The Journal of Systems and Software 166 (2020) 110587

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

STEP-ONE: Simulated testbed for Edge-Fog processes based on the Opportunistic Network Environment simulator

Jakob Mass, Satish Narayana Srirama*, Chii Chang

Mobile & Cloud Lab, Institute of Computer Science, University of Tartu, Estonia

ARTICLE INFO

Article history:
Received 30 November 2019
Revised 17 February 2020
Accepted 21 March 2020
Available online 27 March 2020

Keywords:
Internet of Things testbed
Discrete event simulation
BPMN 2.0
Fog computing
Edge process management
Smart city

ABSTRACT

The Internet of Things (IoT) has evolved from a cloud-based architecture towards multi-layer architectures such as Fog computing, where network edge devices perform processing and messaging tasks. Researchers studied managing and scheduling applications in these architectures extensively, however some issues, especially the effect of mobility, have been less explored. Additionally, evaluation of these mechanisms in realistic scenarios is difficult, as real-world experiments are costly and building composite applications with existing simulation tools is laborious. We call such composite applications which entail functions for handling connectivity and mobility disruptions in the edge network, *Edge Processes*. In this paper, we present STEP-ONE - a set of tools developed to simulate IoT systems where the management and application composition is handled with Business Process Model and Notation (BPMN) related standards. We demonstrate STEP-ONE with a smart-city scenario, where an application modeled as a choreography of processes is executed between different resources - cloud, fog and moving edge devices. We show how STEP-ONE enables constructing such a scenario and how it can be extended with algorithms and decision-mechanisms to adaptively drive the execution of such processes. We also present how detailed performance metrics can be extracted from the discrete event simulations run with STEP-ONE.

Contents

* Corresponding author.
E-mail addresses: jakob.mass@ut.ee (J. Mass), satish.srirama@ut.ee (S.N. Srirama).

https://doi.org/10.1016/j.jss.2020.110587
0164-1212/

1. Introduction

The vision promoted by Internet of Things (IoT) foresees environments that are abundant in devices with sensing, actuating, computing and networking functions. Through these capabilities, IoT technology has become important to a variety of domains: healthcare, intelligent transportation, manufacturing among others. IoT has evolved from a centralised architecture with two layers: 1) cloud and 2) end-devices; towards more de-centralized models with additional intermediate layers. These models, such as edge and fog computing (Bonomi et al., 2012), alleviate the network congestion and latency of passing data through the cloud layer and instead move the application logic closer to the end-users and end-devices, i.e. the *edge* (Mukherjee et al., 2018).

Inherent to IoT is the interoperability issue - devices are heterogeneous in their software and hardware configurations, communication interfaces, making the incorporation of them into cross-platform systems complex (Noura et al., 2018). This has commonly been solved through Service-Oriented Architecture (SOA) (Spiess et al., 2009), which supports syntactical software interoperability through standardized device- and service interfaces based on Web Service (WS) technology. Incorporating SOA also paves the way towards semantic interoperability (Andročec et al., 2018) (e.g. by describing the resources with the Resource Description Framework (RDF) data model; the ontology with Web Ontology Language (OWL); and querying the semantic information using SPARQL).

Another key issue is the resource planning and coordination of IoT device operations and data-flows to realize applications and business processes. Management becomes complex as processes in settings such as a smart city may involve several companies and public sector organizations while also providing a high degree of personalization to the end user (Grefen et al., 2018).

Business Process Management (BPM) - the practice of observing, designing, analysing, automating and improving business processes - has seen significant success in enterprise software (Dumas et al., 2018). BPM has been identified as the suitable enabler of effective management of IoT workflows as well (Janiesch et al., 2017; Chang et al., 2016; Grefen et al., 2018). The majority of existing BPM systems (BPMS) and Process-Aware Information Systems use SOA and WS technology standards when automating the sequences of tasks, decisions and messages that form a process. Another important standard in this context is Business Process Modelling and Notation (BPMN) 2.0, which describes the processes themselves - on one hand BPMN 2.0 is a visual standard for intuitive design of processes, while on the other hand it is interpretable by BPMS for automated execution.

Using BPM with IoT has great potential - the abstraction provided by BPM accelerates developing new IoT business solutions, the BPMS monitoring functionality gives analytic insight into how well processes are performing and how to optimize them (Kalbag and Silverman, 2014). (Janiesch et al., 2017) outline the various benefits of joining BPM with IoT, such as automating the detection of starting and ending activities and enhancing planning of

large-scale IoT sytems. BPMS-integrated IoT has gathered attention of researchers from various perspectives: modelling the IoT-aware processes (Meyer et al., 2015; Domingos and Martins, 2017; Brouns et al., 2018; Martins and Domingos, 2017), adapting processes to unexpected events and contextual changes (Marrella et al., 2018; Seiger et al., 2017; Peng et al., 2013), dynamic service discovery, composition and selection (Chang et al., 2015; Dar et al., 2011); and various case studies (Tüysüz et al., 2013; Tranquillini et al., 2015; Pryss et al., 2015). These works reflect the current paradigm in which process-aware IoT systems are usually based on the 2-layer IoT model, where a central Process Engine orchestrates the resources and services. This design follows from the conventional enterprise BPMS, which too takes a centralized approach.

With the emergence of edge and fog computing, there is a need to adapt BPMS to the edge devices (Kalbag and Silverman, 2014), so that processes can be orchestrated and executed directly in the edge, we call this notion *Edge Process Management* (EPM). EPM brings several benefits: 1) following the edge computing principle, edge networks can provide the best performance in terms of communication latency; 2) privacy, as the tasks and data involved in an edge process may be kept on the local devices and network; 3) resilience and fault-tolerance: in cases where infrastructural damage or outages occur, the edge process can continue enacting (consider disaster scenarios) (Marrella et al., 2016; Mass et al., 2016).

However, the edge environment does impose the challenge of highly volatile context: connectivity is intermittent and the mobility of nodes (Giang et al., 2018; Bonomi et al., 2012) causes the number of nearby devices to be in constant flux, affecting the availability of services and radio interference. Thus, the devices need offline execution and autonomous collaboration capabilities (Seiger et al., 2017), which can take into account mobility context.

Adopting BPMS for execution on end-devices such as mobile phones has been researched several times (Hackmann et al., 2006; Dar et al., 2015; Schobel et al., 2016). These works host a standards-compliant process engine on the device. However, considering that IoT and edge systems are to involve huge numbers of devices, the evaluation of most existing related works is done at a scale of up to a few devices (e.g. 2 robots and 1 stationary system in Seiger et al. (2017)). Indeed, large-scale real-world experiments are costly to perform, so the common approach is to simulate the behaviour (Dede et al., 2018). However, to the best of our knowledge, no existing simulation tool has adapted a process-oriented and mobility-aware perspective to IoT systems.

To support development and research of process-oriented scenarios and models for EPM in environments such as smart cities, where mobility aspects are common, we present **STEP-ONE**: a **S**imulated **T**estbed for **E**dge-Fog **P**rocesses based on the **O**pportunistic **N**etwork **E**nvironment Simulator. STEP-ONE is an extension to the Opportunistic Network Environment simulator (ONE) by Keränen et al. (2009), adding process execution support on the hosts. This allows discrete event simulation of scenarios, where the application logic is defined as BPMN 2.0 processes. STEP-ONE supports defining these processes with a set of ready-to-use process components, such as Message tasks for inter-process messaging across hosts, events and signals for time, mobility and connectivity-related situations and configuration options for varying the executed processes, their inputs and the process engine parameters when executing a batch of simulations. The testbed provides feedback in the ONE graphical user interface during execution and a visual modeller for creating processes. For results interpretation, we make use of the report generation features available in ONE. The process execution and visual modelling tools in STEP-ONE are based on the Flowable[1] BPMS.

[1] https://www.flowable.org/

We demonstrate STEP-ONE with a hypothetical Smart City scenario set in the city of Tartu, where the city uses a *edge process*-based approach to monitor the street and road conditions in a decentralised fashion. This testbed enables enacting scenarios with large number of hosts in city map-based environments, thus allowing to explore the question of mobility, resource scheduling and placement effect on processes in the edge and fog.

The rest of the paper is structured as follows: Section 2 describes background on process-oriented management of IoT and how it can be used for fog and edge computing. The section also gives a comparative overview of other simulation tools related to STEP-ONE. The requirements, architecture and design of STEP-ONE are covered by Section 3, while the implementation details of the testbed are given in Section 4. Section 5 explains the Smart City scenario and the processes it is composed of, then, in Section 6 we present the performance evaluation of cloud- and fog configurations simulated in the scenario. The paper is concluded in Section 7.

2. State of the art and related works

This section explains the core ideas behind Edge Process Management, highlighting the need for a testbed, then we discuss the suitability of existing simulation tools for mobility- and networking simulation to EPM.

2.1. Process-aware information systems

The widespread standard for defining and executing processes is BPMN 2.0 (O.M. Group, 2011). Core components that define the control flow of BPMN 2.0 processes are Activities (Tasks or Subprocesses), Events, Gateways and flows. Tasks represent units of work with some time duration (e.g. downloading some data; sending a message and waiting for it to be received), while events are instantaneous (e.g. a connection has been lost). Along with gateways, these components can be assembled into sequences. In addition to the control flow, processes involve a data flow - some process variables attached to the process. The variables may change during execution or new variables may be created, e.g. a Receive Message task may create process variables based on the information in the message, and subsequent tasks, gateways may influence the control flow based on the variables.

Process-aware Information Systems involve a *process engine* component which provides among other functions: interpretation of process models, assigning resources to process tasks and scheduling the execution of processes, logging a history of process executions, which later allows to analyze performance and shortcomings.

2.2. Process management for Internet of Things

The survey by Chang et al. (2016) gives an overview of different process execution approaches in the IoT context.

The approach commonly found in process-aware IoT systems follows the 2-layer architecture, hosting a process execution engine at the data center. In this case, edge IoT devices can participate in the process execution as providers of atomic services, which the engine invokes. Alternatively, the process execution could be performed on devices and gateways. However, hosting a standards-compliant process execution engine usually sets higher requirements for device hardware. For less capable devices, the process model can instead be converted to executable code corresponding to the control- and data flow of the process and then be run on the device. Compared to the previous option, this does limit the flexibility of the system.

In the next subsection we will be focusing on related works where the process engine is hosted on the device.

2.2.1. Embedded process engine based execution

Since the emergence of devices such as PDAs or smart phones, the possibility of automated execution of processes on these devices has been studied, allowing direct involvement of business activities and information at the mobile device. In earlier mobile-hosted BPMS, such as Sliver (Hackmann et al., 2006) the dominant standard for defining work flows was WS-BPEL.

As sensor-enriched devices have become commonplace, mobile process engines have begun integrating sensor support at the process-level. For example (Schobel et al., 2016) developed a mobile process engine for questionnaire-based data collection, using the mobile's integrated sensors.

In the above cases, processes are defined at a central location and deployed to the devices. Execution on the device can then be done offline, irrelevant of connectivity issues. This is fine for processes where the tasks only involve activities local to the device/user.

More complex processes may involve several devices, external sensors or a choreography of processes hosted by different devices. Here, execution frameworks with collaboration and choreography features have arisen. This is addressed in CiAN Sen et al. (2008), an extension of Sliver,where the process is de-composed into sub processes and distributed among hosts.

In Seiger et al. (2017), the distributed process-execution system consists of *peers* and *superpeers*. While both involve a process engine, the superpeers are more complex and include management of peers and distribution of processes among its peers. The proposed framework includes context-aware aspects to provide self-healing through a MAPE-K feedback loop.

Another reason for distributing the process can be the case when hosting a full process engine on the end device is not feasible due to hardware constraints. Here, out-sourcing the process to other nearby resources with process execution capabilities (Zaplata and Lamersdorf, 2010) is viable. Dar et al. embed a BPMN 2.0 compliant engine on Android smart phones (Dar et al., 2015). They proposed a framework where multiple devices hosting BP engines can collaborate, as demonstrated through an Ambient Assisted Living scenario, monitoring a patients status as they move within their home and outside of their home, with the process execution accordingly executed on different devices.

