Received: 16 December 2017 | Revised: 11 March 2018 | Accepted: 27 March 2018

DOI: 10.1002/spe.2586

RESEARCH ARTICLE

WILEY

Metropolitan intelligent surveillance systems for urban areas by harnessing IoT and edge computing paradigms

Rustem Dautov[1] | Salvatore Distefano[1,2] | Dario Bruneo[2] | Francesco Longo[2] | Giovanni Merlino[2] | Antonio Puliafito[2] | Rajkumar Buyya[3]

[1]Kazan Federal University, Kazan, Russia

[2]University of Messina, Messina, Italy

[3]University of Melbourne, Melbourne, Australia

Correspondence
Rustem Dautov, Kazan Federal University, Russia.
Email: rdautov@it.kfu.ru

Summary

Recent technological advances led to the rapid and uncontrolled proliferation of intelligent surveillance systems (ISSs), serving to supervise urban areas. Driven by pressing public safety and security requirements, modern cities are being transformed into tangled cyber-physical environments, consisting of numerous heterogeneous ISSs under different administrative domains with low or no capabilities for reuse and interaction. This isolated pattern renders itself unsustainable in city-wide scenarios that typically require to aggregate, manage, and process multiple video streams continuously generated by distributed ISS sources. A coordinated approach is therefore required to enable an interoperable ISS for metropolitan areas, facilitating technological sustainability to prevent network bandwidth saturation. To meet these requirements, this paper combines several approaches and technologies, namely the Internet of Things, cloud computing, edge computing and big data, into a common framework to enable a unified approach to implementing an ISS at an urban scale, thus paving the way for the metropolitan intelligent surveillance system (MISS). The proposed solution aims to push data management and processing tasks as close to data sources as possible, thus increasing performance and security levels that are usually critical to surveillance systems. To demonstrate the feasibility and the effectiveness of this approach, the paper presents a case study based on a distributed ISS scenario in a crowded urban area, implemented on clustered edge devices that are able to off-load tasks in a "horizontal" manner in the context of the developed MISS framework. As demonstrated by the initial experiments, the MISS prototype is able to obtain face recognition results 8 times faster compared with the traditional off-loading pattern, where processing tasks are pushed "vertically" to the cloud.

KEYWORDS

big data, cloud computing, distributed smart camera, edge computing, intelligent surveillance system, IoT, smart city, Stack4Things, stream processing

1 | INTRODUCTION

Information and communication technologies (ICT) have been continuously changing people's everyday activities and improving their quality of life. This trend is underpinned by a large variety of "smart" devices increasingly present around us and involving people into socio-technical (decision-making) systems. The resulting fusion of things, information, people, and processes is transforming the world into a complex cyber-physical-social environment. One of the main driving factors of this process is the Internet of Things (IoT), which aims at establishing a global ecosystem of connected intelligent objects that can be (semantically) described, discovered, selected, and grouped to enable novel data-driven business processes and applications.

As a result, a global network of pervasive smart sensing devices on a daily basis generates avalanches of raw data, among which a considerable amount of traffic belongs to streaming video, generated by various kinds of image capturing devices. This renders cameras as one of the main generators of (big) data and traffic on the network. Indeed, hundreds of millions cameras are already installed in modern urban spaces, continuously monitoring the surrounding environment. Goals of this monitoring are manifold, ranging from the increased demand for public safety and continuous surveillance to protect against crimes and terrorist attacks to traffic monitoring, infrastructure, and utility management (eg, parking, ticketing, billing, etc). Relevant statistics report that out of 245 million CCTV cameras operating globally in 2014, more than 20% were network-connected cameras, remotely accessible via the Internet.[1] Moreover, the statistics does not consider personal *mobile* cameras, which have ubiquitously penetrated the market since then. Such mobile cameras may be installed in vehicles (eg dash cams in private cars, public transport, or police cars), unmanned aerial vehicles (eg drones), as well as in personal portable (eg, smartphones, tablets, etc.) and wearable (eg, action cams) devices. Taken together, this plethora of video/image capturing devices can easily reach billions of smart connected cameras in total, millions of which are deployed in urban environments.

This pervasive presence of cameras is underpinned by an increased use of various kinds of surveillance systems, which, despite the global penetration of cameras, seem to remain isolated from each other and closed to the outer world. That is, there often seems to be no mechanisms to share with and retrieve information from other neighbour sources. For example, large stores, business centers, or public organisations typically have their own independently managed intelligent surveillance systems (ISSs) that may even overlap with each other in terms of coverage area, thus generating excessive amounts of redundant non-reusable video footage. This "siloed" vertical approach makes this trend technically unsustainable, that is, as the network of cameras is exponentially growing, sooner or later, a point of "saturation" will be reached, when there will be more cameras than citizens. In this light, it is becoming increasingly important to *govern* this ever-growing collection of ISSs deployed in urban environments, thus aiming at a unified city-wide ISS.[2]

These issues introduce new challenges as to how to manage the growing complexity of surveillance networks and calls for an all-encompassing coordinated approach at a metropolitan scale on the basis of an open shared architecture, thus paving the way for the emergence of a metropolitan intelligent surveillance system (MISS). This novel MISS scenario takes existing challenges of the surveillance application domain to a new level, amplified and extended in a wider urban context. In this respect, main user requirements for surveillance applications can be identified as follows.

- **Performance**: surveillance services are time-constrained, often operating in a real- or near real-time mode, and are required to handle extreme amounts of continuously generated raw data.
- **Security**: data/resources integrity and confidentiality have to be preserved to avoid intrusions and service impairments.

These identify new system requirements for the MISS to be addressed in the urban context, including (1) heterogeneity and interoperability of vision systems and smart cameras, (2) different administrative and networking (subnets, NAT, firewalls, etc) domains, (3) customizability and programmability of smart cameras (eg, remote injection of application logic code), and, last but not the least, (4) aggregation and orchestrated management of both resources (ie, smart cameras) and data at a wide metropolitan scale to create a unified MISS ecosystem of vision systems.

In this paper, we aim at making a first step towards the implementation of this novel MISS scenario, by providing a layered technological stack, exploiting IoT technologies enhanced by cloud, fog, and edge computing approaches and big data management. The proposed idea is based on enabling a unified ecosystem of surveillance/vision systems, where data management is pushed down to edge nodes, orchestrated through cloud/fog computing mechanisms to achieve better performance and response time while meeting security (ie, integrity, confidentiality, and privacy) requirements. This frames the research effort presented in this paper into the domain of *Distributed Smart Cameras*, a research discipline exploring various aspects of real-time distributed embedded systems that perform computer vision using multiple cameras.[3]

The rest of the paper is organized as follows: Section 2 provides an overview of the problem and related work, whereas Section 3 introduces the proposed solution and the MISS framework, which consists of a 3-layer technological stack, further explained and discussed in Sections 4 to 6. Section 7 puts these descriptions of the 3 components together by detailing the integration, deployment, and operation phases of a MISS workflow. A case study on a MISS prototype is reported in Section 8, which compares this preliminary implementation with other approaches and discusses the feasibility of the proposed solution. Section 9 summarizes the main contributions and results of this paper and outlines a few directions for future work.

2 | OVERVIEW OF THE PROBLEM AND RELATED WORK

Modern cities start resembling tangled ICT "jungles" consisting of heterogeneous and isolated systems. This specifically applies to existing surveillance systems, which differ in their architectures, technologies, functionalities, interfaces, data formats, and workflows. Furthermore, the recent trend towards using mobile cameras, either deployed in vehicles (eg, for parking assistance, surveillance, and safety) or used as portable personal devices (eg, action cams and smartphones), even further increases the heterogeneity and complexity.

Common standard components of a camera[4] usually include optics, solid-state image sensors, an image processor with supporting hardware, an output generator, and communication ports sometimes connected with other sensors and actuators such as pan-tilt motors (pan-tilt-zoom/robotic cameras), Global Positioning System (GPS), and noise/temperature/humidity sensors to name a few. *Smart* cameras extend this set of features with improved processing, storage, and networking capabilities. A smart camera typically includes a dedicated signal processing unit, storage, and networking components, which enable communication with other devices; all these features are usually programmable (or at least customisable) to some extent. This way, a smart camera is able to perform *application-specific information processing* (ASIP), conceived for a problem at hand.

In the work of Shi,[4] a taxonomy of cameras and vision systems is proposed, identifying 4 different types:

- **Embedded vision systems** include stand-alone cameras, which either perform ASIP on-board or on an external unit, such as a single-board computer (eg, Raspberry Pi).
- **PC-based vision systems** are smart cameras, consisting of a general-purpose video camera and a back-end PC performing ASIP.
- **Network-based vision systems** are composed of multiple interconnected cameras usually equipped with intelligent features (eg, CCTV surveillance systems).
- **Hybrid vision systems** are special types of smart cameras, which may rely on human involvement to provide high-accuracy data output.

In the context of this paper, we extend this taxonomy by including mobile cameras, ie, personal and vehicular ones, as a subtype of embedded vision systems. As these cameras are already employed in urban surveillance systems installed in police cars or unmanned (aerial) vehicles (drones), it is therefore correct to assume that citizens can also be actively involved in surveillance activities by, for example, sharing their personal video footage captured in an area of interest with a surveillance application, following a *mobile crowdsensing* pattern.[5]

Video surveillance systems, before becoming truly intelligent, have gone through a number of evolutionary steps, as summarized in the work of Rinner et al.[6] The early age of video surveillance is characterised by an increased amount of manual work and mandatory involvement of human administrators in the surveillance process. In the worst case, the recorded video was collected from multiple cameras, viewed and analysed only in those cases, when there was a previously reported incident. With this kind of "post-mortem" analysis, the recorded footage was supposed to help in recovering the details of the incident. In the best case, video streams from a network of distributed cameras were expected to be continuously viewed in real-time mode by dedicated security staff, thus aiming to detect or prevent incidents with minimum time delays (eg, as it typically happens in large department stores or supermarkets). The higher the number of surveillance cameras, the more staff is needed to watch at the incoming video streams.

