E-Mobile: A heterogeneous agent-based, remote off-loading framework for efficient energy management of mobile devices.

Rohit Koul and Raja Bala
Department of Computer Sciences,
University of Wisconsin Madison, Madison, WI, USA
{rkoul,bala}@wisc.edu

ABSTRACT
Web applications are becoming increasingly popular on mobile/wireless devices. As more desktop applications that are inherently computational intensive and resource-demanding are being ported to the mobile platforms, the energy consumption of the devices is shooting up, thereby decreasing the battery life. An alternate approach is to off-load the resource intensive computations to remote server(s) and simply work off the results returned. However such solutions currently existing in the market do not handle multi-media applications like video-streaming very well often causing sub-optimal user experience. This paper proposes a framework for university and office settings that works transparently with the existing mobile clients and attempts to provide an energy efficient and fairly rich user experience by offloading computationally intensive tasks to desktops in their vicinity.

General Terms Design, Energy, Architecture

Keywords Thin client computing, pervasive web, mobile devices, resource-constrained devices, off-load, remote display, video

1. INTRODUCTION
The past several years have witnessed a significant increase in the usage of mobile platforms for connectivity and entertainment on the move. The growth of social media has further fueled this revolution with multimedia capabilities being the norm now-a-days. Users are increasingly using their mobile devices for browsing, gaming and rendering multi-media applications. However, due to form factor constraints, required portability and battery lifetime, mobile devices have limited resources.
Despite the rapid development of smaller and more efficient hardware, software requirements on CPU, memory and disk space cause significant hardware needs and battery drains. Numerous technologies have been proposed to prolong the battery life on mobile devices notably of which include dimming the display when not in use, optimizing the I/O devices and even slowing down the CPU. While all these technologies focus on the hardware part, computation offloading method which uses the wireless communication capability of mobile devices is a software method to deal with the aforementioned problem.
The idea presented in this proposal is to extend the mature thin client technology ([1],[2],[6],[7]) for use on not-so-thin mobile devices, where the heavy computations are offloaded to a distant server(s) over a wireless network and the applications need little or no alteration. This way the computation cost of mobile devices decreases and thereby the energy consumption. Our primary goal is to use the idle computation power of the common desktop PCs instead of dedicated servers. We have also used OpenFlow [14] which enables researchers to run experimental protocols in a network, to design and evaluate our architecture.
Our focus is primarily on the multimedia technologies like live video streaming, viewing PDFs, maps and multimedia gaming**.**
The paper is organized as follows:
The underlying background that has influenced our design is presented in Section 2 whereas Section 3 presents the system architecture of our proposal. In Section 4, we describe where we are with respect to the implementation of our ideas. Section 5 describes work that we intend to do in future whereas Section 6 enumerates related hardware approaches for energy optimization in mobile devices.

2. BACKGROUND
This section focuses initially on some of the offloading techniques for mobile devices that have influenced the design of our framework. We then present a short discussion on why offloading video is a very important piece in our architecture.

2.1 CURRENT LITERATURE

We have come across some stark similarities and fundamental differences in various approaches that have been proposed in the past.

Almost all the application level offloading frameworks make use of *remote display updates* from the server in response to user input on the mobile client ([1],[2],[3],[4] and [6]). Most remote display systems process the user input (sent from the client side), perform the display processing, encode the pixel data in the *framebuffer* and send it back to the client. Fig(1) represents this scheme.

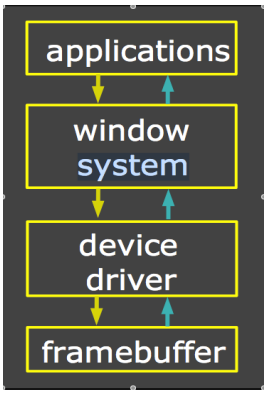
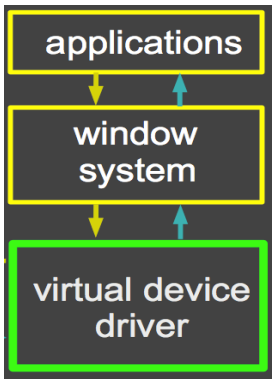


Fig (1)

THIN-C [1] innovates in this space and provides much better quality for multimedia applications when compared to other remote display systems like VNC (Virtual Network Computing), ICA (Independent Computing Architecture by Citrix) and the X-Window System. It accomplishes this by using a simple virtual display driver that intercepts drawing commands, packetizes and then sends them over the network. Fig(2) shows the resulting scheme.



