IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received January 7, 2018, accepted March 10, 2018, date of publication March 13, 2018, date of current version March 28, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2815617

Lightweight SOA-Based Multi-Engine Architecture for Workflow Systems in Mobile Ad Hoc Networks

TONGGUANG ZHANG[1], SHUAI ZHAO[1], BO CHENG[1], (Member, IEEE), MAURIZIO FARINA[2], JIWEI HUANG[1], (Member, IEEE), JUNLIANG CHEN[1], BINGFEI REN[1], AND SHOULU HOU[1]

[1]State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China
[2]Ericsson IT Solutions & Services SpA, 80143 Napoli, Italy

Corresponding author: Tongguang Zhang (jsjoscpu@163.com)

This work was supported in part by the National Natural Science Foundation of China under Grant 61501048, in part by the Beijing Natural Science Foundation under Grant 4182042, in part by the Fundamental Research Funds for the Central Universities under Grant 2017RC12, in part by the National Natural Science Foundation of China under Grant U1536111, Grant 61372115, Grant U1536112, and Grant 61502043, and in part by the Beijing Natural Science Foundation under Grant 4162042.

ABSTRACT At present, most mainstream workflow systems adopt client/server architecture where the workflow system server (WSS) runs on a server in a fixed network or cloud and the workflow system client (WSC) runs on a PC or a mobile smart device (MSD), such as a smartphone. However, in special circumstances (e.g., battlefields, earthquakes, tsunamis, and floods) communication infrastructure can be damaged or it does not exist; consequently, traditional workflow systems cannot meet the need. MSDs are now more powerful than ever and can be used to construct mobile ad hoc networks (MANETs) in special circumstances. To provide communication using workflow technology in MANETs, we present a lightweight service-oriented architecture-based multi-engine architecture for workflow systems in MANETs. One characteristic of the architecture is that certain MSDs play dual roles, functioning as both a WSS and a WSC. We provide the architecture design details and implement the workflow engine on Linux/Android platforms. Because the multiple workflow engines must be able to cooperate closely, we present a multi-engine trigger mechanism. The test results validate the effectiveness and availability of the workflow engine and verify the feasibility of the algorithm for the multi-engine trigger mechanism.

INDEX TERMS MANET, multi-engine, SOA, trigger mechanism, workflow system.

I. INTRODUCTION

In recent years, mobile smart devices (MSDs), such as tablets and smartphones, have gradually become more powerful and popular computing platforms, and MSDs are predominantly used for web services. As a result, MSDs are increasingly used as work equipment in many industries and government organizations worldwide. Assisted by specifically developed mobile applications, employees in many industries have improved their productivity by participating in the business process regardless of the location and time. However, in certain situations, MSDs do not work, and workflow systems are unavailable because of a lack of necessary infrastructure (such as in underground areas, sparsely populated areas, etc.) or infrastructure failure. Most mainstream workflow systems currently use client/server architecture in which the workflow system server (WSS) application runs on a server within a fixed network or cloud, while the workflow system client (WSC) runs on a PC or MSD. However, in special circumstances (e.g., battlefields, earthquakes, tsunamis, and floods) communication infrastructure can be damaged or it does not exist; consequently, traditional workflow systems cannot perform their roles or functions. Now, however, MSDs can be used to construct mobile ad hoc networks (MANETs) in these special circumstances.

MANETs are both self-forming and self-healing and can enable peer-level communications between MSDs without relying on centralized resources or fixed infrastructure. Several routing algorithms are available and include OLSR (optimized link state routing protocol), OSPF (open shortest path first), B.A.T.M.A.N. (better approach to mobile ad hoc networking), and AODV (ad hoc on-demand distance vector) among many others.

Currently, a workflow engine can be executed on an MSD. Some studies have focused on single-engine (mobile process engine, MPE) in mobile networks. Schobel *et al.* [1] presented a lightweight mobile process engine for executing data collection instruments on MSDs. The mobile process engine allowes for an offline execution of deployed process models as well as for the storage of the collected data on the MSD. Wipp [2] proposed Workflows on Android (WOtAN), a modular and flexible framework for business process management running on Android MSDs, and showed an application scenario where WOtAN was used to properly support a mobile data collection application. Castelán *et al.* [3] described a Software Reference Architecture for WfLMS (Workflow Learning Management Systems) with Mobile, Cloud and Collaborative functionalities in order to develop a WFLMS as a native application for the iOS platform. The other studies have focused on the cooperation among multiple engines. Thai *et al.* [4] deployed a number of engines in the cloud to orchestrate the workflow, the engines support decentralisation by allowing intermediate data to be transferred between one another. Bi *et al.* [5] proposed process models fragmentation approaches based multiple execution engines. IBM InfoSphere Information Server [6] is a data integration platform, multiple workflow engine processes can be run across the cluster where the InfoSphere Master Data Management Collaboration Server is running. More than one workflow engine processes run on different servers and share the load of items that are moving through workflows. However, although the work mentioned above studied or implemented the cooperation among multiple engines, due to the constrained resources and the unstable network connectivity of MANETs, the cooperation approaches are not quite suitable to be used in MANETs. In this paper, we study the cooperation among multiple engines in MANETs and face two challenges: (1) the design and implementation of a lightweight workflow engine that can be executed efficiently on MSDs; and (2) combining multiple workflow engines to work as single workflow engine such that users perceive only a single workflow engine.

For communications among people using workflow technology in MANETs, we present a lightweight service-oriented architecture (SOA)-based multi-engine architecture for workflow systems in MANETs. One characteristic of the architecture is that certain MSDs play dual roles, functioning both as a WSS and as a WSC. In this study, we design a scenario in which the communication infrastructure is absent as shown in Fig. 1-A. Then, MSDs are used to construct a MANET as shown in Fig. 1-B, where if *MSD-A* is elected as the server, other MSDs can visit the WSS in this server. Assuming that *MSD-A* eventually becomes disabled, as shown in Fig. 1-C, the MANET must be reconstructed as shown in Fig. 1-D. After *MSD-B* has been elected as the server, other MSDs can visit the WSS in this server. The MANET construction/reconstruction processes are transparent to mobile subscribers. To achieve this goal, we present a lightweight multi-engine architecture based on SOA [7], [8]

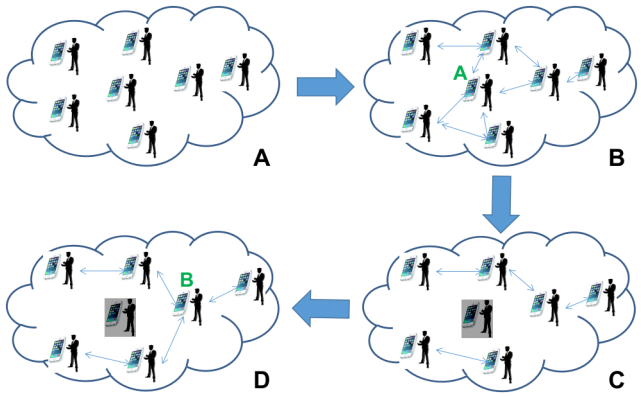


FIGURE 1. Application of the multi-engine architecture in MANET.

to construct the workflow systems in MANETs. Special scenarios [9] occur in which the multi-engine architecture would become beneficial, including but not limited to battlefields, post-disaster relief management, and collaborative working sites under field conditions.

