An Infrastructure as a Service for Mobile Ad-hoc Cloud

Venkatraman Balasubramanian

School of Electrical Engineering and Computer Science
University of Ottawa
Ottawa,Canada
vbala038@uottawa.ca

Ahmed Karmouch

School of Electrical Engineering and Computer Science
University of Ottawa
Ottawa, Canada
akarmouc@uottawa.ca

Abstract— **In this era of growing mobile device technology, the direction of growth is moving towards providing powerful computational capabilities and expanding memory in the device. Nevertheless, this growth has objectively put a lot of the device computational power to an unused state which calls for a better management of intra-device resources. Over a period of time, it has been studied that a mobile "edge-cloud" formed by these devices could be as productive or close to the productivity of the public cloud in terms of providing a service. However, the ease of access to this pool of devices is much more arbitrary and based purely on the needs of the user. This could categorically be summed as the building block of a cloud built for providing an infrastructure for various services that can be processed with volunteer node participation. This representation of cloud formation to engender a constellation of devices in turn providing a service is the basis for the concept of Mobile Ad-hoc Cloud Computing. In this manuscript, an Infrastructure as a Service paradigm in Mobile Ad-hoc Cloud Computing is delineated. A novel architecture for discovering a dedicated pool of devices and the dependencies it should satisfy while formation of this pool for computation is designed. Moreover, a peer-to-peer composition algorithm to form this dedicated resource pool is proposed.**

Keywords— Mobile Cloud Computing; Mobile Ad-hoc Cloud; Infrastructure-as-a-Service;Composition;Mobile-Phone-Virtualization;

I. Introduction

In today's accelerated growth of mobile device technology, there is a need to establish a firm ground for these devices to stay committed to application computation and completion. From [12] it can be inferred that the rate of mobile device usage has increased over the decades. Additionally, the growth of mobile application such as real-time gaming, face recognition, and music OCR also gives a similar picture. With an overall growth rate of 29.8% each year noticed in [2], by the end of 2017 there would be more than 4.4 billion mobile application users. Out of these, there are around one in four mobile applications that are downloaded once and never used again. These applications are primarily discarded due to the growing application needs that have gone beyond the mobile device capabilities. Thus, even if the device is able to process its OS, the remaining resources are finding it difficult to process these intensive applications and resort to costly remote cloud services. Remote cloud services rely on large consolidated datacenters that provide compute and storage. However, these data centers represent a point of centralization that has serious shortcomings. It can end-up as a single point of failure in times of disasters or data center's geographical location is often- times out of limits for the customers using it.

Moreover, public clouds have frequent issues such as infrastructure cost and high Round trip time (RTT) while considering time sensitive classes of applications. In [1][3][4] authors discuss many services and applications which ascertain using remote clouds as infeasible. These are services where applications are solely dependent on the time and place in which the applications need to be executed. Such place-bound activities are best addressed at the user level. This class of cloud computing that deals with the formation and deployment at the user's level is known as Mobile Ad-hoc Cloud (MAC). An MAC is a pool of device with high computational capabilities and is closer to the user. This low-cost computational environment is deployed over a network where all nodes cooperatively maintain the network. Hence, wireless local area networks (WLANs) and Mobile Ad-hoc networks (MANETs) are predominantly considered [6] where users can form a wireless network at any place. For example, a P2P network enabling a computational environment for mobile nodes could be referred to as Mobile P2P Cloud.

There are many features that differentiate cloud models in mobile ad hoc networks with public clouds, however, the most integral out of these are (i) Both consumer and provider nodes are mobile (ii) Service composition would change dynamically depending on the available resources (nodes).

Now, consider the case of a music concert where a crowd has gathered for watching an artist perform. As shown in Figure 1 nearly all connect to the closest wifi access point. It's a common sight in such venues when artists are trying to enthrall the crowd by making the attendees present therein sing for them or interact with them through a mobile wave. Various interactive applications that are used at concerts not only play back pre-recorded notes but also convert audio to text or a music OCR(optical character recognition) for notes or lyrics viewing on the spot at the gig site. Some artists have also begun to call the use of smartphone application in concerts as the new applause [15].These applications are not only compute intensive but are also bound by place and time.

What if devices present therein are able to provide compute and storage facilities to one another? A pool of idle intra-device resources put together would more likely provide a low-cost service in lesser time than a remote cloud. Thus, every device has the potential to act as a service provider in mobile ad-hoc clouds. This has been a motivating factor in harnessing the idle device resources that are not completely capable of performing intensive computation but display the ability to collaboratively perform a compute intensive application execution. Just as a Cloud Service Provider (CSP) is an entity that is responsible for providing an Infrastructure as a Service (IaaS) to the consumers, in a mobile environment each device behaves as an IaaS provider. This paper illustrates the IaaS paradigm in a mobile ad-hoc cloud environment.