Fig(2)

But THIN-C and its subsequent work ([1],[2] and [6]) assume that we have thin 'dumb' clients, which rely on remote servers for their applications as well. This, however, is not true in the present-day mobile devices market, wherein every device comes with a bunch of softwares for multimedia playback, internet browsing, etc. Clearly, current mobile devices, be it laptops, PDAs or mobile phones, aren't thin clients.

[3] takes this into account and suggests tweaking the Firefox mobile browser to offload multimedia, PDF and Flash content requests to a remote server. They accomplish this by using a plug-in on both the client and remote server side. This removes any video format based plug-in dependency on the client side, which is very useful for mobile browsers in particular. Essentially, the server performs the energy-intensive video, PDF or flash decoding computation and sends remote display updates to the client. The client has thus been relieved of a lot of the work and only needs to process these display updates, which are mapped to the rectangular space in the browser (done by the plug-in).

The authors present impressive statistics in favor of offloading PDF and Flash computation. When Flash animation (similar for PDF) is played locally, the browser consumes around 40% client CPU resources. When the remote plug-in is used, the average client CPU utilization is reduced to 15.4% in LAN and 4.5% in wireless (802.11g/b) settings. The CPU utilization after offloading is mainly a result of the amount of display update received and parsed. In a LAN setting, more display updates are sent to the client and hence the higher utilization and quality. When a video file is played locally, it consumes very less CPU as mobile devices have special chips for video encoding/decoding, thus not requiring the CPU to perform these operations. When the remote plug-in is used for video playback, 20.4% CPU utilization is seen. The quality of dynamic media (Flash and video) is measured by the amount of display update received and at values of 41MB for Flash and 102MB for video, it doesn't seem a scalable solution.

We argue that CPU utilization is not a clear indicator of energy used as local video processing in mobile devices uses specialized DSPs or SoCs which are optimized for power, speed and die area. Thus, there is a need for energy measurements of local video playback versus that of processing the numerous display updates to decide if this is a viable method. The communication costs must also be considered.

[5] focuses in offloading certain functions of the encoding pipeline for H.264 compressed videos.

They take into account the energy cost if encoded locally and the communication costs if performed remotely. Their findings show that there is a huge incentive in actually sending the un-encoded data (much bigger size adds in transmission side costs) from the client to a server to perform the encoding, and receiving the encoded data back (receiver side costs). The figures seem exaggerated though, as it claims savings of at least 65%. Note that there isn't any need for remote display updates here, as we require the complete encoded data in the end.

[9] studies the popular video compression codecs for playback in mobile devices from an energy perspective. The authors find that H.263+ and DivX consume the least energy when compared to MPEG4 and WMV. They also analyze the effects of bit rate, frame rate and frame size (resolution) while encoding videos. The authors find that for better picture quality, encoding videos with a higher bit rate results in just a little increase in energy consumption. Increasing the resolution in need of quality has the highest increase in energy consumed, while frame rate falls in between.
This study allows us to think in the direction of optimizing the video content for a particular energy profile of the mobile device before sending it and thus adds a new dimension to the tasks that can be performed remotely.

[7] suggests that there are essentially two basic approaches to handle multimedia data in resource-constrained devices: proxy-based or end-to-end solution. The architecture we propose is a hybrid of both.

2.2 The focus on VIDEO traffic

Video presents a big challenge in offloading, unlike static media like PDF. Studies [16] show that mobile data traffic will double every year through 2013, with around 64% of the world's mobile data traffic slated to be video by 2013. Thus, even a small energy-level optimization in either hardware or software will have a big effect. The need for power efficient and fast video processing which could not be met in the traditional ARM (Advanced RISC Machine) processors in mobile phones led to specialized SoC (Systems on Chip) and DSPs (Digital Signal Processors) being used for video encoding and decoding. They exploit the inherent parallelism of video, which at a low level is nothing but a bunch of frames. Existing compression techniques save tremendous amounts of bandwidth by exploiting the innate redundancy in video, with ratios approaching 200:1.
Also, video encoding is more complex than decoding and includes most of the decoder. Hence, offloading the encoding computation will have higher energy benefits.

Any hardware improvement in the energy consumption in video processing assumes that the video is being played locally. Software offloading looks at this problem from the opposite perspective. Regardless of hardware advancements, we attempt to offload this computation to a resource that is not energy constrained. The tradeoff in energy spent in communication for offloading versus computation for local playback hasn't been comprehensively studied to the best of our knowledge.