The situation changed with the introduction of ISSs,[7,8] whose main goal *"is to be able to functionally mimic the human eyes and brain and to interpret what the camera 'sees' through artificial intelligence."*[4] This means that the task of analyzing and interpreting the video information is off-loaded to machines, thus replacing an army of previously required security personnel. As a result, the involvement of human operators has been minimized to avoid such shortcomings as high labor cost and limited capability for multiple screens. Moreover, ISSs rely on existing technological achievements in computer

vision, pattern recognition, and artificial intelligence, which are used to identify certain patterns (eg, abnormal behavior, crime suspects, missing people, etc) in distributed video streams. Upon detection and recognition of a specific object or an event, a notification/alert may be sent to interested parties such that human operators can take a final decision to react to the occurred situation. To implement this semiautomatic operation, it is expected that relevant video footage, associated with the detected situation, is also available for deeper manual analysis and/or permanent storage. As it will be discussed in more details below, the requirement to persistently store captured video footage is application-specific and depends on the underlying business logic and available hardware resources.

A relevant survey on distributed surveillance systems,[8] dated back to 2005, outlines a generic workflow for related surveillance applications and classifies existing architectures. Since then, recent ICT trends in smart cameras and vision system technologies, on the one hand, and in networking, cloud, big data and the IoT, on the other, radically changed the perspective for distributed ISSs, significantly impacting on related workflows and architectures. More specifically, sensor networks, the IoT and stream processing allowed to directly interconnect smart cameras, orchestrate and "fuse" their data to enable intelligent multicamera video surveillance systems, as reviewed in the work of Wang,[9] whereas cloud computing and related technologies enlarged the scope of distributed ISSs to wider geographical areas such as modern urban spaces. In this direction, an ISS application is reported in the work of Eigenraam and Rothkrantz,[10] where an agent-based approach is adopted to coordinate a set of smart cameras interconnected as a sensor network for the purposes of traffic monitoring. Similarly, in the work of Lu et al,[11] traffic conditions of an urban area are monitored in real time by exploiting the capabilities of smart cameras that are able to preprocess and filter information before sending them to a remote server. Furthermore, the work of Hu and Ni[12] introduced an algorithm for license plate recognition through smart cameras on the basis of a lightweight filter for digital camera sensors to reduce network bandwidth and latency. As far as large metropolitan areas are concerned, data and information fusion techniques on the server/cloud side can be used to properly investigate underlying physical phenomena and events captured by an ISS as discussed in the work of Fan et al.[13] Indeed, enhanced elaboration can be performed by exploiting big data techniques as shown in the work of Shao et al,[14] where a novel workflow exploiting (big) surveillance data based on event detection and alarming messages from front-end smart cameras is proposed.

Other relevant applications of distributed ISSs in urban contexts are typically related to public safety as surveyed in the work of Räty.[15] In this context, and generally for any ISS, (real-time) performance and security requirements are of primary importance for the effectiveness of the ISS as discussed in the work of Rantala et al.[16]

Enabling collaborative behavior in distributed smart cameras has been recognized as a key challenge when performing various object tracking activities, that is, it is important to be able to pass information about an object being tracked over the network as it moves from one camera's *field of view* (FoV) to another. Existing works in this direction,[17-22] however, mainly focus on the aspects of image fusion and feature extraction, as well as on maximizing the overall FoV by combining the FoV of individual cameras, and seem to neglect the networking aspects associated with intercamera information exchange. That is, the existing approaches typically take the dynamic network configuration and routing for granted or target at rather static scenarios, where the network communication between distributed cameras is trivial. As it will be further explained, in our work, we aim at addressing this gap by specifically focusing on the problem of device discovery and integration at runtime. By doing so, we envision an extensible and reusable architecture that can support and complement the existing approaches in terms of dynamic camera integration and coordination.

In all these surveyed works, however, limitations typically emerge when talking about the wide urban area coverage, scalability. and sustainability of approaches and solutions at a large scale. In this light, the solution proposed in this paper aims at addressing these challenges by a novel approach and a framework able to implement an urban scale ISSs by aggregating several existing, multitenant, heterogeneous ISSs into a single shared MISS. Furthermore, it also proposes to exploit (clustered) edge computing capabilities in video stream management and processing to effectively cope with the main 2 challenges. First, by pushing data analytics and processing to an extreme edge of an urban ICT network, thus achieving faster response time, it aims to address the pressing big data challenges by means of a novel stream processing architecture. Second, by minimizing network communications, the approach also reduces the amount of potentially sensitive data transferred over public networks, thus contributing to the domain of data security and privacy.

3 | PROPOSED APPROACH

Intelligent surveillance systems are intended to replace manual object recognition processes performed by human operators with highly automated, scalable, and efficient workflows. The organization of such ISS is often implemented using

a bio-inspired approach, in which an image analysis performed by the human neural system is "reproduced" using digital technologies. More specifically, when performed by a human eye, the majority of image analysis takes place at the retina level in an unconscious manner, and only a small amount of collected visual information (that really trigger attention) reach higher cognitive levels of the human brain.[23] From this perspective, the retina level can be seen as a filtering step, exempting the human nervous system from transferring and "processing" less significant visual information. Similarly, the existing smart surveillance systems aim to draw parallels with and benefit from this human-inspired model by keeping image processing locally at the edge of an ISS network (ie, at the "retina level"). In this research effort, we also follow this bio-inspired approach and aim to even further utilize local edge resources of source cameras and collocated edge devices. This way, we minimize the amount of data transferred over the network to a cloud/edge-based processing server, similar to signals transported through a neural network to the brain in a human body.

As discussed previously, to implement this human-inspired MISS vision, we need to deal with several cross-cutting issues (among which performance and security are seen as the primary ones) at the following 3 interconnected layers.

- **Infrastructure layer** deals with problems related to the heterogeneity of the underlying hardware and vision systems in the metropolitan area. The challenge here is to build an ecosystem of interoperable vision systems, which are able to interact with each other regardless of underlying hardware/software architectures, as well as networking and administrative domains, while providing customizability and programmability facilities. This way, a unified MISS ecosystem spanning the urban area can be established by aggregating multiple independent ISSs from different domains, tenants and stakeholders. To this purpose, solutions coming from the IoT and cloud computing areas could be exploited.
- **Platform layer** is expected to provide advanced mechanisms for the MISS infrastructure management (eg, device discovery and selection, node churning, etc), as well as data management, fusion, accumulation, and processing. Approaches and techniques should give priority to distributed patterns, aiming to push computation closer to the source, thus addressing both security and performance issues. This is feasible in the MISS context if the underlying hardware is equipped with reconfigurable processing resources. To this purpose, additional local resources such as network servers, "cloudlets," and neighbouring edge devices (such as smartphones and IoT gateways) can be used. To this end, reference approaches and solutions could be borrowed from the fog and edge computing paradigms.[24,25]
- **Software layer** is responsible for implementing the video surveillance functionality and corresponding data processing workflows for ISS applications and services. In particular, given the security and performance requirements, as well as resource constraints of (smart) cameras and involved edge devices, a potential solution could be based on in-memory data processing techniques, which aim to process large amounts of streamed data dynamically "on the fly". The main focus of the proposed approach is on enabling timely (ie, near real-time) results by means of the stream processing technology, which assumes that collected data is processed immediately upon arrival and typically is not intended to be stored on a hard drive. Nevertheless, keeping collected data sets in the context of sensitive ISS scenarios is typically seen as a key feature for performing various forms of historical analysis. In this light, it is important to note that using stream processing is not intended to contradict with the established practice of permanently storing captured video streams but rather can be complemented by the latter. In this circumstances, data will be stored centrally on a server following application-specific data storage, protection, and expiration policies.