In this paper, the main contributions of the current work are fourfold. First, we provide a lightweight SOA-based multi-engine architecture for workflow systems in MANETs. Second, we design and implement a lightweight workflow engine on the Linux/Android platform in C/C++. Third, because multiple workflow engines must work as single workflow engine, we present a multi-engine trigger mechanism through which they can cooperate closely such that users perceive only a single workflow engine. Finally, we design and implement a *Disaster Response System* to evaluate the effectiveness and availability of the workflow engine.

The remainder of the paper is organized as follows. Section II reviews related work. Section III introduces the design and implementation of the workflow engine to resolve the first challenge. Section IV presents the trigger mechanism to resolve the second challenge, which activates a workflow engine based on the connection states of mobile nodes in MANETs. Section V describes two experiments performed to validate the effectiveness and availability of the workflow engine and verify the feasibility of the multi-engine synergistic algorithm. Section VI concludes the paper and briefly provides directions for future work.

II. RELATED WORK

At present, a process engine can be enabled to run on MSDs. Schobel *et al.* [1] conducted studies on a lightweight mobile process engine for executing data collection instruments on MSDs. Wipp [2] presented Workflows on Android (WOtAN), a modular and flexible framework for business process management running on Android MSDs. Xu *et al.* [10] presented a mobile workflow support system based on mobile stream data management system for Android devices and applied vehicle data services and information sharing in collaborative works. In mobile environments, the mobility of portable devices and the unstable connectivity of mobile networks

can influence web service selection. To address this problem, Deng *et al*. [11] proposed a novel offloading system to allow for the robust selection of mobile services. This approach considered the dependent relationships between component services and aimed to optimize the execution time and energy consumption of the running mobile services. Park and Nam [12] proposed a framework that supports ubiquitous access to medical systems using MSDs and integrated medical systems. Tao *et al*. [13] introduced a mobile workflow management system and proposed the active service for a mobile workflow. Reference [14] propose a web service recommendation approach based on collaborative filtering and make QoS prediction based on user mobility. This approach considers user mobility and data volatility to adapt to mobile edge computing environments. However, due to resources are often constrained, the proposed approaches mentioned above are not suitable for MANETs.

Some studies have been performed in hosting web services in MSDs. The potential for mobile web services was first discussed by Berger *et al*. [15]. Mohamed and Wijesekera [16] conducted studies on hosting web services in MSDs based on both SOAP and REST [17], [18]. Wagh and Thool [19] proposed Android-based framework for hosting mobile services using RESTful web services [20], [21] to enable Web service provisioning. Verma and Srivastava *et al*. [9] proposed an approach to manage web service directories hosted on MSDs that would enable MSDs to manage service registries without assistance, which can drastically reduce the cost of and dependency on infrastructure. In addition, this approach could facilitate the provision of services in dynamic networks, such as vehicular networks or MANETs. However, most of the work performed by a mobile web service uses a standard directory system with UDDI for web service discovery, which requires high computational costs. In addition, centralized management could cause a single point of failure when discovering web services in MANETs. Therefore, in our proposed multi-engine architecture, we use a WSLT (Web Service LisT) to manage web services in resource-constrained MSDs in MANETs.

The literatures [22]–[24] on web service execution have proposed theories and tools to model the context or environment, and studies on running processes in MSDs have led to more flexible process deployment in MSDs. Even when the MSDs are disconnected from the central process engine, they can still conduct the assigned activities. However, the designed or generated processes target fixed devices and thus fail to adapt to the changing environment of MSDs. In our work, we adopt a multi-engine trigger mechanism to overcome this shortcoming.

III. SOA-BASED MULTI-ENGINE ARCHITECTURE

In this section, we detail the design and implementation of the multi-engine architecture for workflow systems in MANETs. Fig. 2 shows the three-layered model of the architecture, in which MWS represents a mobile web service, WA represents wrapped APP, M-MPE represents master MPE, S-MPE represents slave MPE, M-MSD represents master MSD, and S-MSD represents slave MSD.

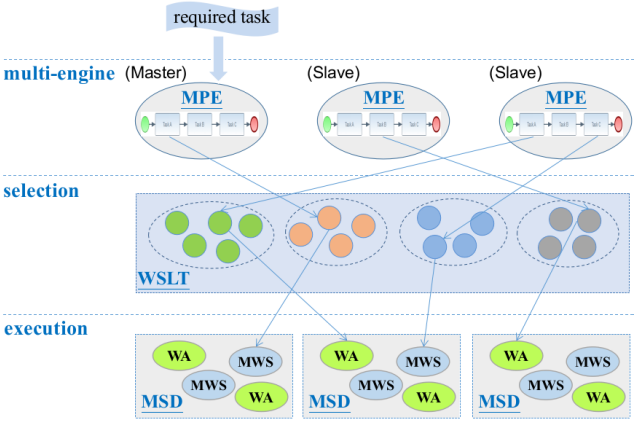


FIGURE 2. Three-layered model of the multi-engine architecture.

The first layer of the architecture focuses on multiple workflow engines. We discuss the design and implementation of the workflow engine in detail. Because MPEs must be able to cooperate, we present a trigger mechanism to activate the workflow engine based on the connection states of mobile nodes. The layer also focuses on process execution. The second layer aims to solve the problem of web service selection in MANETs. The third layer focuses on the execution of web services in mobile environments. Because of space constraints, we mainly focus on the first layer and detail the design and implementation of the lightweight SOA-based multi-engine architecture in MANETs as well as the multi-engine trigger mechanism.

A. DESIGN CONCEPTS

The design concepts of the multi-engine architecture address six key points: lightweight BPM (business process management), a master/slave engine, a multi-engine trigger mechanism, an SOA-based strategy, a web server hosted on an MSD, and a RESTful web service.

- ***Lightweight BPM:*** BPM takes a holistic approach to managing all resources involved in a process, including people, information, content, and applications. In an integrated BPM suite, a technology platform can be developed that supports an entire BPM lifecycle with the ability to design, model, execute, and monitor business processes. A BPM application is similar to an evolved workflow system. Because resources are constrained in MANETs, we propose the lightweight BPM. The analysis, design and modeling of business processes are performed using a PC. The business processes are executed and monitored in MPEs in MANETs.
- ***Master/slave engine:*** In MANETs, each MSD may run either a master engine or a slave engine. However, only one MSD runs the master engine; the other MSDs concurrently run the slave engines. We detail which MSD should be elected to run the master engine in Section IV.

- ***Multi-engine trigger mechanism:*** To avoid a single-point fault, we provide a multi-engine trigger mechanism. If the MSD running the master engine becomes invalid, another MSD is elected to run the master engine as soon as possible. The trigger mechanism is presented in Section IV.
- ***SOA-based strategy:*** Alignment with an SOA strategy is prevalent in current workflow systems. In SOA, components are decomposed into web services, making them reusable over multiple platforms. Together with the SOA, the BPM enables the decoupling and composition of complex business logic.
- ***Web server hosted on an MSD:*** In MANETs, each MSD can act as a service provider and a service consumer. Therefore, a web server must be hosted on an MSD.
- ***RESTful web service:*** Enciso-Quispe *[*18], AlShahwan and Moessner [25], and Srirama *et al.* [26] showed that RESTful web services are relatively more suitable for mobile environments, which is consistent with our proposal to offer a directory service (i.e., WSLT) on an MSD.