Statistics show that approximately 75% of online video is viewed using Adobe's Flash technology [13]. In the context of the web, Flash programs are active objects that are embedded within web pages. An IDE compiles a Flash program into a *.swf* formatted file which contains the media objects, user interface elements and the program *bytecode.* A Flash virtual machine, generally the Flash player plugin in web browsers, parses and executes the *.swf* file.
[4] uses a proxy to splice active content [Flash] out of web pages and replace it with an AJAX-based remote display component. Some of the latest mobile devices (phones) still do not have full support for Flash, like Windows Mobile which only supports FlashLite. The iPhone doesn't have any native Flash support yet. [4] allows such devices to watch Flash content as it is delivered through a proxy and also prevents exploitation of security flaws that have plagued Flash.

[12] suggests that for relatively small picture sizes, such as QCIF (176×144) videos, video encoding consumes about 2/3 of the total power for video communication over Wireless LAN. Hence, there is a high probability that communication costs (energy) incurred for offloading will not exceed the local computational cost.
Thus, we hope that the remote display update mechanism for video decoding (of downstream data) is a viable option for any kind of video, be it hardware accelerated or not.

3. ARCHITECTURE

In Figure 3, we present a high level, conceptual illustration of our proposed E-Mobile architecture.

Our architecture, at its core, is an agent-based model and uses the computation power of common desktop PCs (as opposed to a dedicated set of machines) for auto-offloading the tasks. We argue that an architecture that takes into account the energy state (battery level) of a mobile device and adapts its responses to it, coupled with various optimizations to the already existing and proven mechanisms, would give a higher throughput per joule of energy spent by the mobile device. Drawing from our literature survey and understanding of the display systems in mobile devices, we also argue that our model is extremely useful in the case of video streaming and maps.

The setup essentially consists of the following components:-

Client: The mobile client (such as a laptop/cell phone/PDA), a device that has 802.11a/b/n/g capabilities.

Server: The physical machine that hosts the content being requested by the mobile device (hence forth also referred as simply 'client').

Desktop pool: A set of generic machines in a LAN (say a university or a giant corporate office setup) which are used by other users on a daily basis. As per the statistics we got from a large university's (University of Wisconsin Madison) distributed, high performance computing team, we argue that this pool has sufficiently large amount of idle resources (CPU power, disk space and memory) at any given time, which could be utilized by our architecture for its computation tasks without any substantial perceivable performance impacts on the users working on these machines.

OpenFlow: Since OpenFlow is based on an Ethernet switch, with an internal flow-table, and a standardized interface to add and remove flow entries, we envision OpenFlow to be a key component of our architecture, especially for selecting the best machine amongst the desktop pool and during migration of the computation between machines. We, hence are of the opinion that adding OpenFlow as a feature to existing switches and routers, or using an OpenFlow enabled switch is essential for better energy management of the mobile clients We describe the OpenFlow usage in detail in section 3.1

NOX: NOX[15] is an open-source controller for OpenFlow. It provides a simplified platform for writing network control software in C++ or Python**.** The OpenFlow enabled switches are logically connected to NOX enabling it to establish flows in the switches.

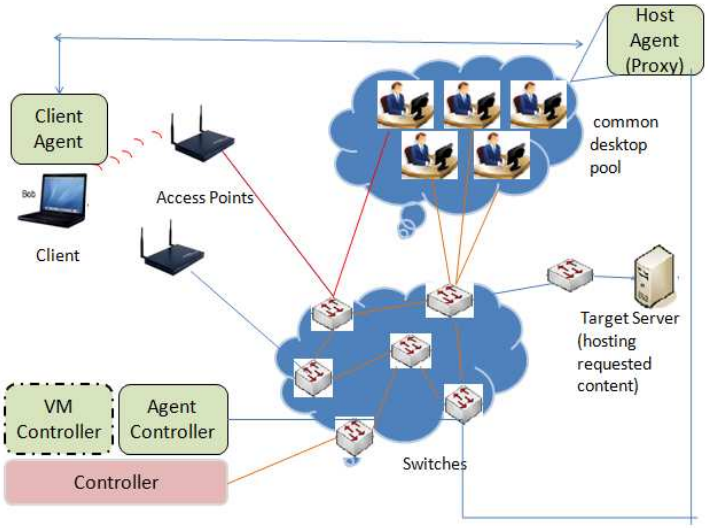


Fig (3): High level E-Mobile architecture.

The openflow controller facilitates selection of an appropriate desktop for offloading the computation.