Considering the need for spontaneity, coordination and storage and computation as the most essential requirements, this paper makes the following contributions in that direction:

- An architecture to address the IaaS based mobile ad-hoc cloud requirements is proposed.
- A composition algorithm called, DARC (Distributed Ad-hoc Resource Composition) is developed.
- An offloading application that performs a coordinated task execution is used to evaluate the performance of the composition algorithm.

The rest of this paper is structured to delineate the complete mobile ad-hoc cloud model with the discussion on related work in Section II, followed by Section III that discusses the system overview. In Section IV, the system architecture is elaborated and in Section V the IaaS algorithm functionalities are provided with an offloading use case to evaluate the performance of our algorithm and Section VI concludes the paper and shows the future potential of this work.

II. Related Work

In [3], authors propose an architecture that provides a centralized framework by making requests to the server with an Ad-hoc client that runs on each device processing the task. Similar to this in [9] all mobile nodes are connected to cloudlets by WiFi, modeling their architecture based on a cloudlet's presence. In contrast to these, our architecture attempts to establish an autonomous and decentralized network with a dynamic composition procedure. In [5] authors demonstrate a similar approach as ours but show a performance degradation at the time of offloading, in contrast, our architecture shows better performance as evaluated later in the paper. [16] is closest to our work, however its purpose was only to provide insight on security issues in mobile ad-hoc cloud.

In [7] authors give importance to cryptography (AES based) for ensuring security following a validation of trust and photograph based certification with manual key modifications for trust establishment. It is our belief that a level of security is achieved with the hashing of files and we consider the remaining aspects of validation and certification to be adding to the overall time of the execution. In the proposed architecture we implicitly achieve security with the key based resource allocation mechanism.

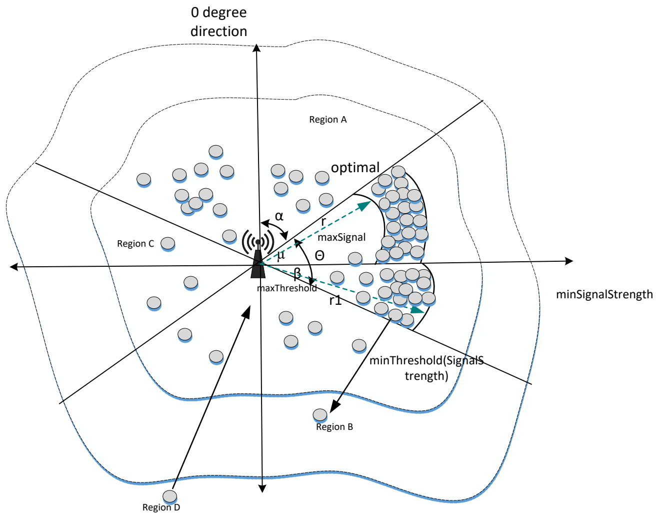


Fig.1. A typical concert venue, those coming into the area from Region D,Region C represent the new attendees who are aware of which Access points they are connected to where all friends gather.

[11] discusses an ad-hoc cloud formation protocol that makes use of separate Cloud proposal broadcast after analyzing all the replies from the initial broadcast. Our algorithm (Distributed Ad-hoc Resource Composition –DARC) obviates the need to analyze all the replies. There are various other DHT (Distributed Hash Table) protocols like Kademlia[10], Krowd[12] that are used for content sharing that might not look pertinent for comparison at first but are similar in structure . [12] is a modified version of [10] (although authors draw out performance characteristics (bandwidth and latency) of Krowd and Kademlia to provide reasons to separate their system from the DHT family). The initial parts of DARC algorithm bear resemblance to the autonomy and lightweight discovery of [12] but have extended the features of DHT (that [12] overlooks) at the resource allocation level to form the resource pool. Further, the DARC algorithm prioritizes node-computing capacity owing to the architectural requirements.

III. System Overview

In this section, the foundations, concepts and the integral components on which the mobile ad-hoc cloud architecture is built are discussed.

A. Concepts

Logically, a distributed cloud infrastructure can be pictured as large dispersed individual computers connected over a network. These cloud frameworks have characteristics similar to P2P systems, some of which are observed in the previous sections. As we are dealing with such a system in a crowd-sourced mobile environment it is defined as a Mobile P2P cloud or Mobile Ad-hoc cloud. A Mobile Ad-hoc cloud harvests resources that are available in the vicinity. As mentioned previously, the mobile devices are responsible for playing the role of the cloud IaaS providers. The role of requesting a cloud service from the providers is of the consumer. Thus the major actors in any cloud computing paradigm are the cloud providers and consumers.

