Modular and Highly Configurable Computation Mobility Framework for Internet of Things

Prithviraj Patil, Aniruddha Gokhale, Akram Hakiri
Email: {prithviraj.p.patil,a.gokhale}@vanderbilt.edu, ahkiri@laas.fr

***Abstract*—Computation offloading or cyber foraging is a key capability required to achieve effective resource utilization in mobile cloud computing. It enables the dynamic offloading of computations to either neighboring mobile nodes or remote cloud-based servers, retrieve results from the offloaded computations, and thereafter continue execution of the mobile business logic. A number of computational mobility solutions have emerged recently for mobile cloud computing involving smart phones and tablets. However, these solutions incur limitations in the context of Internet of Things (IoT) due to the significant heterogeneity illustrated by the range of objects involved in IoT and the fact that existing solutions tend to be tightly coupled to their underlying frameworks, which makes it hard to seamlessly adapt these solutions to the IoT scenarios. To address these concerns, this paper makes three contributions. First, it presents a novel modular and highly configurable framework for providing seamless computational mobility in the IoT realm. Second, it provides implementation details for key capabilities of this framework. Third, it provides qualitative evaluation of the framework's capabilities.**

Keywords—Mobile cloud computing, IoT, software engineering, modular and configurable framework.

I. Introduction:

Computation mobility or cyber foraging [2] is a capability that is critical to the success of mobile cloud computing (MCC). Computation mobility refers to the ability of the infrastructure to seamlessly migrate a computation from one node to another node. The major difference between computation mobility and traditional client-server distributed computing model using remote procedure call (RPC) is that in RPC no computation is being executed at the client while in computation mobility, the computation is performed mostly at the client but may get migrated to the server machine for a variety of reasons. Computation mobility is desired in cases where computing devices are available in every size and shape, and where sharing of computing resources can be beneficial.

In the context of MCC, smart phones and tablets can take advantage of computation mobility. By offloading or moving a computation, MCC attempts to alleviate restrictions on smart phone devices (e.g., their computation power, battery life, memory capacity, etc.) by offloading all or some part of its computation to either a neighboring smart phone device or to the remote cloud. A number of computation mobility solutions have been proposed for MCC like MAUI [5], CloneCloud [4], ThinkAir [11], Scavenger [12] and MobiCloud [10]. All these offloading solutions rely on partitioning the application running on the smart device (phone or tablet) at different granularities, such as a method call or a process, which is called as a *computation unit.* Depending on the offloading policy programmed in these frameworks, these computation units will be offloaded to the other devices such that optimum usage of device resources is achieved.

Although these computation offloading solutions have shown significant promise in traditional mobile cloud computing[1], they incur limitations in the context of *Internet of Things (IoT).* IoT aims to achieve total connectedness in the environment we live in [8]. Computation as we know today is constrained to only personal/server computers or mobile phones/tablets. In the realm of IoT, every person is expected to encounter thousands of new entities that are going to be smarter, powered with computational ability, and which will be able to sense/process/store/send/receive data in one form or another [1].

To list a few instance of such smart devices, people are expected to interact in a variety of ways with TVs, watches, air conditioners, clothes, car keys, refrigerators, house doors, house floor/ceiling, air conditioners, vehicles, roads, kiosks, vending machines, bank ATMs, postal boxes, medical devices and many others. This non exhaustive list of smart objects or "things" makes it immediately apparent that these objects will come in a variety of shapes and sizes. The computing power on these objects will range from few MHz to GHz. Similarly, these objects will illustrate substantial variability in their storage capacity, memory, power, network bandwidth etc.

IoT is thus positioned to be a far larger superset of mobile entities than just phones and tablets. Consequently mobile cloud computing will continue to play an important role for IoT, in turn implying that computational mobility will be a required capability – in fact a much more critical capability – for IoT to effectively utilize the resources. However, contemporary computation mobility solutions based on offloading [4], [5], [10]–[12] have been studied in the context of mobile cloud computing involving just smart phones (and tablets) but not the broad range of objects expounded by IoT. These contemporary solutions have been developed with very specific assumptions about the role, power, mobility, trust etc. of the offloading client and offloading server which is very dynamic in the IoT. To address these challenges, this paper makes the following contributions:

- We present a novel modular and highly configurable framework for providing seamless computational mobility in the IoT realm.
- We describe key capabilities of this framework such as modularity and dynamic configurability for dynamic IoT scenarios in terms of node mobility, trust, cost model etc.

[1]Although mobile cloud computing is also a relatively new field, we use the term "traditional" to refer to use cases that involve only smart phones and tablets.