Host Agents: A Host Agent is essentially a multi-threaded application that is run on every participating machine in the desktop pool. Host Agents act as a light weight proxy and are responsible for performing the computations on behalf of the mobile client. The interactions of Host Agents with other components in our architecture is described in sections 3.1, 3.3 and 3.4

Client Agents: A Client Agent is an entity that runs on the mobile client and communicates with the host agents. It could either be implemented in the form of a web-browser plug-in or a packet-intercepting application altogether. The role of client agents is described in detail in section 3.3

Agent Controller: Agent Controller is an application running on the same physical machine as the OpenFlow controller and communicates with its modules using IPC mechanisms. Agent Controller is responsible for fetching host resource statistics from the host agents and passing it to the controller, thereby facilitating the best proxy

selection. Agent Controller also helps in content migration and flow expiry. Section 3.1 outlines these functions in detail.

Migration Controller: Migration Controller is a NOX module running alongside the controller. It used IPC mechanisms to talk to the Agent Controller. It also facilitates the content migration between hosts in case of network/host overloads, latency and proximity changes and system crashes. We describe the functionalities is a little more detail in section 3.4

3.1 Agent Controller and Host Agents' Interactions.

Agent Controller can interact with the Host Agents to fetch the host statistics (CPU load, memory availability, disk space). Agent Controller uses TCP sockets to communicate with the Agents and fetch the host statistics.

In our model, all hosts were running Linux and hence we used system command like *top*, *free* and *uptime* to gather the host statistics periodically. These commands will work for other *nix based systems. In a Windows based environment one could imagine similar kinds of command line tools to be used. Also, a simple C program using the *glibtop* library could essentially be used in such systems.

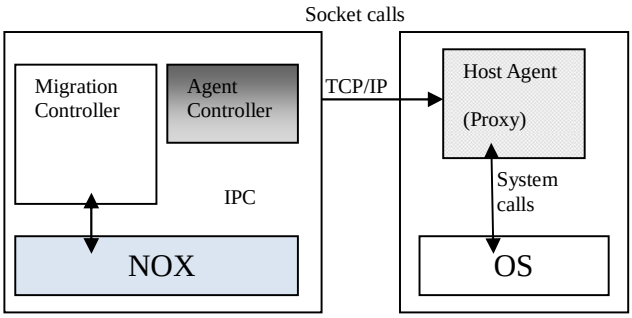


Fig (4) Agent Controller and Host Agent Interactions.

One question that arises is that as to how does the Agent Controller know where the Host Agents are? In our model we pre-configured the Agent Controller with the addresses of the Host Agents since there could be desktops in the pool choosing not to run an agent. However, one important thing to note here is that if it is imperative that any desktop in the pool, runs a Host Agent and is essentially a potential candidate for off-loading computation, one need not pre-configure Agent Controller with the addresses of the Host Agents. OpenFlow framework allows the controller to get to know when a host joined or left the network. Hence one could leverage the controller functionality to get these addresses and communicate with the Host Agents.

One important design decision that we faced was the choice of the frequency of these host statistics updates. We initially started with Host Agents sending these statistics at a pre-defined frequency (say 10s) to the Agent Controller. However this quickly flooded the network especially in the wake of a large number of such hosts. Hence we discarded this approach. Another approach that logically followed was to let the Agent Controller poll these host agents at a fixed interval. This allowed the flexibility of pick-and-choose amongst the agents depending on the already existing load. However this approach suffered from the potential problem of having to work off stale data. There could be a case wherein while examining the best possible desktop for offloading content, the controller could potentially use statistics from the last gathered update – which could have significantly changed since then for a particular host and hence could cause the establishment of sub-optimal routes to overly loaded servers. In the end, we decided to poll for the statistics from Agent Controller, but only when a new flow is to be established. We made provisions for establishing a flow with an infinite expiry time (i.e a dedicated route) until the host agent itself explicitly asks for expiring the route to it. The Host agent could do this if it finds the host getting severely loaded. This reduced the number of update packets in the network and gave more control to the Host Agents.

Apart from the normal _stat packets that are sent by the Host Agents to the Agent Controller (in response to a STAT command), we have also defined three another types of commands involving Host Agents 1) PRUNE with appropriate parameters is sent to Agent Controller when the Host Agent wants a flow to be expired. 2) MIGRATE is sent from the Agent Controller to a Host Controller asking it to offload its contents to a new host during migration (in case of using a dictionary based approach) 3) COPY is sent from one Host Agent to another when it detects a migration scenario. Migration is discussed in detail in section 3.4

3.2 Interactions between Agent Controller and OpenFlow Controller (NOX).

When a switch (say S0) gets a packet from a mobile client (say C0), it checks whether it has a route for the packets coming from the interface to which the client is connected (say I0). In case the switch has no such entry in its forwarding table, it asks the OpenFlow controller (NOX). NOX sends a signal to the Agent Controller to fetch host level statistics from the Host Agents it knows of. NOX simultaneously uses its *switchstats* module to gather the switch statistics connecting S0 to all other switches to which the desktop PCs are connected to. Using these pieces of information NOX is able to put together a selection profile for each of the desktop PCs running host agents (i.e based on the network, host loads and proximity of the desktop PCs). It then selects a desktop PC with the best selection profile (least load) and installs a flow with an infinite expiry time in S0 to route all incoming packets from C0 on I0 to the Interface to which the next hop for the selected machine is connected. The flows are established in the intermediately switches as well. So how long do the packets from the client C0 go to the chosen machine? Once an end-to-end flow is established, all packets from C0 are routed to the selected machine, thereby establishing a dedicated route (until an explicit trigger is generated to severe the route as explained in the previous section.)