With the concept of Mobile IoT, the similar principle expands from smartphones to devices such as vehicles, drones, robots, etc. Here handling the intermittent connectivity and service availability remains a challenge. In this light, (Zhang et al., 2018) provide resilience through a multi-engine mechanism, where nodes in a MANET host process engines, synchronizing the state of the execution across the engines, while a single master engine is responsible for the execution. Another work which addresses resilience through recovery from unanticipated exceptions is the SmartPM system (Marrella et al., 2016), this is achieved through artificial intelligence methods such as situation calculus and classical planning. Connectivity-related events can be predicted based on contextual information such as mobility can be used to establish task execution schedules, e.g. when offloading tasks to fog servers (Mass et al., 2019).

2.3. Edge-Fog computing, edge process management

Fog Computing, which brings virtualized, low-latency execution environments near the edge network to support low-latency IoT applications (Bonomi et al., 2012) has arisen to complement the widely used Cloud Computing concept. The Fog architecture relies on certain *fog-enabled* infrastructure, e.g. WiFi routers, set-top boxes, network gateways - proximal computational resources which provide an environment for hosting applications (e.g. *Cisco IOx*). While Fog complements Cloud, here we focus on the lower layers of the Fog architecture - the edge devices and fog intermediary gateways supporting them.

Among the fundamental research challenges of fog computing (Mukherjee et al., 2018), we highlight some that could effectively be implemented with process-aware information systems:

- how to distribute computational tasks/applications within the network?
- how to offload tasks (e.g. which tasks to offload, to where), while assuring SLA (Service-level Agreement), QoS (Quality of Service) and QoE (Quality of Experience) (Mahmud et al., 2019) such as latency (Rodrigues et al., 2017), resource utilization (Du et al., 2018), task priority (Adhikari et al., 2019), energy efficiency (Yang et al., 2018), profit-aware (Mahmud et al., 2020), etc.?
- how to manage the deployment of tasks and execution in heterogeneous infrastructure?
- how to migrate computation within Fog nodes

From the processes perspective, this means distributing execution of some of the (sub)processes to Fog nodes situated in proximity to the end clients, by hosting the process engine in the Fog infrastructure, this is feasible using light-weight BPMS such as Flowable, that can be deployed within a second (Flowable, 2009).

Since decision-making and exception handling are integral to business processes, the algorithms and models for these issues (such as SLA and QoS control of Fog applications) can be integrated into process engines and be directly invoked at the process abstraction level. A lot of fog applications, which deal largely with messaging and deployment tasks can comfortably be modeled as processes. Since the definition of processes is standardized, deployment across devices is simplified.

More recent works have begun exploring this process-based fog computing direction as well. Cheng et al. (2018) describe a IoT execution environment which hosts process engines on the smartphone, while Fog nodes manage service discovery, deployment and event management.

Node-RED, which is a (work)flow-based approach to defining IoT application behaviour has been used by Giang et al. (2018), they consider dataflow-oriented applications, with the components distributed among different nodes within a Fog network.

We have previously studied execution scheduling of process sub-tasks based on context data about fog infrastructure and host movement (Mass et al., 2019) and run-time migration of processes between devices (Mass et al., 2016).

2.4. Simulation tools

As we mentioned, simulating these systems is important to evaluate their performance at reasonable cost. Simulating mobility behaviour is usually based on discrete-event simulation, where the modelled world state changes are caused by events occurring at discrete points in time. As opposed to continuous simulation, a discrete event simulator (DES) jumps from one event to the next, the system state between events is not modelled. In this subsection we outline the better-known networking and mobility simulators relevant to edge-fog computing.

2.4.1. Ns-3

NS-3[2] is a general-purpose network DES. It includes simulation models that closely reflect the Linux networking stack and as a re-

[2] https://www.nsnam.org/

sult can be used as a real-time network emulator, enabling plugging it into real-world networks and integrating with real-world protocol implementations.

Ns-3 primarily targets IP networks at its core, but also includes simulation models for wireless technology such as Wi-Fi, LTE, WiMax. The ns-3 project has a large community and consortium behind it, contributing development of various modules, including an energy framework and mobility modeling.

The mobility module currently supports randomwalk and random/non-random waypoint mobility models. While ns-3 does not support map-based movement models or schedule-based user movement out of the box, it can be integrated to mobility simulators such as SuMO, importing map-based mobility traces generated by these tools.

The energy framework of ns-3 takes into account several PHY layer properties in the case of WiFi.

As such, ns-3 offers multiple options for modeling physical layer properties of wireless links, includes a very detailed modeling of the OSI layers 2–4, and provides base classes for defining applications (layers 5–7).

2.4.2. OMNeT++

OMNeT++ (Varga, 2010) DES provides a core set of network modelling and simulation blocks which have been used in several specific extension frameworks for IP networks, vehicular networks, wireless networks, but also from domains such as sickness dissemination.

INET INET is the OMNeT++ framework for IP-based networks and protocols at the layers OSI 2–4 in wired and wireless networks. It includes a rich set of implementated models for wireless technology such as IEEE 802.11 or 802.15.4. In addition to basic set of deterministic mobility models[3], INET provides trace-based and stochastic mobility models. Further, it has energy models[4] - both power generators and consumers.

VEINs VEINs is an OMNeT++ based framework for inter-vehicular communication simulation, using SUMO simulator for road mobility modelling. It can import OpenStreetMap scenarios with buildings, speed limits and traffic light restrictions.

2.4.3. ONE Simulator

The Opportunistic Network Environment simulator (ONE) was developed for modelling behaviour of opportunistic networks and routing protocols there-in. The networking is captured at the link layer, so the PHY layer modelling is not very complex, as ONE focuses on store-carry-forward networking.

The radio links have communication range and bit-rate properties and don't include aspects such as signal attenuation. However, some of the provided models reflect interference and signal strength by updating the bit-rate based on the distances of hosts or by the number of other hosts nearby.

The set of movement models includes both random-based movement as well as map-based movement models, and for instance user behaviour modelling (e.g. workday model) or group-based behaviour, such as a group of pedestrians using public transport.

2.4.4. SUMO

Simulation of Urban Mobility (SUMO) (Behrisch et al., 2011) is a renowned simulation framework for road vehicles, and pedestrians and their routing, accounting for aspects such as traffic lanes, intersections, traffic lights and multi-modal traffic. While SUMO does not provide any networking features, the rich mobility modelling of SUMO has been recognized by the above mentioned networking simulators (OMNeT++, Ns-3), as they provide means to integrate SUMO-based traces into them.

SUMO supports importing maps from OpenStreetMap, VISUM, VISSIM and NavTeq.

2.4.5. iFogSim, EdgeCloudSim

Unlike the above mentioned tools, iFogSim (Gupta et al., 2016) specifically targets IoT, edge and fog computing environments. The project was developed for evaluating resource management policies with respect to latency, energy consumption, bandwidth usage and operational costs. iFogSim extends the CloudSim project, as such it inherits its features such as executing and migrating applications / VMs, and managing virtualization, resource management and scheduling functions for cloud datacenter.

However, iFogSim does not capture lower level networking properties such as interference. The infrastructure is defined as a hierarchy, where messages can be passed among child-parent pairs, in bottom-to-top direction (e.g. edge to fog). It does not support device-to-device communication, with the exception of the cloud layer, where data centers can exchange messages. iFogSim provides a set of entitiy classes for the main node types, such as FogDevice, Sensor and Actuator, and classes to model the Fog applications, which are a set of application modules that form a directed acyclic graph.

FogDevice classes can be described by their memory, processor, storage size, uplink, and downlink bandwidths properties, while sensors, actuators have a latency property.

While iFogSim does not provide any mobility modeling features, another tool based on CloudSim, called EdgeCloudSim (Sonmez et al., 2017), which specifically targets the edge computing environment, does include a nomadic mobility model. Further, it provides a networking module to reflect edge WLAN/WAN communications and an Edge Orchestrator module which performs decision-making for incoming requests, tasks.

2.4.6. Discussion

While tools such as NS-3 provide the most detailed networking simulation, the learning curve and set-up can be considerable. In novel areas such as Edge Process Management, quickly evaluating a selection of approaches at a higher abstraction level can be more important than the specific, realistically defined (and usually smaller scale) scenarios (Caro, 2003). In fact, more detailed simulations can sometimes obscure effects of fundamental changes in the algorithms, since they involve managing a large amount of simulation parameters in addition to the algorithms themselves (Caro, 2003).

Meanwhile, the existing fog-oriented tools like iFogSim provide powerful datacenter/virtualization modeling features derived from CloudSim, however, the definition of application logic tends to be limited to certain types of applications. Table 1 summarizes this comparison of the related tools.

Considering the above, our testbed is based on ONE. On one hand, its less-detailed lower-level networking favors rapid prototyping considering the complex vs abstract simulation debate mentioned above. Secondly, we considered the programming languages. Several industry-recognized process managament software suites such as jBPM, Camunda, Flowable and Bizagi BPM are based on Java, as is ONE. This makes integrating initial results from the testbed into the real systems (and vice versa) easier than it would in the case of NS-3 or OMNeT++. Fig. 1 highlights how our tool expands upon ONE while highlighting the focuses of the other related tools.

[3] https://inet.omnetpp.org/docs/users-guide/ch-mobility.html
[4] https://inet.omnetpp.org/docs/users-guide/ch-power.html

Table 1
Comparison of networking and mobility-oriented simulation tools. Acronyms: OSM - OpenStreetMap.

	Wireless Networking	Mobility	Edge & Fog Computing	Learning Curve	Languages
NS-3	Detailed	random, way-point base, map-based with SuMO traces	None	High	C++, Python
iFogSim	Basic	None	Resource Management policies	Medium	Java
EdgeCloudSim	Medium	Nomadic models	Edge Orchestration	Medium	Java
SUMO	None	Detailed map-based & random models, incl. lanes, traffic	None	Medium	C++
ONE	Medium	Map-based & random, OSM support	None	Medium	Java
OMNeT++	Detailed (INET)	Trace-based, stochastic, OSM support with VEINs + SUMO	None	High	C++

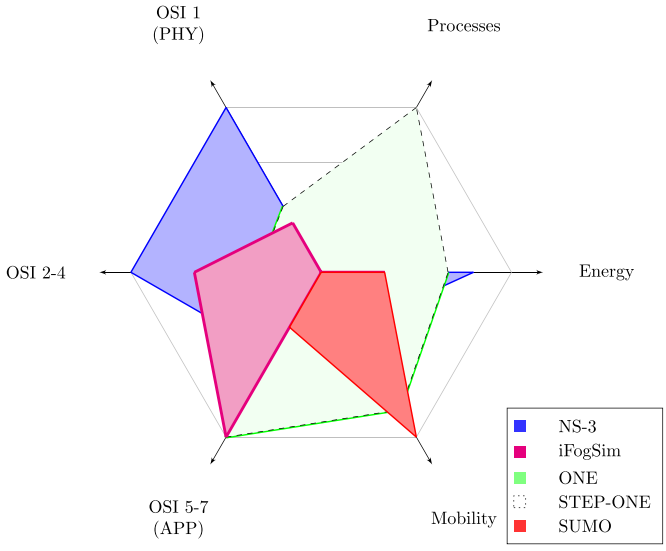

Fig. 1. Comparison of STEP-ONE with related simulators in the level of detail addressing OSI network layers, mobility, energy and processes modeling.

3. STEP-ONE software requirements & design

Given its mobility, networking and communication features (as discussed in Section 2.4), we have based the core of the testbed on the ONE simulator. Before showing how STEP-ONE expands on ONE in order to realize process-based scenarios, we first describe ONE's main concepts. Then we define requirements for STEP-ONE and proceed with a detailed explanation of how the tool fulfils the requirements.