TABLE I. INTERACTION SCENARIOS AND DIMENSIONS OF COMPLEXITY IN IOT FOR COMPUTATIONAL MOBILITY

No.	Complexity Dimension	Participants	Scenario
1	Relative strength and weakness of devices	smartTV, smartPhone, smartCar and smartGas-station	(1) smartTV is a weaker device while smartPhone is a stronger device; (2) smartphone is a weaker device while smartCar is a stronger device; (3) smartCar is a weaker device while smart Gas-station is a stronger device
2	Relative mobility of devices	smartTV, smartPhones and smartCars	(1) smartPhone is a mobile device for smartTV; (2) smartPhone is non-mobile device for the smartCar (as long as the vehicle is on the road)
3	Dynamically changing relative strength and weakness of devices	multiple smartPhones	One smartPhone is stronger than the another smartPhone at time t. But at time $t+1$, the situation reverses possibly due to the dynamic load on both the phones or remaining battery power.
4	Communication paradigm used	Multiple smartCars, smartPhones inside each smartCar and cloud	The smartCars communicate with each other using gossip protocol, e.g. at every traffic signal. The smartPhones communicate with each other using peer to peer protocol. A smartPhone and cloud talk to each other using client-server protocols. A smartphone and smartCar communicate with each other using publish/subscribe protocol.
5	Communication channel used	smartPhone, smartTV, smartCar and smart Gas-stations	The smartTV and smartPhone communicate over WiFI, smartPhone and smartCar communicate by direct physical connection, smartCar and smart Gas-station communicates over cellular.
6	Trust and security	smartPhone, smartTV, smartCar and smart Gas-stations	The smartTV and smartPhone always trust each other but smartCar and smart Gas-station may not trust each other all the time.
7	Stronger in one resource but weaker in another	The smartCar and RSU (Road Side Unit) used as a static relay point	The smartCar is a stronger computational device but has less network bandwidth. The RSU on the other hand is almost a dumb device but has maximum network bandwidth and stable connectivity to the wired network.
8	Application partitioning	The smartPhone and Cloud	Some applications running on the smartPhone are partitioned (e.g. using annotations) by developers while others are not.
9	Business or cost	smartPhone, smartTV, smartCar and smart Gas-stations	A smartCar and smartPhone offload to each other with no money involved; while a smartCar and smartGasStation may offload to each other based on some pricing model.

- We validate our solution by showing how it can be applied to multitude of IoT scenarios.

The rest of the paper is organized as follows: Section II-A describe the similarities and differences in using computation mobility in traditional mobile cloud computing and IoT, and outlines the key challenges that call for a modular and highly configurable framework for computational mobility in IoT; Section II-B describes the components of a computation mobility framework for the IoT scenario; Section III describes the architecture, configuration and implementation of mobility framework; Section IV compares our solution to related efforts in the area of computation offloading in traditional MCC and IoT; and finally Section V provides concluding remarks describing potential future directions and open research problems in this realm.

II. DESIGN CONSIDERATIONS FOR A IOT COMPUTATION MOBILITY FRAMEWORK

We now elicit the design considerations for realizing a computation mobility framework for IoT. First, we understand the problem space by illustrating the different dimensions of complexity that arise in the IoT realm. We then present key desired features of such a framework.

A. Computation Mobility Challenges for IoT

Section I mentioned several computation offloading solutions [4], [5], [10]–[12] that have been developed for the traditional MCC uses cases. In these scenarios, multiple mobile smart phones (or tablets) are distributed across geographical locations and connected to a remote cloud machine via cellular or WiFI network. The computation on a smart phone is offloaded either to another smart phone or to the cloud server machine. All these solutions make specific assumptions about the characteristics of smart phones and cloud capabilities as follows:

1) Smart phone is a "weaker" device.
2) Smart phone is a "mobile" device.
3) Cloud machine is a "stronger" device.
4) Cloud machine is a "non-mobile" device.
5) Cloud machine is "trusted" by smart phones.
6) Cloud machine is always a "remote" device in the context of a smart phone device.
7) Communication between smart phone and cloud obeys "client-server" protocol where smart phone is always a client.
8) A smart phone communicates with the cloud machine over WiFi or cellular networks.

In the above assumptions, the notion of stronger and weaker can be construed as relative quantitative measures of the computation power, storage power, memory, battery power or any other measurable device resource. A weaker device in the context of traditional MCC is basically a client while stronger device is the server.

Although these assumptions are valid for the traditional MCC uses cases, they apply in only a small subset of the much larger IoT realm. IoT consists of thousands of devices capable of sensing, computing, storing, sending and/or receiving data with different capacity. These devices and the services offered within the realm of IoT illustrate several additional characteristics that must be taken into consideration for computation mobility solutions, such as mobile/non-mobile, trusted/un-trusted, weaker/stronger, client/server, WiFi/cellular, free/for profit and many others. In such a complex scenario, different devices talk to each other at different times in different ways which gives rise to a very broad range of complex interactions. The increased complexity limits the applicability of contemporary computational mobility solutions in the IoT context.

In Table I we have attempted to capture many such interactions in IoT. For example in scenario 5 in the Table I, we encounter heterogeneity of communication protocols used by different devices.E.g The smartCars communicate with each other using gossip protocol(at every traffic signal e.g).

The smartPhones communicate with each other using peer to peer protocol. A smartPhone and cloud talk to each other using client-server protocols. A smartphone and smartCar communicate with each other using publish/subscribe protocol.

The list of interactions in Table I is by no means exhaustive; instead we should expect many more such complex interactions among devices as more and newer devices get attached to the IoT. Consequently, in such a varied and dynamic IoT device interactions scenario, computation offloading solutions developed for the traditional MCC are not readily applicable to all IoT interactions. Hence, we need highly configurable, modular and dynamic computation mobility solutions, which can be tailored for and used in each and every use case of IoT.