3.3 Interactions between Client Agent and Host Agent.

In the model that we propose, we are of the opinion that there needs to be an agent at the client side as well that could let the framework know its energy constraints**.** We argue, If the mobile device had a way of telling the offloading desktop "My battery is low, I need you to format the videos/images I ask for in a way best suited for my energy profile", it could then get energy-aware data responses. We also hold that or model could use the idea of energy profiling along with optimized versions of some other techniques to get a better performance.

One such technique is the use of packet caches ([10],[11]) at the Host agent. [11] has already established that by pushing redundancy elimination capacity to the end hosts, one could obtain most of the middleboxs' (host agents' in our case) bandwidth savings. We also argue that one could use the same technique for reducing the inbound video streaming traffic at the mobile client end. This would require the Host Agents to implement a finger-printing dictionary and push it to the mobile client to which it serves. When a new video stream arrives, the host agent just needs to send the fingerprint downstream and the client agent retrieves the actual payload from its local cache. This will lower the inbound traffic and hence save considerable energy due to lesser usage of the wireless receiver power. We also hold that if the payload is stored already in a decoded format, the client could save a significant amount of energy saved during the decoding phase by simply re-using the streams from the payload store.

Our model is highly adaptive. We, therefore reason that by default the clients do not have a packet cache turned on instead work via remote-display mechanism. It is the Host Agents that make energy aware decisions on the basis of the client.

We also introduce the concept of tolerance at the client level. The rationale is that one can trade data quality against resource consumption. Tolerance limits define the degree to which the client is ready to accept the degradation in data. E.g An iPhone streaming video application could configure a tolerance limit of 20% essentially meaning that the Host Agent can serve content with 20% decrease in resolution. The tolerance limits can be configured per application level.

The client Agent also lets the Host Agent know its current battery levels at specific intervals. There are APIs available to get this data. The Host Agent uses this information about the current battery level of a client and the tolerance limits to make energy aware decisions for the client. E.g Host Agent may decide to send an image with lossy compression, or reduce the size of an image/video depending on these parameters.

One can argue that obtaining the battery levels for certain closed-source mobile platforms is not possible or the energy used in these calls needs to be taken into account. We argue that these calls are rather infrequent and are not computation intensive. Also, if the need may arise, we could leverage the power of OpenFlow to predict battery changes.

e.g assuming the Client Agent just sends current battery level only when it is disconnected from a charging source and then sends infrequent half-hourly / hourly updates, we argue that OpenFlow allows us to monitor the network traffic.Depending on the incoming and the outgoing traffic to the client and standard battery discharge models, we can predict with significant accuracy the battery level of a client at a given point in time and make decisions accordingly. However, for the sake of

simplicity we may ignore this prediction model for now and work off the values sent by the Client Agent.
The Client Agent could be implemented as a separate application altogether or a browser plug-in (in case of images) For different video players and formats, we envision that the Agent essentially comprises of a series of modules some of which are specific to a player and video format in question.

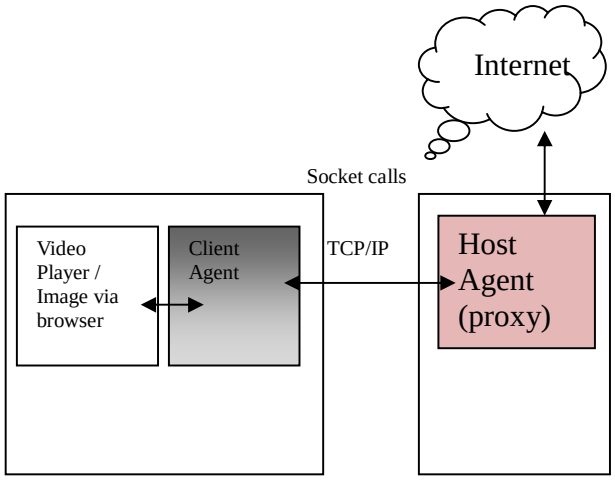
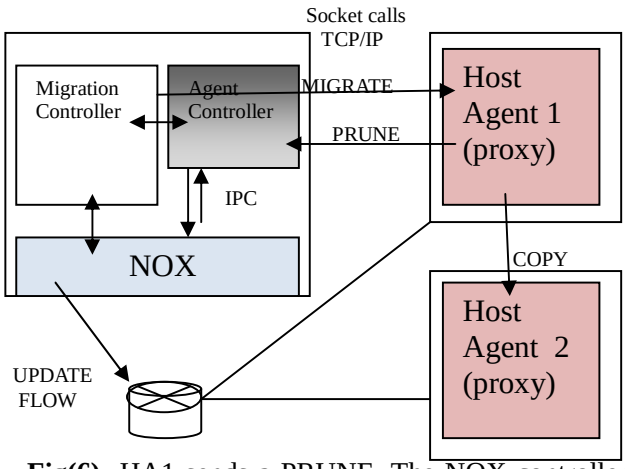


