Improving Network Load Using a Cloud-Edge MAS-Based Architecture for Industrial Safety Applications

Gibson Barbosa, Djamel Sadok, Luis Ribeiro

Improving Network Load using a Cloud-Edge MAS-based Architecture for Industrial Safety Applications

Gibson Barbosa[1][0000−0001−9023−4019], Djamel Sadok[1][0000−0001−5378−4732], and Luis Ribeiro[2][0000−0002−0248−8180]

[1] Networking and Telecommunications Research Group (GPRT), Federal University of Pernambuco, Brazil
{gibson.nunes,jamel}@gprt.ufpe.br
[2] Division of Product Realization, Department of Management and Engineering, Linköping University, 581 83 Linköping, Sweden
luis.ribeiro@liu.se

Abstract. Internet of Things, in particular, the concept of Industrial Internet of Things (IIoT), is one of the key technological pillars of the Fourth Industrial Revolution, also known as Industry 4.0. In this context, one of the areas of interest is safety, whereby multiple intelligent sensors may be permanently connected to a central system to autonomously or semi-autonomously identify safety hazards. Vision systems are a popular sensor in the safety domain as they can simultaneously monitor many different safety concerns. However, the continuous video stream transmission and the increasing number of intelligent devices in IIoT networks introduce additional pressure on the network. There is a risk that the network resources may become overloaded. This paper proposes and discusses a reference architecture for identifying safety risks. The architecture allows multiple sensors to be plugged into the system. The input of the different sensors is then dynamically weighed as the risk situation evolves. The architecture explores sensor-level intelligence (at the edge layer) to mitigate the network overloading problem. Edge agents quickly assess the risk, deciding whether or not to forward their signals to a local cloud agent for further processing. The cloud agent can then selectively request more information from other edge agents. The architecture is tested in a use case for operators' safety in the assembly of aircraft components and uses intelligent vision systems as safety devices. In the selected use case, the accuracy of the system and its impact on the network load are assessed.

Keywords: Multi-agent systems · IoT · Industry 4.0.

1 Introduction

The Industry 4.0 [18] is the fourth step in different evolutionary milestones that revolutionized the industrial activities. The Internet of Things (IoT) is a key

enabling concept of Industry 4.0 and a set of technologies that permits sensors, machines, or everything with some processing capability to be connected to a network. However, connectivity alone does not solve any industrial problem, and Industrial IoT frameworks and architectures must provide the flexibility to address many different use cases. One such case is safety [21]. Multi-sensor environments provide an interesting framework for evaluating safety risks in complex scenarios.

A way to do that is using vision systems that can be used to identify safety events. However, the use of more complex devices, such as vision systems, in industrial networks comes at the cost of network load. Because of that, some works like [5], [3] and [23] propose strategies that bring the image processing algorithms closer to the cameras on the edge of the network. This alternative is called Edge Computing [20] [8]. With that, processing can be distributed to devices at the edge of the network, which reduces the need to transmit heavy raw data. However, such systems are dedicated to improving the streaming of image processing. In many contexts, it is interesting to be able to combine data from different sensors in a coherent way.

This work proposes a reference architecture that considers a Multi-Agent System (MAS) structure for information exchange between agents in low-end Edge devices and high-end Servers. Each agent represents a processing unit in the Edge computing layer that can process locally or pass along the data to a server in a cloud for further heavy-duty and high accuracy processing. This proposal is thought to reduce the overload in IIoT networks avoiding unnecessary data being transmitted to the centralized server. A safety-related use case is used to show-case the proposed architecture, and the architecture contributes in the following directions (1) a MAS-based reference architecture for IIoT edge-cloud computing, (2) a system to detect collision risks in industrial environments, (3) a multi-camera dataset that describes a human interaction activity in the industry and (4) Evaluation of the utilization of edge devices to retransmit camera stream in a WiFi network.

This work is organized as follow. Section 2 presents the works related to this proposal. Section 3 describes proposed architecture. Section 4 details the use case and prototype implementation. Section 5 discusses the experiments and results. Finally, Section 6 summarizes the main findings of the work.

2 Related works

Human safety using IoT networks is a relatively unexplored domain. The work in [22] applies an IIoT in the mining industry. It monitors and analyzes previous and real-time atmospheric and ground stability information. The system alerts people at risk using alert lights and controls the fan ventilation in the mine using the MQTT protocol. In [15] an IIoT-based industrial monitoring system is proposed. Their IIoT structure has sensor nodes that transmit data to the gateway node, responsible for forwarding the information to a monitoring software in a cloud server using MQTT and AWS.

Some works propose systems that use IoT devices to transmit frames in the network. For example, [6] proposes a raspberry Pi Based video streaming service that can transmit to multiple mobile devices. This system streams to android applications using a JSON structure to transport images. The system successfully supports transmitting video streams for different devices, but the device processing power impacts transfer time and the Quality of Service (QoS). The work [10] applies edge devices to Real-time traffic flow data analysis in an intelligent transport system. It is used for heterogeneous and congested traffic conditions evaluation. The Mez system [7] employs an on-demand video frame transmission from IoT camera nodes to an edge server. The application in the edge server can specify network latency upper bound and accuracy lower bound that the application can tolerate for the transmission of the frames. The boundaries for latency and accuracy are achieved by modifying the video frames, analyzing quality parameters, like resolution, color space, blurring, or reducing the size, with the possibility of artifact removal and Frame differencing. The SlugCam proposed in [1] uses camera agents that transfer information based on pre-implemented computer vision algorithms. It is a solar-powered camera agent that communicates in a wireless network. The user defines the algorithms to extract needed information from images.