In a mobile ad-hoc clouds, due to the resource limitations in a mobile, compute-intensive applications require external assistance for execution. For instance, the tasks which cannot be processed locally and require a resource rich environment would need to be offloaded to the cloud providers. Thus, the consumer makes the IaaS request. Considering scenarios such as those with a high density of users (cloud IaaS providers) these requests submitted by the consumers are exposed to the IaaS providers.

Once the IaaS request is received, the IaaS provisioning entity will discover cloud IaaS providers. These are resources whose ownership is with individuals that are available in the vicinity. As the major challenge is to turn this diverse collection of resources into a usable cloud infrastructure a composition algorithm is proposed. The composition algorithm performs the key functionalities pertaining to the services provided to the consumer. The consumers of IaaS have access to virtual resources available in the devices as explained below.

The IaaS is deployed over a wireless network formed with the assistance of an Access point (AP). Many volunteers (who wish to offer their VMs) may exist in the vicinity that provides a unique service to the consumer who requests the infrastructure. The physical resources are known as volunteer resources because of their ability to offer their VMs. For example, one user who is at the concert will have many friends or like-minded people who would be ready to offer their resources. Out of the many friends, the IaaS would select only the nearest devices. These friends (volunteers) will provide their device VM/VMs. In this paper, one request is either dedicated to a single VM or be a part of many VMs.

In this way, the salient features of cloud computing i.e. on-demand self-service and service orchestration is realized with mobile ad-hoc cloud computing.

B. Virtualization and Mobile Ad-hoc Cloud

The architecture relies on the Mobile-Phone-Virtualization concept. As observed in [13] the hardware virtualization approach for smartphones (Virtual Phones) have isolation and light-weight characteristics similar to the Virtual machines. Therefore, in this paper, we refer to virtual phones (VPs) as virtual machines (VMs) of the devices. One device might have multiple background VMs/VPs each offered to different customers. The light-weight VMs from devices are harnessed to deploy IaaS. These VMs have adequate storage and compute capabilities. The process of obtaining a VM and the dependencies it should satisfy is illustrated in figure 2. It is as follows:

The first part of the IaaS algorithm is a composition that is responsible for discovery, selection and P2P formation. On discovery (1), the volunteer submits the details (2) of the VMs, node id, and the IP addresses to the IaaS. After this, routing and management is done with the assistance of the information (key, value) in storage. Concurrently, considering the dependencies the metafile is created. It uses the sub-task information (2a) and the resource information obtained from (2). The meta-file is retrieved (3a). It is then hashed and the keys are used for taking the meta-file (3b) to the correct device. It also has the location of the source file which is used by the VMs for downloading and processing the job (3c).

After (1,2) , the virtual machines are composed(3) followed by (3b), the jobs are obtained with a get request from the user device (3c), processed, executed (4) and the results are sent back(5) after which the resources are released.

In this way, by making the resources available to the cloud consumer, the ability to use the mobile ad-hoc cloud to execute any applications lies with the consumer. This conforms to the cloud IaaS paradigm.

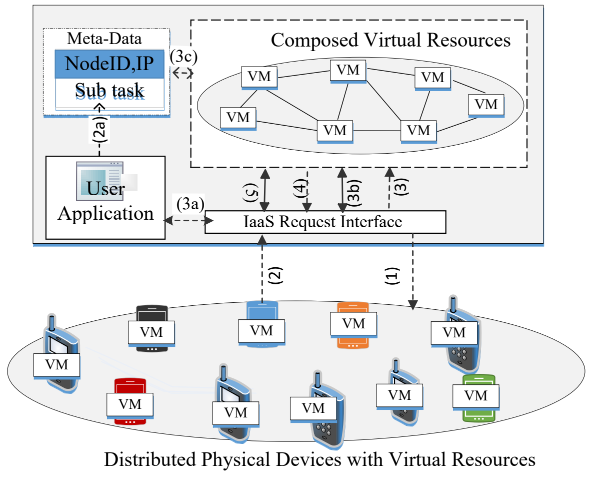


Fig.2. Mobile Ad-hoc Cloud System Overview

IV. System Architecture

In this section, the details of the system framework depicted in figure 3 are discussed. The architecture has two integral parts that are responsible for 1.) IaaS request generation 2.) IaaS composition and provisioning

A. IaaS request generation

In this part, the request is generated. It consists of the following:

1. Application Layer- These are any user application that could make use of services in a crowd-sourced environment. The applications are agnostic to the provisioning mechanism and the interactions between the layers below.