Given this challenging set of features to implement, we propose a solution reflecting this layered structure as shown in Figure 1. The lower infrastructure layer is performed by an *IoT cloud management* framework that establishes an

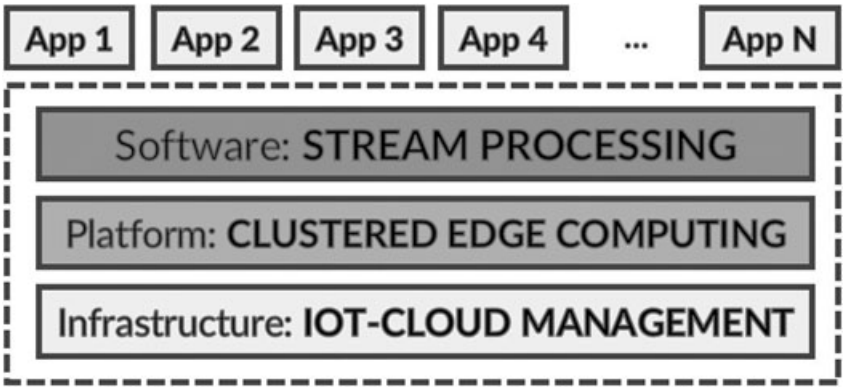

FIGURE 1 Conceptual architecture of the metropolitan intelligent surveillance system framework. IoT, Internet of Things

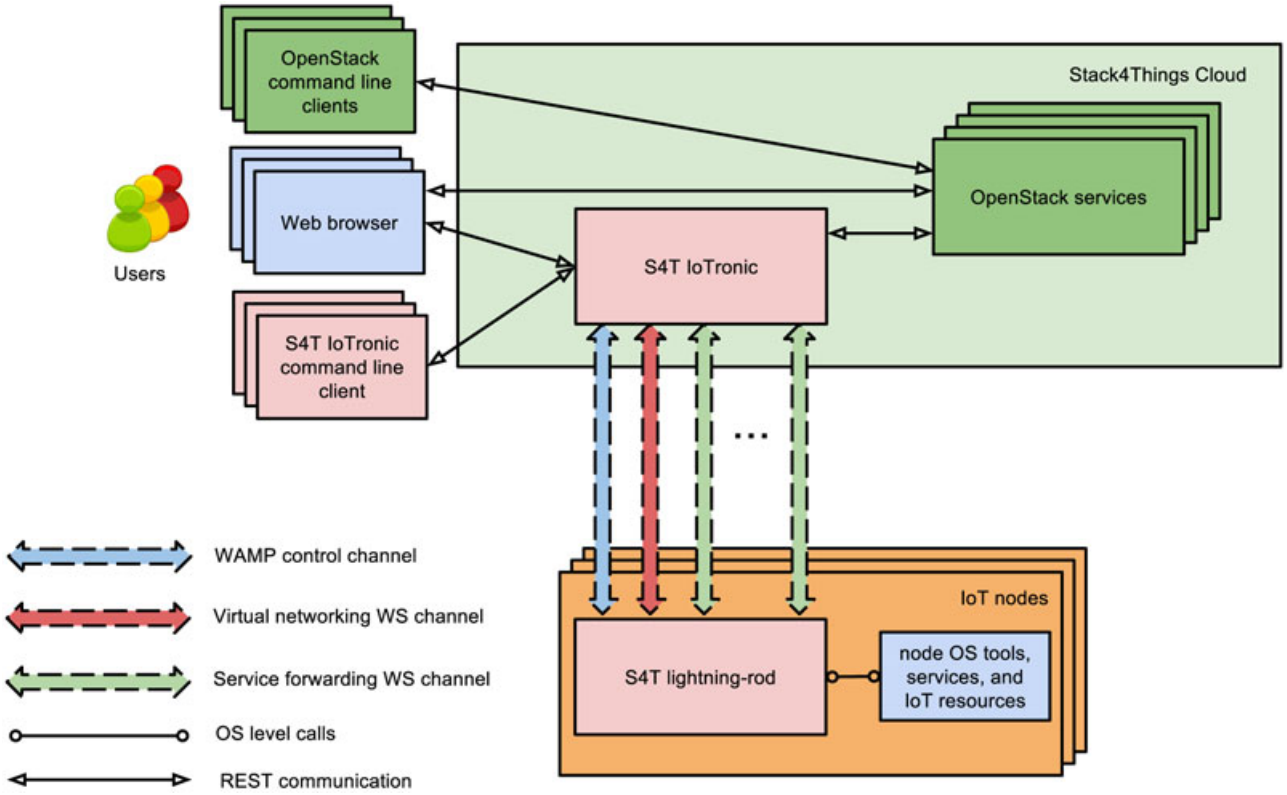

FIGURE 2 A bird's eye view on the Stack4Things (S4T) architecture. IoT, Internet of Things; REST, Representational State Transfer; WAMP, Web Application Messaging Protocol; WS, WebSocket [Colour figure can be viewed at wileyonlinelibrary.com]

interconnected "homogeneous" environment of programmable vision systems by integrating the existing vertical ISS. Then, to implement device orchestration and data management, we further extend the fog and edge Computing paradigms by introducing clustered computation at the extreme edge of the network that includes smart cameras and other non-camera edge devices. This way, a novel clustered edge computing (CEC) paradigm, which enables data management and processing on local devices rather than on communication and networking units (ie, fog computing) or remote servers (ie, cloud computing), is envisioned and implemented. At the highest application level, stream processing techniques are used to to orchestrate MISS data stream workflows among participating clustered edge nodes that include both source cameras and noncamera worker nodes. The implementation of the proposed 3-layer MISS stack will be based on existing solutions, adapting and extending them to the problem at hand, as discussed in the following.

4 | INFRASTRUCTURE LAYER: IoT-CLOUD MANAGEMENT

The infrastructure layer provides networking facilities to interconnect and manage edge nodes (such as smart cameras), possibly belonging to disjoint ISSs disseminated in an urban area. To implement basic facilities for IoT nodes allowing to establish clusters at the extreme edge of the network, we employ *Stack4Things* (S4T),[26] a framework adapting and extending the OpenStack* middleware with support for IoT infrastructures, towards a novel Cloud of Things paradigm.[27] Figure 2 shows the overall architecture of S4T, as defined in the work of Longo et al,[26] highlighting interactions between end users, the cloud, and sensor- and actuator-enabled IoT nodes, both static and mobile.

The S4T agent, called the *lightning-rod*, is deployed on edge nodes. It runs under the device-native (typically SDK-enabled) environment available for developers and interacts with OS tools and services available on the device. It represents the point of contact with the cloud infrastructure allowing end users to manage node-hosted resources even

*https://www.openstack.org/

when nodes are behind a NAT, a strict firewall or similar network barriers. This is ensured by WebSocket–based tunneling and Web Application Messaging Protocol–based messaging between the S4T lightning-rod and its cloud counterpart, namely the *IoTronic* service. WebSockets are also exploited to implement virtual networking channels among IoT nodes as described in the work of Merlino et al.[28] The S4T IoTronic is designed as an OpenStack service, providing end users with the possibility to remotely manage one or more nodes. This can happen both via a set of command-line clients, including the *IoTronic command line client*, as well as via a Web browser and Representational State Transfer (REST) APIs, as provided by the core OpenStack services and IoTronic itself.

Stack4Things thus provides and implements infrastructure-enabling facilities suitable for clustering and further management of edge nodes, including smart/networked cameras and any kind of (programmable) nodes belonging to the MISS, providing support for the following:

- **tenants and projects** by offering mechanisms to delegate and revoke authorisations for users and groups to access and manage selected resources (eg, hosted video/image capturing devices and corresponding video streams) with tunable granularity;
- **customizability** by enabling on-demand reconfiguration of edge devices and smart cameras at runtime, from low-level firmware/OS configuration (ie, also interacting with the initialisation and package management subsystem) up to the business logic, through mechanisms for injection of pluggable software components from the cloud;
- **virtual networking** by instantiating and managing cloud-mediated or fully peer-to-peer (ie, cloud-initiated) virtual networks among smart cameras and other edge devices, thus deploying standard applications on IoT nodes as if they were on the same local area network. This way, it overcomes network barriers and issues through (reverse) tunneling, while setting up the functional equivalent of private isolated secure VPN environments with the flexibility of low-level composable tools, for internode communication in the MISS context.

5 | PLATFORM LAYER: CLUSTERED EDGE COMPUTING

To enable time-critical MISS scenarios, a potentially promising approach would be to maximize the amount of computation performed on edge devices, that is, as close to the original source of data as possible, such that minimum amount of data is sent over the network to a (cloud) server (similar to image analysis on the retina level), and results are achieved immediately on the spot. In practice, smart cameras can typically perform image filtering and aggregation operations before transferring data to the Cloud, and in the best case, the available on-board processing capabilities of a single smart camera might be sufficient to localize individual faces and/or detect facial expressions. However, the performance will drop as the number of faces/objects to be analyzed increases, a typical situation in CCTV systems that operate in crowded public areas. Besides, even the "smartest" camera, regardless of its processing capabilities, physically cannot cover a wide surveillance area because of a limited FoV. That is, to fully cover an area of interest at a metropolitan scale, it is inevitably required to involve a maximum number of available cameras. Accordingly, this integration of edge devices into a common MISS system at the platform layer serves 2 purposes: (1) multiple distributed cameras act as sources and provide an extended FoV on the covered area of interest and (2) noncamera devices contribute their processing capabilities to a shared pool of clustered resources to support timely image processing.

As a solution, starting from the assumption that IoT devices such as modern smartphones and smart cameras are equipped with sufficient networking capabilities, we propose that edge devices can build up distributed clusters of edge nodes to share computational tasks and achieve better performance results, enabling near real-time application scenarios commonly present in the surveillance domain. This approach, CEC, revises and extends the traditional IoT view and reference model (that is based on deploying fog/edge computing facilities into networking and communication units such as switches, routers, "cloudlets," etc) toward the very extreme edge of the network as depicted in Figure 3. More specifically, (1) CEC assumes that data processing takes place directly on (stand-alone or clustered) edge nodes, thus reducing overheads and network latency; (2) fog computing at network/communication processing units serves to provide additional support to edge devices; (3) cloud computing, similarly, may provide even more resources to lower-level devices in situations, when their pooled resources are not sufficient.

As computation is pushed closer to the extreme edge from the cloud and network-level processing units, network latency and the overall processing time decrease. It is also worth noting the security and data privacy issues. Arguably, since minimum amount of sensitive data is transferred over the public network, the amount of associated security threats is also expected to be minimized.