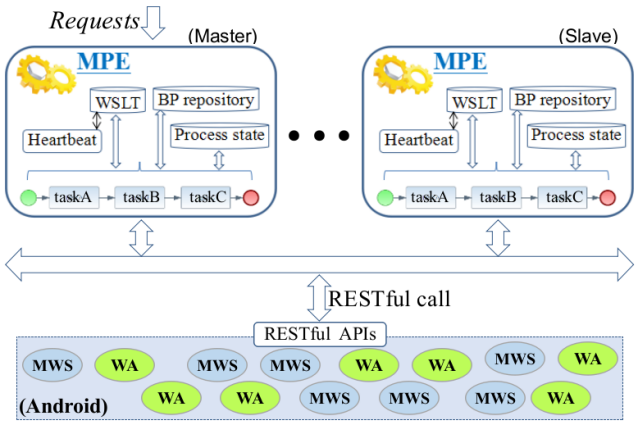


FIGURE 3. Lightweight SOA-based multi-engine architecture.

B. LIGHTWEIGHT SOA-BASED MULTI-ENGINE ARCHITECTURE

As shown in Fig. 3, the lightweight SOA-based multi-engine architecture includes both an M-MPE and S-MPEs. In the SOA, each application and its functions are modeled and published as web services and are reusable across multiple platforms. Each web service has an interface with a public description. The architecture supports the asynchronous invocation of web services that have disadvantages when used in mobile applications. For example, these web services may use the more verbose XML and SOAP protocols. Several performance problems of web services in MSDs are discussed in [27]–[29]. SOAP is verbose and carries high performance costs [30]. Therefore, SOAP is difficult to use in resource-constrained situations. In these cases, we adopt a RESTful API and JSON to solve the problem. The client sends a request (JSON message) to the server, and the server sends a response (JSON message) back to the client.

The characteristics of the architecture are as follows. First, technical personnel can make full use of various technologies, such as C/C++, PHP, JavaScript, Java, python, Node.js, Websocket, Nginx, lighttpd and Android. Second, the invocation interface of web services adopts the RESTful style [31]. Third, the workflow engine is lightweight and can be efficiently run in an MSD.

The M-MPE executes the business process and invokes the tasks belonging to the process. A business process is composed of a group of tasks, resources and a logical relationship, and these processes are described using XPDL/BPMN2. With XPDL/BPMN2, we can describe the services that are used and specify their order and the data flow occurring in the web services.

Single points of failure may occur because of server shutdowns or a removal from the MANET. Therefore, we present a multi-engine trigger mechanism between the M-MPE and S-MPEs to solve these issues.

C. MAIN COMPONENTS OF THE WORKFLOW ENGINE

The main components of the workflow engine and their relationships are described in Fig. 4. The M-MPE manages the selection, invocation, execution order, and faults of web services. The S-MPEs synchronize the state/execution data of the processes with the M-MPE. In this section, we discuss the design and implementation of these components. The MPE consists of six main functional modules: Process Execution, WSLT, Get Web Service URI, Heartbeat, Communication Protocol and Trigger.

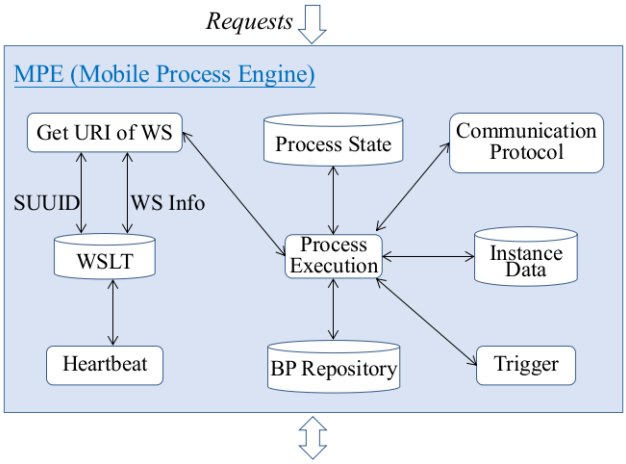


FIGURE 4. Components of the workflow engine.

1) PROCESS EXECUTION

This component is the heart of the workflow engine, and it is responsible for parsing and executing the process description. In this module, the process business logic is described using XPDL, and all XPDL activities are implemented. Each implementation can change the process state. All the messages use the JSON data-interchange format. When a JSON message arrives, the module traverses the process description document and takes appropriate actions. Web services are identified by the RESTful API, and they communicate by

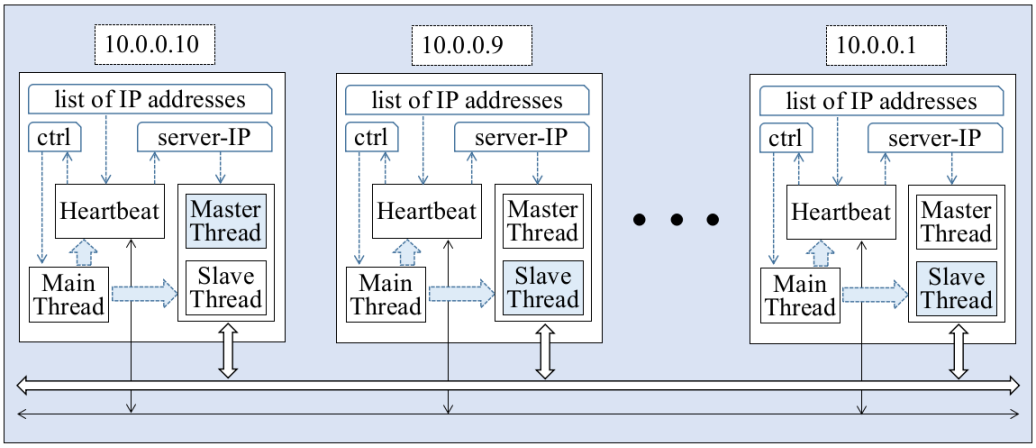


FIGURE 5. Components of the multi-engine trigger algorithm.

interchanging JSON messages. Therefore, the MPE parses and executes business processes and then remotely invokes web services running on other servers via the RESTful API. The module includes three subsidiary modules: (1) *BP (Business Process) Repository*, which supports process management and enables persistent storage for process descriptions; (2) *Instance Data*, which stores the values of currently running process instances; and (3) *Process State*, which stores the process state data. The MPE monitors the state of each request it receives. The state consists of the discovery state and execution state of each service in the request.

2) WSLT

The WSLT (Web Service LisT) manages the status information of web services. After a web service is developed in a PC, it is uploaded to the MSD. Meanwhile, the information (SUUID, URI, service function description, etc.) of the web service is inserted into the WSLT. When designing a business process, a suitable web service must be identified for each activity. Thus, a suitable web service in the WSLT must be identified based on the service request; the SUUID of the matched web service must be returned, and the activity node in the business process must be associated with the SUUID. When executing a business process, the *Process Execution* module finds the appropriate URI in the WSLT based on the SUUID of an activity and then invokes the web service via the URI.

3) GET WEB SERVICE URI

This module obtains the corresponding URI of a web service from the WSLT by its SUUID and then calls the web service via the URI. The web service execution results are returned to the invoker in a certain data-interchange format (such as JSON).

4) HEARTBEAT

This module is used to check the validity of a web service running in a server, and the results are used to update the activity state of the web service in the WSLT.

5) COMMUNICATION PROTOCOL

To support external communication protocols, a *Communication Protocol* module is provided. This module must support both the basic internet protocols (e.g., HTTP) and mobile protocols (e.g., Wi-Fi). Nginx is used as the web server in the MSD. We provide an *Nginx* module that connects the standard web server with the MPE. Thus, an incoming HTTP request can be forwarded to the MPE and the reply message can be routed back to the right application.

6) TRIGGER

This module is the concrete implementation of the multi-engine trigger mechanism detailed in the next section.