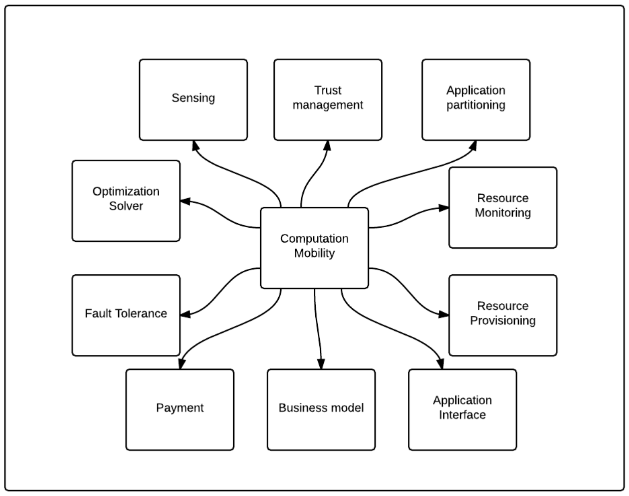

Fig. 1. Components of a Computation Mobility Framework

B. Key Desired Capabilities of IoT Computational Mobility Framework

Based on the different dimensions of complexity alluded to in Table I, we describe the key desired capabilities that can be realized as components or modules of the computation mobility framework for IoT.

Figure 1 shows the various capabilities of the framework at an abstract level. Table II lists these capabilities (in columns) indicating how the different scenarios of IoT device interactions (slightly simplified) give rise to the need for these capabilities. To capture the smallest possible variation or heterogeneity, in Table II, we use individual IoT interactions, i.e. interaction involving two IoT devices, to realize IoT complexities in a modular structure. We can also see that some modules of computation framework are needed in every IoT device interaction, e.g.optimization solver, fault tolerance, application interface and partitioning.

In this subsection, we describe these modules in more detail. We also show the possible variations of each one of those capabilities required to cater to different configurations in the IoT scenario. We will use these component abstractions to design a concrete modular and configurable framework described in Section III. Each capability and its responsibility is described below.

Sensing for Discovery and Data Collection: Node sensing is an important part of the computation mobility framework for IoT. Each node dynamically needs to (1) discover and (2) gather/update resource information of other nodes. Different nodes may use different sensing mechanisms, e.g.,

1: node discovery using multicast/broadcast protocols.
2: node discovery using publish/subscribe paradigm.
3: discovery based on node mobility predictions (e.g. in vehicular networks).
4: peer-to-peer node discovery (e.g. node discovered by one node can be used by other nodes).
5: node discovery through gossip protocol.
6: distributed node discovery where multiple nodes work together to find new nodes.
7: similar variations as above may apply during gathering of resource information from discovered nodes.

Computational Granularity: The computational mobility framework needs to decide the granularity at which it should offload to or compute offloaded computations. Below are the possible granularity variations that the framework should be support.

1: process
2: thread
3: class object
4: class method
5: application-level component
6: entire application

Optimization Solver: The purpose of an optimizer solver is to dynamically decide the matching between (1) nodes and (2) computations such that the given optimization function cost is minimized. The optimization function can be defined using the following constraints.

1: number of nodes
2: resources (computation time, power usage, network load, memory, disk etc.)
3: actual cost in dollars if cost/business model is present (see below)
4: computational granularity
5: fault tolerance policy (see below)

Cost or Business Model-based Trade-offs: Some nodes in the IoT may want to offer their resources for profit Different nodes may require different cost models.

1: A node may want to trade in dollars for their time
2: A node may want trade in dollars for their resources
3: A node may want to trade symbiotically, e.g. node A may want to use equal amount of resources in future that it is letting other node use at present
4: A node may want to trade what it has more with what he lacks, e.g. node A wants to trade in its storage space against computational power.

Payment Methods: For those nodes who want to do business using the computation framework, a payment option must be available. For cost models like those mentioned in bullet 1 and 2 of the above list, any eFinancial service can

TABLE II. CAPTURING & MODULARIZING HETEROGENEITY IN IoT

Individual interaction in IoT.	Sensing	Trust Mgmt	Business Model	Payment Payment	Mobility Mgmt	Comm Mgmt	Resource Monitor	Resource Provision	Fault Mgmt	App i/f	Opt Solver
smartCar & smartPhone in the same smartCar	✗	✗	✗	✗	✗	✔	✔	✗	✔	✔	✔
smartCar & smartPhone in the different smartCar	✔	✔	✔	✔	✔	✔	✔	✗	✔	✔	✔
two smartCars both at rest	✔	✗	✗	✗	✗	✔	✔	✔	✔	✔	✔
two smartCars both on the road	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
two smartphones at home	✔	✗	✗	✗	✗	✔	✔	✔	✔	✔	✔
a smartPhone & smartTV (physically connected)	✗	✗	✗	✗	✗	✗	✔	✔	✔	✔	✔
a smartPhone & cloud	✗	✗	✗	✗	✗	✔	✔	✗	✔	✔	✔
a smartPhone & smart Gas-station	✔	✔	✔	✗	✗	✔	✔	✔	✔	✔	✔
a smartCar & smart RSU (Road-Side-Unit)	✔	✔	✗	✗	✔	✔	✔	✗	✔	✔	✔

be used. However, for cost models like those in bullet 3 and 4, we need a secure way agreed by both parties involved. We may need to implement secure software service on the basis of bit-coin for such cost models.

Fault Tolerance Strategies: Different nodes may have different fault management strategies in the offloading framework. For instance, a node N1 may want to offload the same computation C1 to two nodes N2 and N3 so as to tolerate one fault. Node N2 in turn may also want to offload C1 (which was offloaded to it by N1) to N4 (in addition to computing it by itself) to tolerate a fault and retain its credibility in the system. This is desirable especially if nodes are getting charged/paid for offloading computation. Nodes may want to protect their computations against following faults.

