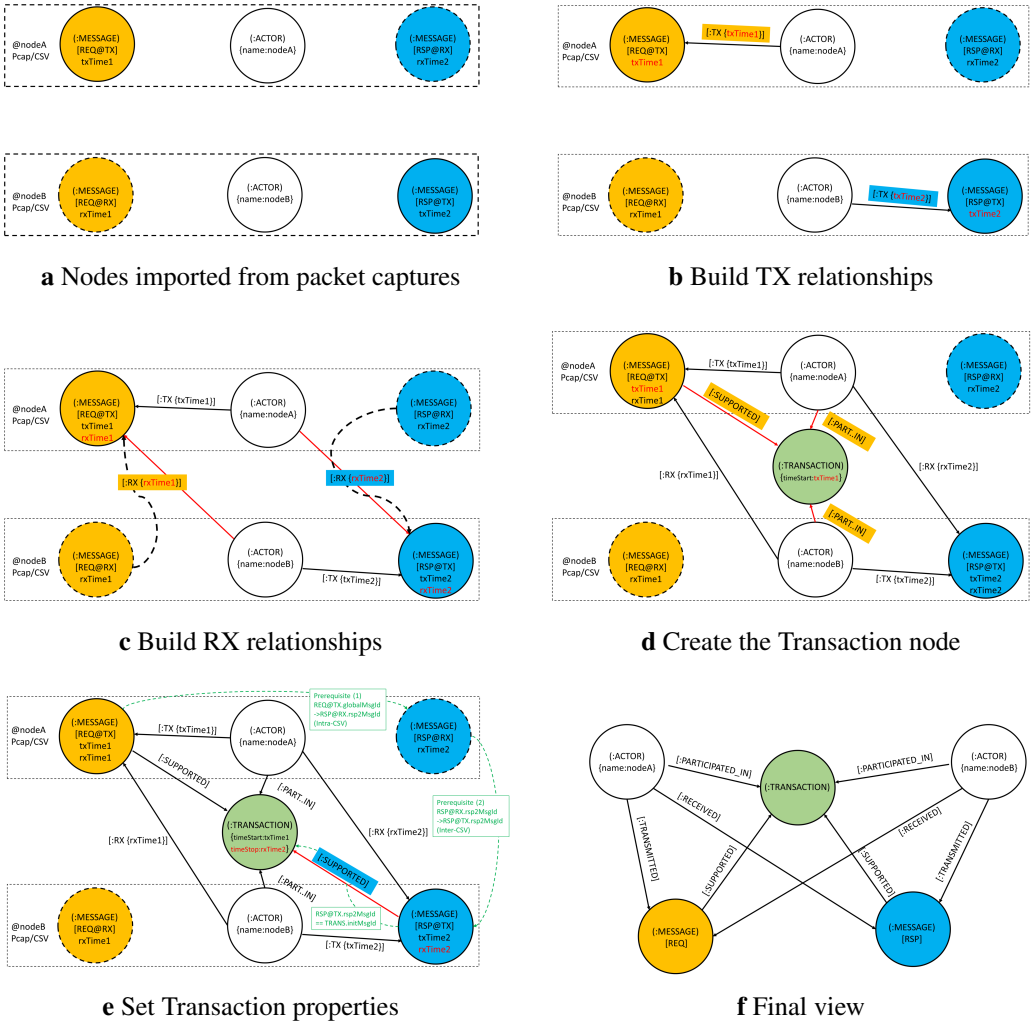
C.2.4 Cypher Queries and Python-Neo4j Interface

Cypher is Neo4j's query language that interfaces with the GDB runtime. With a SQL-like syntax, Cypher also provides drivers in common programming languages, such as C, Python, and Java, which use the bolt protocol.

In the GDB-related operations, we use Python 3.6.6 to send Cypher queries to the running graph in Neo4j Desktop 3.5.6 which returns the requested data for further analysis. There are a few available Python packages on the Python-Neo4j interface, such as neo4j and py2neo. Their main difference is the format of embedding Cypher queries in a Python script.