2. Cloud Consumer Layer –This layer receives the mobile device application's offloading requests. The Profiler does the decomposition of heavy application tasks into light-weight jobs. It gives the information of execution profiles to the offloading manager. The complexities of the profiler are beyond the scope of this paper.

The Task scheduler constructs a first-in first-served queue that maps the execution profile to the node profiles. The foremost goal in scheduling the jobs is considering the network parameters (3G,4G,Wifi or Wifi-direct) at the time of offloading for the purpose of minimizing the cost. It schedules the decomposed jobs to compute/storage resources obtained.

The Offloading manager accepts the information from the profiler and coordinates with the task scheduler input to queue these requests. The decision of whether to offload or not is made here. Once a decision to offload is taken, an IaaS request is made to the cloud provider. As the VMs of the devices are

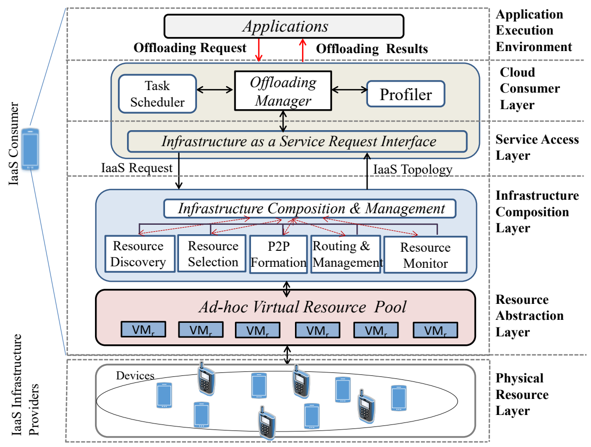


Fig.3. Mobile Ad-hoc Cloud Architechture

received and utilized by this module, the offloading manager behaves as a consumer.

B. IaaS Provisioning and Composition

In this part, the infrastructure composition is assembled and made available in usable form to the consumer. It consists of the following:

1. Service Access Layer –

At this layer, the **IaaS request interface** is responsible for handling the IaaS request. This acts as a conduit between the cloud consumer and the IaaS provider. This is because the IaaS request cannot be made directly to the individual heterogeneous providers. Therefore, it is responsible for providing a set of service interfaces and resource abstractions (e.g. Virtual Machines) obtained from the vicinity to the consumer in a usable form. It is only concerned with the receiving of service requests and provisioning of services. In general, this layer could be defined as the uppermost layer in the IaaS provisioning mechanism.

2. Infrastructure Composition Layer - This layer has the **Infrastructure Composition and Management** module which is the core of the architecture. This module is responsible for composition and management. It constitutes the essential functions of the architecture. This includes discovering resources, forming the physical layer and populating the ad-hoc virtual pool, the formation of the P2P network and managing the resources. As these resources are diverse in nature with different capabilities, a composition algorithm to unify them is proposed. Additionally, due to the heterogeneity, co-ordination between the resources is necessary. Thus, a key based routing mechanism is followed. This modeling approach allows easier resource management and spontaneous IaaS provisioning. Moreover, a composition strategy in IaaS provisioning is essential as the Service layer does not have the logic required for the unification of the disparate resources. This layer creates the composed resources from the ad-hoc virtual resource pool. The generated composition is the only view for the layers above. Each of these modules is explained below with their functionalities.

The **Resource Discovery** module is responsible for an examination of available resources in the vicinity. That is, it follows a publish/subscribe mechanism to search for the IaaS providers. It is the first step towards the deployment of the IaaS composition. The search involves discovering volunteers and populating the ad-hoc virtual resource pool. These volunteers together become part of the volunteer ad-hoc resource pool (VARP). Once discovered, the volunteers offer their intra-device virtual machines.

The **Resource Selection** module optimally selects required virtual resources from the resource pool that is created. These are the VMs of the devices which satisfy the IaaS request. Once selected out of the volunteer pool, these are used as participants in the composition. Once the participants are selected, the **P2P formation** module performs the composition of the selected device VMs. These VMs have an interface and a computing capability similar to the underlying device. This paper only considers compute and storage services. The composed participant topology (CPT) is formed by combining multiple virtual resources from the vicinity that were formerly part of VARP. How the composition of these virtualized resources takes place is elaborated in the next section.

The **Routing and Management** module's role begins once the P2P network is formed. It accesses the storage that has the dependencies specific to a request and integral for the managing of the resources. For example, as seen in the previous section, a meta-file is taken into consideration that acts as a dependency. Once resources are composed, the requests need to be serviced with the assistance of IaaS providers that require co-ordination and management. It takes the decision about the route to take and the devices to be chosen when using the composed service. Hence, the routing and management algorithm makes use of a key that eventually takes the dependencies to the VMs. The IaaS algorithm comprises of this key based routing described later with an example.