IV. MULTI-ENGINE TRIGGER MECHANISM

Certain single points of failure may occur in MANETs, such as when the *master MSD* is shut down or removed from the MANET. In these situations, *another MSD* must be elected as the M-MPE, and multiple MPEs must be able to cooperate to send and receive messages in synchronous or asynchronous modes. Therefore, we study and present a multi-engine trigger mechanism to support the process execution in MANETs, design the algorithm for the trigger mechanism, and then write it in C. The source code is available in [32]. The simulation experiment is presented in Section V.B. The multi-engine trigger mechanism is shown in Algorithm 1.

The main functional components of the algorithm and their relationships are described in Fig. 5. There are three variables: *ctrl*, *server_IP* and *IP_list*. The *IP_list* variable includes the IP addresses of the nodes which may be M-MPE or S-MPEs, in addition, the *IP_list* variable is mainly used by the *Heartbeat Thread* to find which node is the M-MPE. The values of all the *IP_list* variables are the same. The value of the *server_IP* variable is the IP address of the M-MPE. The value of the *ctrl* variable is very important, $ctrl = -1$ denotes that the algorithm is in the initialization stage, $ctrl = 0$ denotes that the node is running as the S-MPE, $ctrl = 1$ denotes that the node is running as the

Algorithm 1 For the Multi-Engine Trigger Mechanism

```
VAR: ctrl = −1
VAR: server_IP
VAR: IP_list //the list of IP addresses
BEGIN:
 1. procedure: MAIN
 2.  create_thread(Heartbeat);
 3.  while ctrl == −1 //Initialization stage
 4.   sleep 1 s;
 5.  while true
 6.   if ctrl == 0: create_thread(Slave);
 7.   if ctrl == 1: create_thread(Master);
 8.  end while
 9. end procedure MAIN
10. procedure: Heartbeat
11.  for ip in IP_list
12.   if ctrl == −1 //Initialization stage
13.    while true
14.     who_is_Master(ip); //later joins in MANETS
15.     if ip is Master
16.      ctrl == 0; server_IP == ip;
17.     if no ip is Master: the max_ip is elected as Master
18.      if self_ip != max_ip
19.       ctrl == 0; server_IP == max_ip;
20.      if self_ip == max_ip: ctrl == 1;
21.     end for
22.    end while
23.   if ctrl == 0 //Slave
24.    while true
25.     sleep 1 s; request_to_Master(server_IP);
26.     if no response from Master
27.      for ip2 in IP_list > self_ip //descending order
28.       if ping_ok(ip2): server_IP == ip2;
                                        //switch Master
29.       if cannot ping all ip2: ctrl == 1; //become Master
30.      end for
31.    end while
32.   if ctrl == 1 //Master
33.    while true: response_to_Slave();
34.  end for
35. end procedure Heartbeat
36. procedure: Master
37.  while true //communication between Master & Slave
38.   if database updating
39.    send synchronous data to Slave;
40. end procedure Master
41. procedure: Slave
42.  while true
43.   if receive synchronous data from Master
44.    update database;
45. end procedure Slave
END
```

M-MPE. There are four functional components: *Heartbeat Thread*, *MAIN Thread*, *Master Thread* and *Slave Thread*. When the algorithm (the *MAIN Thread*) is starting up, the initial value of the *ctrl* variable is −1 and the *Heartbeat Thread* is created firstly. After the algorithm is initialized, the value of the *ctrl* variable is 0 or 1, if *ctrl* = *0*, the *Slave Thread* is created, if *ctrl* = *1*, the *Master Thread* is created. Because the value of the *ctrl* variable is tuned by the *Heartbeat Thread*, the *Heartbeat Thread* plays an important role. The *Heartbeat Thread* is detailed in the algorithm.

V. TESTING

The test environment is an IBM Server with a 32-core 2.0 GHz Intel Xeon CPU, 64 GB of memory, and a 64-bit Fedora 26 operating system. We design and implement the workflow engine (cBPM4Linux), process designer, and multi-engine synergistic algorithm. cBPM4Linux is short for C/C++ Business Process Management for Linux. The core source code of cBPM can be obtained from [33]. We conduct two experiments: one to test the effectiveness and availability of cBPM4Linux and the other to validate the feasibility of the multi-engine synergistic algorithm in a MANET.

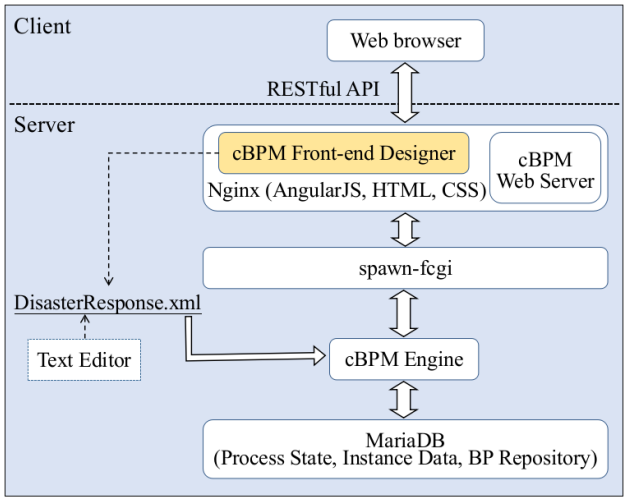


FIGURE 6. Functional components of the running cBPM4Linux System.

A. DESIGNING AND TESTING THE WHOLE CBPM4LINUX SYSTEM

As shown in Fig. 6, the whole cBPM4Linux System consists of two parts: a Web Server and a Web Client. Users can design and execute a process through a Web Client that accesses the Web Server. The *cBPM Front-end Designer* module provides process design interface for users. The *cBPM Web Server* module provides business process execution interface for users. *spawn-fcgi* is used to spawn a *cBPM Engine* which is a FastCGI application written in C/C++. MariaDB is used to store the process state, process instance data, and so on.

To test the effectiveness and availability of the entire cBPM4Linux System, we design and implement a *Disaster Response System* (DRS) as shown in Fig. 7. A DRS provides

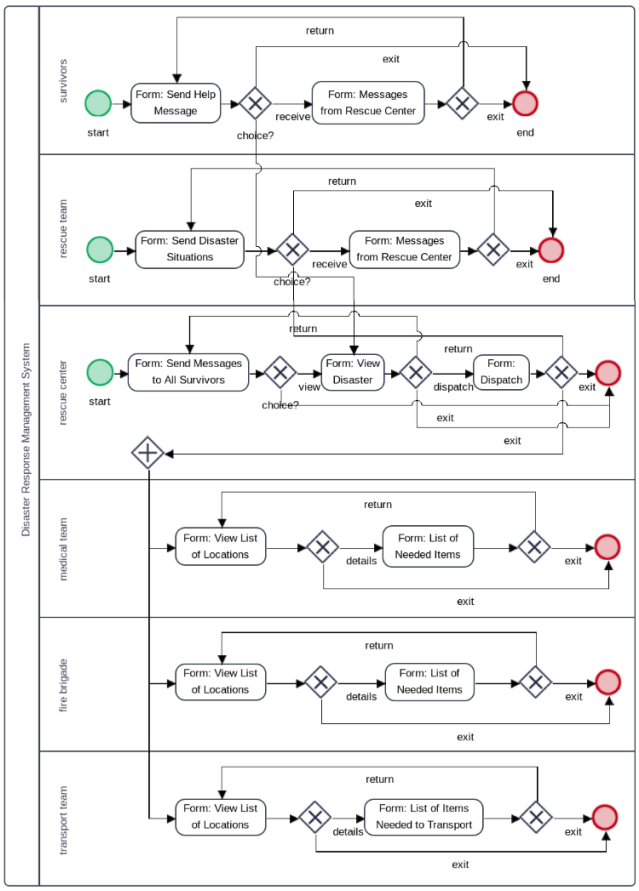


FIGURE 7. Process of a disaster response system.

survivors and *emergency personnel* with the information to locate and assist each other in a disaster area. The DRS allows *survivors* or *rescue team* to submit help requests to the *rescue center*. The *rescue center* optimizes the available resources to serve every incoming request, generates an action plan for the rescue mission, and then sends *medical team*, *fire brigade* or *transport team* to perform the rescue mission.