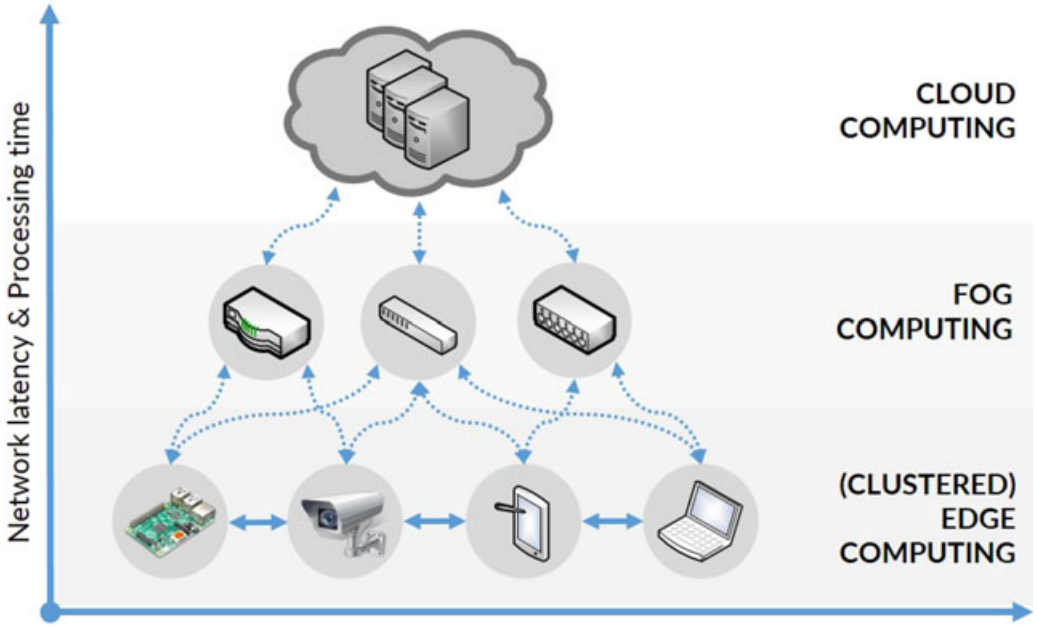

FIGURE 3 Clustered edge computing and horizontal off-loading [Colour figure can be viewed at wileyonlinelibrary.com]

5.1 | Vertical and horizontal off-loading

A fundamental underpinning of the described concept of CEC is the notion of *computational task off-loading*.[29] Broadly speaking, off-loading is a process of splitting the computation workload among several neighboring nodes within a network topology, so as to increase the performance and shorten processing time. In the context of IoT environments, we can distinguish between 2 main types of off-loading (also depicted in Figure 3).

- **Vertical off-loading** is the traditional and established way of sending computational workload over the network to a nearest server, data center, cloud, etc. In this case, no computation takes place on edge devices, but rather, raw data is transferred from the bottom of the topology upwards to the fog and cloud (hence, vertical off-loading), which may also be responsible for the top-down coordination and orchestration activities. This type of off-loading is represented by bidirectional dashed lines between nodes at the 3 levels of the IoT network in Figure 3.
- **Horizontal off-loading** relies on the principles of cluster computing, where multiple computational instances (ie, a decentralised cluster) share workload among themselves, thus keeping the computation closer to the original source and thereby increasing the performance. This type of off-loading is represented by solid lines between edge nodes in Figure 3. Horizontal off-loading lies at the core of the proposed approach since it minimizes the amount of data sent over the network to the fog/cloud and time delays because of network latency.

5.2 | Implementing CEC

To implement the aforementioned patterns, we start from S4T, extending them toward CEC. Indeed, S4T provides a UNIX-style virtual file system-based abstraction of IoT resources, including remote node-hosted ones (ie, as if these were local), and their underlying (lower-level) interfaces by leveraging the *File system in UserSpacE* platform. These interfaces are enabled by the virtue of S4T-specific *drivers*, accessible via a command line client and a Web browser through REST APIs. For example, resources may be general-purpose input/output pins (for embedded boards), SDK-specific sensors API (for mobiles), real-time video capturing API, and corresponding device nodes (for smart/IP cameras). An S4T driver is composed of a JSON configuration file and a Node.js module, the former associating each method in the latter with a Portable Operating System Interface–compliant file system primitive performed on one of the files in the corresponding virtual file system. These interfaces provide methods for the stream processing subsystem to rewire not only the networking topology underneath edge nodes but also the application-level (ie, data) processes by letting translate a workflow in the corresponding (runtime) mapping of node-hosted resources (eg, device nodes or streams) to files as a way to instantiate the processing/forwarding pipelines. Further details are discussed below.

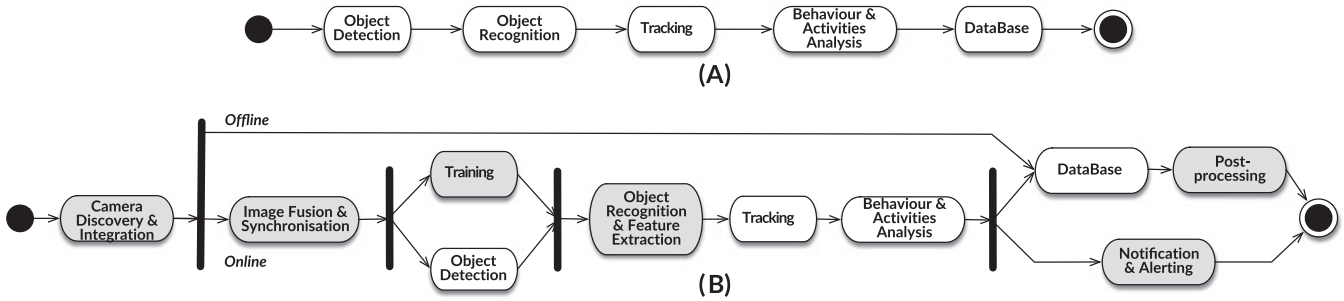

FIGURE 4 (A) Intelligent surveillance systems and (B) metropolitan intelligent surveillance system reference workflows

6 | SOFTWARE LAYER: STREAM PROCESSING

The traditional ISSs have been fully or partially implementing an established pipelined workflow, in which images, captured by a camera, go through several processing phases. More specifically, this pattern includes the following steps[8] as shown in Figure 4A: *object detection*, *object recognition*, *tracking*, *behaviour and activities analysis*, and *database*. This workflow, however, refers to the previously described rather outdated scenarios, in which isolated ISS infrastructures operate independently from each other, thus not taking into consideration potential opportunities for dynamic discovery of neighbor devices and collaboration with them. To this end, this paper extends the established reference workflow with several novel phases, which reflect the proposed MISS vision, as well as underpin the proof-of-concept implementation.

As it follows from Figure 4B, the MISS is enabled by the following key activities.[†]

- **Camera discovery and integration**: Neighboring camera nodes are discovered over the network and integrated into a common co-operating ensemble to capture images and videos. Once images have been captured, they can be either processed online (ie, the lower branch in Figure 4B) to enable time-critical application or offline (ie, the upper branch in Figure 4B). Moreover, in many situations, both threads can run in parallel such that captured data is processed online for timely decision taking and at the same time persisted to a storage system for later postprocessing.
- **Image fusion and synchronization**: Video streams/static images are combined together to provide a more holistic view on an area of interest. They can also be sampled or grouped for further (distributed) processing. The specifics of image fusion go beyond the scope of the presented research, and the interested reader is referred to other works[22,30,31] for an overview of the existing image fusion techniques and approaches. In an ISS scenario, image fusion can be seen as a part of a more generic data fusion process that can take place at different levels, ranging from lower-level sensor fusion to higher-level information and knowledge fusion.
- **Object Detection**: Objects of interest are detected in the input video stream, according to an algorithm in place such as the ones proposed in the work of Wahyono et al.[32]
- **Training**: The system is trained to recognize detected objects and activities.
- **Object Recognition and feature extraction**: Detected objects of interest are recognized on the basis of a training set and/or can be further processed for feature extraction.
- **Tracking**: The detected and recognized objects of interest are continuously captured and tracked across multiple frames in a video stream, for example, adopting the technique proposed in the work of Gao et al.[33]
- **Behavior and activity analysis**: The system is able to recognize continuous actions performed by objects in a video stream.
- **Notification and alerting**: Corresponding interested parties are notified of the object detection, recognition, and/or behavior analysis results.
- **Database**: These results may also be persistently stored for later analysis.
- **Postprocessing**: This step performs offline processing of captured data, either as an independent workflow branch or as a continuation of online processing (eg, for deeper manual investigation of detected issues).

This way, the original ISS workflow as specified in the work of Valera and Velastin[7] has been extended with new activities to adapt it to the wider urban context targeted by the MISS as highlighted in Figure 4B. As it clearly follows from this

[†]Please notice that this reference workflow does not necessarily imply that all of its steps are required to be implemented. Rather, it represents a generic architecture, where some steps can be omitted depending on the application requirements.