1: FT against unreliable network
2: FT against random node mobility
3: FT against dynamic load
4: FT against malicious nodes
5: FT against network partitioning

Application Interface: Applications running on the device may need to communicate with the framework in a standard way for one or more of the following purposes:

1: report computations which are compute-intensive/missing deadlines.
2: report the required granularity/partitioning of an application, computation priority and other configuration information
3: receive information about offloaded computation like expected time for results
4: receive results back from the framework of the offloaded computation

Security/privacy/trust of the Computation: Nodes use different protocols/strategies for various trust management tasks like

1: establish a trust with other device
2: communicate with a trusted device
3: communicate with an untrusted device

Framework should be flexible enough to accommodate these variation among nodes.

Communication Protocols & Channels: Different devices employ different strategies for sending and receiving data to other devices. Framework should be flexible enough to take these into consideration. The variation here also affects the way nodes discover each other (described under the first bullet point)

1: publish-subscribe based data dissemination
2: client-server based data dissemination
3: peer-to-peer based data dissemination
4: multi-path data dissemination
5: network channels(cellular, WiFi, radio, wired)

Resource Monitoring: Framework also need to make sure it has latest resource usage information of the device. Hardware resources that need to be monitored could be following

1: network b/w
2: processor usage
3: power usage
4: memory usage
5: disk usage
6: sensors

Resource provisioning: Nodes also need to make sure they have enough resources before they accept the computation from another node. This is relevant especially to those nodes which scale up and down based on the load like cloud server, e.g. smart Gas-station is serving no smartCars at time t0 and hence running on low power mode or possibly on hibernate. But at time t1, n number of smartCars arrive at gas station, discover a smart Gas-station, sense an opportunity for faster computation and offload some of their heavy computations to it. The gas station needs to make sure it is able to scale up its computing resources to serve those cars and financially benefit from it [14].

III. IoT COMPUTATION MOBILITY FRAMEWORK: ARCHITECTURE, CONFIGURATION AND IMPLEMENTATION

In this section we concretely describe the internals of the framework by focusing on its software engineering, i.e., class structure of its various components.

A. Architecture

We describe top-level or primary classes (entities) that exist in the software architecture of the IoT computational mobility framework that capture the IoT complexities. The need for these classes/entities in our software framework is motivated by the key capabilities described in Section II-B. These classes/entities components encode one or more of these capabilities. We show the mappings between abstract modules we envisioned in the previous section mapped to concrete class structue of the framework in the Table III.

TABLE III. MODULE TO CLASS STRUCTURE MAPPING

Class	Sensing	Trust Mgmt	Business Model	Payment	Mobility Mgmt	Comm Mgmt	Resource Moni-tor	Resource Provi-sion	Fault Mgmt	App. Inter-face (pari-tioning & granu-larity)	Optimi-zation Solver
node_repository :	✔	✔	✔	✔	✔	✕	✕	✕	✕	✕	✔
mobility_info	✕	✕	✕	✕	✔	✕	✕	✕	✕	✕	✔
trust_model	✕	✔	✕	✕	✕	✕	✕	✕	✕	✕	✔
resource_info	✕	✕	✕	✕	✕	✕	✔	✔	✕	✕	✔
cost_model	✕	✕	✔	✔	✕	✕	✕	✕	✕	✕	✔
application_info	✕	✕	✕	✕	✕	✕	✕	✕	✔	✔	✔
communication_info	✔	✕	✕	✕	✕	✔	✕	✕	✕	✕	✔

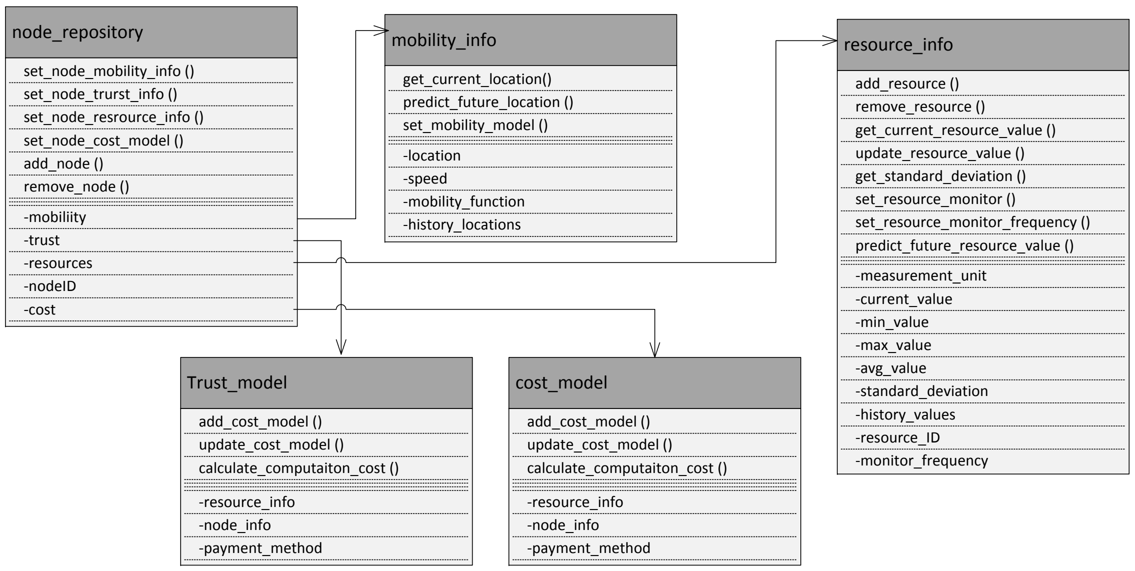

Fig. 2. Node Repository Class Diagram

1. Node Repository: Figure 2 shows the class structure of the node repository. The node repository object is used to store the information about the nodes that device can connect to. The important information of a node required for computation mobility including node mobility, resources, cost, trust etc. is stored in here. It will have functions to set/get information about node and to add or remove nodes.