The processes of the DRS are described by the XML file *DisasterResponse.xml* which can be generated by using the *cBPM Front-end Designer* or written manually using a text editor. Because the *cBPM Front-end Designer* is in the process of development, we designed the process description file for this study using a text editor.

As shown in Fig. 8, the commands in lines 1 and 2 are used to start the services nginx and mariadb. The command in line 3 is used to navigate to the folder where the *cBPM Engine* (executer) and the code of the *cBPM Web Server* are stored. The command in line 4 is used to set up the environment variables used by the *cBPM Engine*. The command in line 5 is used to spawn the *cBPM Engine*. Next, we introduce how users access the DRS supported by the cBPM4Linux System.

Create DRS Process: First, the *rescue center* creates the DRS process (the XML file *DisasterResponse.xml*) as shown in Fig. 9 (A). Fig. 9 (B) shows the process activities.

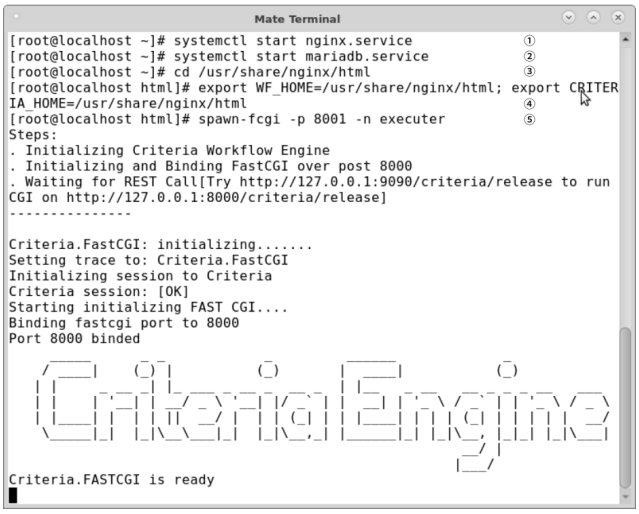


FIGURE 8. Start up the whole cBPM4Linux System; the key is to run the cBPM Engine (Criteria Workflow Engine).

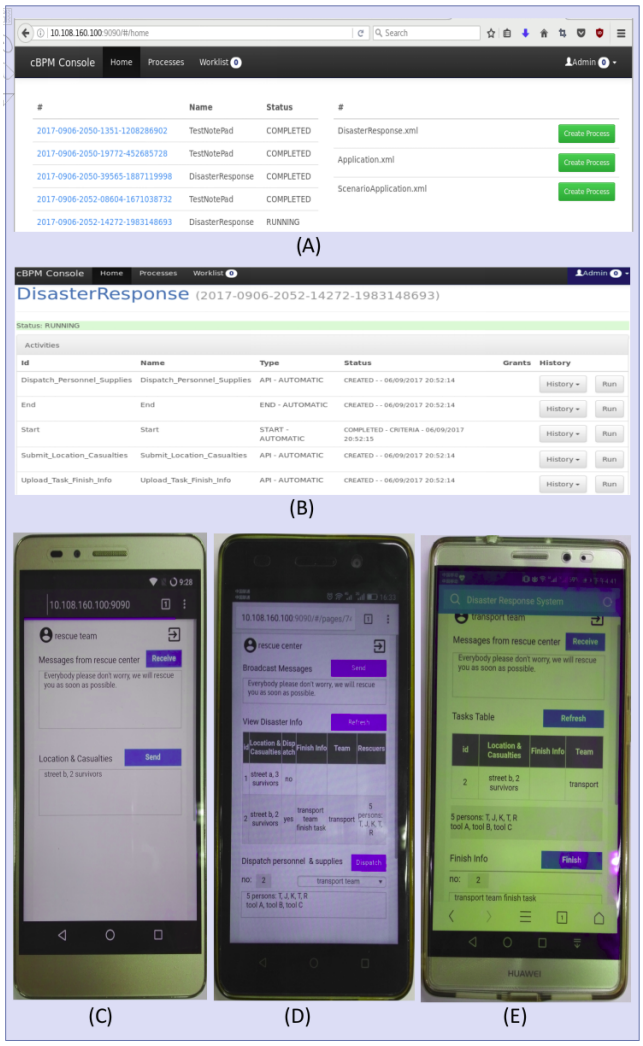


FIGURE 9. Test of the disaster response system.

Submit Help Requests: The *survivors* or *rescue team* complete a help request form and then submit the form as shown in Fig. 9 (C).

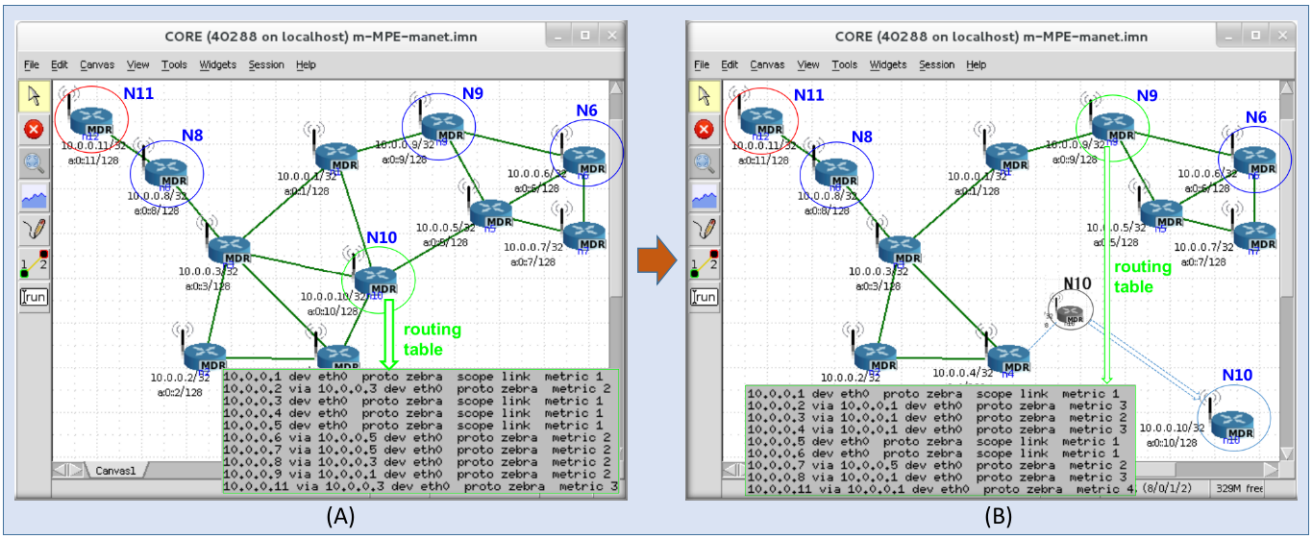


FIGURE 10. Node N10 is moving to a new location in the virtual MANET.