The **Resource Monitoring** module's interaction will involve frequent exchanges with the VARP, defined in the algorithm below that will be essential for recognizing failures and reconfiguring the CPT. Additionally, as CPT is a sub-set of VARP, it is also possible to adjust the configuration of the composition by joining new resources in the pool. As this paper is considering very few to no disruptions it is impertinent to delve into the node failures at this juncture.

3. **Resource Abstraction Layer-** This layer contains the mobile phone virtualization [13] components that the cloud IaaS providers use to provide and access the physical resources. It represents a collection of virtual resources collected from the volunteers forming the ad-hoc virtual resource pool. Here, the devices that offer their VMs/VPs have the same characteristics to the respective physical Node IDs in the Ad-hoc Virtual Resource Pool. VM_r represents the reference to the VMs present in the devices. Thus, in general, it could be said that the ad-hoc virtual resource pool is a combination of VARP and CPT. The cloud IaaS providers have control over these abstractions. There could be multiple such abstractions which the cloud IaaS provider can offer. In this way, flexibility in service orchestration is achieved.

4. Physical Resource Layer – Physical resource Layer includes the physical devices obtained from resource discovery. This is the lowermost layer with hardware resources such as phones, tablets, and other physical computing infrastructure elements. These are the entities providing the virtual abstractions for computation. In other words, these are the cloud IaaS providers who own the virtualized resources.

In the next section, we discuss the composition algorithm and define its major functions with a use case.

V. IaaS Algorithms

These algorithms are essential for IaaS deployment in the mobile ad-hoc cloud. It's first stage is *Composition* followed by *Routing & Management*. The advantage of these algorithms is to extend flexibility and simplicity. It achieves these characteristics by orchestrating the discovered devices to satisfy the IaaS request. Additionally, the Routing& Management algorithm ensures co-ordination among the composed resources. Therefore, requests can be submitted at any time, and ad-hoc cloud can be formed on the fly. For *Composition,* once the service request is received, a resourceDiscovery() broadcast is made after which resource information is obtained. The resources obtained any time later than *send(msg, t)* are considered to be evicted. The resource information of Node ID, IP, and port from listener nodes (line 5) is used later to form a P2P network(similar to the joining mechanism in Kademlia), post the selection of resources and session establishment.A bootstrap construct is sent to the provider nodes (line 8-10). Once bootstrapped the VMs/VPs form a P2P network. This is how the resources are composed. Eventually, request-specific dependency retrieval is performed and mapped to VMs where the computation can be processed. This is how *Routing and Management* performed. It makes use of a consistent hashing scheme for generation of the key. Thus, for a given value a corresponding key will take the dependency to the correct VM, where it is downloaded. For one request, consider an example of a meta-file formed with the resource information obtained assimilated along with the job information. This meta-file is the (that is stored in the key-value storage) value for keys generated. (lines 3 to 7). As the keys point to the meta-data values, if there are two similar keys then they'd be pointing to the same location from where the dependency needs to be downloaded. The unique node Ids that are bootstrapped to the consumer device distinguishes between devices.

Once composed the *Routing & Management* module armed with the information from the storage then informs the composed participants in the VM pool where the actual data exists.The individual VMs can then begin downloading the files for execution. Failed nodes can be determined by the resource monitoring module. However, that aspect will be addressed in our future work. One major enabling factor for managing the composition is the composition score given by (1) Where α is the weight given to the systems total time since the last failure ,T_{dept}- Time of departure, T_{arr}- Time of arrival. Qj is defined for each service therefore can have a number of QoS criteria such as delay in delivery, bandwidth, accuracy etc.

$$\text{Composition Score} = \alpha\,(T_{dept} - T_{arr}) + \Sigma_{i\in I}\,(\sum_{j\in J} P_i + (D_i^R * w_{qual}))Q_j \text{ ....(1)}$$

w_{qual} is given to each QoS factor as a pre-defined value. P_i is the Popularity factor of the device who has served 'k' number of requests at time 't'. Let R be resources provided with D_i^R being the device Resources ('R' could be the CPU,RAM, Storage). Logically, the signal strength of a node plays a major role in deciding its popularity factor. S_{str} represents the signal strength. Where str ε Z { as Z ranges from 1 to z , where 'z' is the nodes in the vicinity of a device}.

$$P_i^t = \sum_{str\in Z} S_{str} \;\; * P_i^{t-1} \text{ ......(2)}$$

Due to space limitations we shall not be going into detail about composition score in this manuscript.