We store mobility information of a node in the mobility_info object which has parameters like current location, current speed, frequency for location update and history location data. It also has parameter called mobility function. It is used to predict the mobility of node if required by other components e.g. cost function may need to determine network connectivity during given time period in the near future to determine the realistic cost of offloading to a node. This function can be used in two ways. For devices with specialized mobility patterns like a moving car, train, airplane or phones, this function can be set/adapted according to the route set by the user of the moving device. For other devices where mobility is random (like phone with a walking person), this function is derived from the recent history location data.

The Node repository stores resource information about nodes into the resource_info object. Resource information can be configured with various standard and non-standard resources.

2. Resource Info: Figure 3 shows the class structure of the resource_info. It has functions to add resource, remove resource, get/update/predict resource value, add a resource monitor, set frequency for resource monitoring. This class can be used to collect information about various general resource types like network, disk, memory, power, computing power, sensors etc. It can also be used to define special-purpose resources by overriding functions in the base resource class. For example, in the use case 5 from Table I, smartCar may want to offload its computation to other smartCar based on the temperature/fuel efficient of the other smartCar. This object has parameters like measurement_unit, max value, min value, history values, monitor and monitor frequency.

3. Application Repository: Figure 4 shows the class structure of the application repository. Every node in IoT that runs its application(s) would like to use the offloading framework. So the framework stores information about such applications. It provides an API interface to the running applications to talk to the framework. This class of framework needs to be linked

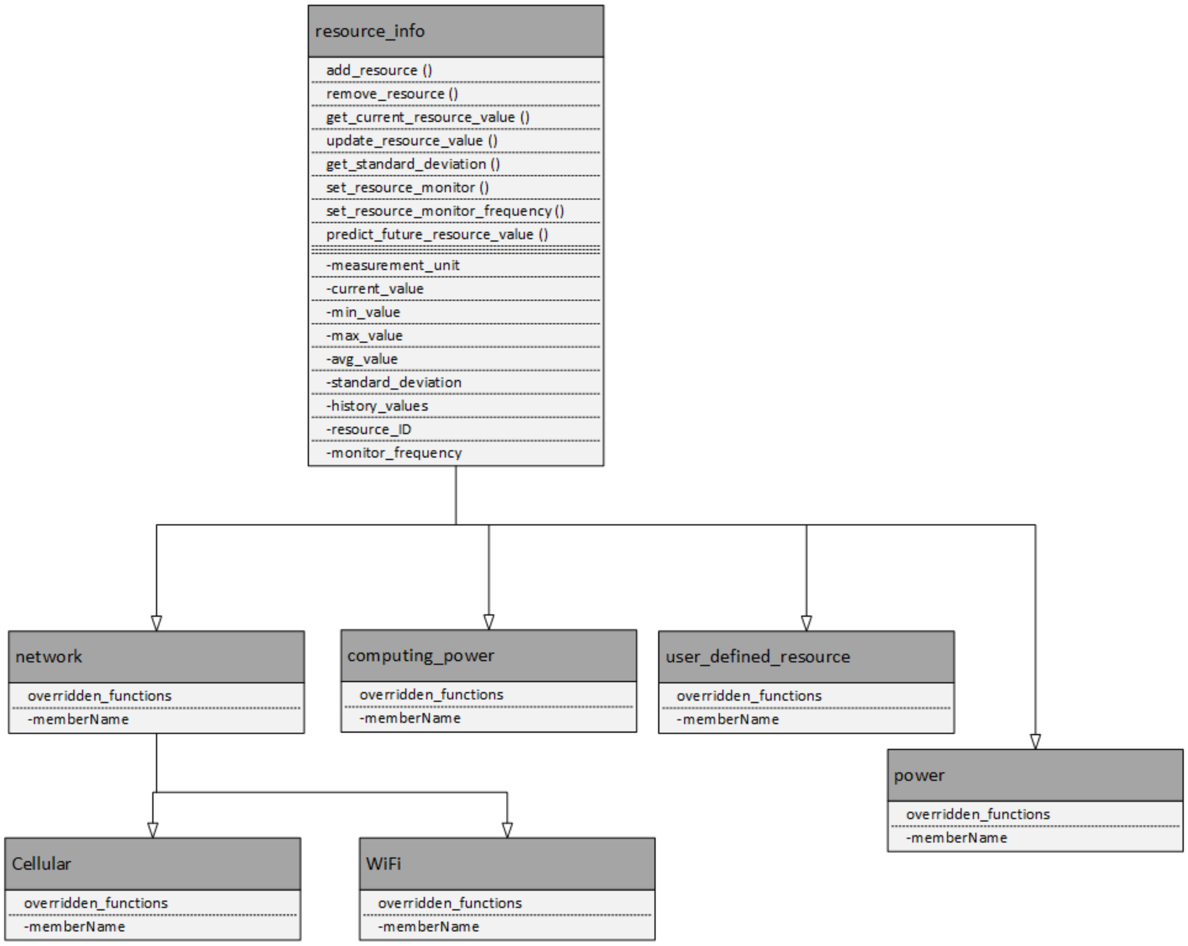

Fig. 3. Resource Info Class Diagram

to the application running on the device which wants to use the computation mobility framework.