Different frameworks also are proposed for the IIoT context. The framework in [17] intends to offer efficient and resilient communication over critical disaster circumstances. It looks to use timely detect abnormal events, locate the event site and workforce, generate alerts to workers and emergency service providers, and guide humans to safe places. They consider a four-layer IIoT network: data acquisition, host computing system, cloud computing, and application. The framework proposed by [13] uses ontologies to automate reasoning and decision-making. The user, who does not need to be an expert in ontologies, can insert safety actions into the processing workflow, which uses the new knowledge in the decisions to mitigate the possible hazards. The system is demonstrated in an application to recommend the best protection equipment in a risky situation.

There are many contact points between these previous works and the work discussed in this paper. However, the present work seeks to evaluate the impact of local processing in network load and discusses an architecture that can potentially scale to a large number of sensors that are selectively enabled over a network to improve the detection accuracy of events of interest. The architecture is also not restricted to specific use cases and offers a modular and re-configurable structure that allows users to quickly customize a system to change conditions or the addition and removal of new sensors and actuation devices.

3 Architecture

Fig. 1 presents an overview of the architecture. The architecture assumes that sensors have their computational infrastructure or can be connected to an edge computing device. The architecture also assumes a network infrastructure that

makes computational resources with high computational capabilities available in a conventional cloud/edge setup.

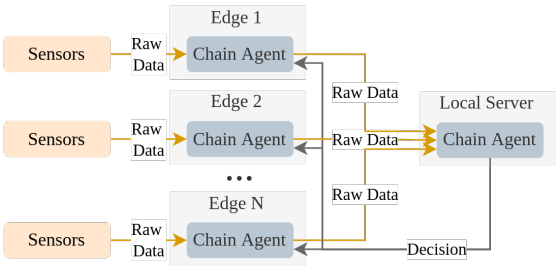

Fig. 1: High-level visualization of the Architecture for multiple Edge devices.

The Chain agent in Fig. 1 is the main active entity in the architecture. Chain agents can be located at edge or cloud levels and are responsible for processing relevant system information. Edge chain agents pre-process and evaluate raw data collected directly from one or several sensors and decide whether said data must be forwarded up the chain for a more accurate evaluation. Later, the Cloud chain agent will perform a more thorough data analysis and make a decision on the system. Such decisions may include: acting upon the system, asking additional Edge agents for data, and informing Edge agents that decisions have been taken and their data is no longer needed.

The general idea is that the edge agents will act as the system's first responders providing a continuous, local and quick evaluation of a developing system situation. Therefore, the edge layer is a filter that limits potentially unnecessary network traffic.

Even if the use case, later detailed, explores risk identification in an industrial environment, the architecture itself applies to many other scenarios where network utilization needs to be balanced in the presence of sensors that require high network bandwidth. In this context, chain agents, their processing and actions are configured through description files and use specific interaction interfaces enabling the creation of more complex decision chains. Holonic aggregation of chain agents is therefore possible as depicted in Fig. 2 providing additional flexibility in other domains.

The communication between the chain agents uses a publish/subscribe pattern whereby the agents lower in the hierarchy subscribe to the topics of agents higher up. MQTT (Message Queuing Telemetry Transport) is the protocol used with the current implementation of the architecture using the Mosquitto server [11].

The internal architecture of chain agents is depicted in Fig. 3 and is composed of the following modules: Interfaces, Rx Window, Ensemble, Decision, and Attention. As earlier mentioned, the behavior of the agent and particularly of its modules is configured by the description file.

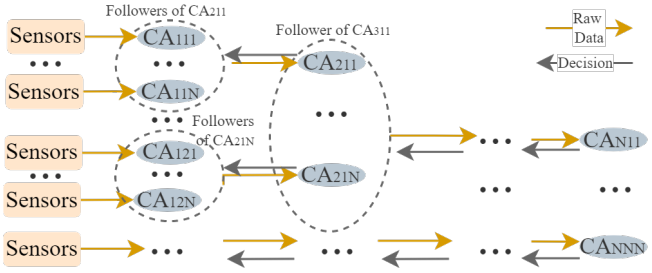

Fig. 2: Architecture multilevel chaining.

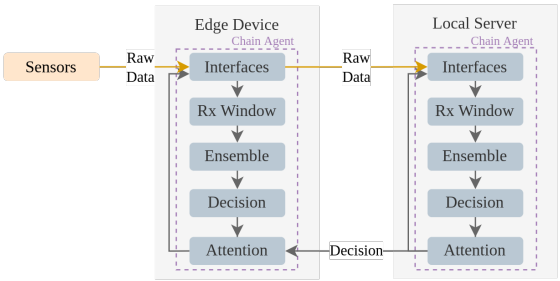

Fig. 3: Reference architecture internal modules interaction.

Interfaces, as the name suggests, are software integration mechanisms for the inclusion of new sensors and actuators in the system. The integration code for the new sensor, or actuator, must implement a specific software interface defined by the platform. Through such an interface, the raw data from the sensor becomes harmonized with the information processing and execution semantics of the agent. In the case of actuators, system actuation information is converted to native commands. Interfaces need to be created only once for each new sensor or actuator integrated into the architecture. The interface module then channels the data internally to other modules, forwards it further up the chain, or actuates on the system. Interfaces, even for the same device, may expose/make available several variables of interest for the system.