Fig (5) Client Agent and Host Agent Interactions. Host Agent acts as a light weight proxy

3.4 Interactions between Migration Controller and Host Agents

The Migration Controller mechanism was first highlighted in a SIGCOMM OpenFlow dem [18].
In this mechanism, the VM client/ server bindings are generated using a wsdl. Zolera SOAP Infrastructure (ZSI) modules are used in conjunction with the NOX controller to update the VM bindings to a host when the VM is to be migrated. The demo makes use of VMWare's VM migration features. The VMs are hosted on ESX server machines which are under the control of a Virtual Center server. The MobileVMs code communicates with Virtual Center, when MobileVMs wants a VM to move it sends a move request to Virtual Center via a SOAP call. VMWare's mechanism involves a bulk transfer of the VM, followed by a series of delta to reflect the state changes that have occurred since the bulk transfer began, finishing in a switchover between the two instances. We argue that a similar kind of infrastructure could be leveraged in our case as well. The marked difference is that we neither have a separate VM virtual control server nor we envision dedicated VMs being used for off-loading (though such a scenario is also viable). In our case, migration can be triggered as follows:
If the host is down, the controller figures it out owing to *host_leave / datapath_leave* events. In this case migration involves updating the flow table entries at real time. Since the host is down, no dictionary based content can be copied over to the migrated host and the computation gains are lost.
The Host Agents can also trigger a migrate event when the statistics indicate that the host is being over-loaded. In this case the Host Agent sends a PRUNE command to the Agent Controller which then passes the appropriate parameters to the Migration Controller via IPC mechanisms. PRUNE causes updating the flow tables with the interface of the migrated host and a MIGRATE command with the identity of the new host is sent to the initial Host Agent as a response to this PRUNE In case a dictionary based redundancy elimination method is being used, migration will also involve copying the dictionary over from the original host to the migrated host. We argue that the Host agent is the best place to bundle this functionality. Hence a MIGRATE from Agent Controller causes the Host Agent to start the dictionary copy over in parallel (via a COPY command). One thing to be noted here is that this requires the host agent (HA1) of the initial server to communicate to the Host Agent (HA2) of the migrated server. For the sake of simplicity we can assume that all the Host Agents listen on a fixed port (say 1337) and hence HA1 just requires the IP address of the host running HA2 to establish a connection and send the data. If we do not mandate this requirement, all the host agents need to know the existence of one another in some other way. Fig (6) shows the mechanism:



Fig(6) HA1 sends a PRUNE. The NOX controller UPDATES flow entries, Migration Controller sends a MIGRATE back. HA1 COPIES the data to HA2.

3.4 Example Scenario: Putting it all together

A mobile device (client) requests some web page. If it is the first time the controller hears about this device (in the last 'X" minutes or so), it finds out the best desktop (in its pool) to take care of its needs based on the factors of i) desktop usage & ii) network congestion to reach that particular desktop. Once it has found the best match, flow table entries are added by NOX to all the OF switches in the path between the mobile device and the desktop.

Essentially, this desktop acts as a light weight proxy (no caching of content). If the request is a normal web page (no multimedia content), it just forwards the request and sends back the reply as any other proxy would do. If the request consists of multimedia objects/pdf/flash , the desktop decodes the response data and sends it back as display updates to the mobile device [whether this is a viable method for videos is questionable, but it is a start nevertheless]. Thus, there is some tweaking in the mobile device's browser that is necessary to accommodate this which we implement as a client side Agent which sends battery statistics and application tolerance levels to the Host Agents. We think this would be particularly useful in applications like Maps and Videos.