An application can register itself with the framework, set computational granularity for offloading, set priority, set a partition-er strategy or define a specialized partitioner, and provide partitioning points. This class object has parameters like application_id, computation_info, and partitioning_info. It can then register or remove computations with the framework for offloading which is the object of the class computation_info. The computation_info stores all the information related to a computation such that it can be offloaded, e.g. code_size, code, time_limits, resource_requirements, history_run_times, computation_id and application_id.

Communication Repository: Figure 5 shows the class structure of the communication repository. This captures all the variations in the node to node communication. It has functions to discover nodes, update nodes, get node information (mobility information, trust information, resources information etc). It also can offload a computation to a node and receive results or receive a computation from a node and send the results back. It can be configured with various network communication protocols (publish-subscribe, peer-to-peer, gossip protocol) and network channels (cellular, WiFi etc). Different communication protocols can be defined by overriding functions in the base class like discover_nodes.

B. *Configuration:*

Now that we have designed all the required components for the IoT computational mobility framework, we describe how to configure these components to satisfy requirements of individual node or application. Note that there could be different/better ways to configure the framework. The primary purpose of describing it is not to show how to configure framework but to show how easily the framework can be configured.

Algorithm 1 describes one possible approach to configure the communication component. At the start, Algorithm 1 sets the communication channel and communication protocol. Values for these two parameters are provided by the configuration object. The framework reads the configuration values from the .xml file and then populates the configuration object to be used by other components in the framework.

At line 3, it sets a trigger to notify that discovery is required. Framework can use different types of user-defined triggers like

1) timeout: e.g. after every 10 seconds.
2) resource value: e.g. if cpu usage crosses 90%
3) application/process level fault: e.g. if a process/application misses a deadline.

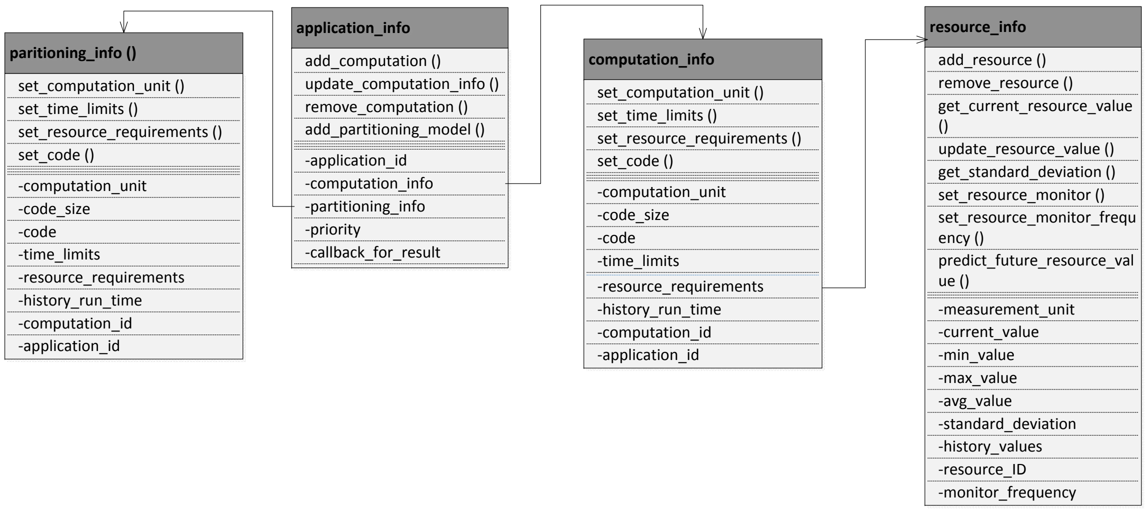

Fig. 4. Application Repository Class Diagram

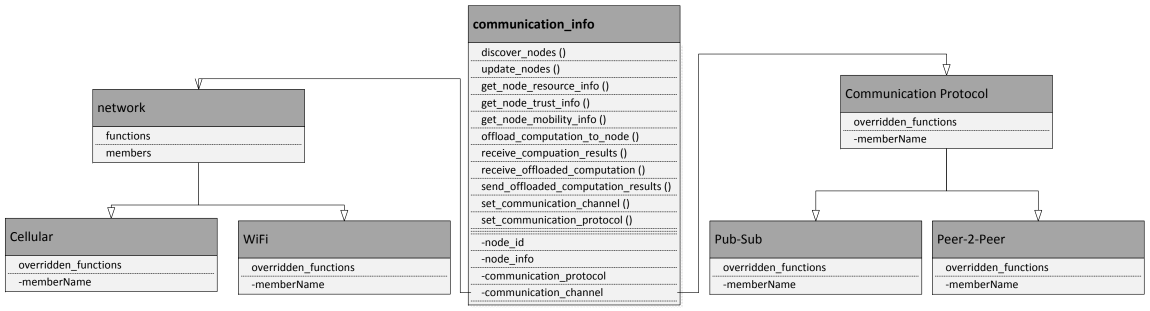

Fig. 5. communication info: class diagram

4) offloadable computation added: e.g. if a new computation is added to offload-able computation list

Then the algorithm proceeds to check if a discovery event has been triggered. If triggered, it continues to discover nodes and get their trust information. If the trust level of node is as per the requirements, then it stores these nodes in the trusted list otherwise in the non-trusted list. It also gathers more information about nodes like mobility, resource, cost etc.