3.1. Existing ONE simulator features

Simulations in ONE are defined by configuration files that describe the nodes involved in the simulation, their behaviour and capabilities in terms of communication interfaces, (delay-tolerant) routing mechanisms and movement. Given a configuration file, ONE simulations can be run: **a)** using a graphical user interface (GUI), which displays the movement of the nodes in the simulated world, connectivity and messaging events or **b)** in batch-mode, without a GUI, which is useful for running permutations of simulation configurations at higher performance. The simulation results are captured in report files, which can be selectively enabled. These reporting modules produce summaries of messaging statistics, e.g. delivery ratios, performance (latency), movement and connectivity information, such as contact times.

Hosts in ONE can run ***applications***: custom programmable modules, that handle incoming messages, create new messages, or run custom logic at every discrete time step. The implementation of applications in ONE is left to the developer. In addition to applications, messages can be produced from an external event generator.

3.2. Requirements for STEP-ONE

The above functions provide a foundation for the test bed, however, to achieve process management support, we define the following requirements. The test bed should:

- **REQ1:** support executing BPMN 2.0-defined processes on simulated hosts.
- **REQ2:** allow executed processes to interact with the simulated world and other simulated processes through messages, events and signals.
- **REQ3:** provide means to specify the process-related parameters for the simulation from a configuration file.
- **REQ4:** provide means to create process definitions and related messages using a visual editor.
- **REQ5:** allow specifying the computational complexity of certain process sub-tasks and specify the computational capabilities of host groups.
- **REQ6:** support generating reports based on the process execution-related logs.
- **REQ7:** be extendable with decision mechanisms and algorithms to study their effects on the scenario and process execution(s).

3.3. Process engine application

To address REQ1, we provide a specific ONE *application* implementation - a Process Engine Application (PEA) (See Fig. 2). During simulation runtime, the PEA handles deployment of BPMN 2.0 process definitions or starting execution of individual instances of deployed processes through respective messages from other hosts (their processes). Alternatively, these actions can also be performed at initialization of the simulation based on the configuration file. The PEA enables various functionality to executing process instances based on REQ2, for example: connectivity events within ONE are mapped to events which processes can catch and *Message tasks* defined within a process will create a ONE message to be sent to other hosts.

3.3.1. Messaging

Messaging is the core functionality of both ONE and STEP-ONE. STEP-ONE processes exchange *Process Messages*, that are encapsulated within standard ONE messages. This allows the message routing to be handled by ONE networking capabilities.

ONE messages are defined by an ID, size, recipient host address and custom extension properties (key-value pairs). STEP-ONE uses these extensions to attach extra data which define a Process Message, for example: attaching process variables (optional) or *name* of the message (mandatory), processes which are interested in receiving messages have to subscribe to them by name.

STEP-ONE distinguishes four types of Process Messages, all of which can be sent from or received within a process execution, except "ONE Application Message", which can only be sent from processes.

- **Generic Process Message** - A generic message which originates from a "Send Message" task within a process. The receiving pro-

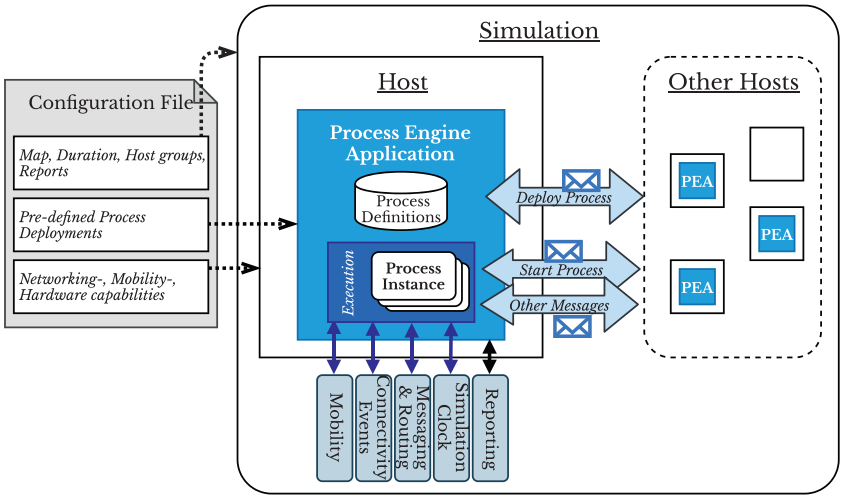

Fig. 2. Overview of STEP-ONE architectural Components.

cess engine forwards the message to the process instance which is subscribed to the message (if any).

- **Deploy Process Message** - when a process engine receives a "Deploy process" message, it attempts to deploy a process definition based on the resource file path attached to the message.
- **Start Process Message** - when a process engine receives a "Start process" message, it attempts to start a new process instance, given the identifying process key contained in the message. This assumes that a process definition with a start event corresponding to the message name has been deployed.
- **ONE Application Message** - this message can be sent from a process and is targeted at other ONE applications (that aren't PEA-s). For example, invoking an atomic service from a central process engine would be done with this type of message.

Section 4.1 discusses the implementation details how these messages can be defined within a Process and how the process engine maps them to ONE messages and vice versa further.

3.3.2. Events, signals

The process engine also handles events from the simulation world and notifies running process instances which are subscribed to related events. This is particularly useful for driving the process execution based on the events - waiting for a connectivity-related event before proceeding with a message sending task or interrupting a flow in reaction to an event. In the following, we list the events STEP-ONE processes can handle.

- **Connectivity Events** - When a network connection is established or broken between hosts, the events are translated into BPMN signals.
- **Timer Events** - Processes which contain timer-based events (e.g. timeouts or cyclic repeating behaviour) are notified of time elapsing based on the simulation clock.
- **Location Events** - When a moving host arrives at a location to which a running process instance is subscribed (based on coordinates), it is broadcast to the process engine as a BPMN signal.
- **Message Events** - This allows processes to wait for a given message to arrive before proceeding with the control flow.

3.3.3. Tasks & activities

Processes are sequences of tasks. STEP-ONE provides a set of useful BPMN tasks which help quickly define simulated scenarios. STEP-ONE provided tasks are listed as follows:

- **Send Message Tasks** - for creating each of the above message types, a *Send Task* subvariant exists.
- **Simulated Work Tasks** - to simulate long-running, compute-intensive tasks (See REQ5 in 3.2), this task type is defined by a work size parameter, how large the task is in terms of millions of instructions (MI).
- **Generic Service Tasks** - Service tasks are generic tasks, whose implementation is up to the developer. This allows creating experiment-specific functionality, programmed in Java. For instance, a scheduling algorithm may be implemented as a Service task, which based on some process variables, produces a scheduling decision and writes it into another variable, used in the rest of the process flow.

In general, the duration of tasks in simulation clock time is instantaneous, except for Simulated Work Tasks, whose duration depend on the node's hardware configuration and Send Message tasks configured not to finish execution before the message has been received at the destination.

3.4. Discussion

Since in addition to the above-mentioned STEP-ONE messages, tasks and events, BPMN 2.0 allows to define various control flow structures such as AND or (X)OR gateways, and custom logic can be attached through Service Tasks, execution listeners, etc, complex behaviour can be modelled and enacted.

As we will show in Section 5, one class of applications to model as processes are Fog computing scenarios, where ONE-s networking abilities and STEP-ONE's messaging, simulated work tasks will be used.

4. Implementation

STEP-ONE is packaged as a Gradle-based Java project and is available as a public GitHub repository[5], with additional documentation and reference materials on how to use it.

The Process Engine Application embeds an instance of Flowable BPMS (version 6.4.2), which is a light-weight process software that supports a variant of the BPMN 2.0 standard - namely some Flowable-specific extension elements have to be included in the process model to enable automated execution.

To improve simulation performance, STEP-ONE uses Flowable with an in-memory database and features such as process history tracing can be turned off.

[5] https://github.com/jaks6/step-one

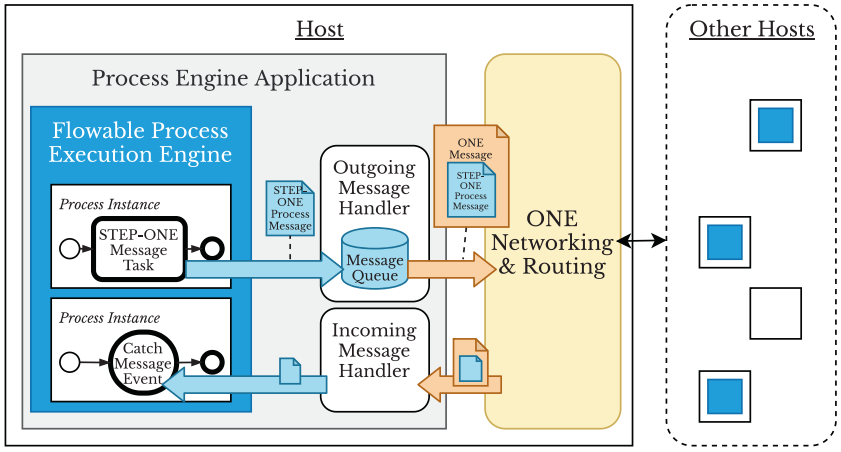

Fig. 3. Overview of how Process Messages are mapped to ONE messages and vice versa.

Implementation of the tasks mentioned in Section 3.3.3 is based on the *Service Task* BPMN construct. Custom Service Tasks in Flowable must implement the `JavaDelegate` interface, which defines an `execute()` method called whenever the process execution flow reaches that task (Listing 2 shows an example). When describing a Service Task within a BPMN 2.0 process definition, a XML attribute specifies the Java class which implements the `JavaDelegate` interface. Thus, we created Java implementations for the various task types.

4.1. Messaging implementation

STEP-ONE provides an implementation for each of the mentioned Message Tasks. When executed, the Java code constructs a ONE message and embeds a Process Message into it. After constructing the message object, it is added to an "outgoing messages" queue. The PEA queries the queue on each simulation update, and hands them over to networking and routing modules of the ONE, which handle the message transfer to other hosts (Fig. 3).

When the target PEA receives a ONE message, the contents are parsed and if the message contains a STEP-ONE Process Message, the appropriate Flowable API (e.g. deployment, process start, message received) is invoked with the message contents and attachments.

4.1.1. Message parameters

When defining a STEP-ONE Task, in addition to selecting the appropriate Java implementation, a number of task-specific parameters must be specified, as BPMN *extensionElements*, more precisely Flowable's *extension fields*, which are parsed by the Java implementation.

The compulsory parameters for all types of message tasks are: Message Name, Message Size, and Destination Address.

Additional parameters vary by message task type. Both the compulsory and additional values are part of the Process Message object that is encapsulated in a ONE Message. The full list of parameters is shown in Table 2. These parameters can either be defined with static values, or alternatively using the *dynamically evaluated expressions* feature of Flowable. For example, instead of hard-coding a message recipient address, it can be inferred from a process variable during process runtime. Listing 1 shows an example of a BPMN 2.0-defined message task using the extension element parameters - the name and message size are defined with fixed values, while the destination address is a dynamic expression.

4.2. Events & signals implementation

Movement and connectivity related events are handled via Flowable/BPMN *signals*. Like messages, signals have names and processes can subscribe to them based on their names. Further, signals can have process variables attached to them.

Time-based events on the other hand are handled with a modification to Flowables time/calendar APIs that use ONE-s simulated time instead of the real world time. Time-based events are defined with an extension element, which specifies the value and whether it is a repeating time period, a single duration.

Table 3 lists the exact signal names, attached process variables and expected time format for Timer events.

4.3. Simulated work task & cost management

STEP-ONE provides a basic modelling of computational work for Simulated Work tasks mentioned in 3.3.3.

A host's computing capability is defined by: number of CPU cores: *NoOfCpus*, and speed per CPU: *CpuSpeed* (in DMIPS, Dhrystone Million Instructions Per Seconds - a commonly used metric for performance evaluation (Lee et al., 2017)). All CPU jobs, as defined by Simulated Work Tasks, are placed in a registered job queue. On every simulation update, the PEA manages the execution of job units in the queue using the algorithm shown in Algorithm 1. While a single job can only be run by a single CPU at once, multiple jobs can be processed in parallel by different CPUs and a single CPU can work on several jobs sequentially within a single update iteration if the job sizes are sufficiently small.