The Host Agent also keeps track of the replies that it receives (for various requests by the mobile device) and builds up a small dictionary based on the redundancy profile of the received data. If there is a good amount of redundancy, it suggests to the mobile device that it can save on wireless receiver power. Essentially, the Host Agent sends this dictionary (containing the encoded fingerprints and the payload it points to) to the mobile device. Further responses from the desktop now use these fingerprints (in a shim layer) to represent redundant data. The mobile device will now expand those regions based on lookups in the dictionary. So, is this computation done by the mobile device worth the receiver power saved? We have no idea presently. We think that looking to save on the transmission side makes lesser sense, as a mobile user, you tend to transmit less and receive more data while browsing.

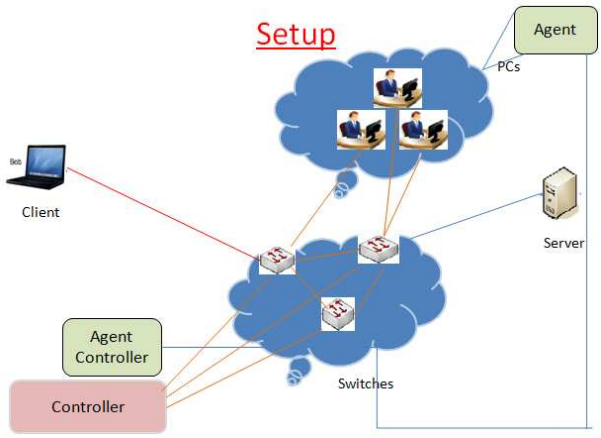
Now, migration comes to the picture assuming the dictionary scheme is useful. Suppose the desktop that we've offloaded the mobile requests to is no longer able to cater to its needs (proximity, disk space, computation crunches etc) we would need a new desktop with the existing dictionary intact, thus using migration.

3.5 Scalability

With every agent-based distributed model, one concern that network researchers generally have is whether the model is scalable enough. Our contention is that use of OpenFlow inherently makes our model scalable. We could also use Flowvisor[19] for connecting multiple NOX controllers across network slices. Also, one extension could be to designate certain desktop hosts as intermediate components which would collect statistics from the underlying hosts and respond to the Agent Controller when queried, essentially creating a hierarchy of Host Agents.

4. CURRENT IMPLEMENTATION

Owing to a complex architecture with lots of moving pieces, we choose to focus on implementing in steps, essentially going from simple to complex, building the end-to-end model in a systematic manner. As of now, we have implemented the selection of off-loading server (Host Agent) using OpenFlow. Our setup is depicted in Figure



Fig(7) Implementation of host selection..

We essentially created a topology consisting of 3 switches and 3 desktop hosts. The client used was a laptop. So far we have only incorporated the Agent Controller and the Host Agents, focusing on selecting the least congested host. Although we used a VM based setup for testing, in the long run we need to have actual OF switched, wireless clients and WAP's.

One thing to note here is the allocation of IP addresses. Since this topology would most likely exist in conjunction with the already existing

campus production network, we are most likely to be allocated a block of IP addresses in a subnet for mobile clients. Since we cannot have multiple DHCP servers in a single broadcast domain, we need to somehow distinguish a mobile client which is part of the setup from other clients. One can use OpenFlow for this distinction. Using OpenFlow we can identify all DHCP requests that are from one of our wireless APs. If not, then the request is broadcast as usual and will be handled by the campus wide DHCP server. Otherwise, we could check the client against a list of MACs registered with us. If the client is registered, the request can be forwarded to our own local DHCP server and an IP is allocated. Replies are unicast to the client (if possible).

In such a deployment, we could run a DHCP server on the same physical server as the NOX controller.

5. FUTURE WORK

Our present model is purely theoretical with little or no statistical evidence of its feasibility. Our future work will mainly focus on implementing and rigorous testing all possible aspects of the architecture.

Also, our current implementation uses wired OpenFlow devices with a NOX controller. Deploying this successfully in a university-wide setting requires the use of OpenRoads, an enhancement of NOX. OpenRoads provides better support for alternate handover mechanisms, such as bi-casting to two APs simultaneously, and provides a mechanism to ascertain information not normally provided via OpenFlow, such as wireless channel strength.

6. RELATED WORK

Section 2 focused purely on the software approaches of saving energy in mobile devices.

"Hardware methods to reduce energy consumption in mobile devices" has been a hot topic for more than a decade. Video coding is usually the most power-hungry application a mobile processor has to run. Therefore, minimizing the maximum power consumption figure of a chip often concentrates on finding the most power-efficient way to implement video processing algorithms.