For example, the Python code using ph2neo can be written in the following way.

```
from py2neo import Graph
g = Graph("bolt://localhost:7687", auth=('neo4j', '1234'))
query = """
MATCH (p:Message)
RETURN p.msgId as ID, p.ipSize as Size
limit 3
```



a Nodes imported from packet captures

b Build TX relationships

c Build RX relationships

d Create the Transaction node

e Set Transaction properties

f Final view

Fig. 62. Coupling procedures to pair network captures in a communication transaction.

```
"""
a = g.run(query).data()    % return the query data as a list of
                                dictionaries
a = g.run(query).stats()   % return the query statistics
```

In the above sample code, the query asks for the "msgId" and "ipSize" information of at most 3 Message nodes from the graph. A Neo4j Graph object is first initialized with the graph address and authentication information. The Cypher query is then set as a string variable and sent to the database by the run() function. The data() functions returns the query result as a list of dictionaries. Each dictionary, i.e., one matched Message node out of the returned 3, contains key-value pairs, e.g., the "ID" and "Size" keys which are the alias of the requested properties, respectively.

D. Testbed Equipment Specifications

Table 11. Testbed equipment hardware and software specifications. Numbers in the parentheses denote the quantities of equipment components.

Equipment	Hardware		Software	
Central data collector (1)	hostname	RIVA	OS	Ubuntu 18.04.1 desktop 64-bit
	CPU	Intel Xeon E5-2620 2.4 GHz (1)	UR3 TCP/IP client	RTDE Python driver
	Memory	16 GB DDR4 2400 MHz (4)	Packet capture & measurement	TShark
	Hard drive	1 TB SSD (1)	Time sync	linuxptp
	Network	Gigabit Ethernet ports (10)		
Data processor & GDB server (1)	hostname	Fishbone2	OS	Windows 10 Enterprise 64-bit
	CPU	Intel Xeon E-2186G 3.8 GHz (1)	GDB server	Neo4j Desktop 3.5.6
	Memory	16 GB DDR4 2666 MHz (4)	Python	3.6.6
	Hard drive	1 TB SSD (1)		
	Network	Gigabit Ethernet ports (2)		
Collaborative robots (2)	Make/Model	Universal Robots UR3 6-DoF CB-series [32]	Software	3.12.0
	Peripheral	Robotiq 2-Figure 2F-140 gripper (1) [33]		
	Peripheral	OnRobot 6-axis HEX Force Torque sensor (1) [34]		
Supervisor PLC (1)	Make/Model	Beckhoff CX2020 [35]	OS	Windows Embedded Standard (WES) 7
	Terminal	Beckhoff EL6688 external sync interface (1) [30]	PLC IDE	Beckhoff TwinCAT 3.1 Build 4020.28
	Terminal	Beckhoff EL1808 digital input (1)	TC3 libraries	TC1200, TF1800, TF6421
CNC emulator PLCs (4)	Make/Model	Beckhoff CX9020 [36]	OS	Windows Embedded Compact (CE) 7
	Terminal	Beckhoff EL1808 digital input (1)	PLC IDE	Beckhoff TwinCAT 3.1 Build 4022.14
			TC3 libraries	TC1200
Ethernet-WLAN adapter /WLAN sniffer (7)	Make/Model	Shuttle DL10J NUC "Intel NUC" [37]	OS	Ubuntu 14.04 server, Linux kernel 4.10, 64-bit
	CPU	Intel Gemini Lake J4005 Dual Core	Packet capture& measurement	TShark, Tcpdump
	Memory	4 GB DDR4 2400 MHz (2)	Traffic generation	Iperf v2.0.14a
	Hard drive	250 GB SSD (1)	Time sync	linuxptp
	Network	Gigabit Ethernet port (1); Intel Wireless-AC 9560 (1)		
PTP-capable switches (2)	Make/Model	Cisco IE-4000-8GT4G-E	IOS	ie4000-universalk9-mz.152-4.EA5
	Network	Gigabit Ethernet ports (12)		
Non-PTP switches (2)	Make/Model	D-Link 8-port DGS-108		
	Network	Gigabit Ethernet ports (8)		
Wireless AP (1)	Make/Model	Netgear AC1900 WLAN router R7000 (1)	Firmware	DD-WRT V3.0-R40559
TAP devices (7)	Make/Model	Shark TAP Gigabit network sniffer		
Time server (1)	Make/Model	Meinberg M900/GPS PTPv2 grandmaster [38]	Firmware	6.24.014
	Network	NTP (1); PTP (1)		