4.3.1. Cost estimation helper

It is also possible to define cost parameters to a node's computational and networking resources. For compute resources, cost is defined per CPU usage at a granularity of either hours or seconds, (e.g. $0.02 per hour). For networking, cost is defined for each interface as cost of transmitting 1MB. If these values are specified, it is possible to use the *CostHelper* class, which provides methods:

1. getCostForTransmission(x, y) - returns the cost of transmitting *x* bytes over network interface *y*.
2. getCostForTask(z, w) - returns the cost of processing a task with size *z* (in MI) on host *w*. The time of processing the given task is rounded up to the specified cost granularity level.

The above methods can be used in Applications or Process Service tasks for making decisions based on costs and budgets. For instance, if a process involves deploying and starting another process including Simulated Work tasks to some external host, the

Table 2
BPMN 2.0 Message Task parameters in STEP-ONE. "Req." - Required.

Parameter name	Applies to	Req.	Description
Name	All Messages	+	String. Used to forward the message to all process instances that have subscribed to this name.
Size	All Messages	+	bytes
Destination	All Messages	+	ONE host address
Target Execution ID	Generic Messages	-	Process Instance Execution ID. Allows the message to be forwarded to the exact corresponding process instance.
Included Process Variables	Generic Message, Start Process Message	-	comma separated list of values attached to the message. The variables become available to the destination process instance
Resource Path	Deploy Process Message	+	location of BPMN 2.0 process definition file (on simulation host machine classpath).
AppID	ONE App Message	+	ONE Application identifier - which ONE application this message targets

```
<serviceTask id="exampleMessageTask" name="Example Message" flowable:class="ee.mass.
    epm.sim.task.MessageTask" flowable:triggerable="true" flowable:type="stepone_msg">
  <extensionElements>
    <flowable:field name="msg_name">
      <flowable:string><![CDATA[Hello World]]></flowable:string>
    </flowable:field>
    <flowable:field name="msg_size">
      <flowable:string><![CDATA[1]]></flowable:string>
    </flowable:field>
    <flowable:field name="msg_destination">
      <!-- Using Dynamic expression evaluation -->
      <flowable:expression><![CDATA[${myDestinationVariable}]]></flowable:expression>
    </flowable:field>
  </extensionElements>
</serviceTask>
```

Listing 1. Generic Process Message Task example.

```
class CostAwareTask implements JavaDelegate {
    @Override
    public void execute(DelegateExecution execution) {
        DTNHost cloud;
        int taskWorkSize;
        int taskBandwidth;

        // Assign values to above, e.g. from process variables,
        // process field expressions or from simulation instance
        ...
        double cpuCpost = CostHelper.getCostForTask(taskWorkSize, cloud);
        double networkCost = CostHelper.getCostForTransmission(taskBandwidth, cloud.
            getInterface(1));

        // Make decision based on cost - e.g. whether to invoke cloud or not,
        // set process variables accordingly, etc.
        ...
        }
}
```

Listing 2. Java example of a Service Task implementation using the Cost-related functions in STEP-ONE.

CostHelper can be used to first perform some calculations to determine the most cost-efficient qualifying node.

4.4. Other features

4.4.1. Full duplex network interface

Network interfaces provided by ONE simulator operate in half-duplex mode and the message routing is restricted to sending 1 message from a single host concurrently. This becomes an issue if a single host is communicating large messages with several hosts simultaneously. Thus, we implemented a full-duplex network interface `SharedBandwidthInterface`, which considers network speed of the upload and download links separately, and divides the bandwidth between all active transmissions. `SharedBandwidthInterface` is to be used in combination with STEP-ONE-s `DirectMultiTransferRouter`, a message router that enables transferring multiple messages at a time.

Table 3
Event & signal constructs in STEP-ONE[6].

Name	Description	Defining in a process
Timer Events	Notifies host of time passing. Defined with ISO-8601, e.g. PT45S represents a time duration of 45 seconds	Define the Timer Event and specify the duration
New Connection Signal	When a connection is established, both hosts receive signal with name *Device Connected* and the other device's address attached as a process variable	Define a signal catching event for respective signal name
Disconnected Signal	When a connection is lost, both hosts receive signal with name *Disconnected* and the other device's address attached as a process variable	Define a signal catching event for respective signal name
Location Signal	When host position reaches a given coordinate, the host process engine receives the signal with name *New Coordinate*.	Define a signal catching event with extensionElement named "coordinate" giving the coordinate pair as value and specify a location execution listener.

Algorithm 1: Processor routine on every simulation update.

```
On every simulation update: for cpu_i ← 1 to NoOfCpus do
    instructionsLeft ← CpuSpeed ;
    while instructionsLeft > 0 and not (empty(activeJobs[cpu_i])
    and empty(registeredJobs)) do
        if empty(activeJobs[cpu_i]) then
            activeJobs[cpu_i].add(registeredJobs.dequeue());
        end
        currentJob ← activeJobs[cpu_i].peek();
        currentJob.workLeft −= instructionsLeft ;
        if currentJob.workLeft ≤ 0 then
            activeJobs[cpu_i].dequeue() ;
            processEngine.notifyJobFinished(currentJob));
            instructionsLeft −= currentJob.workLeft ;
        else
            instructionLeft = 0;
        end
    end
end
```

4.4.2. Reporting tools & GUI additions

To complement the existing set of reporting facilities in ONE, we have added two additional modules: *BpmAppReporter* and *DetailedBpmReporter*.

The former gives aggregate statistics of processes run on all hosts during the simulation, such as total number of processes started/completed/cancelled, number of Process Messages sent/received, number of Process Activities started, completed, cancelled, and so forth. DetailedBpmReporter, on the other hand, records data about every individual executed process instance: the start and end timestamp of the process, which host executed it and various aggregate counts such as messages received/sent, activities started/-completed as part of that process.

Alongside the reports, another way of getting feedback on process executions in STEP-ONE is using the GUI. We have modified ONE-s GUI, which provides routing & message-related details, to also show process states during runtime (See Fig. 4). The GUI panel captures running and finished process instances of a host, their process variables, process start time and currently executing subtask (in case of running processes).

4.4.3. Configuration

STEP-ONE uses ONE-s configuration file approach to enable setting the parameters of its modules. STEP-ONE introduces the following new configuration options:

- The number of CPUs and CPU speed of hosts
- Compute and networking costs of hosts
- The Upload and Download speed of `SharedBandwidthInterface`
- Process definitions to be auto-deployed at beginning of simulation
- Processes that should be auto-started at beginning of simulation, including process variables that may be attached when starting them.
- Some of the features, such has generating signals for connectivity and movement events can be selectively disabled, as they have some effect on performance

An example of these configuration options is shown in Listing 3.

4.5. Modelling processes

Manually writing the XML of BPMN 2.0 models including the STEP-ONE/Flowable-specific extension elements and classpaths of STEP-ONE implementation classes can be tedious. The easiest way to attain a compatible process model is to use the Flowable modeller, which is a web-based application for visually defining the process - its' tasks and their parameters. Hence, STEP-ONE includes an extended version of the Flowable modeller, which adds the main STEP-ONE task types to the modellers palette and allows configuring their parameters in a similar fashion to the overall experience of Flowable modeller.

With the modeller, custom task behaviour can be defined by specifying execution listeners or specifying the Java class implementation for a task. The STEP-ONE extension pre-fills the Java implementation values for the STEP-ONE tasks.

Fig. 5 shows a screenshot of a STEP-ONE model designed with the modeller. As depicted on the bottom of the image, parameters such as the name, size of the message can be defined in text fields, note that these fields can also be specified as dynamic variables, e.g. the message Destination Address on the figure has been defined as the value of variable `videoAnalyserAddress`.

5. Case study

We illustrate the usage of STEP-ONE with a hypothetical Smart City scenario:

The city of Tartu performs monitoring of street and road conditions along public transport routes using image processing techniques. Video footage captured by camera-equipped buses is analysed for objects such as potholes, cracks, snow and ice build-up using image processing techniques (Zakeri et al., 2017). The analysis is done individually per road segment, each segment starting and ending at a bus stop. Based on the location, traffic frequency and weather conditions, the information system schedules analysis requests for individual road segments at different rates. A segment analysis request initiates video

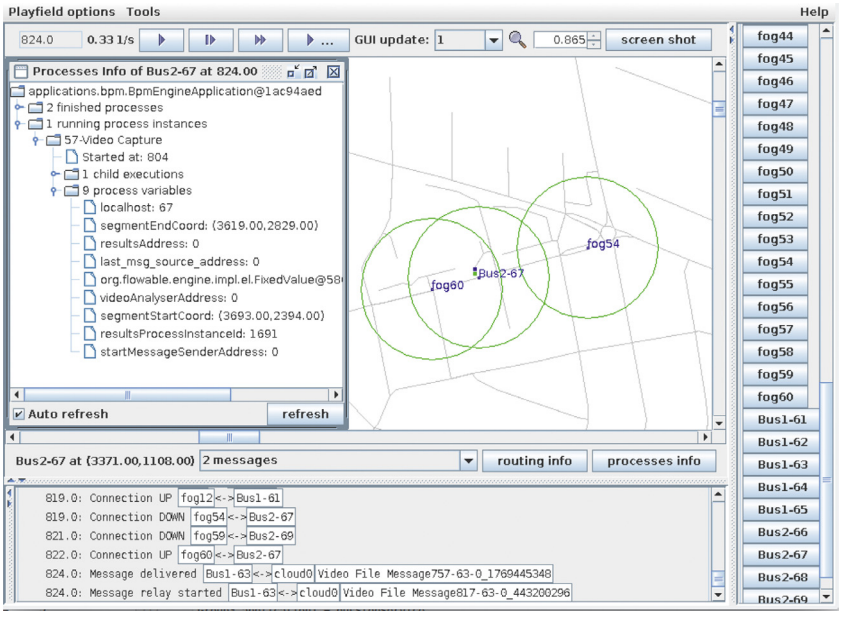

Fig. 4. ONE Simulator with the Process Info Extension of STEP-ONE visible.

```
# Configuring the Process Engine Application
fogProcessEngine.type = bpm.BpmEngineApplication
fogProcessEngine.generateMovementSignals = false
fogProcessEngine.generateConnectionSignals = true

# Processes to run at simulation start:
fogProcessEngine.autoDeployedProcesses = ./bpmn/Road_Analysis_Process.bpmn20.xml
fogProcessEngine.autoStartedProcessKeys=roadanalysisprocessV2
fogProcessEngine.autoStartedProcessVars=segmentStartCoord=(12.00,23.00)&segmentEndCoord
    =(22.00,23.00)

# Network interface
wlanInterface.type = SharedBandwidthInterface
wlanInterface.downloadSpeed = 6750k
wlanInterface.uploadSpeed = 6750k
wlanInterface.transmitRange = 45

# Assign a group of hosts to use the above engine and interface
Group1.application1 = fogProcessEngine
Group1.interface1 = wlanInterface

# Setting compute capacity for the process engine
fogProcessEngine.noOfCpus = 4
fogProcessEngine.cpuSpeed = 31500

# Specify costs for compute and networking
fogProcessEngine.cpuCostGranularity = hour
fogProcessEngine.cpuCostPerTimeUnit = 0.017

fogProcessEngine.networkInterfacesCostsPerMb = 0.015
```

Listing 3. Example excerpt of a STEP-ONE configuration file.

capturing from a vehicle whose route corresponds with the requested segment. Once the segment video is captured on the bus, it is forwarded to an external image processing service. The processing results are recorded in the monitoring systems database and are used to plan timely maintenance, repair, snow clearing etc. operations.

We show how this scenario can be realized as a process-based application in STEP-ONE as: 1) a 2-layer cloud-centric design and 2) a 3-layer fog computing design. We limit the scope of the case study, so that our process begins when a new request for analysing a road segment is received and ends when the video processing results are produced, we replace the dynamic scheduling of road segment analysis requests with some fixed request rates and leave out the management of the maintenance activities that occur based on the results.