Accept and Dispatch: Based on information from the *survivors* or *rescue team*, the *rescue center* dispatches *medical team*, *fire brigade* or *transport team* to perform the rescue mission as shown in Fig. 9 (D).

Upload Rescue Results: After completing the rescue missions, the *medical team*, *fire brigade* or *transport team* upload the rescue results to the *rescue center* as shown in Fig. 9 (E).

The test result shows that the whole cBPM4Linux System is effective and available.

B. TEST FOR MULTI-ENGINE SYNERGISTIC ALGORITHM

To validate the feasibility of the multi-engine synergistic algorithm, we use the Common Open Research Emulator (CORE) [34] to build the virtual MANET. Then, we verify its feasibility via CORE.

As shown in Fig. 10 (A), the virtual MANET is composed of 11 mobile nodes. The nodes labeled N1–N9 are S-MSDs. Node N10 is elected as the M-MSD. Each node acts as a miniature Linux machine with OSPFv3MDR installed. OSPFv3MDR is an extension of the popular OSPF routing protocol, and it is used to construct MANETs. In addition, we run *server-manet*, which is the program implementation of the multi-engine synergistic algorithm, on nodes N1–N10 and run *client-manet* on the node N11. The program *client-manet* is used to simulate a *workflow system client* that visits the *workflow system server* by sending data to the *server-manet* running in node N10. Meanwhile, node N10 sends the synchronous state/execution data of the process to nodes N1–N9. By double clicking on nodes N6 and N8–N11, we open terminal windows as shown in Figs. 8–11. The test steps are as follows.

1. Run *server-manet* on node N10 as shown in Fig. 11.

2. Run *server-manet* on nodes N6 and N8. The outputs are shown in lines 1 and 7 in Fig. 12 and in lines 2 and 3 in Fig. 11.

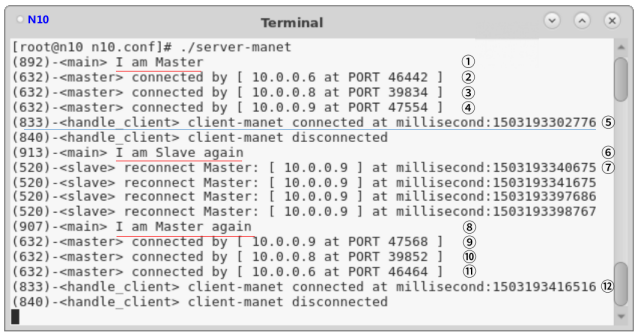


FIGURE 11. Node N10 acts as the M-MSD/S-MSD.

3. Run *server-manet* on node N9. The outputs are line 1 in Fig. 13 and line 4 in Fig. 11.

4. Run *client-manet* on node N11. The outputs are shown in lines 1 and 2 in Fig. 14. Node N11 sends data to node N10, which then sends data to nodes N1–N9. The outputs are shown in lines 2 and 8 in Fig. 12 and line 2 in Fig. 13.

5. Now, node N10 moves to a new location as shown in Fig. 10 (B), which changes the network topology. The *server-manet* program is executed and elects node N9 as the new M-MSD, as shown in line 3 in Fig. 13. Node N10 changes to act as an S-MSD, as shown in line 6 in Fig. 11; however, it cannot visit M-MSD (N9) because of its disconnection from other nodes as shown in line 7 in Fig. 11. The *server-manet* program is executed on nodes N6 and N8 in Fig. 12. Then, nodes N6 and N8 are elected as S-MSDs again (lines 3 and 9 in Fig. 12) and connected to the M-MSD (lines 5 and 6 in Fig. 13).

6. The *client-manet* is executed again on node N11 (line 3 in Fig. 14). The outputs are shown in lines 4 and 5 in Fig. 14. Node N11 sends data to node N9, which then sends data to nodes N1–N8. The outputs are shown in lines 4 and 10 in Fig. 12.

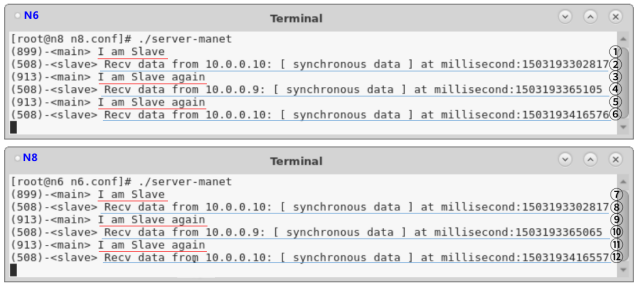


FIGURE 12. Nodes N6 and N8 act as S-MSDs.

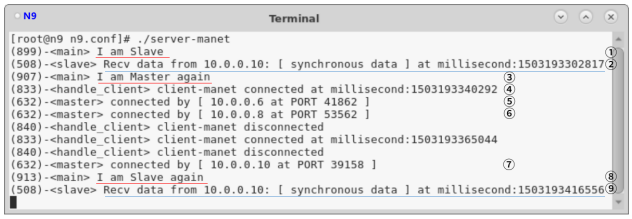


FIGURE 13. Node N9 acts as the S-MSD/M-MSD.

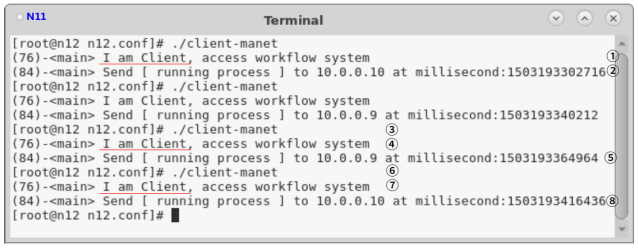


FIGURE 14. Node N11 acts as a client and visits the M-MSD.

7. Now, node N10 moves to its original location, which again changes the network topology, and node N10 is re-elected as the M-MSD (line 8 in Fig. 11).

8. The *client-manet* is executed again on node N11 (line 6 in Fig. 14). The outputs are shown in lines 7 and 8 in Fig. 14. Then, node N11 sends data to node N10, which, in turn, sends data to nodes N1–N9. The outputs are shown in lines 6 and 12 in Fig. 12 and in line 9 in Fig. 13.

We examine the packet loss, network latency and jitter in the virtual MANET (shown in Fig. 10) by performing ping tests from node N11 to nodes N1–N10. After many tests, we find that the round-trip time (RTT) between any two adjacent nodes is approximately a constant value of 40Âś0.1 ms. The results of the ping test vary depending on the quality of the connection in real networks. However, stable connections occur within the virtual MANET. The packet loss is 0.0%, and the delay between any two adjacent nodes is 20 ms. In addition, in terms of the relationship between hops and RTTs (shown in Fig. 15), we observe no delay jitter in the virtual MANET. Therefore, we test the multi-engine synergistic algorithm in an ideal network environment.

We then study the time points until data synchronization between the M-MSD and S-MSDs to show the sensitivity of the algorithm. As shown in Fig. 16, the time required

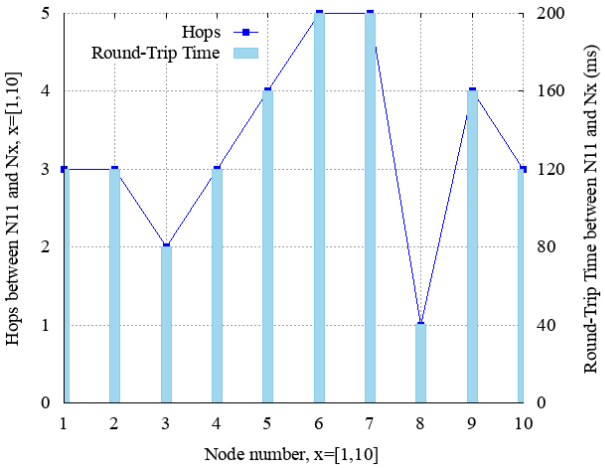


FIGURE 15. Relationship between hops and round-trip times.