Now let's look at a use case to understand the algorithms.

C. Use Case – Offloading Application

To utilize the available resources in the composition, a consumer must submit a request to get the service. In the use case, a dummy task is offloaded, however it could be any

IaaS Algorithm

```
1:Input  msg,T,msg1,host,,x,arrival, receive, key,value,ip,port,id
 //Composition initiation→resourceDiscovery(msg,T) by
   send(msg,t) "broadcast message with task information
   with time"
 ninfo ← nodeInformation()
 sendto(msg1,ip)   > msg1 "Session establishment message"
//Maintain a database of resources :resourceDict[addr[0]]
2:begin Listener (nid,ip)
//once a ready offloaded files are queued and bootstrapping
follows; btI← bootstrapI(id,initiator)
queue.poll () with parameters N, arrival and message   received
update() updating the array resourceDict[]
3:send(msg,t)         >msg " resource information within time t)
//for each meta-file a hashed: initiator.set(key,value) is
followed; function hashMetafile(st,f) is initialized
4: while True:
5:     data,addr←ninfo
6: end while
7: begin Session()
// Routing and Coordination inside →composition is made
initProtocol←InitializeDARC()
8:    Exchange session acknowledgement
// For offloading use-case this protocol is initialized calling
bootstrap();initiator behaves as server <-initiator.bootstrap
[(ip,port)].
9: for all t > T do     >t is the time of nodes arriving earliest, x  is
the late replies.
10:      calc = t+x
11:   { N,arrival,receive}←queue.poll()
12:    updateInitiator()
13:     send(msg,t)    > with acknowledgement message
14:  end for
15: end
16:end
```

S.No.	Parameter	Value
1.	Area	1000x1000 sq.units
2.	Channel capacity	2 Mbps
3.	Transmission Range	250m
4.	MAC Protocol	IEEE 802.11
5.	Packet Size	Upto 100Kb (6 different traces)
6.	Algorithm	Distributed Adhoc Resource Composition(DARC)
7.	Node Speed	0-30m/s

executable application. An option to request IaaS service through the IaaS request interface is realized.We evaluate the performance of the IaaS algorithms with the offloading of an application. The entire code is written in python. Consider a constant IP for a session. Two environments that support python well- Network Emulator for Mobile Universe (NEMU[8]) and Mininet[14] are used for emulation. Two of these environments are considered to evaluate the features of NEMU and compare its realism with Mininet. In NEMU environment, all nodes (VMs) of differing sizes are used. P2P network emulation within an environment height and width of 1000x1000 with small step changes every 5 seconds is used for modeling an ad-hoc network. VM images modeled between 256 MB to 512 MB with 1 CPU, that act as 12 nodes embedded onto NEMU environment.

For this paper, just one AP is considered to avoid the case of congestion in local networks. The procedure starts from the initiator (host/offloader), first discovering the devices and composing the VMs using composition algorithm. The experiment starts by offloading to 2 nodes, then to 4 nodes gradually, offloading is done to 12 nodes to check for performance with an increase in the number of nodes. As increasing by a single node was not producing any visible difference in performance, nodes are increased by a factor of 2. These nodes behave as providers (volunteers/offloadees). The composition of P2P network and its routing & management is done with the IaaS algorithms. The sequence of the function call is shown in Fig 5.We evaluate the composition algorithm after the initiator discovers the VMs through IPs established for a session. We model dummy tasks as small files of 100 kb. A number of iterations were performed only a few are shown for brevity. The graph is normalized to local execution (value 100). The time is measured during the *get*-result request phase because getting back the serviced results would alone ensure completeness of the process.

D. Performance Evaluation

The performance results are shown in figure 7a. Both NEMU(Figure 6a) and Mininet (Figure 6b) show approximately the same realism. We compare our work with [5]. The results show that as the number of nodes goes on increasing for a particular task, the increase in performance

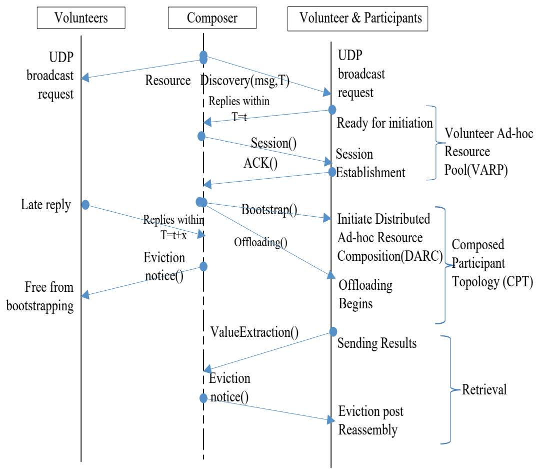
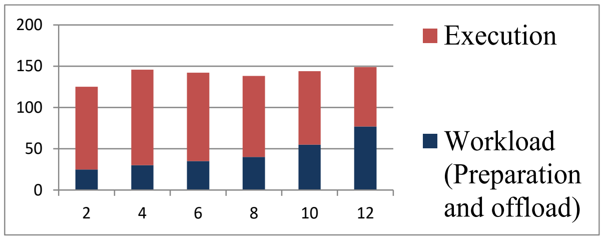