E. Network Diagrams in Testbed Experiments

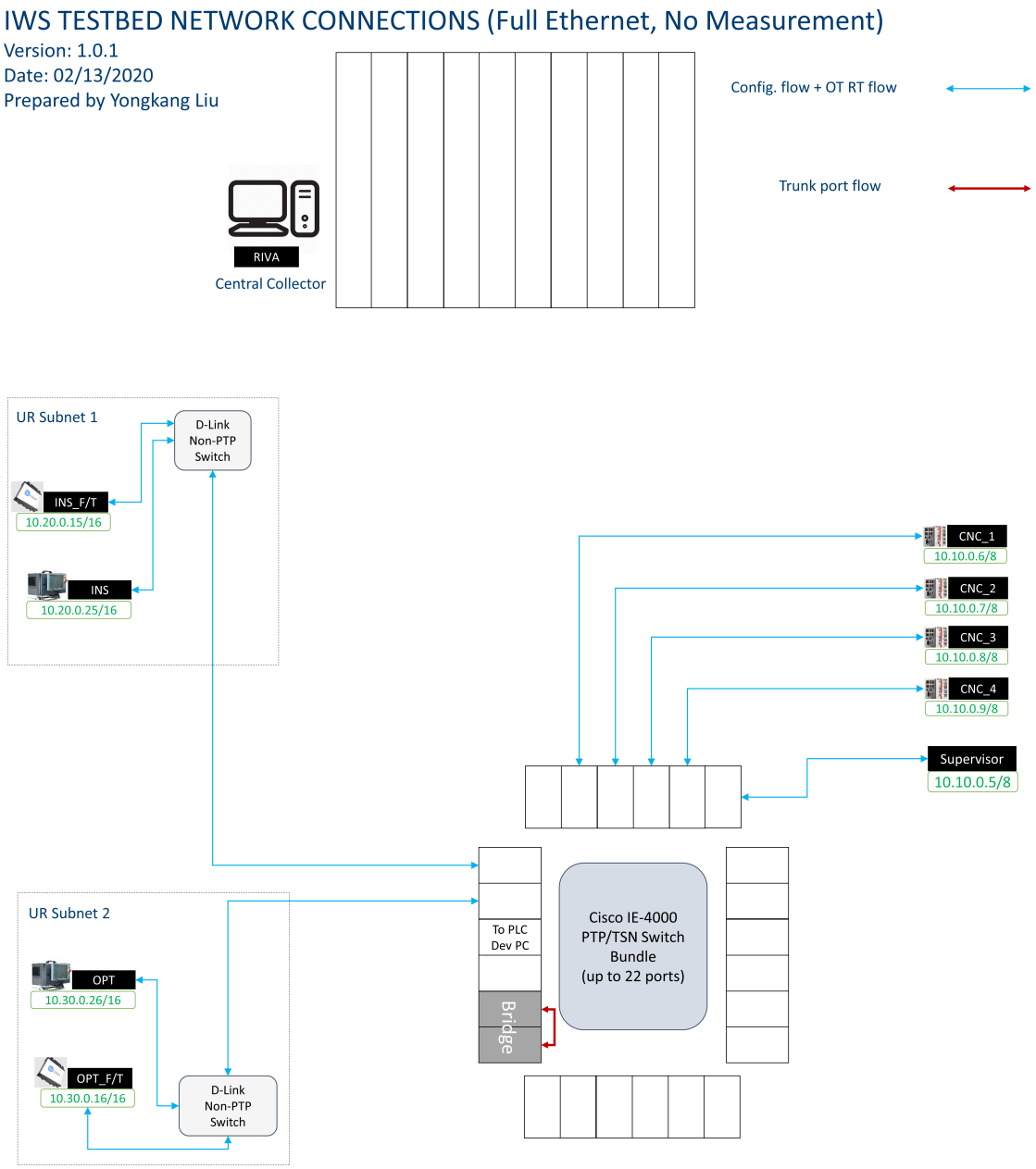


Fig. 63. Testbed network diagram with only work-cell connections