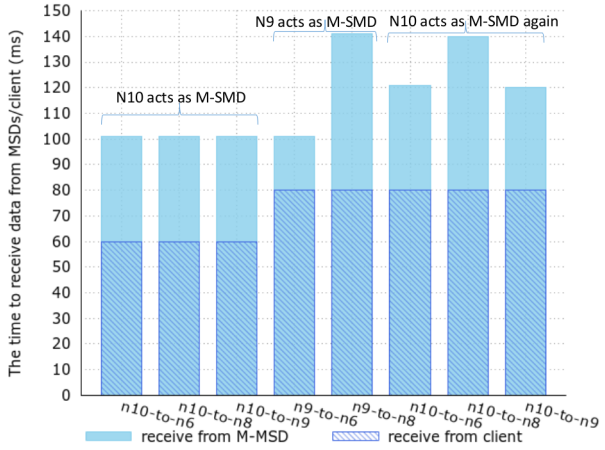


FIGURE 16. Time points until data synchronization between the M-MSD and S-MSDs.

for node N11 to visit nodes N10/N9 is the baseline time. (1) Node N10 acts as the M-MSD. When node N11 visits node N10, after 60 ms, node N10 is successfully connected to node N11 and sends synchronous data to nodes N6, N8 and N9, respectively. After another 40 ms, nodes N6, N8 and N9 receive the synchronous data. (2) The analysis process for node N9 acting as the M-MSD is similar to that of node N10 acting as the M-MSD. (3) Node N10 acts as the M-MSD again. When node N11 visits node N10, after 80 ms, node N10 is connected to node N11 and sends synchronous data to nodes N6, N8 and N9, respectively. After another 40 ms, nodes N6 and N9 receive the synchronous data. As shown in Fig. 10 (A), the minimal number of hops between node N10 and nodes N6, N8, and N9 is two. However, after 60 ms, node N8 receives data from node N10 because after node N10 moves to its original location again, the routing table generated by CORE chooses a three-hop path from node N8 to node N10. The previous analysis shows that the algorithm is highly sensitive. The delay time is mainly spent in data transmissions.

The above test results show that the multi-engine synergistic algorithm is available for data synchronization between MSDs in MANETs. The source code of the algorithm is available in [32].

VI. CONCLUSIONS

When communication infrastructures are damaged or do not exist, traditional workflow systems cannot fulfill their roles or functions. This study was performed to provide communication by using workflow technology in MANETs. The primary findings can be summarized as follows. First, a lightweight SOA-based multi-engine architecture for workflow systems in MANETs was presented, and the architecture was described. Second, the workflow engine that runs on Linux/Android was implemented using C/C++. Third, a trigger mechanism between multiple workflow engines was applied; the algorithm for the trigger mechanism was designed and implemented in C. Two experiments were conducted to validate the effectiveness and availability of the workflow engine and to verify the effectiveness of the multi-engine synergistic algorithm in MANETs.

Our group is performing ongoing work to combine Android, Docker, NS-3 and Fedora to construct an experimental platform [35], [36] that more closely matches an actual MANET environment. This platform will run cBPM4Linux, server-manet, MariaDB and Nginx in Docker and run cBPM4Android, server-manet, SQLite and Nginx in Android. Subsequently, we will be able to undertake additional studies to reveal the issues related to lightweight SOA-based multi-engine architectures for workflow systems in MANETs.

REFERENCES

[1] J. Schobel, R. Pryss, M. Schickler, and M. Reichert, "A lightweight process engine for enabling advanced mobile applications," in *Proc. OTM Confederated Int. Conf.*, 2016, pp. 552–569.

[2] W. Wipp, "Workflows on Android: A framework supporting business process execution and rule-based analysis," M.S. thesis, Faculty Eng., Comput. Sci. Psychol., Ulm Univ., Ulm, Germany, 2016, pp. 23–34.

[3] E. Castelán, M. Brigos, and J. Fernandez, "The design and development of a mobile workflow learning application," in *Proc. Int. Conf. Edu., Res. Innov.*, 2015, pp. 2881–2889.

[4] L. Thai, A. Barker, B. Varghese, O. Akgun, and I. Miguel, "Optimal deployment of geographically distributed workflow engines on the cloud," in *Proc. IEEE 6th Int. Conf. Cloud Comput. Technol. Sci.*, Singapore, Dec. 2014, pp. 811–816.

[5] J. Bi, Z. L. Zhu, and Y. S. Fan, "Multiple BPEL execution engines based on fragmentation approach and application of service process models," in *Proc. IEEE Int. Conf. Ind. Eng. Eng. Manage.*, Hong Kong, Dec. 2009, pp. 1292–1296.

[6] *InfoSphere Information Server*. Accessed: Aug. 2017. [Online]. Available: https://www.ibm.com/analytics/information-server

[7] M. Daagi, A. Ouniy, M. Kessentini, M. M. Gammoudi, and S. Bouktif, "Web service interface decomposition using formal concept analysis," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Jun. 2017, pp. 172–179.

[8] M. P. Papazoglou, "Service-oriented computing: Concepts, characteristics and directions," in *Proc. IEEE 4th Int. Conf. Web Inf. Syst. Eng. (WISE)*, Dec. 2003, pp. 3–12.

[9] R. Verma and A. Srivastava, "A novel Web service directory framework for mobile environments," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Jun./Jul. 2014, pp. 614–621.

[10] J. Xu, Y. Nakamoto, and S. Akiyama, "Workflow support based on mobile data stream management system," in *Proc. 4th Int. Symp. Comput. Netw. (CANDAR)*, Hiroshima, Japan, 2016, pp. 332–337.

[11] S. Deng, L. Huang, J. Taheri, and A. Y. Zomaya, "Computation offloading for service workflow in mobile cloud computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 12, pp. 3317–3329, Dec. 2015.

[12] E. Park and H. S. Nam, "A service-oriented medical framework for fast and adaptive information delivery in mobile environment," *IEEE Trans. Inf. Technol. Biomed.*, vol. 13, no. 6, pp. 1049–1056, Nov. 2009.

[13] H. Tao, C. Jian-Guo, and X. Wei, "Modeling mobile workflow based on business friend domain," in *Proc. 5th Int. Conf. Comput. Inf. Sci. (ICCIS)*, Jun. 2013, pp. 512–515.

[14] S. G. Wang, Y. L. Zhao, L. Huang, J. L. Xu, and C. H. Hsu, "QoS prediction for service recommendations in mobile edge computing," *J. Parallel Distrib. Comput.*, pp. 1–11, Oct. 2017. [Online]. Available: https://doi.org/10.1016/j.jpdc.2017.09.014

[15] S. Berger, S. McFaddin, C. Narayanaswami, and M. Raghunath, "Web services on mobile devices-implementation and experience," in *Proc. 5th IEEE Workshop Mobile Comput. Syst. Appl.*, Oct. 2003, pp. 100–109.

[16] K. Mohamed and D. Wijesekera, "A lightweight framework for Web services implementations on mobile devices," in *Proc. IEEE 1st Int. Conf. Mobile Services (MS)*, Jun. 2012, pp. 64–71.