The *Rx window module* is responsible for processing the data generated by different sensors and reaching the chain agent through the different interfaces. Different sensors will have different data throughput rates. For example, in a sensor outputting data with a frequency of 10 Hz, the Rx window module guarantees that every 10Hz that data from the sensor is processed. As mentioned, this module manages all the available interfaces. By defining the size of the data reception window, it is possible to control the frequency at which data is forwarded and analyzed by the subsequent modules in the chain agents' internal architecture. The reception window is configurable, and hence the user can decide, given the sensor available in the system, how much time the system should wait to compile data from the different sensors. Due to different throughput

rates, different sensors will buffer a distinct number of data points in the same reception window.

The *Ensemble module* is responsible for summarizing all the information received in a given data reception window. The presence of multiple sensors with different data throughput rates may give rise to conflicting information. The purpose of this module is, therefore, to try to decide on what is, most likely, the status of the system, given the data received on each reception window. Internally the module takes a two-step approach. First, it decides on the information coming from each interface. Different interfaces may have buffered a different amount of data points for the same window. With a decision on the value for each interface, then the module attempts to fuse the information from all the different interfaces. The fused information is then forwarded to the decision block for further processing. Many different strategies can be considered for fusing information. In the present case, to demonstrate the feasibility of the architecture, a set of simple mechanisms are considered. In the first step, the current implementation considers values as literals and uses majority voting for fusing the data. On the second step, a unanimous vote in a specif direction is required. For example, in the use case later explained, the edge chain agent can only claim that there is no risk if all the interfaces unanimously vote for no risk; otherwise, the second stage evaluation defaults to the presence of risk. The ensemble fuses the information based on the variables made available by the interfaces. If different interfaces provide data related to the same variable, the ensemble guarantees that the different sources are fused. Therefore, the ensemble's output is the fused value for all the variables exposed by the interfaces.

The *Decision module* receives the response from the *Ensemble* and applies user-defined decision rules. Such rules are specified in the description file. The outcome of the decision module is a set of user-specified actions. These local actions can only be applied to the system if a higher-order chain agent does not create an overriding decision. The decision from higher-order layers invalidated the local decision. The attention module manages which action ends up taking effect in the system.

As mentioned, many of the actions of the chain agents can be configured by the user through the usage of Description Files. An excerpt of a description file, which will be later detailed, can be found in Listing 1.1. Such files are JSON documents where the following parameters need to be defined.:

- **id**: represents the agent identification.
- **leader**: the id of an agent upper in the hierarchy if there is one; otherwise takes the value "none".
- **rxperiod**: the value of the reception window for the Rx Window module, which needs to be regulated as a function of the throughput rate of the sensor present in the system and the desired reactivity of the edge chain agents.
- **mqtt**: the network information of the MQTT server supporting system communication.

- **interfaces**: the names of the classes containing the software integration interfaces to be used by the chain agents for determining the value of the different variables of interest.
- **perceptions**: represents the variables' names the interfaces have to use to send information to the chain agents. It is a structure where each key is the name of the variable, and the value for that key is an array of strings that represents the values that an interface can send when using that variable.
- **actions**: represents the variables' names the interfaces have to use to receive information from the architecture. It is a structure where each key is the name of the variable, and the value for that key is an array of strings representing the values that an interface has to implement to receive data.
- **rules**: represents the decision rules that will be triggered by the system. It is a list of structures with the "if" and "then" tags. The "if" and "then" assumes a structure with one or more components. The components of the "if" need to have the names of the variables defined in perceptions as keys. The value for that key has to be one of the values defined for that variable. In the "then" tag, it has to assume one of the variables defined in the actions and put the value that this variable assumes when the "if" conditions are triggered. The value has to be in the defined values for that action variable.

In the forthcoming sections, a concrete application case of the proposed architecture is discussed in the context of an industrial safety application.

4 Use case

4.1 Demonstration Scenario

The instantiation of the architecture was demonstrated in a mock-up scenario that simulates assembly operation in an aircraft cargo door. Due to the nature of some operations, only one person may be in the vicinity of the cargo door, a safety risk arises otherwise. In the current scenario, whenever a second person is detected in the vicinity of the cargo door, the system exhibits a message and a sound noticing that. However, more appropriate real-world actions would include activating emergency stops in motorized tools in operation, alerting the operators, raising alarms, etc.

The current system is monitored by four security cameras connected to a network and placed around the cargo door (Fig. 4). Standstill pictures from the live stream of the four cameras can be seen in (Fig. 5). In this case, the four cameras correspond to four inputs in the system and four sources of information for the same set of variables. The previous creates a scenario of information redundancy common in safety applications. Even if the scenario explored relates to a safety application, the authors would like to stress that the system, as is, does not meet any criteria for real-world usage (i.e., is not a safety system), and it serves only the purpose of demonstrating the architecture which also has a much broader potential applicability than just the one considered.

In the scenario considered, one operator is working on the cargo door when a second operator invades the scene and comes in contact with the first. The system will work with the live stream from the cameras, but for the purpose of analysis and comparison of results, the scenario was also recorded into a 30 seconds dataset. In the first 16 seconds, Person 2 goes around the aircraft cargo door. Then, it collides with Person 1 for 10 seconds. Finally, Person 2 walks away for 4 seconds. Each camera perspective is recorded for the dataset at 10 fps and a 320x320 pixels resolution.