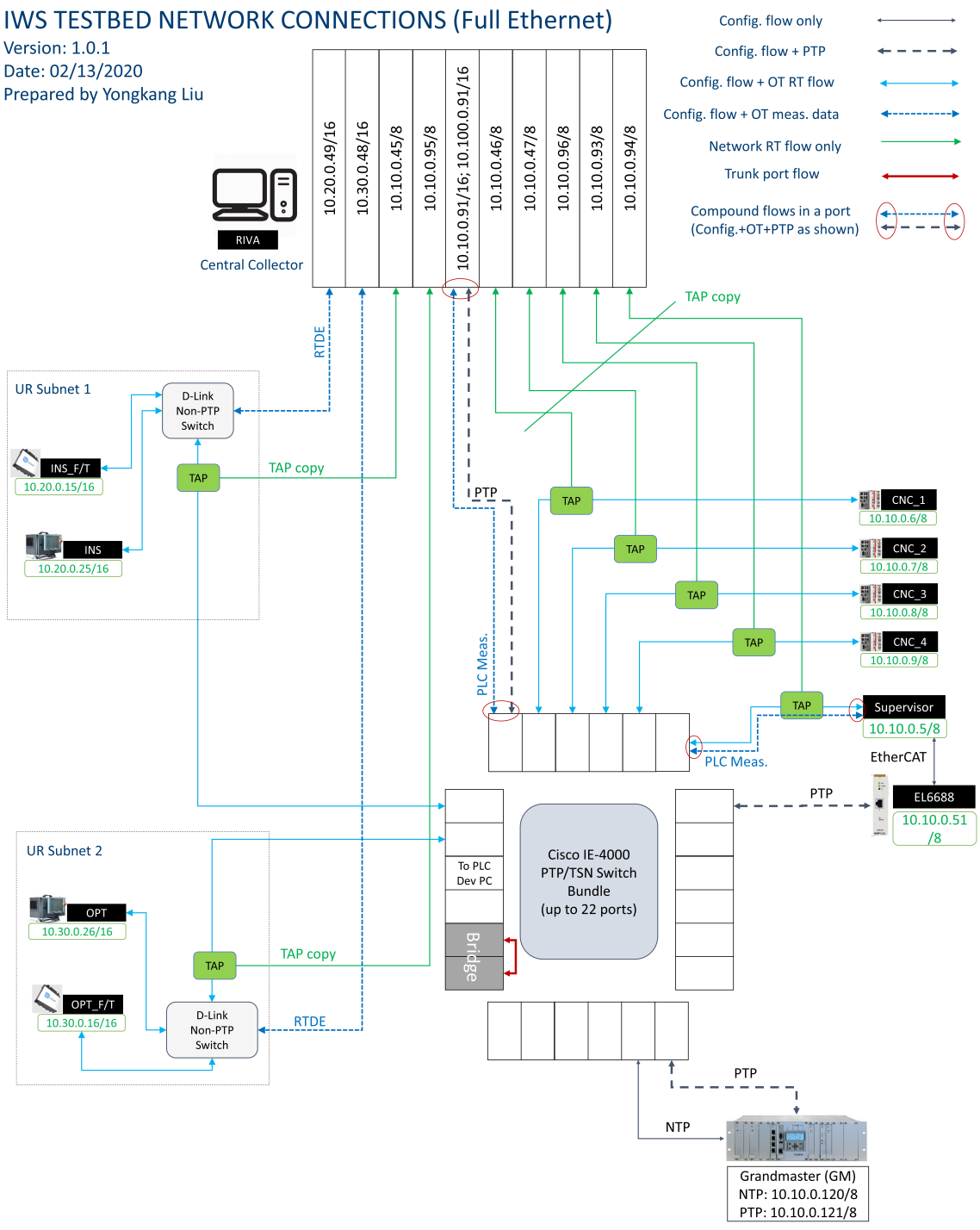


Fig. 64. Testbed network diagram with full wired connections and measurement data links