The high-level process of a single segment analysis is shown on Fig. 6. It can be seen as a composite application with the following sub-components:

- Choosing a vehicle to perform video capture
- Capturing the video
- Performing video analysis

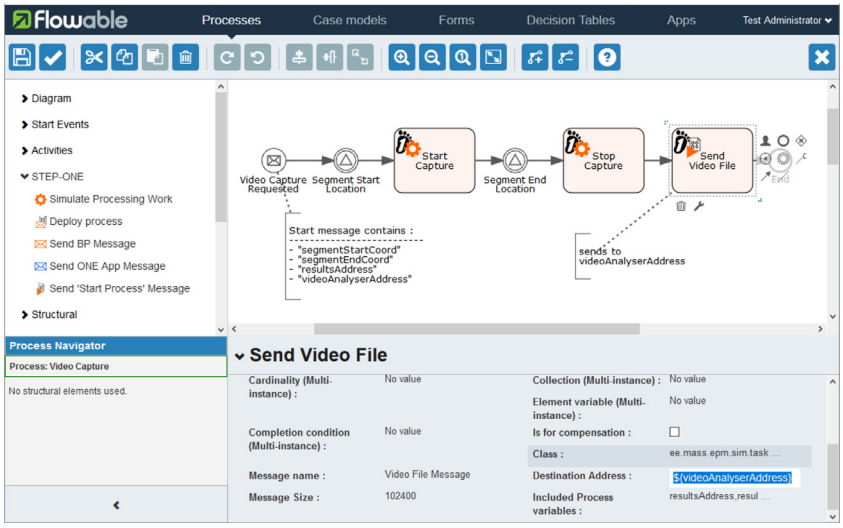

Fig. 5. STEP-ONE version of Flowable Modeller.

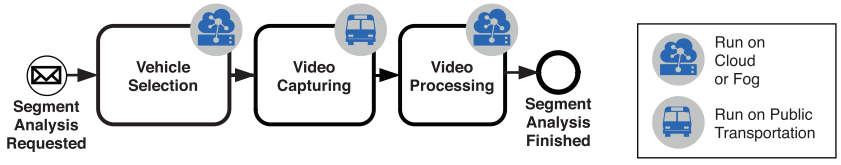

Fig. 6. Single Road Segment Analysis Process Overview.

The subparts may run on different types of devices (bus, fog server, cloud server). In the cloud approach, vehicle selection and video processing are run in the datacenter. However, in the Fog design, bus stops in the city are equipped with Fog servers. Vehicle selection is done by a Fog server located at the beginning of the road segment, while the Fog node at the segment end performs the video processing. In this case, the full raw video file is processed at the Fog, saving the bandwidth of transferring it from the bus to the cloud and also shifting more computational load to the Fog.

5.1. Implementing scenarios in STEP-ONE

The outline of steps needed to establish and simulate such scenarios is the following.

1. Establishing the simulation world map - the necessary.wkt format file can be attained by e.g. downloading a map from OpenStreetMap and converting it to `.wkt`. With software such as OpenJUMP, these files can be edited or created manually.
2. Preparing the process definitions - Using the Flowable modeller extension for STEP-ONE, the BPMN 2.0 files can be visually designed.
   - The process defines the tasks, message exchanges, signal & event listeners.
   - If the process includes custom code, these should be created accordingly in Java.
3. Configuring the ONE simulation
   - Network interfaces & routing algorithms on the nodes
   - Movement models of the nodes
   - Assigning applications for nodes, including `bpm.BpmEngineApplication` for STEP-ONE engine
   - Configuring STEP-ONE engine application
     - Auto-deployed processes for hosts (host groups)
     - Auto-started processes for hosts and their process variables
     - Choosing which reports to generate
4. Running the simulation(s)
5. Interpreting the results
   - From GUI, including STEP-ONE-s process info pane, which shows running and finished processes, their execution states and variables
   - From Generated Reports, including Process Reports

5.2. Establishing the simulation world map

For the Tartu Smart City scenario, we extracted a 50 km^2 area `.osm` file from *OpenStreetMap*, from which we extracted vehicle-navigable streets and converted the result to `.wkt` format, ending in a file with 20,440 points and 3296 features (polylines). Then, we defined two bus routes using OpenJUMP by marking points on the map which correspond to a bus stop, this produced files with 38 and 51 points (stops) respectively. Finally, we also specified locations for fog servers as points, coinciding with the bus stops. The results are shown on Fig. 7, where the bus routes are shown in green and red, bus stops are marked as squares, and Fog servers are marked with blue circles.

5.3. Describing processes definitions

Before we discuss the technical implementation aspects of the processes in this scenario, we first introduce their behaviour - the tasks, messages, signals events and variables used. The visual BPMN process description used in our case study is presented on Fig. 8. It features several BPMN pools, the *Master Process* leads the orchestration and deployment of other processes, such as the ones depicted in the *Vehicle Selection* and *Video Processor* pools.

5.3.1. Master process

The entire analysis is started by sending a Road Segment Analysis Request message, to which the Start Event of the Master Process is subscribed to. The request message includes an

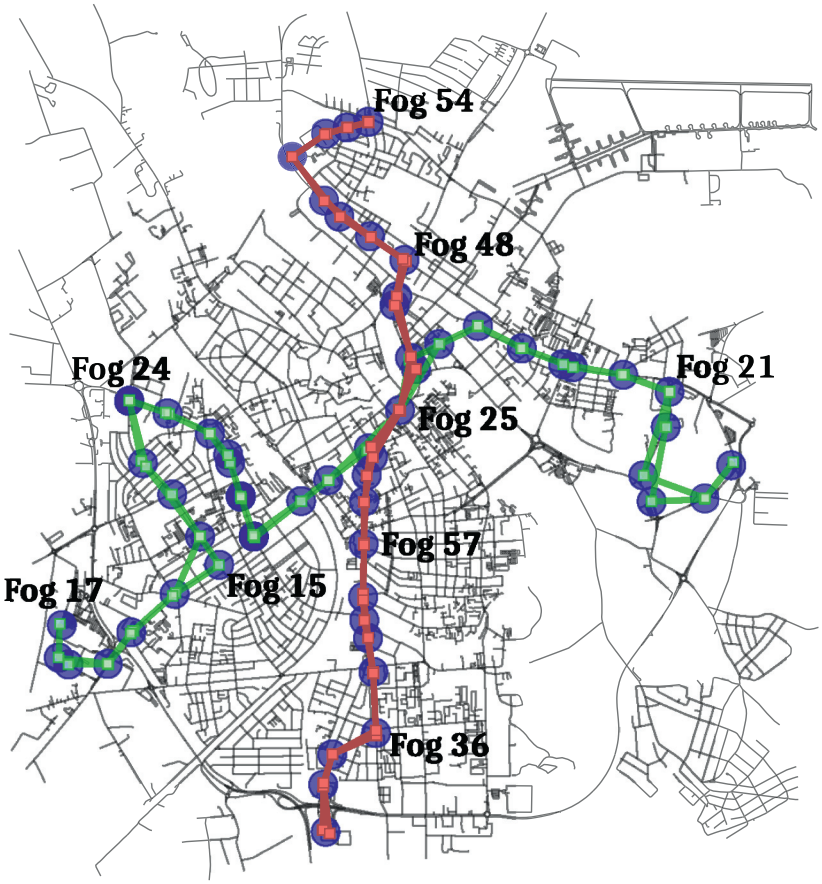

Fig. 7. Simulation World Map Preparation: Tartu City streets, two bus lines (shown in green and red), bus stops shown as squares, fog servers shown as blue circles. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Analysis Instance Descriptor (AID), which is a set of process variables: coordinates of the segment's start and end, the Master Process executor address. Using information from these variables, the first task in the process, *Select Placement Strategy*, determines whether the components of the analysis (Vehicle Selector, Video Procesing) should be placed on one of the fog nodes or be Cloud. As the output of this task, the process variables `Video Processor Address`, `Vehicle Selector Address` and `Placement Strategy` (Cloud or Fog) are added to the AID, these will be used to initiate resources during later stages of the process.

Next, the deployment and initiation of external processes takes place - first, the Video Processor is deployed to the previously assigned address and secondly, if applicable, the Fog version of the Vehicle Selection process is deployed at the road segment start Fog server. The Cloud variant is assumed to be pre-deployed.

After the external processes have been deployed, the Vehicle Selection Process is initiated through a Start Process Message Task. From the Master Process perspective, what follows are simple receive message events (*Recording Started, Recording Finished*), which reflect the progress of the segment analysis. These messages are sent from the external processes which the Master Process has initiated by this stage, described below.

5.3.2. Vehicle selection processes

The *Vehicle Selection Process* has the goal of choosing a bus for deploying the "Video Capture Process" and invoking it. The message starting this process instance includes process variables from the AID.

Cloud variant In the Cloud variant of this process, the bus located closest to the segment start and whose route includes the road segment is selected, based on AID object and queries to the Transport Information System. Then, the process is deployed and started via messages.

Fog variant The Fog variant of the process catches new connection (WiFi) events, and in each case, queries the other device (bus) what its' next movement destination (stop) is. If the response message indicates that the next stop of the bus matches the requested road segment, the video capture process is deployed and started on that device via messages.

Finally, the Video Requester Process sends a notification message to the Master Process about its progress.

5.3.3. Video capturer processes

After receiving a start message with the AID, the capturer process awaits for a coordinate-based signal event (*Segment Start Location Signal*) matching the segment start coordinate. Once the signal is received, the Start Capture tasks initiates video capture on the device. Next, another location-based signal triggers the subsequent "Stop Video Capture" task, after which the recorded file is sent to the Video Processor as indicated by the address from the message which started the process. The Video Processor will be either the Cloud (4G) or the Fog server (WiFi).

5.3.4. Video processor

This process starts when receiving a message including a road segment video file and AID. Before processing, the Master process is notified that the video has been captured. Then, the resource-intensive video processing Simulated Work task is executed, the

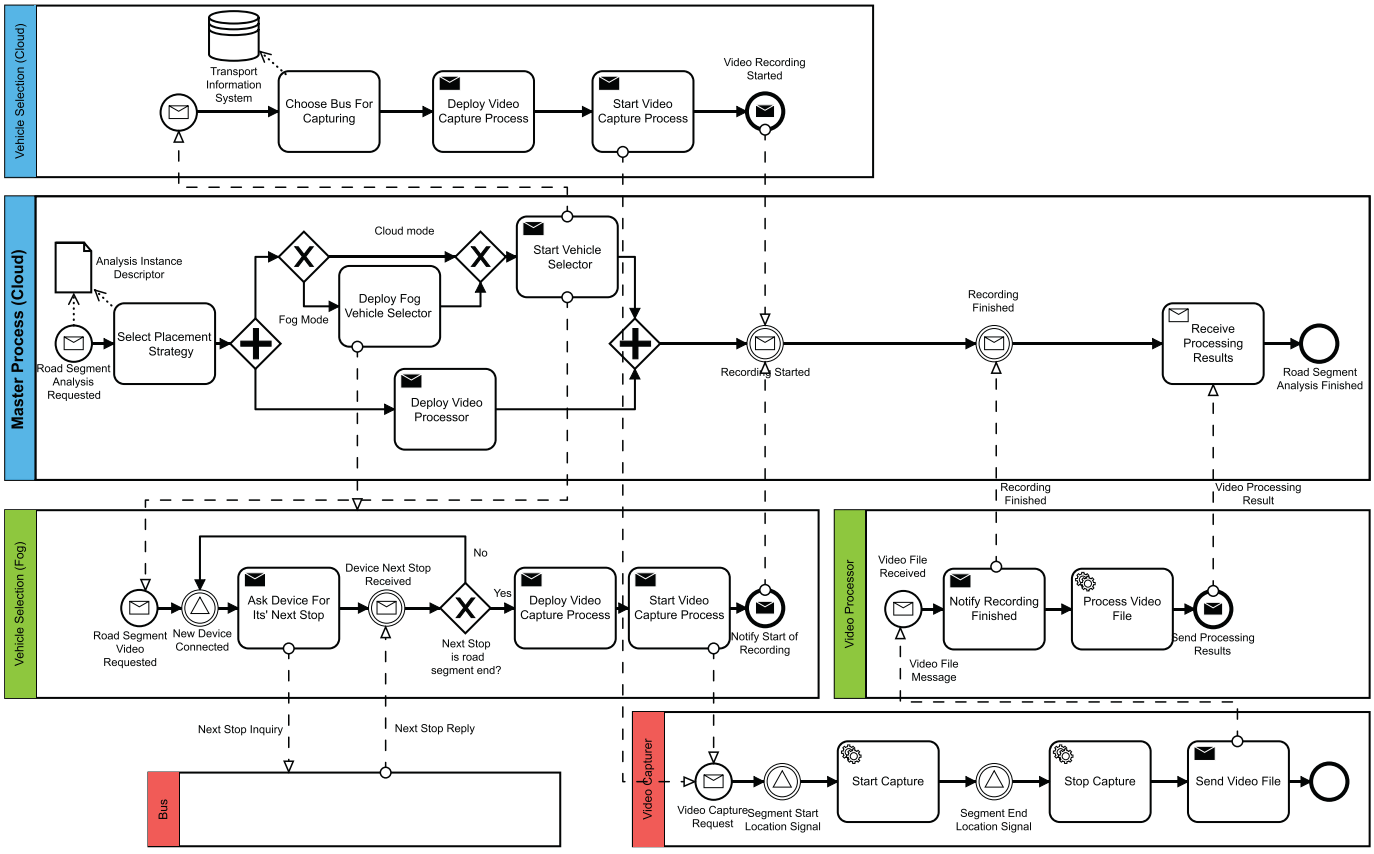

Fig. 8. Smart City Road Monitoring Process Collaboration Diagram.

results of which are sent to the Master process through the end message event.