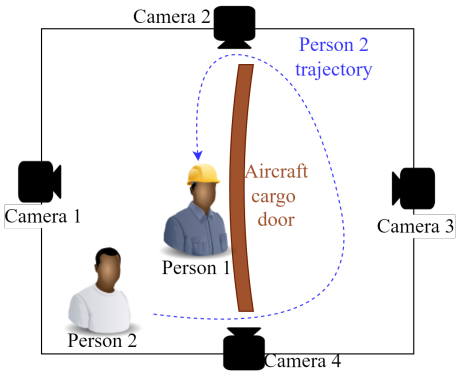

Fig. 4: Scenario.

Fig. 5: Scenario visualization from different cameras' perspectives.

The technical infrastructure is summarized in Table 1. There are four Edge devices (raspberry pis), a WiFi router, and a Local server machine. Each camera is directly connected to an Edge device by cable using the Ethernet interface. The Edge communicates with the Server via a WiFi network (802.11n).

Table 1: Specification of the devices in the scenario.

Device	Model	Specification
Camera	Foscam R2M	Resolution: 1920 x 1080 (2.0 MP), Frame rate: 25fps (1080P)
WiFi Router	D-Link GO-RT-N300	Standard: 802.11b/g/n, Speed: 300 Mbps 2.4 GHz
Edge	Raspberry Pi 3 B+	Processor: Quad Core 1.2GHz Broadcom BCM2837, RAM: 1 Gb, SO: Raspberry Pi OS 32bit
Server	HP Workstation Z2 G5	Processor: Intel(R) Xeon(R) W-1250 CPU @ 3.30GHz, RAM: 16GB SSD, SO:Ubuntu 20.04.3 LTS GPU: NVIDIA Quadro RTX 4000 8GB

4.2 Interfaces and Description Files

Specific interfaces were defined for this scenario (Fig. 6). As mentioned, integration interfaces guarantee the harmonization of raw sensor/actuator data with the execution semantics of the system. At the edge device, the chain agent has the interface *H-H Collision Edge* which allows the system to receive data from the cameras and detect collisions. The second interface (*Transfer Frame*), working in tandem with the first, allows redirecting the video frames up the chain for further processing. The Cloud chain agent in the server also has a collision detection interface (*H-H Collision Server*) with a more accurate collision detection algorithm in comparison to the edge device and an interface to raise alarms (*Print State*) when a collision between two operators is detected.

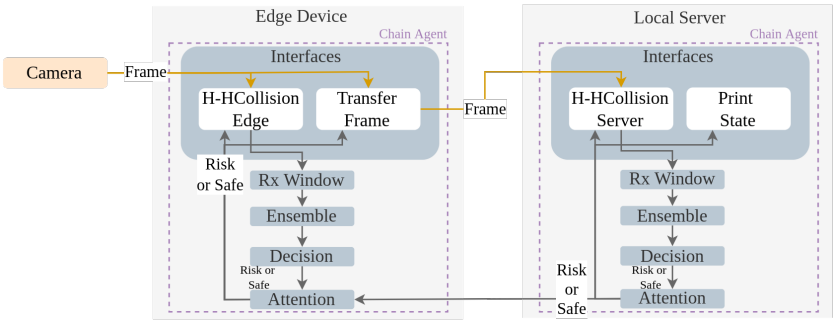

Fig. 6: Reference architecture with the interfaces for the specific application.

The Description File for this system is presented below in the Listing 1.1. The id of this edge device is "edge1". His leader, up the chain, has the id "server1". It considers a window time to receive data of 0.1 seconds. The MQTT server is running in a machine with IP: "192.168.0.6" and port: "1883". It executes the interfaces "HHCollisionEdge" and "TransferFrame". There is only one possible perception variable, that is called "collision", this variable can assume two literal

values, "true" or "false". Also, there is only one possible action variable, that is called "risk", that can be "true" or "false". For the last, there is two rules. The first rule says that if a collision is true ("if": {"collision": "true"}), then there is a risk ("then": {"risk": "true"}). The second rule says that if a collision is false ("if": {"collision": "false"}), then there is not a risk ("then": {"risk": "false"})

Listing 1.1: Description File for edge agent.

```
{"id": "edge1",
 "leader": "server1",
 "rxperiod": "0.1",
 "mqtt":{"ip":  "192.168.0.6", "port": "1883"},
 "interfaces": ["HHCollisionEdge", "TransferFrame"],
 "perceptions": {"collision": ["true", "false"]},
 "actions": {"risk": ["true", "false"]},
 "rules":[
    {"if": {"collision": "true"}, "then": {"risk": "true"}},
    {"if": {"collision": "false"},"then": {"risk": "false"}}] }
```

The implementation of the interfaces is discussed now. The Transfer Frame that is in the Edge device begins to transmit the frames to the server when the decision received is {"risk":"true"}. It uses the OpenCV library [4] to read the frames and applies the ImageZMQ library [2] to transmit OpenCV frames to the server. Otherwise, it does nothing. The Print State interface, in the Local Server, is a visual feedback to print in the screen the safety state in the environment, so it prints Risk when receives {"risk":"true"}, else it prints Safe.

The H-H Collision is in the edge device and the Local Server. The difference between this interface in these devices is only the applied model for object detection and how it reads the frames. In the Edge devices, it uses a simpler model that requires low memory and processing power to return responses, that is, the Mobilenet v2 [19]. To read the frames in the Edge device, it uses the OpenCV library. For the Local Server, it uses the YOLOv3 base version [16] that requires more memory and processing, but it is more precise than the Mobilenet v2. To read the images, the Server uses the ImageZMQ library.