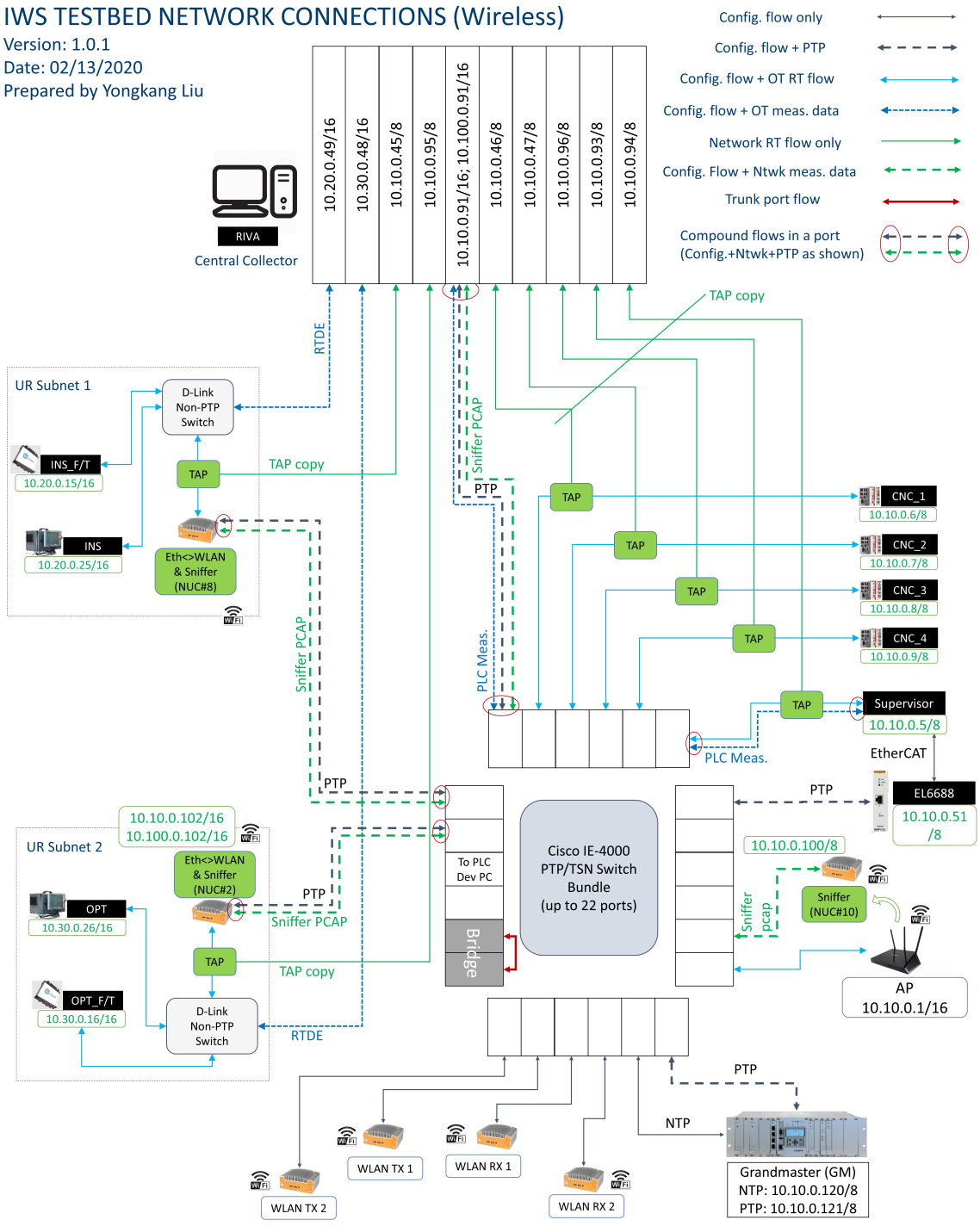


Fig. 65. Testbed network diagram with wireless connections

F. Acronyms

5G	The fifth-generation
ADS	Automation device specification
AGV	Automated guided vehicle
AMS	Automation message specification
AP	Access point
ATI	International atomic time, or “temps atomique international” in French
BC	Boundary clock
CLI	Command-line interface
COMM	Communication
CPS	Cyber-physical system
CPU	Central processing unit
CSV	Comma-separated values
DC	Distributed clock
DDR4	Double data rate 4
DIAG	Diagnostics
DoF	Degree-of-freedom
E2E	End-to-end
EDT	Eastern daylight time
ERP	Enterprise resource planning
EST	Eastern standard time
EtherCAT	Ethernet for control automation technology
F/T	Force torque
FEAT EXTR	Feature extraction
GDB	Graph database
GDB INSRT	Database insertion
GRF TUN	Graph tuning
GUI	Graphic user interface
GVL	Global variable list
HDD	Hard disk drive
HIL	Hardware in the loop
HMI	Human-machine interface
HW	Hardware
I/O	Input and output
ID	Identification
IDE	Integrated development environment
IEEE	Institute of Electrical and Electronics Engineers

IN	Input
INIT	Initialization
INS	Inspector
IOI	I/O module interface
IP	Internet protocol
ISO	International Organization for Standardization
IT	Information technology
IWS	Industrial wireless systems
LAN	Local area network
MAC	Medium access control
MB	Megabyte
MBAP	Modbus application protocol