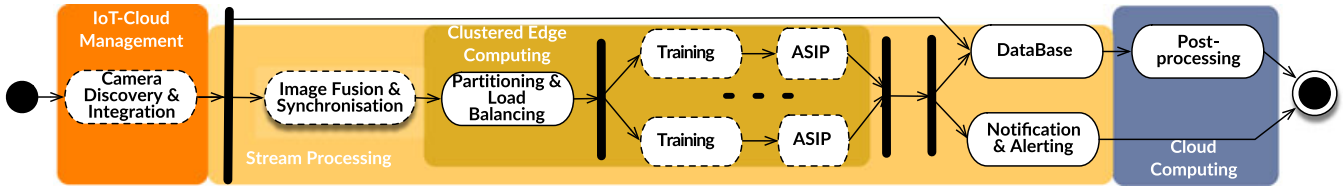

FIGURE 5 Convergence of IoT-Cloud management, clustered edge computing, stream processing, redand cloud computing to enable the MISS reference workflow [Colour figure can be viewed at wileyonlinelibrary.com]

diagram, the MISS workflow with its ordered sequence of data transformation and processing steps naturally fits the stream processing paradigm, at least in the online branch, whose main principle is to process data dynamically as it arrives, without storing it to the hard drive first. This is perfectly aligned with the resource-constrained nature of edge devices, which are typically not expected to permanently store large amounts of data. This further reduces potential overheads (ie, no data stored on the devices and no multiple read/write operations) and even further decreases processing times.

Stream processing focuses on data flow processing and timely reaction.[34] The former assumes that data is not persisted but rather continuously flows and is processed in memory, and the latter means that stream processing systems aim at operating in real time under pressing time constraints. These 2 key features have led to the emergence of a family of middleware systems specifically conceived for processing data streams on the basis of a set of predeployed rules. Given the large volume and high velocity of streaming data, such stream processing frameworks have been primarily designed to be deployed on powerful server machines or clusters of such machines. The situation is changing, however, with the active development of embedded technologies, which resulted in a number of stream processing implementations, specifically developed to be deployed and executed on portable and embedded devices with potentially limited hardware capabilities.

One of such lightweight middleware implementations is Apache NiFi.[‡] This open-source stream processing framework is specifically designed to implement a wide range of IoT scenarios. NiFi is based on the notion of *flow-based programming*, where a *data flow* conceptually represents a sequence of processing steps, through which data is streamed. These steps, known as *processors*, can be seen as data processing operators, ranging from simple mathematical operations to more advanced ones such as natural language translation or data format conversion. To date, there are more than 100 built-in NiFi processors that can be further extended with user-customized ones. Processors are equipped with input/output ports, which serve to connect them and thus create complex data flow topologies. To implement an even more flexible behavior, NiFi supports various types of flow file prioritization, that is, on the basis of a flow file's attributes, it may be routed to different processors.

7 | MISS FRAMEWORK INTEGRATION, DEPLOYMENT, AND OPERATION

The convergence of the 3 enabling technologies, namely IoT cloud management, clustered edge computing, and stream processing, provides a promising foundation for implementing the MISS reference workflow, depicted in Figure 4B, and has to be properly integrated into a MISS framework. Using S4T, it is possible to discover and interconnect previously disjoint edge devices, which are then grouped into clusters as suggested by CEC and the horizontal off-loading patterns, whereas NiFi's support for flow-based programming at the highest level serves to create complex data flow topologies, made of data sources and processors, to be executed on clustered devices. The resulting integration of the 3 underpinning technologies is illustrated by Figure 5 that highlights the roles of each supporting technology and distinguishes between built-in (solid line) and user-customized (dashed line) components. Within the overall workflow, 3 key phases can be identified in this respect, as discussed below.

7.1 | Integration

Constituted by multiple mobile and portable smart devices that can move across different geophysical and network locations, the urban IoT ecosystem is very dynamic in its nature. As a result, topologies of such networks are typically not

[‡]https://nifi.apache.org/

fixed, but rather continuously change with respect to nodes joining and leaving the network at unpredictable rates. As a generic, scalable, and flexible solution, in the presented approach, dynamic runtime clusterization is aided by overlay networking facilities provided by S4T. Since the dynamic nature of such topologies is underpinned by mobility patterns, possibly inducing the traversal of different network domains, it is important to take into account some issues that may arise as a result of these conditions such as (sudden) introduction of address/port translators or security-oriented appliances (eg, firewalls) between any 2 nodes, which may outright block or significantly modify internode communications, hindering the process of node discovery and clusterization.

To provide support for (transparent) network communications among edge devices traveling across heterogeneously-administrated subnets (eg, in a metropolitan area network or even smaller scope such as a university campus), with the help of an (overlay) networking coordinator, which gets contacted by all nodes at startup to establish an always-on command-and-control stream of messages, compliant to WebSocket-based Web Application Messaging Protocol. Available commands to be sent by the coordinator include requests for nodes to establish (reverse) tunnels to the coordinator. Indeed, in this solution, WebSockets are leveraged to actually pierce "middle boxes" and implement overlay networks among IoT nodes by transporting (node-initiated) tunnels as described in the work of Merlino et al.[27] In particular, transparent Layer-3 networking is enabled by the overlay coordinator instantiating, managing and routing coordinator-terminated tunnels to each smart camera and worker nodes. Network barriers are overcome through WebSocket-based (reverse) tunneling, setting up the functional equivalent of private isolated secure VPN environments.

This way, IoT cloud management brings together formerly disjoint individual ISSs and supports their integration into a single unified MISS as if they all were on the same physical network. The network integration enables interconnecting previously disjoint collocated edge devices into a stream processing cluster using the Apache NiFi middleware and enable horizontal off-loading as suggested by CEC. Accordingly, the next challenge is to implement communication between the S4T lightning-rod and NiFi middleware within a single edge node. Assuming that each node is already running a default NiFi instance within an isolated S4T container, it is now required to dynamically reconfigure it with (1) cluster network settings and a data flow topology and (2) processors involved in this newly added topology. The former configuration is implemented using NiFi's REST API that supports runtime modifications to cluster and topology settings. The latter is somewhat more complicated and requires compiled processors to be first downloaded from the S4T IoTronic service to the local S4T lightning-rod and then uploaded to the NiFi instance. This is implemented using S4T's support for plugins, that is, downloaded NiFi processors are "injected" into the running NiFi instance as pluggable packaged components. Having received the received updated settings, the NiFi instance will reboot and start running the new stream processing topology.

The resulting integration of the 3 main technologies reflects the 3-level conceptual architecture in Figure 1. At the lowest level, S4T handles the network and infrastructure heterogeneity, acting as an interface for the NiFi middleware that operates at the platform level and serves to group collocated edge devices into a stream processing cluster. At the highest level, complex data flow topologies are executed on an edge cluster to enable various (ISS) application scenarios.

7.2 | Deployment

Once the previously described components are integrated into a MISS framework, it is time to deploy them into a real setting. Referring to Figure 5, the main workflow activities and related deployment can be described as follows.

- **Camera discovery and integration**: In the MISS context, these are enabled by S4T. CCTV cameras and other edge devices, all equipped with a S4T lightning-rod, are able to dynamically discover each other's network locations through the cloud broker and establish direct network links afterwards. Each node applies this newly received cluster configuration and reboots in order for the new settings to take place. Once all nodes are back online, they start exchanging heartbeats to identify peer cluster nodes and elect a coordinator; these activities are enabled by the NiFi middleware that is responsible for most cluster-related activities. Upon the creation of a cluster, participating nodes are now able to run a common NiFi workflow, differentiating between source nodes (ie, CCTV cameras) and noncamera worker nodes (ie, Raspberry Pi boards). Accordingly, it is now required to design the required MISS reference workflow using NiFi processors (an activity that is typically undertaken by a human operator at design time) and deploy it on the resulting cluster for execution.
- **Image fusion and synchronization** (user-customized): An important challenge to be addressed by a MISS is the seamless combination of separate video streams/images potentially captured from different perspectives and ranging in their frequency, quality, encoding, etc. This kind of "multiplexing" is expected to take place on the cluster coordinator

and can be implemented by a custom NiFi processor, which takes as input multiple video streams and produces an "aligned" output. Admittedly, developing a sophisticated data merging algorithm, underpinning this process, is an inherently difficult research challenge in its own right, going beyond the scope of the presented paper.

- **Partitioning and load balancing** (built-in): This is one of the key built-in processors in NiFi (`DistributeLoad`) as it provides even workload distribution across all nodes in the cluster with respect to a distribution strategy in place (eg, "round robin"). Accordingly, in the MISS context, this processor is configured to distribute incoming video/images for further processing on the basis of the "next available" distribution strategy. To support distributed processing and follow the "exactly once" semantics, NiFi uses internal repositories for tracking the task processing progress. This way, incoming MISS tasks are guaranteed to be accomplished, which is particularly important for mission-critical surveillance domains.
- **Training and ASIP** (user-customized): The latter includes Object Detection, Object Recognition and Feature Extraction, Object Counting, Tracking, Behavior and Activities Analysis, and any possible/relevant application-specific processing of images captured by the MISS. All these ASIP activities are undertaken by a NiFi cluster composed of CCTV cameras (acting as sources) and other noncamera edge devices (acting as worker nodes), following the horizontal off-loading principles of CEC. Each of these activities can be implemented as a custom NiFi processor, responsible for an individual processing step, and interconnected by input/output ports for information exchange. Admittedly, this is the most computationally intensive part of the MISS workflow and is therefore expected to benefit from parallel processing if possible. There is also expected to be (1) a load balancing processor at the beginning of this subworkflow to distribute the incoming image processing tasks among cluster nodes (ie, this can be thought of as "mapping") and (2) a merging processor to collect and synchronize the processing results (ie, this can be thought of as "reducing"). Both processors are available in NiFi and can be configured according to the MISS application requirements.
- **Notification and alerting, database** (built-in): Once the processing results are collected by the cluster coordinator, corresponding NiFi processors are required to either notify interested parties (eg, there is a wide range of NiFi built-in processors, including JMS, MQTT, AMQP messaging, as well as e-mail notifications and logging) or persist these results in a dedicated storage using one of the many built-in processors (eg, `PutSQL`, `PutMongo`, `PutFTP`, `PutHDFS` to name a few). Both alerting and writing to a database can also be performed by individual cluster nodes even before the "reduce" step if required to achieve faster performance. Alternatively, as previously described, storing video footage to a database can take place immediately upon capturing to enable offline postprocessing at a later stage.
- **Postprocessing**: This is usually outsourced to third-party services that require captured video footage to be vertically off-loaded to remote datacenters and/or cloud platforms for storage and processing. Because of the increased network latency and time delays, this step is not included in the stream processing workflow but is rather seen as optional (yet desirable) activity to be implemented by the MISS workflow depending on the underlying infrastructure resources and application-specific policies.