5.4. Process implementation in STEP-ONE

The Master Process, two variants of Vehicle Selection Processes, Video Capturer and Video Processor workflows were created as separate BPMN 2.0 processes designed with the STEP-ONE Flowable designer.

The Message tasks were implemented as STEP-ONE Process Message tasks, to which the relevant variables of AID were attached. For Deploy message tasks, e.g. "Deploy Video Capture Process", the STEP-ONE Deploy Message task was used, with the attached resource specified by a file path in the XML.

For the Start Capture, Stop Capture, and Process Video File tasks we used the Simulated Work task. The capture-related tasks had a worksize of 500 millions instructions (MI), while the Video Processing Task had a 7000 billion instructions task size.

With the exception of "Video File Message", the message sizes are relatively small, below 1024 kB. In our setup, the Video File Message has been fixed at 100 MB.

For the "New connection" signal catching events in the Fog Video Requester we used the STEP-ONE connection signals, while the Video Capturer Location signals use the Coordinate-based STEP-ONE signals that are based on an custom execution listener *ee.mass.epm.sim.LocationSignalExecutionListener* using the segment end coordinate variable from the AID. Other process parts, such as the gateways and their evaluation expressions or start events were defined following typical process modelling approaches for Flowable.

5.4.1. Custom code for Ttartu scenario

To realize this scenario, two custom ONE applications supplement the above processes. First, the scheduling of Road Segment Analyses is by an application, which maintains the set of all road segments based on the world map, and then periodically sends the Segment Analysis Request messages to the Cloud node, initiating the Master process.

The second custom application is the Bus response application, which responds to "Next Stop Inquiry" messages with a message containing the bus's next stop coordinates.

5.5. Configuring networking and reporting

The configuration of ONE takes places with the `settings.txt` file, STEP-ONE-specific settings are also to be defined there.

For Fog and Cloud nodes, we used the *SharedBandwidthInterface* network interface for 54Mbps wired network links in the simulation. Additionally, Fog and Bus nodes are equipped with a WLAN interface with a max transmit speed of 54Mbps, while Bus and Cloud nodes are interconnected with a interface representing 4G LTE connection with a 12Mbps speed.

We captured the full details of processes using the DetailedBpmReporter, and some messaging data with ONE-s DeliveredMessagesReport.

6. Evaluation

We analysed the behaviour of the presented scenario in varying configurations: how often new analysis requests are scheduled and how many analyses are scheduled in parallel. Further, we investi-

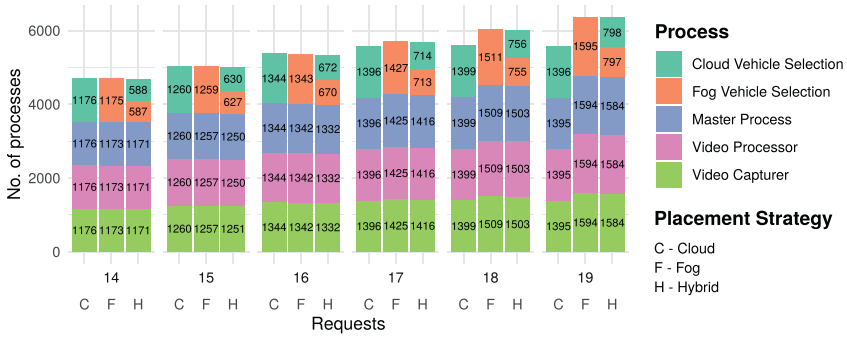

Fig. 9. No. of different Processes run across experiments.

Table 4
Evaluation parameters and their values.

Parameter	Value(s)
$\|S\|$	87
i	400
n	14.19
$ratio_{cloud}$	0.0, 0.5, 1.0
Fog Computation Conf.	CPUs: 4, CPU Speed: 31500 DMIPS
Cloud Computation Conf.	CPUs: 8, CPU Speed: 35500 DMIPS

gated how the processes perform with different process placement strategies: when run on Cloud, Fog, or both interchanged.

6.1. Experiment setup

First, we describe how Road Segment Analysis Requests were scheduled and the parameters which affected the process placement strategy.

6.1.1. Rate of starting processes

A custom ONE application, *RoadmonitoringApp*, maintains the set of all road segments - S, and the set of currently running road segment analyses - $S_{running} \subseteq S$.

At an interval of every i seconds, it randomly chooses a set of new analysis requests to schedule $S_{new} \subseteq S \setminus S_{running}$, $|S_{new}| \leq n$, where n is the no. of requests to start each interval.

For each $s \in S_{new}$, a Master Process instance in the cloud is started, and the set of running analyses is updated, $S_{running} := S_{running} \cup S_{new}$

When a segment request process finishes, it is removed from $S_{running}$.

6.1.2. Placement strategy - fog or cloud

The output of the "Select Placement Strategy" task is a preconfigured as a simulation parameter: $ratio_{cloud}$. The ratio determines what fraction of processes use cloud in the placement. For instance, if the value is 0.0, all processes use Fog placement strategy, if it is 0.5, then every other process uses fog placement.

This acts as a placeholder for a dynamic resource scheduling algorithm, that could be easily implemented in STEP-ONE. For this paper, such an implementation is out of scope, instead, we wish to show how the testbed supports evaluating such algorithms through its general features and refer the reader to our previous work where placement and scheduling decisions were made using STEP-ONE (Mass et al., 2019).

In our evaluation, we considered the values shown in Table 4 for $|S|$, i, n, $ratio_{cloud}$, resulting in a total of 15 simulations. In each simulation the no. of hosts is 71: 1 cloud, 60 fog servers, and 10 buses.

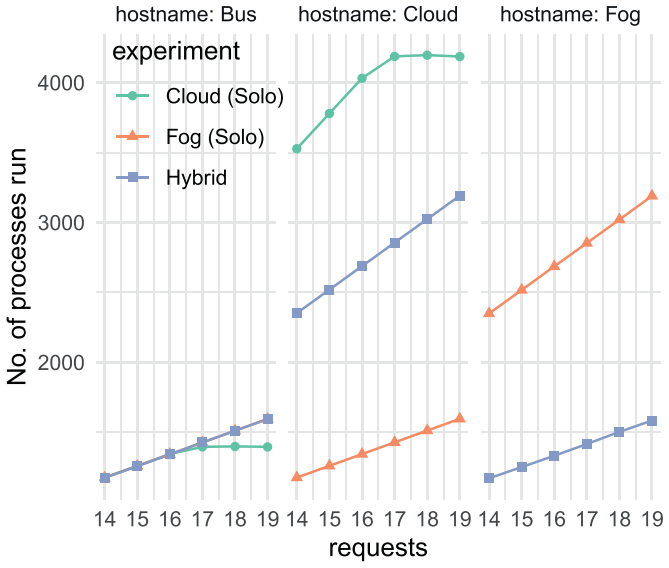

Fig. 10. Processes run in different layers.

6.2. Case study

Fig. 9 shows a breakdown of how many processes were run by type, across all hosts. As can be expected based on the process design from Fig. 8, for each Road Analysis Master Process, there is 1 Video Processor, 1 Video Capturer and 1 of either Cloud- or Fog Vehicle Selection process - depending on the process placement decision. For instance, scheduling 15 requests at 400 second intervals started a total of 1250 road analysis processes.

At up to 16 parallel requests, all process placement strategies run the same no. of processes (with a small amount of jitter). At above 16 requests, the no. of Solo Cloud processes becomes smaller than the other cases and peaks at near 1400. We discuss the reason for this below in Section 6.4

6.3. Process allocation

Using the Fog placement strategy reduces the load for Cloud in terms of total number of processes run, as visible in Fig. 10, which shows for each host type the number of processes executed in each experiment. The effect is significant in the "Solo Fog" case, where the load for cloud is reduced by 2/3 compared to Solo Cloud. The hybrid approach does reduce load for cloud, but by 1/3.

Looking at the distribution of processes among fog hosts, Fig. 11, we can note that fog server 25 is handling more processes than others. Server 25 is actually located at the crossing of the

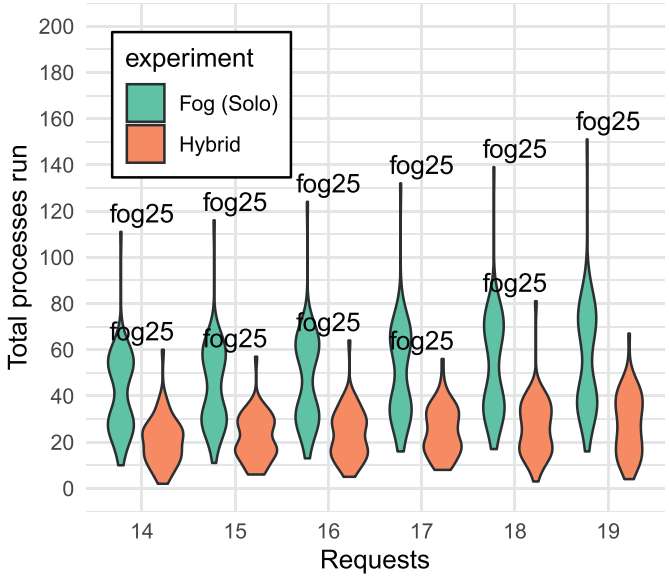

Fig. 11. Process distribution among fog hosts.

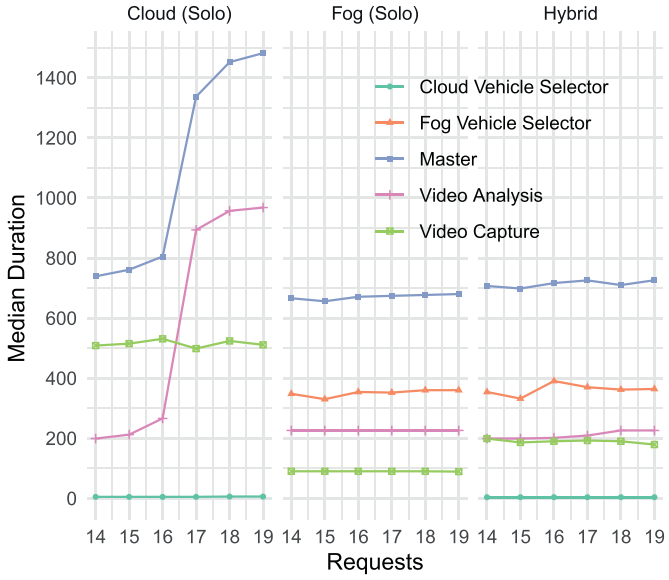

Fig. 12. Median Durations of Different Processes.

two bus-lines so higher utilization of this server is expected (recall Fig. 7).