Algorithm 2 describes a way to configure the framework to implement optimizer solver. The optimizer solver described in the 2 is a general purpose solver. It takes optimizer function from the configuration object as input. It then tries to find the best mappings of offload-able computations to the computation nodes so that specified optimization function is optimized. Of course, we can implement optimization in various other ways however main point here is to show the flexibility of framework to realize such fine-grained configurations and optimizers.

C. Proof-of-Concept Implementation & Validation:

To validate the feasibility of our framework, we have implemented some components of the framework for providing a proof of concept. We have also validated it in the smaller simulated IoT environment. This validation shows that our framework is highly configurable and can be applied to various scenarios in IoT seamlessly.

For validation, we have configured our framework with "process" as a computation granularity. We used existing process level migration solutions for java applications. It can then be used to migrate a process running on one JVM capturing all its context, stack, register, heap values and migrate it to another remote JVM. We ran compute-intensive application (for calculating highest prime number) on all nodes. This application has 110 lines of code and would take 300 seconds on the normal desktop machine.

We validated this framework using the (mini) IoT scenario with multiple computational nodes with varying properties like mobile/static, trusted/un-trusted, closer/remote etc. We simulated computational nodes using the docker containers running in virtual machines and communicating with each other with virtual network interface. We simulated network for these nodes in the Mininet (and NS2) simulator. For mobility, we used random way-point model for mobile nodes. We did not use dynamic trust model but statically defined some nodes as trusted while other as un-trusted. We also did not use dynamic cost model but statically defined cost model for every node before simulation. We used different simulated communication

Algorithm 1: Communication Setup & Discovery

```
Data:
1. comm_info = an object of communication_info class
2. r_node = remote node
3. config = object of class configuration_info
Result:
1. nodes are discovered according to communication protocol and
communication channel
2.
comm_info.set_communication_protocol(config.proto);
comm_info.set_communication_channel(config.channel);
comm_info.set_discovery_evnet_trigger();
while true do
    if comm_info.discovery_event_trigger == true or time_to_last_discovery >
    Y seconds then
        comm_info.discover_nodes();
        while comm_info.nodes.empty() == false do
            r_node = comm_info.get_nodes ();
            r_node.get_node_trust_info;
            if r_node.check_trust == true then
                r_node.get_node_resource_info();
                r_node.get_node_cost_info();
                r_node.get_node_mobility_info();
                add_to_trusted_node_list();
            else
                add_to_untrusted_node_list();
            end
        end
    else
        discovery_event not triggered;
        sleep(x seconds);
        continue;
    end
end
```

Algorithm 2: Offloading Optimization Solver

```
Data:
1. node_infos = an array of communication_info class objects
2. comp_infos = an array of computation_info class objects
3. comm_info = an communication_info class object
4. configuration = an configuration class object
Result:
1. offloadable computations are matched against discovered nodes
2. matching is optimized for network traffic, cost, reliability
comm_info.set_communication_protocol();
comm_info.set_communication_channel();
total_cost = 0;
total_network_load = 0;
total_power_usage = 0;
for comp in comp_infos do
    for node in node_infos do
        if node.resources > comp.resources & node.cost.compute_cost(comp)
        < comp.cost then
            total_cost += comp.cost;
            network_traffic += comp.code_size;
            if configuration.optimizer == cost then
                optimize total_cost;
            end
            if configuration.optimizer == network_load then
                optimize total_power_usage;
            end
            if configuration.optimizer == power_usage then
                optimize total_power_usage;
            end
        else
            this node is not suitable for computation;
            Check next node;
            continue;
        end
    end
end
```

channels (WiFi, 802.11a/b/g, wired etc) and communication protocols(client-server, p2p) for communication among the simulated nodes. This experiment showed that framework is flexibile enough to adapt to various scenarios.

However, we are yet to fully implement various components of the framework including trust management, fault tolerance, payment etc. We plan to achieve full implementation of the framework and exhaustive validation in the future.

IV. Related Work

This section compares our work with prior work on computation offloading frameworks and efforts that provide these to IoT.

A. Computation Offloading Frameworks

There are several survey papers available [6], [7], [15] that discuss different offloading frameworks in the traditional MCC realm including MAUI [5], CloneCloud [4], ThinkAir [11], Scavenger [12], MobiCloud [10]. These prior efforts both similarities and differences in their offloading frameworks.

MAUI [5] uses a combination of VM migration and code partitioning to conserve the power of mobile devices. It is implemented in the .NET environment and uses a run-time, dynamic partitioning algorithm. The partitioning strategy is intrusive and application developers need to annotate the code at method level to let the framework know about offloadable methods. MAUI however does not address the scaling of execution in cloud.

CloneCloud [4] unlike MAUI [5] can be used with unmodified applications but it requires the clone of the entire device on the cloud machine. It uses both static analyzer and dynamic profiler to partition an application so that it migrates and executes in the cloud, and resumes on the device. It uses a simple cost model based on the execution time.

ThinkAir [11] tries to combine approaches used by MAUI [5] and CloneCloud [4] and improves their individual shortcomings. It improves MAUI's lack of scalability by creating virtual machines (VMs) of a complete smart-phone device on the cloud machine. It also removes the restrictions on applications that CloneCloud induces by adopting an online method level offloading. It supports on-demand resource allocation in the mobile cloud based on workloads and deadlines

MobiCloud [10], though not strictly an offloading framework, is a framework for secure, trustworthy mobile cloud in an unsecured network (e.g. MANET). The MobiCloud framework can be used to create a secure communication in the mobile cloud network with trust management, secure routing, and risk management taken care by the framework.