7.3 | Operation

Once the MISS topology is deployed on the cluster for execution, the participating cluster nodes are ready to proceed with the operation step. Image processing tasks run in parallel on worker nodes, returning results to the elected cluster coordinator (ie, CCTV camera), which periodically reduces and aggregates them, according to the business logic. In parallel to this, the coordinator keeps on listening for heartbeat messages from edge nodes (ie, both source and worker nodes) appearing on the network in the meanwhile. Whenever a new node appears, it is expected to go through the same clusterization steps, ie, receive network and cluster settings from the S4T broker, update its internal NiFi configuration, reboot, and attempt to join the cluster either as a source or a worker node. The node will be eventually integrated into the running cluster and will start receiving jobs for processing. The clustered processing will continuously iterate on new tasks until completion. While the dynamic discovery and integration of network nodes is part of the newly developed functionality presented and described in this paper, handling the *churn rate* (ie, the number of nodes leaving the cluster over time) is part of NiFi's built-in "zoo keeping" functionality. Whenever an existing node leaves the cluster and stops sending heartbeats, the rest of the nodes will shortly notice this and will no longer attempt to send a task for processing.

NiFi cluster nodes also maintain state repositories to keep track of processed jobs, logs, cluster topologies, data, etc. These repositories are continuously synchronized across all nodes following the zero-master approach such that even if the currently elected coordinator crashes, the negative impact on the cluster operation will be be minimized, ie, the rest of the nodes will continue operating and will shortly elect a new coordinator.

FIGURE 6 Indicative face recognition workflow in an intelligent surveillance systems

8 | FROM THEORY TO PRACTICE

This section now proceeds with a demonstration of how the proposed MISS vision can be enabled through the convergence of the key 3 technologies as described previously. The case study focuses on a face recognition task implemented by a simple MISS, constituted by 2 CCTV cameras installed in a university campus of the Kazan Federal University (Kazan, Russia). The main goal of our experiments was to demonstrate the feasibility of the proposed approach, and the effectiveness of the MISS framework to handle distributed video streams through the CEC pattern. This way, without loss of generality, we focus on a simple example to validate and evaluate the behavior of the overall MISS through a proof-of-concept prototype. A more extensive experimentation on a larger urban area case study is planned as future work.

Taking the common task of recognizing potential suspects in a crowd as a reference, a typical workflow implemented by an ISS in an automated manner can be represented with the diagram in Figure 6. This process can be seen as a partial instantiation of the generic MISS reference workflow reported in Figure 4B. The process is initiated by each camera that continuously captures video streams. Next, the streams are sampled into separate static frames at a given frequency (depending on the requirements and capabilities), which may contain human faces to be detected. The face detection component, on the basis of its intelligent algorithms, is able to identify human face features in the image (possibly applying some additional transformations such as cropping, resizing, or gray scaling). Finally, the detected faces are processed by the face recognition component, which also applies some intelligent algorithms and might need to be pretrained against a sample set of faces. If required, a corresponding interested party is notified once a potential suspect is recognized.

Taken together, these activities, however, can be classified as rather computationally intensive, requiring advanced processing and storage capabilities, as well as sufficient network bandwidth; these are not necessarily present in a single CCTV camera, and therefore, the intelligent processing can be off-loaded to a different more capable processing location (eg, a server or a cloud). On the basis of the workload distribution, these streams can be processed using the following 3 processing and off-loading models.[6]

- **Clustered edge computing**: Video analysis is performed collaboratively by multiple clustered cameras and other collocated smart devices in a decentralised manner as suggested by the proposed CEC approach. This way, the intelligence is pushed to the very edge of the network, as close to the data source as possible, to minimize the amount of data transferred to the server and thus achieve faster execution results.
- **Cloud/fog computing**: Multiple video streams are analyzed on a central server, be it a local fog node or a remote cloud platform. Since the intelligent ASIP components are deployed remotely, cameras need to stream their data over the network. This process might take quite long, and a corresponding reactive action (eg, the police is alerted) might be too late. The performance of such a smart CCTV system is affected by the quality of its external network connection, worsening with the number of hops, a limitation hardly addressable within the context of the "vertical" off-loading model because of the inevitable requirement to send data to a remote processing location. For example, by the time the cloud-based face recognition software detects a suspect criminal and sends back a corresponding signal, this criminal may have already escaped the initial CCTV-covered area.
- **Edge computing**: A video stream is independently analyzed immediately on the spot by an edge camera itself, without any off-loading and collaboration such that only the final processing results are transferred upwards. Admittedly, this assumes that the smart camera is equipped with rather advanced hardware and software capabilities, allowing it to independently accomplish computationally intensive tasks. An opposite situation is also possible, ie, image processing demands are trivial to an extent making a single camera able to meet them.

These 3 models serve as a reference for us to conduct a set of experiments using corresponding testbeds. Accordingly, in the rest of this section, we compare the proposed approach to enable the MISS via distributed video surveillance at the edge using NiFi (ie, the CEC model) with the other 2 types of workload distribution described above (ie, the fog/cloud and edge computing models). To this end, we benchmark processing times in all 3 testbeds and then discuss the results.

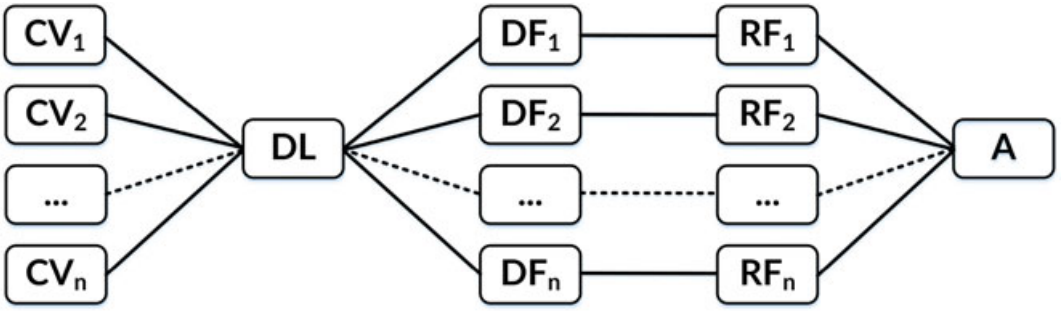

FIGURE 7 The intelligent surveillance systems face recognition workflow implemented using Apache NiFi [Colour figure can be viewed at wileyonlinelibrary.com]

8.1 | Testbed setups

Three sets of experiments have been conducted on different infrastructures following different off-loading and processing models as described in the previous section.

8.1.1 | Edge cluster setup

Experiments on the CEC testbed were conducted using the following equipment: (1) two independent ISSs each composed of a 5-megapixel camera and (2) five Raspberry Pi 3 (1.2 GHz 64-bit quad-core ARM CPU + 1GB RAM) boards were edge processing nodes; all these edge devices were required to be discovered and connected into a cluster.

All devices, connected to a wireless local area network, run on Linux OS with a S4T lightning-rod and the NiFi middleware installed. As previously described in more details, the S4T lightning-rod communicates to the S4T IoTronic service, which acts as a common shared bus for discovering and establishing direct network links to collocated network devices. This way, by communicating through the S4T IoTronic platform, an edge node is able to discover neighboring nodes equipped with required resources, including real-time video capturing API (for source cameras) and CPU/RAM (for worker nodes). Through the S4T IoTronic and lightning-rods, it is possible to configure managed edge devices with corresponding NiFi processors as required by the MISS scenario. This way, the NiFi edge cluster is established.

Please note that the presence of the cloud coordinator does not contradict with the proposed idea of pushing intelligence to the edge to minimize the network latency explained so far. The role of the S4T IoTronic service is to help edge nodes to discover peers and enable communication between them. This means that the nodes interact with the cloud only during the one-off initialization process, and once a local cluster is discovered and established, they start exchanging data within the cluster, ie, no sensitive data is transferred remotely and no face recognition is performed on the cloud.

As a result, the established clustered MISS, composed of 2 independent ISSs and up to 5 Raspberry Pi worker nodes, is configured to run the following topology (see Figure 7). In the context of the presented MISS experiment, the following built-in and user-customized processors were used.