[17] G. Mesfin, T.-M. Grønli, G. Ghinea, and M. Younas, "Usability of composing REST services on smartphones," in *Proc. IEEE 31st Int. Conf. Adv. Inf. Netw. Appl. (AINA)*, Taipei, Taiwan, Mar. 2017, pp. 476–483.

[18] L. Enciso-Quispe, J. Quichimbo, F. Luzón, E. Zelaya-Policarpo, and P. A. Quezada-Sarmiento, "REST architecture in the implementation of a Web and mobile application for vehicular tariff rotating parking," in *Proc. 12th Iberian Conf. Inf. Syst. Technol. (CISTI)*, Lisbon, Portugal, Jun. 2017, pp. 1–6.

[19] K. Wagh and R. Thool, "Mobile Web service provisioning and performance evaluation of mobile host," *Int. J. Web Service Comput.*, vol. 5, no. 2, p. 1, 2014.

[20] M. M. Kazzaz and M. Rychlý, "Restful-based mobile Web service migration framework," in *Proc. IEEE Int. Conf. AI Mobile Services (AIMS)*, Honolulu, HI, USA, Jun. 2017, pp. 70–75.

[21] A. Ruokonen, Z. Wu, and R. Lu, "Describing mobile devices as RESTful services for the end-users," in *Proc. IEEE Int. Conf. Mobile Services (MS)*, San Francisco, CA, USA, Jun./Jul. 2016, pp. 127–134.

[22] A. Russo, M. Mecella, and M. de Leoni, "ROME4EU—A service-oriented process-aware information system for mobile devices," *Softw., Pract. Exper.*, vol. 42, no. 10, pp. 1275–1314, 2012.

[23] H. Viswanathan, P. Pandey, and D. Pompili, "Maestro: Orchestrating concurrent application workflows in mobile device clouds," in *Proc. IEEE Int. Conf. Autonomic Comput. (ICAC)*, Wurzburg, Germany, Jul. 2016, pp. 257–262.

[24] E. Philips, R. Van Der Straeten, and V. Jonckers, "NOW: Orchestrating services in a nomadic network using a dedicated workflow language," *Sci. Comput. Programm.*, vol. 78, no. 2, pp. 168–194, 2013.

[25] F. AlShahwan and K. Moessner, "Providing SOAP Web services and RESTful Web services from mobile hosts," in *Proc. IEEE 5th Int. Conf. Internet Web Appl. Services (ICIW)*, May 2010, pp. 174–179.

[26] S. N. Srirama, C. Paniagua, and J. Liivi, "Mobile Web service provisioning and discovery in Android days," in *Proc. IEEE 2nd Int. Conf. Mobile Services, IEEE Comput. Soc.*, Jun. 2013, pp. 15–22.

[27] H. Hamad, M. Saad, and R. Abed, "Performance evaluation of RESTful Web services for mobile devices," *Int. Arab J. e-Technol.*, vol. 1, no. 3, pp. 72–78, 2010.

[28] R. Mizouni, M. A. Serhani, R. Dssouli, A. Benharref, and I. Taleb, "Performance evaluation of mobile Web services," in *Proc. IEEE 9th Eur. Conf. Web Services*, Luga, Switzerland, Sep. 2011, pp. 184–191.

[29] K. Mohamed and D. Wijesekera, "Performance analysis of Web services on mobile devices," *Proc. Comput. Sci.*, vol. 10, pp. 744–751, Jan. 2012.

[30] Y. Natchetoi, V. Kaufman, and A. Shapiro, "Service-oriented architecture for mobile applications," in *Proc. 1st Int. Workshop Softw. Architectures Mobility*, 2008, pp. 27–32.

[31] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Doctoral dissertation, Faculty Inf. Comput. Sci., California Univ., Irvine, CA, USA, 2000, pp. 76–106.

[32] *Multi-Engine-Architecture*. Accessed: Aug. 2017. [Online]. Available: https://github.com/ztguang/Multi-Engine-Architecture

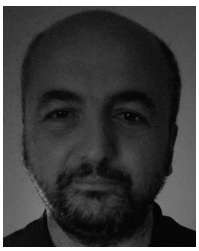
[33] cBPM. *C/C++ Business Process Management (cBPM)*. Accessed: Aug. 2017. [Online]. Available: https://github.com/ztguang/cBPM

[34] CORE. *Common Open Research Emulator (CORE)*. Accessed: Jun. 2015. [Online]. Available: http://www.nrl.navy.mil/itd/ncs/products/core
[35] FEP. *High Fidelity Experiment Platform (FEP)*. Accessed: May 2017. [Online]. Available: https://github.com/ztguang/FEP
[36] T. Zhang, S. Zhao, B. Cheng, B. Ren, and J. Chen, "FEP: High fidelity experiment platform for mobile networks," *IEEE Access*, vol. 6, pp. 3858–3871, 2018.

TONGGUANG ZHANG is currently pursuing the Ph.D. degree in computer science and technology with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. His current research interests include mobile Internet technology, Internet of Things technology, communication software and distribute computing, and embedded system and service computing.

SHUAI ZHAO received the Ph.D. degree in computer science and technology from the Beijing University of Posts and Telecommunications under the supervision of Prof. J. Chen in 2014. He is currently a Lecturer of computer science with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. His current research interests include Internet of Things technology and service computing.

BO CHENG (M'12) received the Ph.D. degree in computer science from the University of Electronics Science and Technology of China, Chengdu, China, in 2006. He is currently a Professor with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, China. His research interests network services and intelligence, Internet of Things technology, communication software, and distribute computing.



MAURIZIO FARINA was the CEO of Adaptive Software Company, Italy. He is currently an IT Solution Architect with Ericsson IT Solutions & Services SpA. His current research interests include business process and rules management, natural processing language, and machine learning.

JIWEI HUANG (S'13–M'14) received the B.Eng. and Ph.D. degrees in computer science and technology from Tsinghua University, in 2014 and 2009, respectively. He was a Visiting Scholar with the Georgia Institute of Technology. He is currently an Assistant Professor with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. He has published over 30 papers in international journals and conference proceedings, e.g., the IEEE TRANSACTIONS ON SERVICES COMPUTING, SIGMETRICS, ICWS, and SCC. His research interests are in services computing and performance evaluation. He is a member of the ACM.

JUNLIANG CHEN received the B.S. degree in electrical engineering from Shanghai Jiao Tong University, China, in 1955, and the Ph.D. degree in electrical engineering from the Moscow Institute of Radio Engineering (formerly Soviet Russia) in 1961. He is currently the Chairman and a Professor of the Research Institute of Networking and Switching Technology, Beijing University of Posts and Telecommunications (BUPT), where he has been since 1955. His research interests are in the area of communication networks and next generation service creation technology. He was elected as a member of the Chinese Academy of Science in 1991, and a member of the Chinese Academy of Engineering in 1994, for his contributions to fault diagnosis in stored program control exchange. He was a recipient of the first, second, and third prizes of National Scientific and Technological Progress Award, in 1988, 2004, and 1999, respectively.

BINGFEI REN received the B.S. degree in software engineering from the Beijing University of Posts and Telecommunications, China, in 2014, where he is currently pursuing the Ph.D. degree with the State Key Laboratory of Networking and Switching Technology. His main research interests include mobile computing, mobile security and wireless network.

SHOULU HOU received the master's degree from the Shenyang University of Technology in 2014. She is currently pursuing the Ph.D. degree in computer science and technology with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. She is also pursuing the Joint-Training Ph.D. degree with Data61, CSIRO, Australia. Her research interests include service computing and Internet of Things technology.