Fig. 7 illustrates the process to make the collision detection. First, an object detection model is applied, which returns an array with the boxes for all the objects identified in the frame. From the identified objects, the boxes with humans are selected. With all the human boxes, the algorithm now evaluates the collision. For that, it evaluates the overlap between each two by two combinations of human boxes. The calculus of the overlap is based on the Intersection over Union (IoU) metric. However, instead of the Union of boxes, it applies the smallest human box in the image as a reference because a smaller box completely covered by a bigger box has to return the maximum level of possible contact. This represents a short person in front of a taller person. The use of IoU would only return the proportion of the smaller box related to the bigger one. Therefore, the overlap is calculated using the Intersection over Smaller (IoS), Eq. 1 refers to the adopted IoS metric:

$$IoS(x, y) = \frac{H_x \cap H_y}{\min(H_x, H_y)} \tag{1}$$

H_x and H_y are the boxes of the humans x and y, respectively, where $x \neq y$.

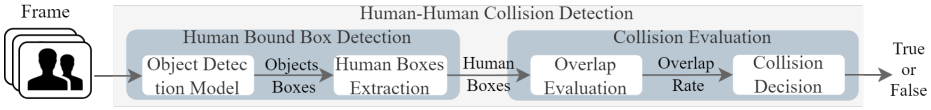

Fig. 7: Human-Human Collision detection module.

With the IoS for each pair of humans in the scene, the Overlap Evaluation step returns the largest IoS among all the pairs evaluated. This represents the Overlap Rate (OR) for that frame. It is calculated following the Eq. 2.

$$OR = \max_{\forall x,y \in P, x \neq y} IoS(x, y) \tag{2}$$

Where P is the set of all People identified in the frame.

For the last, the collision decision considers a pre-established threshold. If the OR is equal or is above this threshold, it returns that the collision is True for that frame. Whereas if the OR is below the threshold, it is False for collision. In the application, we consider a threshold of 5% to be considered a collision.

4.3 Technical Execution Flow

The execution starts with the camera sending a frame to the Edge device. The H-H Collision Edge module process this frame and identifies if there is a collision. It outputs the result to the Rx Window that receives all the detections for the pre-defined window period. Then it passes on all received data to the ensemble. The ensemble converts it into one single perception, with the variable "collision", then forwards it. The Decision step receives the collision information and applies the rules defined in the description file. If there is a collision, it returns that there is a Risk; else, it is Safe. Considering that the server did not send any message, the attention only passes along the Edge decision to his interfaces. If the decision is Safe, the H-H collision Edge continues processing more frames. However, if the response is risk, the H-H collision Edge stops the processing, and the Transfer Frame starts the processing, sending all the received frames to the server.

When the server receives the frames from the edge device, it does the same process reported in the previous paragraph. However, with one difference, the attention output is also passed on to the Edge device and the Print State interface, which prints in the local server if there is risk or not.

The attention module in the Edge now uses the Server decision to be passed along to the local interfaces when the server sends the decision of Risk or Safe to the Edge. If the server had sent risk, the Transfer Frame interface keeps sending the frames, and the H-H collision Edge keeps stopped. However, if the server reports that it is Safe, Transfer Frame stops, and the H-H collision Edge restarts to process.

5 Experiments and results

This section discusses the results obtained while considering the use case in Section 4 and the dataset generated and mentioned therein. The analysis focuses on the impact of the solution on the network traffic with and without the edge devices, as well as the accuracy for the architecture application on the selected use case.

5.1 Network impact

The equipment defined in Table 1 was setup in different ways according to Fig. 8 to validate the extent to which the edge/cloud architecture proposed impacts the performance of the network. The difference among the configurations is mainly how the devices (camera or Edge device) are connected to the router. In the first case, the camera is directly connected to the router using an Ethernet link (IEEE 802.3 Local Area Network). The second configuration uses a wireless camera communicating in the multi-antenna high data rate 802.11n standard. Finally, the third configuration connects the camera to the Edge device Ethernet interface. In turn, the edge device wirelessly connects to the server. TCP is the underlying transport protocol to exchange flows among the devices.

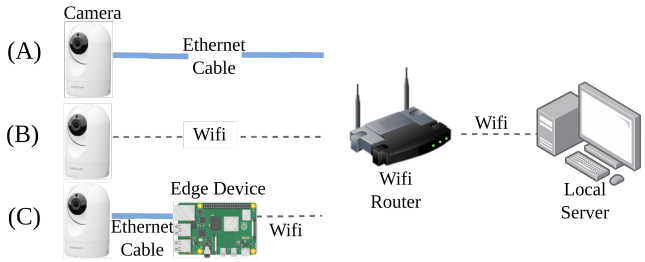

Fig. 8: Configurations for the communication devices used for the network experiment.

The following evaluation metrics are considered: Acknowledgement Round Trip Time (ARTT), packet loss, and the number of exchanged packets. The ARTT is calculated as the time the cameras or edge devices take to respond with an acknowledgment (Ack) for packages sent by the local server. As the cameras do not implement a network interface that can be easily monitored, all packet level measurements are carried out at the server´s network interface. The lost packets metric refers to the level of packets that are dropped during their transmission. We also compute the number of packets successfully received by the server. The higher this is, the more packet processing overhead the CPU requires to handle such traffic. The following parameters are considered to evaluate

network impact: frame resolution 640x640 pixels at 15 fps for a 10 min transmission period. The number of cameras in the experiment is increased from 1 to 4.