- `CaptureVideo` (CV) continuously captures video from a camera, splitting the video stream into separate static frames and sending them to an output port for face detection.
- `DistributeLoad` (DL) is a NiFi built-in processor for load balancing. It equally distributes incoming images between available cluster nodes on a "first available" basis.
- `DetectFaces` (DF) detects and crops human faces in each frame. Once faces are detected, they are serialized and transferred forward for recognition.
- `RecognizeFaces` (RF) is first trained against a predefined set of human faces. Then, the processor is ready to perform the face recognition routine; it takes as input an image, processes it with respect to its training set, and outputs a prediction value for each face in the training set. Simply put, it decides to which extent the detected face resembles faces in the training set.
- `Alert` (A) informs interested parties about potential suspects recognized in the crowd.

To implement the custom processors, an established open-source face detection/recognition library OpenCV[§] was used. It contains a wide range of utility methods for handling various face detection/recognition tasks. The experiments were

[§]http://opencv.org/

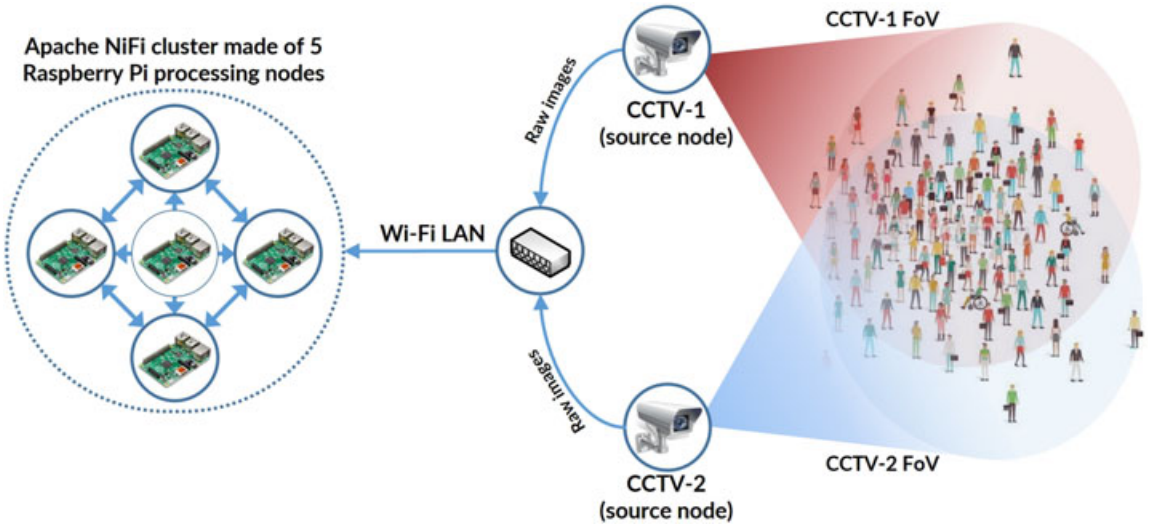

FIGURE 8 A metropolitan intelligent surveillance system scenario implemented by clustered edge computing. LAN, local area network [Colour figure can be viewed at wileyonlinelibrary.com]

conducted in a university campus, located within a densely-populated metropolitan area, during crowded events. Figure 8 schematically depicts this setup.

In computer systems, where large amounts of data are continuously streamed from one location to another, there is always a risk of running into the "bottleneck" problem. Accordingly, in the first proof-of-concept implementation, the camera sampled and streamed images to peer nodes in a pipelined manner, ie, images were *pushed* to the input queue of the cluster, from where they were then picked up by the load balancer for further distribution. This way, the source camera is solely responsible for delivering tasks, and this queue became the "bottleneck," where images would stack before being processed by one of the available nodes. This becomes particularly pressing if the sampling frequency or the image size (ie, resolution) increases. A common solution in these circumstances is to make processing nodes *pull* data themselves from the source node, thus enabling parallel transferring of multiple images. In the NiFi terminology, this is known as the *List/Fetch* pattern, according to which the source node keeps images locally and only sends image IDs to the cluster. The cluster load balancer receives the *list* of IDs and distributes them among the processing nodes, which are then able to *fetch* actual images from the source node in parallel, thereby avoiding the "bottleneck."

8.1.2 | Cloud testbed

A similar ISS scenario was implemented using the traditional "vertical" pattern, which includes a cloud-based face detection/recognition software component, receiving independent image streams from both CCTV cameras, serialized, and transferred via a messaging queue. The cloud-side component is configured to extract images from the queue and then detect and recognize faces in the frames. The detection/recognition workflow is similar to the one in the NiFi cluster setup, ie, the system is first trained on a set of images and then executes the recognition routine with respect to incoming frames. To conduct the described experiments, Heroku[¶] and Amazon EC2,[#] 2 well-established cloud platforms, were chosen.

8.1.3 | Single camera testbed

The single-node ISS scenario was implemented using a camera with a Raspberry Pi board attached to it. To some extent, this setup is similar to a cluster with only 1 processing node present. This way, the camera is able to deliver images for processing almost immediately. However, a potential point of saturation will be reached when the limited processing resources will fail to support higher sampling frequencies or more faces in frames.

[¶] https://www.heroku.com/
[#] https://aws.amazon.com/ec2/

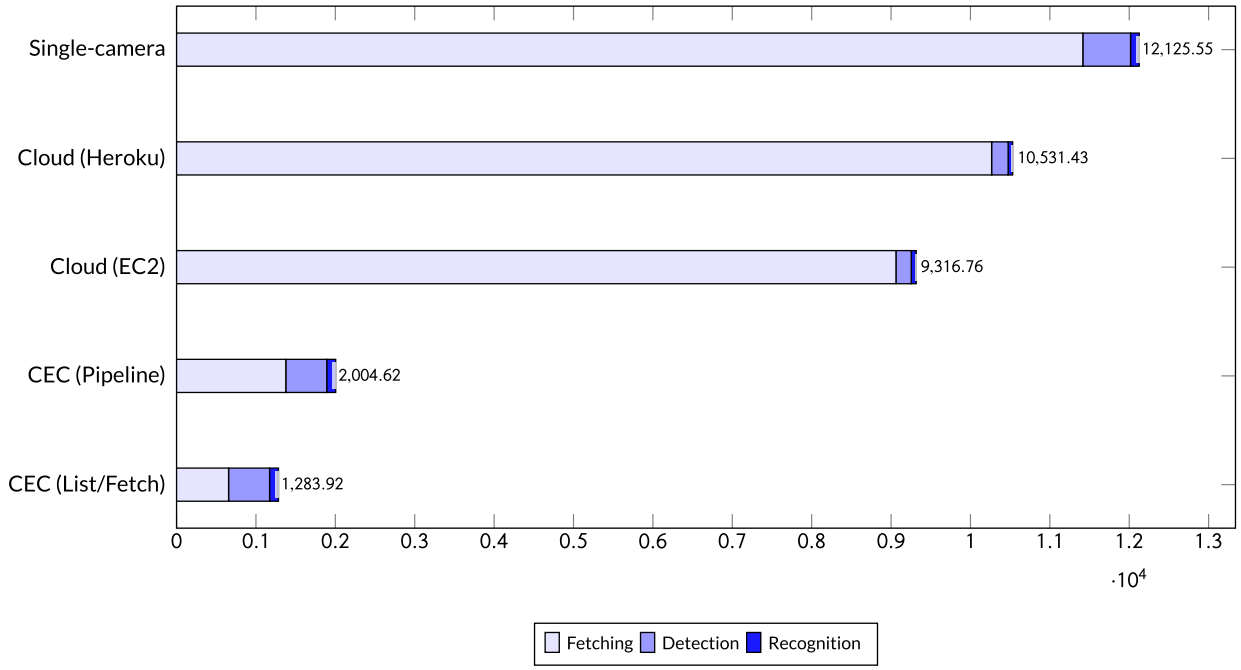

FIGURE 9 Benchmarking results. CEC, clustered edge computing [Colour figure can be viewed at wileyonlinelibrary.com]

8.2 | Benchmarking and comparison

Figure 9 summarizes the benchmarking results obtained from running 1000+ experiments with the sampling frequency of 5 frames per second. In general, the proposed CEC approach demonstrated promising performance results (given this particular MISS scenario). Besides, it is assumed that by minimizing the amount of sensitive data transferred over potentially insecure public network, the presented implementation also contributes to secure and private data management. More specifically, by looking at the histogram chart, the following can be highlighted.

- The proposed stream processing architecture on top of NiFi edge cluster using the List/Fetch pattern performs up to 8 times faster than the Cloud setups on top of EC2 or Heroku and up to 9 faster than a single stand-alone camera.
- The List/Fetch pattern considerably improves the throughput of the clustered system by avoiding the bottleneck of the messaging queue, when using it as a single communication channel in the pipelined implementation.
- In the cloud-based setups, most of the time is spent on fetching images for processing, whereas the proposed CEC approach aims to minimize the network latency. Even though the cloud was able to accomplish the face detection/recognition tasks faster than the local setups, the main time delay was caused by the network latency, inevitably present in remote network communication.
- The increased processing time for a stand-alone camera is caused by the insufficient resources to cope with the sampling frequency of 5 frames per second. In other words, the camera was "saturated" and was not able to process previously arrived frames in time for handling new ones. Accordingly, the frames had been queuing until the system became overloaded and, eventually, crashed.

It is also worth noting that task parallelisation and Stream Processing at the edge are not a 'one-size-fits-all' solution. There may be MISS scenarios, where data cannot be fragmented and tasks cannot be parallelised in general. For such kind of scenarios, a vertically scalable Cloud-based server might prove to be more useful, even at a cost of the increased network latency.