6.4. Process performance - Duration

The issues with the cloud stem from the Video Analysis process. At lower loads, Video Analysis in cloud takes 200s to complete, in all placement configurations. However, for Cloud Solo, already from 17 parallel requests at 400s interval, the cloud cannot complete Video Analysis processes in time before new requests are scheduled, hugely increasing the median duration of the Video Analysis process and by extension the Road Analysis Master Process as well (see Fig. 12).

Fig. 12 also shows the different behaviours of the cloud and fog placement strategies can be seen as well - in the fog case, the Vehicle Selector takes more time to delegate the capturing to a bus compared to the clouds Vehicle Selector. On the other hand, since the bus is already at the segment start once it starts the capture

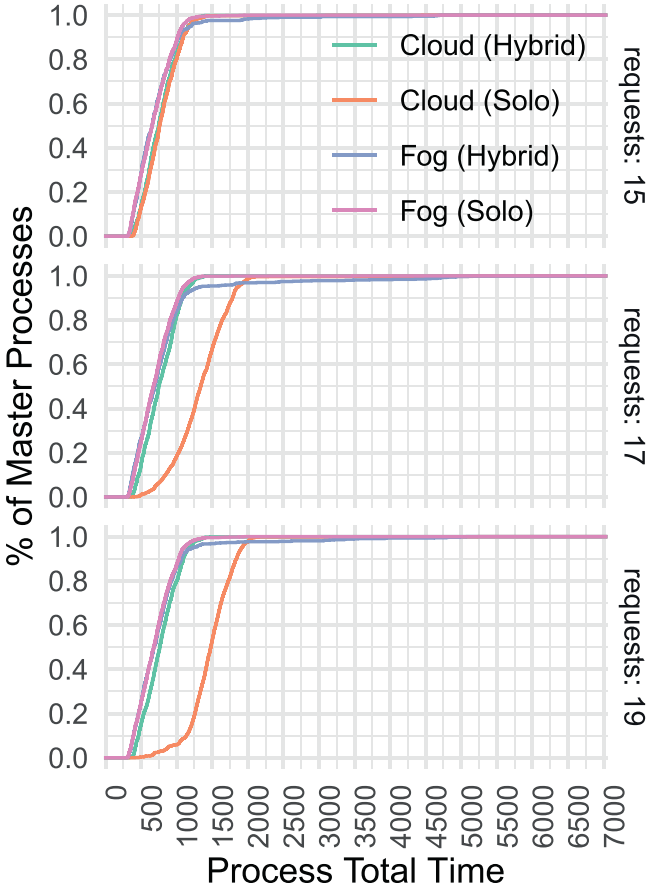

Fig. 13. ECDFs of Master Road Analysis Process.

process, the Video Capture process duration is lower for Fog, since in the Cloud case, the Capturer process includes the time spent travelling to the road segment.

When comparing the Solo Fog and Solo Cloud approaches, the Road Analysis Process finishes faster in the fog case, which in our simulation is caused by the network interface differences Cloud and Fog.

Fig. 13 shows the Empirical Cumulative Distribution Functions (ECDF) of Master Process durations in cases were Cloud was not lagging due to a high request rate. Here, the Hybrid case has been further separated depending on whether the analysis was placed in Fog or Cloud.

We see that all processes take more than 200s to complete, and that in both Hybrid or Solo Fog configurations at 15 requests, 90% of processes run on Fog finish under 1000s, while for cloud, this is around 80%. This suggests that the bus-selection algorithm has room for improvement - the slower behaviour of Cloud is explained by the Cloud choosing a bus which is not optimal. At request rates higher than 15, the slow-down of cloud is again visible.

6.4.1. Bandwidth allocation

The impact of moving some of the communication towards the edge networks can be seen from Fig. 14, which shows the message count on the y-axis, and the bandwidth per message type with text. While the total no. of messages handled by the cloud does not reduce drastically when considering the hybrid or solo fog case, the impactful difference comes from the Video File Messages, which make almost the entirety of the bandwidth consumption. The sum of bandwidth taken by Video File messages is 117600MB, and we can see how either half or the entirety of that is shifted to the Fog nodes depending on the process placement. Shifting large

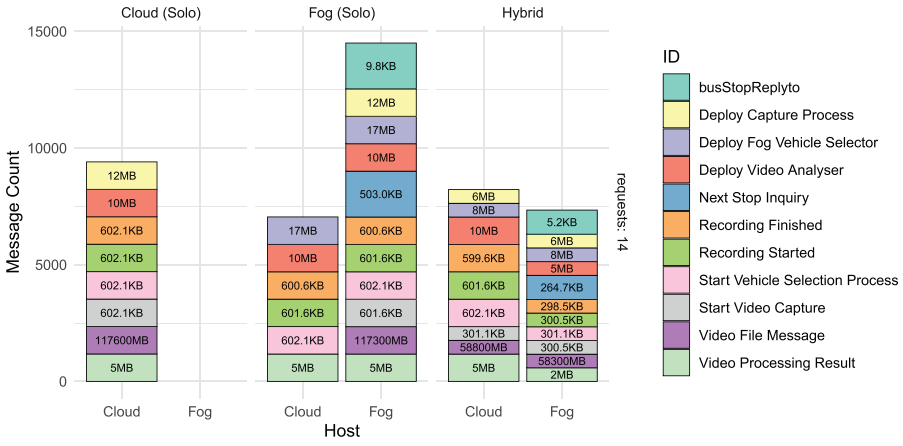

Fig. 14. No. of messages and bandwidth transferred by message type.

file transfers to the local networks has an impact on the overall process speed as well, since we consider Fog servers to have speedier network interfaces than the 4G LTE interface which connects the buses with the cloud.

6.5. Discussion, future research directions

The case study confirmed how both the processing load and bandwidth can be shifted from the cloud to fog. In this setting, we saw shortcomings in scalability of a cloud-based solution - notably that at a 400s interval,the Cloud alone cannot support more than 15 parallel requests. Meanwhile, Fog-oriented placement could support 19 parallel requests without slow-downs.

While in these results Cloud showed worse scalability, it's important to note that in our simulation, the network interface of the cloud was slower than the fogs and that the processing capability of Cloud, while higher than the Fog-s, was still in the same order of magnitude.

The results gave insight into the (over)-utilization of nodes (such as Fog server 25 or the Cloud node), which is useful for planning the geographical placement and hardware needs of nodes.

With these experiments, we wished to show that STEP-ONE can be used for in-depth study on the effect of load-balancing, scheduling etc algorithms within realistic scenarios. The algorithms could be implemented as process sub-tasks. While Fog yielded promising results, an aspect which we did not consider in our study are the capital- and operational expenses (CAPEX, OPEX) of deploying Fog infrastructure. With some additions, other researchers could explore these directions with STEP-ONE as well, e.g. by using the *CostHelper* described in Section 4.3.1.

Another research theme are scenarios where data transfer is subject to temporal quotas - e.g. the cloud or individual fog nodes have a bandwidth budget per time unit, which can be used in process-level decision-making. This would require creating an application or service for ONE nodes that can report their data usage during runtime, then this data can be forwarded to the processes as variables and be used in the decision gateways, deployment and service tasks.

Thus, STEP-ONE can be used to model various vehicular applications (Mendiboure et al., 2019). A direction to analyse, for example, is the effect of other edge nodes such as pedestrians and cars, on the discussed scenario. On one hand, these nodes may also be using Fog services or invoking processes simultaneously with the road analysis processes, effecting the compute load of the fog servers or bandwidth usage of the networks. On the other hand, even if the other nodes aren't directly invoking functions of the fog resources, the radio interference of their presence may still effect the performance.

7. Conclusion

In this paper, we presented a set of tools to define and study applications and algorithms using BPMN 2.0 processes for scenarios where mobility and connectivity issues play a key role. The core of the proposed simulated testbed is a Flowable process engine embedded into the ONE simulator that has been interfaced with the simulators modules. Further, STEP-ONE provides a visual modeller for designing executable processes, reporting tools to capture process metrics and new network interfaces and computing model to ONE, which are useful for modelling edge and fog computing situations.

We showcased the capabilities of STEP-ONE through a smart-city case study, where processes were delegated from a central cloud node to static fog nodes and moving vehicle nodes, while involving movement, connectivity, messaging and processing features of STEP-ONE. Through this case study, we showed how STEP-ONE can be used to compare different placement strategies of processes and assess scalability of multi-layer fog and cloud systems.

As part of future work, modelling of networking and computing for cloud-layer nodes can be improved, by introducing virtualization and cluster-based concepts.

STEP-ONE currently mainly focuses on catching events from the simulation and mapping them to the process execution. To complement messages, more tasks which influence the simulation world could be added, such as a "Move to" task, which would allow to direct host movement from processes and would be interesting to model self-driving cars, drones, robots. Another possible extension is the feature of migrating running process instances between hosts.

Declaration of Competing Interest

There are no potential conflicts of interest.

Acknowledgment

The work is supported by the Estonian Centre of Excellence in IT (EXCITE) and by the Archimedes Foundation Smart specialisation scholarship, both funded by the European Regional Development Fund. We would also like to thank anonymous reviewers and Helen Karatza (Area Editor) of Journal of Systems and Software for their constructive suggestions and guidance on improving the content and quality of this paper.

References

Adhikari, M., Mukherjee, M., Srirama, S.N., 2019. DPTO: A Deadline and Priority-aware Task Offloading in Fog Computing Framework Leveraging Multi-level Feedback Queueing. IEEE Internet of Things Journal doi:10.1109/JIOT.2019.2946426. 1–1

Andročec, D., Novak, M., Oreški, D., 2018. Using semantic web for internet of things interoperability: a systematic review. Int. J. Semant. Web. Inf. Syst. 14 (4), 147–171. doi:10.4018/IJSWIS.2018100108.

Behrisch, M., Bieker, L., Erdmann, J., Krajzewicz, D., 2011. SUMO – simulation of Urban MObility: an overview. In: Aida Omerovic, S..U.o. O., Diglio A. Simoni, R.I.-R.T.P., Georgiy Bobashev, R.I.-R.T.P. (Eds.), Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation. ThinkMind, Barcelona.

Bonomi, F., Milito, R., Zhu, J., Addepalli, S., 2012. Fog computing and its role in the internet of things. In: Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing. ACM, Helsinki, Finland, pp. 13–16. doi:10.1145/2342509.2342513.

Brouns, N., Tata, S., Ludwig, H., Asensio, E.S., Grefen, P., 2018. Modeling IoT-aware business processes. IBM Res. Rep. 42.

Caro, G. A. D.,. Analysis of simulation environments for mobile ad hoc networks, 282003.

Chang, C., Srirama, S., Mass, J., 2015. A middleware for discovering proximity-based service-oriented industrial internet of things. In: Chou, W., Maglio, P.P., Paik, I. (Eds.), Proc. - IEEE Int. Conf. Serv. Comput., SCC. Institute of Electrical and Electronics Engineers Inc., pp. 130–137. doi:10.1109/SCC.2015.27.

Chang, C., Srirama, S.N., Buyya, R., 2016. Mobile cloud business process management system for the internet of things: a survey. ACM Comput. Surv. 49 (4), 70:1–70:42. doi:10.1145/3012000.

Cheng, Y., Zhao, S., Cheng, B., Chen, J., 2018. A service-based fog execution environment for the IoT-Aware business process applications. In: 2018 IEEE International Conference on Web Services (ICWS), pp. 323–326. doi:10.1109/ICWS.2018.00052.

Dar, K., Taherkordi, A., Baraki, H., Eliassen, F., Geihs, K., 2015. A resource oriented integration architecture for the internet of things: a business process perspective. Pervasive Mob. Comput. 20, 145–159. doi:10.1016/j.pmcj.2014.11.005.

Dar, K., Taherkordi, A., Rouvoy, R., Eliassen, F., 2011. Adaptable service composition for very-large-scale internet of things systems. In: Proceedings of the 8th Middleware Doctoral Symposium. ACM, Lisbon, Portugal, pp. 2:1–2:6. doi:10.1145/2093190.2093192.