A video transmission resolution of 640x480 pixels and a frame rate corresponding to 15 FPS were adopted. These represent upper bounds that allow the processing of frames by edge devices in the experiments. An experiment lasts 10 minutes, at the end of which a network trace is captured using the tcpdump [9] [14] network packet analyzer running at the server machine. This duration is estimated as sufficient to extract meaningful insights and identify possible transmission patterns. Running longer experiments has shown no advantages and generates unnecessarily big packet capture files. The number of cameras is gradually increased from 1 to 4 cameras. Observe that in the case of configuration C (Fig. 8), increasing the number of cameras also increases the number of the Edge devices, as each camera is directly connected to an edge device.

Table 2 collects the results generated by the suggested experiments. It shows that the inclusion of Raspberry Pi as an edge device considerably impacts the ARTT. Observe that ARTT deteriorates even further when increasing the number of these devices. Setups A and B (i.e., without the use of Raspberry devices) exhibit no significant difference for this metric. Every time a new device is included, there is a corresponding increase of at least 0.288 ms that is observed in the case of experiment configuration C. Also, the higher standard deviation obtained by the edge device-based configuration shows that introducing additional devices introduces new outliers and increases delay.

Table 2: Network results.

Qty	Conf	ARTT/ packet (ms)	Packet qty	Packet loss	% loss	Qty	Conf	ARTT/ packet (ms)	Packet qty	Packet loss	% loss
1	A	0.03±0.18	63746	0	0.00%	3	A	0.03±0.24	217571	0	0.00%
	B	0.04±0.21	69088	258	0.37%		B	0.03±0.20	194207	384	0.20%
	C	0.08±2.82	296604	398	0.13%		C	0.65±12.01	513148	1565	0.30%
2	A	0.03±0.02	142637	0	0.00%	4	A	0.03±0.19	294433	0	0.00%
	B	0.03±0.21	125440	400	0.32%		B	0.04±0.49	270021	568	0.21%
	C	0.36±8.70	392886	1091	0.28%		C	1.09±17.88	602576	2589	0.43%

The number of packets increases by approximately 75000 for Configurations A and B, while it increases by around 10000 in the case of Configuration C. This last one achieves a more significant amount of packets from a single camera. Although the increase in the number of transmitted packets is linear, more packets are needed for the first connection. The wired connection is loss-free, independently of the number of used cameras. On the other hand, the WiFi configuration suffers some level of packet loss. For example, the configuration with the Edge device suffered a maximum packet loss of 0.43% when using four devices. Note

that this packet loss level does not significantly affect the transmission quality [12].

The obtained results showed that the use of the Raspberry Pi led to a degradation in the network transmission flow compared to other configurations. Nonetheless, it does not result in a significant loss of information for the intended application. Hence considering our application context, the most significant benefit of adopting such devices is their ability to process the video streams locally and relieve the network from their transmission. Our application takes advantage of this result. Most of the time, there is no risk of accidents being detected, and therefore all processing will be local while saving network resources. In the rare event of a safety risk being detected, transmission occurs from these devices.

Additional advantages of using an edge device, such as a Raspberry PI, include the fact that an edge device may select different system configurations. Unlike the used cameras, it may alter frame size and the compression rate parameters. Finally, transmission efficiency may be improved using devices with more processing power and optimizing the installed software libraries, especially those related to the TCP/IP stack.

5.2 Accuracy analysis

This experiment evaluates the impact on the general accuracy when using the proposed architecture. The dataset presented in Section 4.1 is used in this experiment. The accuracy is measured based on the right identification of a Risk or Safe situation in each dataset frame. The experiment compares the results produced when using the Proposed Architecture (PA), with other models: YOLO v3 (YV3), Mobilenet v2 (MN), and Mobilenet v2→YOLO v3 (MN→YV3). The YV3, MN, and MN→YV3 are applied directly to each dataset frame in a single machine (server device described in Table 1). The MN→YV3 first applies the MN in the frame; if this model detects risk, then the YV3 is applied to the frames of different perspectives, with a unanimity ensemble as described in Section 3. The MN→YV3 is near our approach, although it runs on a single computer and has no network between models' execution. The PA is used by applying Configuration C from Fig. 8 for the network communication.

Different cameras are used in this evaluation, considering the positions illustrated in Fig. 4. There is an evaluation for each camera individually, cameras 1, 2, 3, and 4. We also evaluate multi-camera scenarios, one with the four cameras at once (labeled as configuration 1-2-3-4) and the other with the cameras 2, 3, and 4 (labeled as configuration 2-3-4). On all cameras, each frame's resolution is 320x320 pixels, a 10 fps frame rate is considered, and different combinations of models are tested.

The results are evaluated in terms of the metrics: accuracy (ACC), sensitivity (SEN), and specificity (SPE). They are calculated as in the equation 3, based on the classification metrics: True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN).

Also, it is evaluated the mean time per frame and Frames Per Second (FPS) that the models YV3, MN, and MN→YV3 require to process in a single machine.

The MN→YV3, in this case, considers only the MN detection of frames from one single camera perspective for then pass along to be processed by the YV3. Three different machines are considered: the Edge device, the Server device with GPU, and the Server device without GPU. It was executed for the 300 frames of the dataset.