Fig.5. Sequence diagram of the Distributed Ad-hoc Resource-Composition algorithm



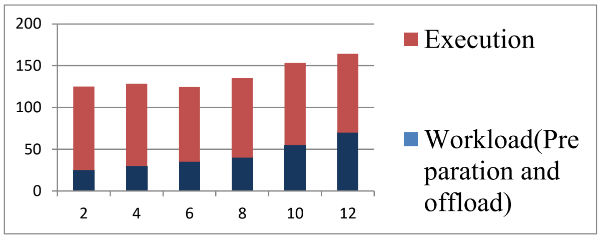


Fig.6a&6b. NEMU&MININET -Local Execution vs Mobile Offloading (normalized to local execution)

stagnates after a point. It suggests that as overheads go on increasing (network overheads, lookups, and creation of P2P network, device overheads etc.) performance attains a stage where there is no increase or decrease but maintains the same level for the same task. However, while offloading to 10 nodes i.e. going from 8 to 10, a change in slope of the graph is observed. This delineates the addition of overheads to the execution of tasks, which in turn causes the deterioration. In Mininet (in Fig 6b.) we model an environment like [5]. The number of nodes is gradually increased same as before where dummy tasks are offloaded. Based on the work done by Canepa et al. [5] it can be learned that for small files using servers like Hadoop would degrade performance.

It also provides an insight to the resource usage. Pooling of resources, when not needed, results in performance degradation. Thus, resource usage should be based on the task's need. Assuming the preparation and offloading time in [5] together as the workload offloading time in our case, we observed a better performance in our system. The Hadoop server performance in [5] could be mapped to the degradation

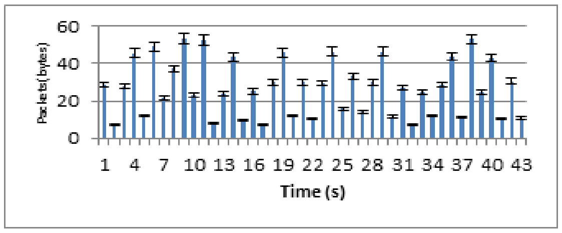
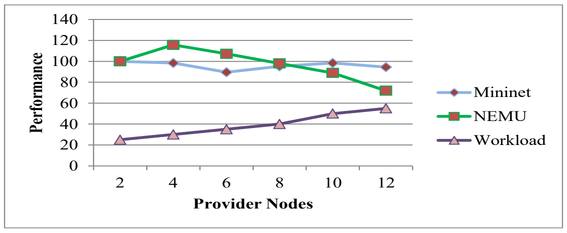


Fig.6c. Packet drop at the Consumer



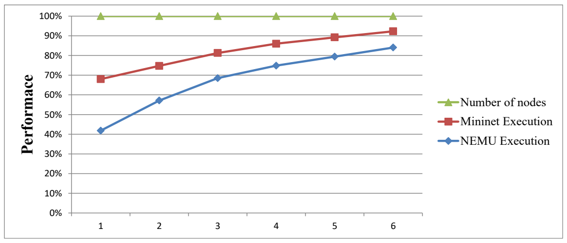


Fig.7a&7b- Performance of Mininet and Nemu providers & Saturation after 5 nodes

in performance observed after 10 nodes. In an ad-hoc environment, extra resources could be used by some other customers, also blocking more than the requisite resources means wastage of resources. There were failures while offloading that caused over 20% packet drops (Fig 6.c). For one, during **Offloader's device error** the initiating host starts the process of offloading and does not *set ("key", value)* a value (meta-file). That is the provider bootstraps or the provider peer which is ready for processing the tasks keeps waiting but doesn't find a value. Secondly, when **Device shuts down post offloading** - In this scenario, the providers who are bootstrapped in the P2P network but do not respond once the host device has stopped. Lastly, during **Late Replies** the provider could be part of another consumer's ad-hoc cloud. However, once offloading begins then there will be another look up initiated for processing the task, if at all the host device is not able to process the sub-task locally. Figure 7b shows how the saturation occurs for one task. This is the case when one task that does not require 6 nodes, is amassing resources which eventually leads to saturation of performance. As observed in 7a this saturation will lead to degradation over time.