Dede, J., Förster, A., Hernández-Orallo, E., Herrera-Tapia, J., Kuladinithi, K., Kuppusamy, V., Manzoni, P., bin Muslim, A., Udugama, A., Vatandas, Z., 2018. Simulating opportunistic networks: survey and future directions. IEEE Commun. Surv. Tutor. 20 (2), 1547–1573. doi:10.1109/COMST.2017.2782182.

Domingos, D., Martins, F., 2017. Using BPMN to model internet of things behavior within business process. IJISPM - Int. J. Inf. Syst. Project Manag. (3) 39–51. doi:10.12821/ijispm050403.

Du, J., Zhao, L., Feng, J., Chu, X., 2018. Computation offloading and resource allocation in mixed fog/cloud computing systems with min-max fairness guarantee. IEEE Trans. Commun. 66 (4), 1594–1608. doi:10.1109/TCOMM.2017.2787700.

Dumas, M., La Rosa, M., Mendling, J., Reijers, H., 2018. Fundamentals of Business Process Management: Second Edition. 10.1007/978-3-662-56509-4

Flowable,. The flowable engine as a serverless function. https://blog.flowable.org/2019/01/29/flowable-engine-as-a-serverless-function/2009.

Giang, N.K., Lea, R., Blackstock, M., Leung, V.C.M., 2018. Fog at the edge: experiences building an edge computing platform. In: 2018 IEEE International Conference on Edge Computing (EDGE), pp. 9–16. doi:10.1109/EDGE.2018.00009.

Grefen, P., Ludwig, H., Tata, S., Dijkman, R., Baracaldo, N., Wilbik, A., D'Hondt, T., 2018. Complex collaborative physical process management: a position on the trinity of BPM, IoT and DA. In: Camarinha-Matos, L.M., Afsarmanesh, H., Rezgui, Y. (Eds.), Collaborative Networks of Cognitive Systems, 534. Springer International Publishing, Cham, pp. 244–253. doi:10.1007/978-3-319-99127-6_21.

Gupta, H., Dastjerdi, A. V., Ghosh, S. K., Buyya, R., 2016. iFogSim: A Toolkit for Modeling and Simulation of Resource Management Techniques in Internet of Things, Edge and Fog Computing Environments. arXiv:1606.02007 [cs].

Hackmann, G., Haitjema, M., Gill, C., Roman, G.-C., 2006. Sliver: a BPEL workflow process execution engine for mobile devices. In: Dan, A., Lamersdorf, W. (Eds.), Service-Oriented Computing – ICSOC 2006. Springer Berlin Heidelberg, pp. 503–508.

Janiesch, C., Koschmider, A., Mecella, M., Weber, B., Burattin, A., Di Ciccio, C., Gal, A., Kannengiesser, U., Mannhardt, F., Mendling, J., Oberweis, A., Reichert, M., Rinderle-Ma, S., Song, W., Su, J., Torres, V., Weidlich, M., Weske, M., Zhang, L., 2017. The Internet-of-Things Meets Business Process Management: Mutual Benefits and Challenges, 2017. arXiv:1709.03628 [cs].

Kalbag, A., Silverman, G., 2014. Enriching business processes through internet of everything. Cisco IT Article.

Keränen, A., Ott, J., Kärkkäinen, T., 2009. The ONE simulator for DTN protocol evaluation. ICST doi:10.4108/ICST.SIMUTOOLS2009.5674.

Lee, J., Ko, J., Choi, Y.-J., 2017. Dhrystone million instructions per second–based task offloading from smartwatch to smartphone. International Journal of Distributed Sensor Networks 13 (11). doi:10.1177/1550147717740073. 1550147717740073

Mahmud, R., Srirama, S.N., Ramamohanarao, K., Buyya, R., 2019. Quality of experience (QoE)-aware placement of applications in fog computing environments. J. Parallel Distrib. Comput. 132, 190–203. doi:10.1016/j.jpdc.2018.03.004.

Mahmud, R., Srirama, S.N., Ramamohanarao, K., Buyya, R., 2020. Profit-aware application placement for integrated fog–cloud computing environments. J. Parallel. Distrib. Comput. 135, 177–190. doi:10.1016/j.jpdc.2019.10.001.

Marrella, A., Mecella, M., Sardina, S., 2016. Intelligent process adaptation in the SmartPM system. ACM Trans. Intell. Syst. Technol. 8 (2), 25:1–25:43. doi:10.1145/2948071.

Marrella, A., Mecella, M., Sardiña, S., 2018. Supporting adaptiveness of cyber-physical processes through action-based formalisms. AI Commun. 31 (1), 47–74. doi:10.3233/AIC-170748.

Martins, F., Domingos, D., 2017. Modelling IoT behaviour within BPMN business processes. Procedia Comput. Sci. 121, 1014–1022.

Mass, J., Chang, C., Srirama, S.N., 2016. WiseWare: A Device-to-Device-Based Business Process Management System for Industrial Internet of Things. In: 2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), pp. 269–275. 10.1109/iThings-GreenCom-CPSCom-SmartData.2016.69

Mass, J., Chang, C., Srirama, S.N., 2019. Edge process management: a case study on adaptive task scheduling in mobile IoT. Internet Things 6, 100051. doi:10.1016/j.iot.2019.100051.

Mendiboure, L., Chalouf, M.-A., Krief, F., 2019. Edge computing based applications in vehicular environments: comparative study and main issues. J. Comput. Sci. Technol. 34 (4), 869–886. doi:10.1007/s11390-019-1947-3.

Meyer, S., Ruppen, A., Hilty, L., 2015. The Things of the Internet of Things in BPMN. In: Persson, A., Stirna, J. (Eds.), Advanced Information Systems Engineering Workshops, 215. Springer International Publishing, Cham, pp. 285–297. doi:10.1007/978-3-319-19243-7_27.

Mukherjee, M., Shu, L., Wang, D., 2018. Survey of fog computing: fundamental, network applications, and research challenges. IEEE Commun. Surv. Tutor. 20 (3), 1826–1857. doi:10.1109/COMST.2018.2814571.

Noura, M., Atiquzzaman, M., Gaedke, M., 2018. Interoperability in internet of things: taxonomies and open challenges. Mob. Netw. Appl. doi:10.1007/s11036-018-1089-9.

O.M. Group, 2011. Business process model and notation (BPMN): version 2.0. Object Manag..

Peng, T., Armellin, G., Betti, D., Chiasera, A., Toai, T.J., Ronchetti, M., 2013. MDO: framework for Context-Aware Process Mobility in Building-Maintenance Domain. In: 2013 17th European Conference on Software Maintenance and Reengineering, pp. 449–452. doi:10.1109/CSMR.2013.69.

Pryss, R., Mundbrod, N., Langer, D., Reichert, M., 2015. Supporting medical ward rounds through mobile task and process management. Inf. Syst. e-Bus. Manag. 13 (1), 107–146. doi:10.1007/s10257-014-0244-5.

Rodrigues, T.G., Suto, K., Nishiyama, H., Kato, N., 2017. Hybrid method for minimizing service delay in edge cloud computing through VM migration and transmission power control. IEEE Trans. Comput. 66 (5), 810–819. doi:10.1109/TC.2016.2620469.

Schobel, J., Pryss, R., Schickler, M., Reichert, M., 2016. A lightweight process engine for enabling advanced mobile applications. In: Debruyne, C., Panetto, H., Meersman, R., Dillon, T., Kühn, e., O'Sullivan, D., Ardagna, C.A. (Eds.), On the Move to Meaningful Internet Systems: OTM 2016 Conferences. Springer International Publishing, pp. 552–569.

Seiger, R., Herrmann, S., Aßmann, U., 2017. Self-healing for distributed workflows in the internet of things. In: 2017 IEEE International Conference on Software Architecture Workshops (ICSAW), pp. 72–79. doi:10.1109/ICSAW.2017.36.

Sen, R., Roman, G.-C., Gill, C., 2008. CiAN: a workflow engine for MANETs. In: Lea, D., Zavattaro, G. (Eds.), Coordination Models and Languages. Springer Berlin Heidelberg, pp. 280–295.

Sonmez, C., Ozgovde, A., Ersoy, C., 2017. EdgeCloudSim: an environment for performance evaluation of edge computing systems. In: 2017 Second International Conference on Fog and Mobile Edge Computing (FMEC), pp. 39–44. doi:10.1109/FMEC.2017.7946405.

Spiess, P., Karnouskos, S., Guinard, D., Savio, D., Baecker, O., Moreira Sá de Souza, L., Trifa, V., 2009. SOA-based integration of the internet of things in enterprise services. In: Proceedings of the IEEE 7th International Conference on Web Services (ICWS). IEEE Computer Society Press, Los Angeles, US, pp. 968–975.

Tranquillini, S., Daniel, F., Kucherbaev, P., Casati, F., 2015. Modeling, enacting, and integrating custom crowdsourcing processes. ACM Trans. Web 9 (2), 7:1–7:43. doi:10.1145/2746353.

Tüysüz, G., Avenoglu, B., Eren, P.E., 2013. A workflow-based mobile guidance framework for managing personal activities. In: 2013 Seventh International Conference on Next Generation Mobile Apps, Services and Technologies, pp. 13–18. doi:10.1109/NGMAST.2013.12.

Varga, A., 2010. OMNeT++. In: Wehrle, K., Güneş, M., Gross, J. (Eds.), Modeling and Tools for Network Simulation. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 35–59. doi:10.1007/978-3-642-12331-3_3.

Yang, Y., Wang, K., Zhang, G., Chen, X., Luo, X., Zhou, M.-T., 2018. MEETS: maximal energy efficient task scheduling in homogeneous fog networks. IEEE Internet Things J. 5 (5), 4076–4087. doi:10.1109/JIOT.2018.2846644.

Zakeri, H., Nejad, F.M., Fahimifar, A., 2017. Image based techniques for crack detection, classification and quantification in asphalt pavement: a review. Arch. Comput. Methods Eng. 24 (4), 935–977. doi:10.1007/s11831-016-9194-z.

Zaplata, S., Lamersdorf, W., 2010. Towards mobile process as a service. In: Proceedings of the 2010 ACM Symposium on Applied Computing. ACM, pp. 372–379.

Zhang, T., Zhao, S., Cheng, B., Farina, M., Huang, J., Chen, J., Ren, B., Hou, S., 2018. Lightweight SOA-based multi-engine architecture for workflow systems in mobile ad hoc networks. IEEE Access 6, 14212–14222. doi:10.1109/ACCESS.2018.2815617.

Jakob Mass is a PhD student at the Mobile & Cloud Lab at the Institute of Computer Science, University of Tartu, Estonia. He received his MSc in software engineering from University of Tartu, Estonia. His research interests include Mobile Information Systems, IoT-aware Business Process Management Systems and Service-oriented Computing.

Satish Narayana Srirama is a Research Professor and the head of the Mobile & Cloud Lab at the Institute of Computer Science, University of Tartu, Estonia and a Visiting Professor at University of Hyderabad, India. He received his Ph.D. in computer science from RWTH Aachen University, Germany in 2008. His research focuses on cloud computing, mobile web services, mobile cloud, Internet of Things, fog computing, migrating scientific computing and enterprise applications to the cloud and large scale data analytics on the cloud. He is an IEEE senior member, was an Associate Editor of IEEE Transactions in Cloud Computing, is an Editor of Wiley Software: Practice and Experience, a 50 year old Journal, and a program committee member of several international conferences and workshops. Dr. Srirama has co-authored over 140 refereed scientific publications in several inter-national conferences and journals. For further information of Prof. Srirama, please visit: http://kodu.ut.ee/~srirama/.

Chii Chang obtained Doctor of Philosophy (PhD) (2014), Master of Information Technology (MSIT) (2008) and Bachelor of Computing (BCom) (2006) from Faculty of Information Technology, Monash University, Australia. Dr. Chang's research involves Fog Computing, Edge Computing, Internet of Things (IoT), IoT-aware Business Process Management System, Services Computing and Mobile Cloud Computing.