9 | CONCLUSIONS AND FUTURE WORK

This paper proposed the vision of the metropolitan intelligent surveillance system, a globally connected and accessible metropolitan ISS spanning urban areas, on the basis of the pervasive network of smart cameras and independent ISSs, to address the big data and security challenges. To implement this novel vision, the paper introduced a 3-layered architecture,

enabling the discovery of heterogeneous edge devices, their integration into local computational clusters of edge devices, and dynamic data processing using a stream processing architecture. The resulting CEC is underpinned by the concept of "horizontal" off-loading, ie, distributing workload among edge devices, rather than transferring to a remote location the task for processing. As demonstrated by the proof-of-concept implementation and a number of benchmarking experiments, the proposed approach has the potential to enable MISS scenarios and outperform the cloud-enabled setup, as well as to guarantee data privacy and security by minimizing the amount of data exchange over potentially insecure public networks. From this perspective, the proposed approach can be seen as complementary to the existing established practice of pushing computation to the cloud, offering promising opportunities for creating hybrid/mixed scenarios, in which computation can partially take place on both local resources and remote cloud infrastructures. Utilising edge/fog computing principles, the resulting convergence will enable a flexible data processing architecture, underpinned by the balanced distribution between local and remote computational resources.

The proposed ideas are just the first step towards enabling novel ISS architectures and services in metropolitan areas and are yet to be further elaborated and proved. Arguably, with the the presented proof of concept, it is not yet possible to evaluate potential benefits when running a MISS consisting of tens and hundreds of smart cameras, as well as to predict emerging challenges related to networking, load balancing, security, etc. To this end, the future work will primarily focus on deploying and testing a MISS in a real camera-rich urban ecosystem. On the one hand, this work will require "injecting" and configuring S4T agents in multiple CCTV cameras, as well as handling other technical aspects. To a great extent, this will be based on the positive experience of the #SmartME initiative[35] that involved deploying smart environmental sensors throughout the municipality of Messina, Italy. Using the common S4T cloud-based middleware, these distributed sensors are remotely discoverable, accessible, and reconfigurable according to emerging business requirements. Similar functionality is expected to benefit the MISS implementation as well. On the other hand, these activities need to be negotiated and arranged with several parties, including local municipal authorities and CCTV service providers.

ORCID


Rustem Dautov http://orcid.org/0000-0002-0260-6343

REFERENCES

1. Jenkins N. 245 million video surveillance cameras installed globally in 2014. https://technology.ihs.com/532501/. Published June 11, 2015. Accessed February 22, 2017.
2. Dautov R, Distefano S, Merlino G, Bruneo D, Longo F, Puliafito A. Towards a Global Intelligent Surveillance System. In: Proceedings of the 11th International Conference on Distributed Smart Cameras; 2017; Stanford, CA.
3. Rinner B, Wolf W. An introduction to distributed smart cameras. *Proc IEEE*. 2008;96(10):1565-1575.
4. Shi Y, Real FD. Smart cameras: fundamentals and classification. In: *Smart Cameras*. Boston, MA: Springer; 2009.
5. Merlino G, Arkoulis S, Distefano S, Papagianni C, Puliafito A, Papavassiliou S. Mobile crowdsensing as a service: a platform for applications on top of sensing clouds. *Futur Gener Comput Syst*. 2016;56:623-639.
6. Rinner B, Winkler T, Schriebl W, Quaritsch M, Wolf W. The evolution from single to pervasive smart cameras. Paper presented at: 2008 Second ACM/IEEE International Conference on Distributed Smart Cameras; 2008; Stanford, CA.
7. Qian H, Wu X, Xu Y. *Intelligent Surveillance Systems*. Vol 51. Berlin, Germany: Springer Science & Business Media; 2011.
8. Valera M, Velastin SA. Intelligent distributed surveillance systems: a review. *IEE Proc - Vis Image Signal Process*. 2005;152(2):192-204.
9. Wang X. Intelligent multi-camera video surveillance: a review. *Pattern Recogn Lett*. 2013;34(1):3-19. http://doi.org/10.1016/j.patrec.2012.07.005
10. Eigenraam D, Rothkrantz LJM. A smart surveillance system of distributed smart multi cameras modelled as agents. Paper presented at: 2016 Smart Cities Symposium Prague (SCSP); 2016; Prague, Czech Republic.
11. Lu X, Ye C, Yu J, Zhang Y. A real-time distributed intelligent traffic video-surveillance system on embedded smart cameras. Paper presented at: 2013 Fourth International Conference on Networking and Distributed Computing; 2013; Los Angeles, CA.
12. Hu L, Ni Q. IoT-driven automated object detection algorithm for urban surveillance systems in smart cities. *IEEE Internet Things J*. 2018;5(2):747-754.
13. Fan C-T, Wang Y-K, Huang C-R. Heterogeneous information fusion and visualization for a large-scale intelligent video surveillance system. *IEEE Trans Syst Man Cybern Syst*. 2017;47(4):593-604.
14. Shao Z, Cai J, Wang Z. Smart monitoring cameras driven intelligent processing to big surveillance video data. *IEEE Trans Big Data*. 2017;4(1):105-116.
15. Räty TD. Survey on contemporary remote surveillance systems for public safety. *IEEE Trans Syst Man Cybern Part C (Appl Rev)*. 2010;40(5):493-515.

16. Rantala A, Kylänpää M, Merilinna J, Nieminen M. Resilient and adaptive public-key infrastructure for distributed city-wide surveillance systems. Paper presented at: 2013 IEEE International Conference on Granular Computing (GrC); 2013; Beijing, China.
17. Esterle L, Lewis PR. Online multi-object k-coverage with mobile smart cameras. In: Proceedings of the 11th International Conference on Distributed Smart Cameras; 2017; Stanford, CA.
18. Esterle L. Centralised, decentralised, and self-organised coverage maximisation in smart camera networks. Paper presented at: 2017 IEEE 11th International Conference on Self-Adaptive and Self-Organizing Systems; 2017; Tucson, AZ.
19. Shiva Kumar KA, Ramakrishnan KR, Rathna GN. Distributed person of interest tracking in camera networks. In: Proceedings of the 11th International Conference on Distributed Smart Cameras; 2017; Stanford, CA.
20. Shiva Kumar KA, Ramakrishnan KR, Rathna GN. Inter-camera person tracking in non-overlapping networks: reidentification protocol and on-line update. In: Proceedings of the 11th International Conference on Distributed Smart Cameras; 2017; Stanford, CA.
21. Cai Z, Hu S, Shi Y, Wang Q, Zhang D. Multiple human tracking based on distributed collaborative cameras. *Multimed Tools Appl*. 2017;76(2):1941-1957.
22. Peng Y, Zhao Y, Zhang J. Two-stream collaborative learning with spatial-temporal attention for video classification. *IEEE Trans Circuits Syst Video Technol*. 2018;PP(99):1-1.
23. Abràmoff MD, Garvin MK, Sonka M. Retinal imaging and image analysis. *IEEE Rev Biomed Eng*. 2010;3:169-208.
24. Garcia Lopez P, Montresor A, Epema D, et al. Edge-centric computing: vision and challenges. *ACM SIGCOMM Comput Commun Rev*. 2015;45(5):37-42.
25. Bonomi F, Milito R, Zhu J, Addepalli S. Fog computing and its role in the internet of things. In: Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing; 2012; Helsinki, Finland.
26. Longo F, Bruneo D, Distefano S, Merlino G, Puliafito A. Stack4Things: a sensing-and-actuation-as-a-service framework for IoT and cloud integration. *Ann Telecommun*. 2017;72 (1-2):53-70.
27. Distefano S, Merlino G, Puliafito A. Enabling the cloud of things. Paper presented at: 2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing; 2012; Palermo, Italy.
28. Merlino G, Bruneo D, Longo F, Distefano S, Puliafito A. Cloud-based network virtualization: An IoT use case. Paper presented at: International Conference on Ad Hoc Networks; 2015; San Remo, Italy.
29. Flores H, Hui P, Tarkoma S, Li Y, Srirama S, Buyya R. Mobile code offloading: from concept to practice and beyond. *IEEE Commun Mag*. 2015;53(3):80-88.
30. Pohl C, Van Genderen JL, Zhang J. Review article multisensor image fusion in remote sensing: concepts, methods and applications. *Int J Remote Sens*. 1998;19(5):823-854.
31. Sahu DK, Parsai M. Different image fusion techniques–a critical review. *Int J Mod Eng Res*. 2012;2(5):4298-4301.
32. Wahyono, Filonenko A, Jo K-H. Unattended object identification for intelligent surveillance systems using sequence of dual background difference. *IEEE Trans Ind Inform*. 2016;12(6):2247-2255.
33. Gao Y, Ji R, Zhang L, Hauptmann A. Symbiotic tracker ensemble toward a unified tracking framework. *IEEE Trans Circuits Syst Video Technol*. 2014;24(7):1122-1131.
34. Cugola G, Margara A. Processing flows of information: from data stream to complex event processing. *ACM Comput Surv*. 2012;44(3):15.
35. Bruneo D, Distefano S, Longo F, Merlino G. An IoT testbed for the software defined city vision: The #SmartME project. Paper presented at: 2016 IEEE International Conference on Smart Computing (SMARTCOMP); 2016; St. Louis, MO.

How to cite this article: Dautov R, Distefano S, Bruneo D, et al. Metropolitan intelligent surveillance systems for urban areas by harnessing IoT and edge computing paradigms. *Softw Pract Exper*. 2018;48:1475–1492. https://doi.org/10.1002/spe.2586