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} SEN = \frac{TP}{TP + FN} \quad SPE = \frac{TN}{TN + FP} \quad (3)$$

Table 3 has the time, and FPS obtained for the execution of different models and processing machines. When using the YV3 model, the Edge device has one frame processed every 4.5 seconds, which does not allow us to run this model in the Edge device for a real-time application. That is why this device uses the MN, which can process 6.3 frames each second. The MN could be applied in the Server to a higher fps. However, for our application is more critical that there we have a more accurate model in the Server, that is the case of YV3.

Table 3: FPS for different models and processing machines.

Model	Edge	Server CPU	Server GPU
YV3	0.223±0.008	9.233±0.423	38.028±2.717
MN	6.306±0.671	102.270±5.327	159.042±9.842
MN→YV3	0.211±0.006	8.432±0.337	32.484±2.098

As the Server CPU process in 9.2 fps with YV3 and Edge with MN is 6.3 fps, The application of at least two Edge devices processing in parallel surpasses the processing of YV3 in a Server without GPU. When the Server has GPU is necessary to have at least 7 Edge devices. However, the network transmission is the limiting in this case, so when processing in the Edge devices, the level of network load evaluated in the Table 2 (configuration C) is avoided most of the time when there is no risk. The FPS of the MN→YV3 is lower than the other models for all devices due to having to process first the MN and then the YV3.

Table 4 presents the results for accuracy experiments. The PA model is the only one with the Standard Deviation once each execution can result in different results due to the network. The others are deterministic models that run on a single computer.

When using only a single camera, the camera 1 perspective has the best ACC of all models, followed by camera 2, camera 3, and the last is camera 4. This is justified, given that camera 1 is positioned tacking the humans side by side, camera 2 and camera 4 have humans occlusion since one human is in front of the other, but for camera 2 this is more favorable once it is a little more inclined to a position that has a better vision of the different human's complete body. Camera 3 clearly does not take the humans interaction, only the passage of the Person 2, which does not represent a collision, so this justifies the low SEN =

Table 4: Accuracy results.

C.	Model	ACC	SEN	SPE	C.	Model	ACC	SEN	SPE
1	YV3	0.96	1.00	0.94	3	YV3	0.68	0.00	1.00
	MN	0.98	0.95	1.00		MN	0.67	0.00	1.00
	MN→YV3	0.97	0.98	0.97		MN→YV3	0.67	0.00	1.00
	PA	0.97±0.01	1±0.01	0.96±0.01		PA	0.68±0	0±0	1±0
2	YV3	0.92	1.00	0.89	4	YV3	0.55	0.05	0.79
	MN	0.67	0.02	0.98		MN	0.63	0.03	0.92
	MN→YV3	0.89	0.71	0.97		MN→YV3	0.60	0.02	0.87
	PA	0.93±0.04	0.85±0.13	0.96±0.03		PA	0.62±0.03	0±0	0.92±0.05
2	YV3	0.82	1	0.734	1	YV3	0.80	1.00	0.70
3	MN	0.617	0.052	0.887	2	MN	0.91	0.95	0.89
4	MN→YV3	0.79	0.711	0.828	3	MN→YV3	0.86	0.98	0.81
	PA	0.82±0.07	0.76±0.17	0.85±0.06	4	PA	0.86±0.05	0.99±0.02	0.8±0.07

0 and SPE=1 for all models, given that it always returns the safe state once it can not see the collision, due to the occlusion caused by the cargo door.

When comparing experiments with multiple cameras (1-2-3-4 and 2-3-4) to experiments with single cameras (1, 2, 3, and 4), the ACC results with single Cameras 1 and 2 surpass those with multi-cameras, being around 11% more accurate for the PA model. However, when using single cameras, they have to trust in their position and be positioned in the best place, while the multi-camera can consider different perspectives and make a unified decision. This makes the multi-camera a good result. Because for example, suppose that the collision occurs in front of camera 3 instead of camera 1. The camera 1 ACC would drop, but the multi-camera model still can keep the ACC as it considers the response of camera 3 too. The configuration 2-3-4 is interesting; once it disregards the camera in the best position, that is camera 1. Moreover, even without it, keep accuracy near the one with 1-2-3-4, except for the MN model.

At first sight, the MN model has the best ACC for most configurations (1, 4, and 1-2-3-4). This is an unexpected result because this model is more restricted, prioritizing processing speed over accuracy. However, the fact is that for Camera 1 all the models have similar results with at most 2% of difference. For camera 4 the MN takes advantage once it most of the time returns the safe state. This improves his SPE, while YV3 commits more mistakes in this case. The models MN→YV3 and CA take advantage of it because they have a previous detection using MN, then apply YV3, so the ACC and SPE are near the MN.

The PA follows the MN→YV3 ACC for all configurations, staying in the standard deviation range. This shows that including the PA does not impact the general ACC of the system.

6 Conclusion

This work presented and discussed the construction of a reference architecture for safety risk identification. The proposal is based on an edge/cloud computing

structure to parallelize sensor processing in an industrial environment and avoid centralizing the information transmission and processing to a local server. One of the primary intentions of the proposed architecture was to alleviate network load and eventually assess whether more mission-critical applications could co-exist and share network resources with other devices. The evaluated scenario showed that, compared to conventional solutions where images are continuously streamed to a cloud infrastructure, the proposed architecture generates more traffic during transmission. However, it does not continuously transmit, using network resources only when needed. Part of the increased traffic can be justified by the lesser efficient protocols used in the current implementation of the architecture and on the choice of edge devices. Still, given that safety occurrences are relatively rare, one can argue that the overall reduction in network load is attainable and concrete.