Some of the techniques used in wireless mobile devices include Dynamic Voltage Scaling (DVS) and Frequency Scaling, wherein the incoming video data always has a jitter component [12]. Since the frames do not come at a constant rate, the voltage or frequency is reduced when there is a delay in receiving the next frame. In conventional system design with fixed supply voltage and clock frequency, clock cycles, and hence energy, are wasted when the CPU workload is light and the processor becomes idle. Reducing the supply voltage in conjunction with the clock frequency eliminates the idle cycles and saves the energy significantly. As power has a square dependency on voltage, i.e. $P = f (V^2 , f)$, this gives a huge advantage over designs that require a higher clock-frequency [17].

7. CONCLUSION

We have proposed an architecture that is suited to university and corporate office settings, wherein desktops in the vicinity of a mobile device can be used to perform energy-intensive tasks like decoding PDF, Flash and video, thus allowing for energy-savings in the mobile device.

8. REFERENCES.

[1] R. Baratto, J. Nieh, and L. Kim. THINC: A Remote Display Architecture for Thin-Client Computing. Technical Report CUCS-027-04, Department of Computer Science,Columbia University, July 2004.

[2] J. Kim, R. A. Baratto, and J. Nieh,"pTHINC: A Thin-Client Architecture forMobile Wireless Web", In Proceedings of the 15th International World WideWeb Conference (WWW2006), Edinburgh, Scotland,UK, May 2006.

[3] Yang Zhang, Xue-tao Guan, Tao Huang andXu Cheng, "A Heterogeneous Auto-Offloading Framework Based on Web Browser for Resource-constrained Devices", Fourth International Conference on Internet and Web Applications and Services, 2009

[4] A. Moshchuk, S. D. Gribble, and H. M.Levy, "Flash-Proxy: Transparently EnablingRich Web Content via Remote Execution",Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services(MobiSys), pp. 81-93, 2008.

[5] Xiaoli Zhao, Pin Tao, Shiqiang Yang and Fei Kong, "Computation Offloading for H.264Video Encoder on Mobile Devices", IMACS Multiconference on "Computational Engineeringin Systems Applications" (CESA), October 4-6, 2006, Beijing, China

[6] A. Baratto, S. Potter, J. Nieh. MobiDesk: Mobile virtual desktop computing, In *Proceedings of the 10th Annual Conference on Mobile Computing and Networking*, (Philadelphia, PA,Sept/Oct, 2004), pp. 1-16

[7] C. E. Perkins, "Handling Multimedia Data for Mobile

Computers" Proceedings of the 20th International Computer Software and Applications Conference (COMPSAC), pp. 147-148, 1996

[8] Xiaohui Gu, Klara Nahrstedt, Alan Messer, Ira Greenberg and Dejan Milojicic, " AdaptiveOffloading for Pervasive Computing Vol.3, No.3, IEEE Pervasive Computing Magazine, July 2004.

[9] Chu-Hsing Lin, Jung-Chun Liu, Chun-Wei Liao, *"Energy Analysis of Multimedia Video Decoding on Mobile Handheld Devices,"* 2007 International Conference on Multimedia and Ubiquitous Engineering (MUE07), Seoul, Korea, April 26-28, 2007, pp.120 – 125.

[10] A. Anand, A. Gupta, A. Akella, S. Seshan, and S. Shenker. Packet Caches on Routers: The Implications of Universal Redundant Traffic Elimination. In *ACM SIGCOMM*, Seattle, WA, Aug. 2008.

[11] Redundancy in Network Traffic: Findings and Implications
Ashok Anand, Chitra Muthukrishnan, Aditya Akella and Ramjee Ramachandran. SIGMETRICS 2009 Seattle, WA.

[12] Wireless Video – Introduction
http://encyclopedia.jrank.org/articles/pages/6943/Wireless-Video.html

[13] Adobe – Solutions: Broadcast and media
http://www.adobe.com/solutions/broadcast/overview.html

[14] OpenFlow Switch Consortium
http://www.openflowswitch.org/

[15] NOX – An OpenFlow controller
http://noxrepo.org/wp/

[16] Cisco Visual Networking Index: Forecast and Methodology
http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360_ns827_Networking_Solutions_White_Paper.html

[17] Video/Imaging DesignLine | Anatomy of a hardware codec
http://www.videsignline.com/199500920;jsessionid=IDLUZT1XPC4TVQE1GHPCKH4ATMY32JVN?pgno=1

[18] SIGCOMM- Openflow poster :
http://guido.appenzeller.net/pubs/sigcomm-2008-openflow-poster.pdf

[19] Flowvisor: A Network Virtualization Layer:
http://www.openflowswitch.org/downloads/technicalreports/openflow-tr-2009-1-flowvisor.pdf