VI. CONCLUSION AND FUTURE WORK

In this paper, we developed a mobile ad-hoc cloud computing architecture that deploys an IaaS request based on an end user's application in a crowd-sourced environment. The idle-intra device resources of nearby users can be benefitting if put together in a collaborative manner. This has been realized in this work. The primary aim to tap the resources and assemble it in a usable form for a consumer has been managed and successfully shown with the IaaS algorithms. There are certain drawbacks in the system that affected the offloading process. Firstly, the mobility between the devices needs to be managed i.e. devices entering and leaving the topology at will. Secondly, for every device making a broadcast, the devices that are available are the ones connected to the same AP in the same region, there is no knowledge of the location of resources in the other APs. Thirdly, as seen in Case 3, resource discovery needs to be done taking QoS metrics into consideration. The conflict of one provider becoming a consumer at the same time is not illustrated in this paper. Although, most of these drawbacks can be easily overcome by re-broadcasting a request at the consumer side, we plan to develop a more efficient solution for mobility along with developing the scheduler algorithms.

REFERENCES

[1] O. Babaoglu and M. Marzolla, "Peer-to-Peer Cloud Computing," pp. 1–9, 2011.

[2] mobiforge.global-mobile-statistics-2013-section-e-mobile-apps-app-stores-pricing-and-failure-rates, 2013

[3] G. a. McGilvary, A. Barker, and M. Atkinson, "Ad Hoc Cloud Computing," *2015 IEEE 8th Int. Conf. Cloud Comput.*, pp. 1063–1068, 2015.

[4] A. Chandra and J. Weissman, "Nebulas : Using Distributed Voluntary Resources to Build Clouds," *Proc. 2009 Conf. Hot Top. cloud Comput.*, vol. San Diego, no. San Diego, CA, June 2009., p. 2, 2009.

[5] G. Huerta-Canepa and D. Lee, "A virtual cloud computing provider for mobile devices," *Proc. 1st ACM Work. Mob. Cloud Comput. Serv. Soc. Networks Beyond MCS 10*, vol. 16, no. 2010, p.

[6] S. A. Abid, M. Othman, and N. Shah, "A Survey on DHT-Based Routing for Large-Scale Mobile Ad Hoc Networks," *ACM Comput. Surv.*, vol. 47, no. 2, pp. 1–46, 2014.

[7] R. Lacuesta, J. Lloret, S. Sendra, and L. Peñalver, "Spontaneous ad hoc mobile cloud computing network," *Sci. World J.*, vol. 2014, 2014.

[8] V. Autefage, D. Magoni, and J. Murphy, "Virtualization Toolset for Emulating Mobile Devices and Networks," *IEEE/ACM Int. MobileSoft*, 2016.

[9] M. Al-Rousan, E. Al-Shara, and Y. Jararweh, "AMCC: Ad-hoc based mobile cloud computing modeling," *Procedia Comput. Sci.*, vol. 56, no. 1, pp. 580–585, 2015.

[10] D. Maymounkov, Petar ; Maziéres, "Kademlia: A Peer-to-peer Information System Based on the XOR Metric," *Proceeding IPTPS '01 Revis. Pap. from First Int. Work. Peer-to-Peer Syst.*, pp. 53–65, Nov. 2002.

[11] B. Zaghdoudi, H. K. Ben Ayed, and I. Riabi, "Ad hoc cloud as a service: A protocol for setting up an ad hoc cloud over MANETs," *Procedia Comput. Sci.*, vol. 56, no. 1, pp. 573–579, 2015.

[12] U. Drolia, N. Mickulicz, R. Gandhi, and P. Narasimhan, "Krowd: A Key-Value Store for Crowded Venues," in *Proceedings of the 10th International Workshop on Mobility in the Evolving Internet Architecture*, 2015, pp. 20–25.

[13] J. Andrus, C. Dall, A. V. Hof, O. Laadan, and J. Nieh, "Cells- A Virtual Mobile Smartphone Architecture," Proc. Twenty-Third ACM Symp. Oper. Syst. Princ. - SOSP '11, p. 173, 2011.

[14] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," Work. Hot Top. Networks, pp. 1–6, 2010

[15] Wave your Phone in the Air: How Technology is Changing Live Music-Craig Rosen Online[Available] yahoo.com/tech/the-lights-go-down-your-pulse-races-in-194316495.html, 2015

[16] D. M. Shila, W. Shen, and Y. Cheng Online [Available], "AMCloud : Towards a Secure Autonomic Mobile Ad Hoc Cloud Computing System.",2016