However, such reduction is not useful if the accuracy becomes compromised. In this respect, the paper has also evaluated the accuracy of different risk detection models. Edge devices must necessarily use simpler detection algorithms due to their limited computational power. The results suggest that it is possible to find a balanced compromise that makes the edge/cloud solution proposed useful. Indeed the higher computational power of cloud resources, particularly using GPU processing, enables very high processing rates. However, these come at the cost of network load. The positioning of the sensors matters naturally, but the results also show the benefits of the sensorial fusion offered by the platform.

Overall, the results suggest that the proposed platform offers an interesting and re-configurable distributed solution for multi-sensor industrial applications.

Acknowledgments

This work was realized with the support of the Conselho Nacional de Desenvolvimento Científico e Tecnológico - Brasil (CNPq); Centro de Pesquisa e Inovação Sueco-Brasileiro (CISB); SaaB; and Division of Product Realization, PROD, Department of Management and Engineering at Linköping University. Under the agreement of CNPq/CISB/Saab scholarships.

References

1. Abas, K., Obraczka, K., Miller, L.: Solar-powered, wireless smart camera network: An iot solution for outdoor video monitoring. Computer Communications **118**, 217–233 (2018)
2. Bass, J.: Imagezmq: Transporting opencv images. https://github.com/jeffbass/imagezmq (2020)
3. Berardini, D., Mancini, A., Zingaretti, P., Moccia, S.: Edge artificial intelligence: A multi-camera video surveillance application. In: International Design Engineering Technical Conferences and Computers and Information in Engineering Conference. vol. 85437, p. V007T07A006. American Society of Mechanical Engineers (2021)
4. Bradski, G., Kaehler, A.: Opencv. Dr. Dobb's journal of software tools **3**, 2 (2000)

5. Chen, J., Li, K., Deng, Q., Li, K., Philip, S.Y.: Distributed deep learning model for intelligent video surveillance systems with edge computing. IEEE Transactions on Industrial Informatics (2019)
6. Filteau, J., Lee, S.J., Jung, A.: Real-time streaming application for iot using raspberry pi and handheld devices. In: 2018 IEEE Global Conference on Internet of Things (GCIoT). pp. 1–5. IEEE (2018)
7. George, A., Ravindran, A., Mendieta, M., Tabkhi, H.: Mez: A messaging system for latency-sensitive multi-camera machine vision at the iot edge. arXiv preprint arXiv:2009.13549 (2020)
8. Hu, Y.C., Patel, M., Sabella, D., Sprecher, N., Young, V.: Mobile edge computing—a key technology towards 5g. ETSI white paper **11**(11), 1–16 (2015)
9. Jacobson, V.: Tcpdump. ftp://ftp. ee. lbl. gov (1989)
10. Khan, A., Khattak, K.S., Khan, Z.H., Gulliver, T., Imran, W., Minallah, N.: Internet-of-video things based real-time traffic flow characterization. EAI Endorsed Transactions on Scalable Information Systems **8**(33), e9 (2021)
11. Light, R.A.: Mosquitto: server and client implementation of the mqtt protocol. Journal of Open Source Software **2**(13), 265 (2017)
12. Mansfield Jr, K.C., Antonakos, J.L.: Computer networking for LANS to WANS: hardware, software and security. Cengage Learning (2009)
13. Mayer, S., Hodges, J., Yu, D., Kritzler, M., Michahelles, F.: An open semantic framework for the industrial internet of things. IEEE Intelligent Systems **32**(1), 96–101 (2017)
14. McCanne, S., Jacobson, V.: The bsd packet filter: A new architecture for user-level packet capture. In: USENIX winter. vol. 46 (1993)
15. Prasad, G.S.C., Pillai, A.S.: Role of industrial iot in critical environmental conditions. In: 2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS). pp. 1369–1372. IEEE (2018)
16. Redmon, J., Farhadi, A.: Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767 (2018)
17. Reegu, F., Khan, W.Z., Daud, S.M., Arshad, Q., Armi, N.: A reliable public safety framework for industrial internet of things (iiot). In: 2020 International Conference on Radar, Antenna, Microwave, Electronics, and Telecommunications (ICRAMET). pp. 189–193. IEEE (2020)
18. Rüßmann, M., Lorenz, M., Gerbert, P., Waldner, M., Justus, J., Engel, P., Harnisch, M.: Industry 4.0: The future of productivity and growth in manufacturing industries. Boston Consulting Group **9**(1), 54–89 (2015)
19. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.C.: Mobilenetv2: Inverted residuals and linear bottlenecks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 4510–4520 (2018)
20. Varghese, B., Wang, N., Barbhuiya, S., Kilpatrick, P., Nikolopoulos, D.S.: Challenges and opportunities in edge computing. In: 2016 IEEE International Conference on Smart Cloud (SmartCloud). pp. 20–26. IEEE (2016)
21. Wu, F., Wu, T., Yuce, M.R.: An internet-of-things (iot) network system for connected safety and health monitoring applications. Sensors **19**(1), 21 (2018)
22. Zhou, C., Damiano, N., Whisner, B., Reyes, M.: Industrial internet of things:(iiot) applications in underground coal mines. Mining engineering **69**(12), 50 (2017)
23. Zhou, Z., Liao, H., Gu, B., Huq, K.M.S., Mumtaz, S., Rodriguez, J.: Robust mobile crowd sensing: When deep learning meets edge computing. Ieee network **32**(4), 54–60 (2018)