Scavenger [12] is another framework that employs cyberforaging using WiFi for connectivity, partitions the application and distributes jobs. However, it strictly uses a client-server architecture and hence offloading to surrogates is possible but not vice-versa.

All of the above frameworks provide computation mobility in a very specific way by considering only a subset of variants that we discussed in Section II and hence it becomes almost impossible to deploy these solutions in a scenario where offloading requirements vary based on mobile node, application, security, computational granularity etc.

Also, as we discussed in Section II-A, all the above frameworks work in a scenario where multiple weak mobile smart-phones work as client device and are connected to a remote powerful cloud machine which work as a server machine. However, IoT brings a whole new set of different scenarios (refer to Section II-A) where the same device may act as a weak or a powerful, a remote or a non-remote, a mobile or a non-mobile device. In such complex and dynamic scenarios, solutions mentioned above would not work.

B. *IoT and computation mobility*

These papers [1], [8] discuss the various aspects of the IoT including big vision, architecture, components and future challenges.

In our literature survey, we could not find any major works which would extend MCC computational mobility like features to the space of IoT. However, we find the concepts described in [3], [9], [13] (e.g. fog computing, data processing at the edge of network, etc.) very relevant to the framework we have proposed in this paper. The authors in [3] propose a fog computing concept which is a highly virtualized platform that provides compute, storage, and networking services between end devices located at the edge of the network and traditional cloud computing data centers. Fog computing is proposed as a compliment to the traditional cloud computing especially in the IoT scheme. Though the paper discusses the need for interoperatibility and mobility support in the fog computing network, it however does not discuss computational mobility in particular.

V. Conclusion & Future Work:

In this paper we discussed the importance of code mobility in mobile cloud computing (MCC), and specifically in the Internet of Things (IoT) environment. We discussed the major differences in the traditional MCC and IoT scenarios arising because of the high degree of hetergeneity in the IoT compared to traditional MCC. We motivated the need to take a fresh perspective at designing a modular and highly configurable computation mobility or offloading framework for IoT since the solutions developed for traditional MCC environment are not applicable in IoT due to tight coupling and lack of configurability. We described the various capabilities of such a framework and show a way to design it by providing a concrete architecture through class diagrams, configuration and proof-of-concept implementation.

In the future, we plan to implement the full implementation of our framework as a reference point with every sub-component implemented so as to cover every scenario or device interaction in the IoT. We also plan to deploy the framework in the real-world or simulated IoT scenario and configure it dynamically and test the performance of computation mobility on the resource usage

Acknowledgments

This work is supported in part by NSF CAREER CNS 0845789. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of NSF.

References

[1] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.

[2] Rajesh Balan, Jason Flinn, Mahadev Satyanarayanan, Shafeeq Sinnamohideen, and Hen-I Yang. The case for cyber foraging. In *Proceedings of the 10th workshop on ACM SIGOPS European workshop*, pages 87–92. ACM, 2002.

[3] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16. ACM, 2012.

[4] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the sixth conference on Computer systems*, pages 301–314. ACM, 2011.

[5] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. Maui: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 49–62. ACM, 2010.

[6] Hoang T Dinh, Chonho Lee, Dusit Niyato, and Ping Wang. A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless communications and mobile computing*, 13(18):1587–1611, 2013.

[7] Niroshinie Fernando, Seng W Loke, and Wenny Rahayu. Mobile cloud computing: A survey. *Future Generation Computer Systems*, 29(1):84–106, 2013.

[8] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645–1660, 2013.

[9] Kirak Hong, David Lillethun, Umakishore Ramachandran, Beate Ottenwälder, and Boris Koldehofe. Mobile fog: a programming model for large-scale applications on the internet of things. In *Proceedings of the second ACM SIGCOMM workshop on Mobile cloud computing*, pages 15–20. ACM, 2013.

[10] Dijiang Huang, Xinwen Zhang, Myong Kang, and Jim Luo. Mobicloud: building secure cloud framework for mobile computing and communication. In *Service Oriented System Engineering (SOSE), 2010 Fifth IEEE International Symposium on*, pages 27–34. IEEE, 2010.

[11] Sokol Kosta, Andrius Aucinas, Pan Hui, Richard Mortier, and Xinwen Zhang. Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *INFOCOM, 2012 Proceedings IEEE*, pages 945–953. IEEE, 2012.

[12] Mads Darø Kristensen. Scavenger: Transparent development of efficient cyber foraging applications. In *Pervasive Computing and Communications (PerCom), 2010 IEEE International Conference on*, pages 217–226. IEEE, 2010.

[13] Ichiro Satoh. A framework for data processing at the edges of networks. In *Database and Expert Systems Applications*, pages 304–318. Springer, 2013.

[14] Mahadev Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The Case for VM-Based Cloudlets in Mobile Computing. *Pervasive Computing, IEEE*, 8(4):14–23, 2009.

[15] Muhammad Shiraz, Abdullah Gani, Rashid Hafeez Khokhar, and Rajkumar Buyya. A review on distributed application processing frameworks in smart mobile devices for mobile cloud computing. *Communications Surveys & Tutorials, IEEE*, 15(3):1294–1313, 2013.