MES	Manufacturing execution systems
NIC	Network interface cards
NIST	National Institute of Standards and Technology
NoSQL	Non-relational or non-SQL
NTP	Network time protocol
NUC	Next unit of computing
OC	Ordinary clock
OPT	Operator
OS	Operating system
OT	Operational technology
OUT	Output
PCAP	Packet capture
PDU	Protocol data unit
PER	Packet error rate
PHC	PTP hardware clock
PHY	Physical
PLC	Programmable logic controllers
POU	Program organization units
pps	Packets per second
PREPROC	Preprocessing
PTP	Precision time protocol
PTPv2	Precision time protocol version 2, also known as IEEE 1588-2008
QoS	Quality of service
RDP	Remote desktop protocol
RSSI	Received signal strength indicator

RTDE	Real-time data exchange
RX	Reception or receiver
SCHDL	Scheduler
SMS	Spectrum monitoring service
SNR	Signal-to-noise ratio
SQL	Structured query language
SSD	Solid-state drive
STA	State machine
SW	Software
TAP	Test access point
TB	Terabyte
TC	Task clock
TCP	Transmission control protocol
TDMA	Time division multiple access
TPKT	ISO transport services on top of the TCP
TSN	Time sensitive networking
TX	Transmission or transmitter
UE	User equipment
UR	Universal Robots
URCaps	UR robot's extended capabilities, i.e., accessories
UTC	Coordinated universal time
WES	Windows embedded standard
WIA-FA	Wireless networks for industrial automation - factory automation
WLAN	Wireless local area network
XML	Extensible markup language