INTERNATIONAL JOURNAL OF HIGHER EDUCATION AND RESEARCH
IJHER, Vol., 7(1), 2017, 58-80
www.ijher.com

IJHER
(ISSN 2277 260X)

CODEN: IJHER ORIGINAL ARTICLE

# WEBSERVER, A ROBUST APPROACH TO INFORMATION COMMUNICATION TECHNOLOGY AND ITS CHALLENGES

**Oyebola Blessed Olalekan[1] and Ajayi Adeola[2]**

[1]Department of Computer Engineering Technology
Gateway (ICT) Polytechnic Saapade, Nigeria
E-mail: blessedolalekan@gmail.com
+2348150732388 or +2348039272829

[2]Institute for Entrepreneurship and Development Studies
Obafemi Awolowo University, Nigeria
E-mail: ajayifunso4christ@gmail.com

**ABSTRACT**

*A dedicated server is a single computer on a web hosting network that is leased or rented, and dedicated to just one customer. A service provider monitors the computer's hardware, network connectivity, and routing equipment, while the customer generally controls and maintains the server software. Dedicated servers are usually used for websites who have outgrown their standard hosting account and require something more powerful, they are also used by larger companies that require separate servers for mail, web, and database servers. Web hosting companies may have their sites spread of one or several servers. Dedicated servers are housed in data centers, where service providers can monitor them close-up and have hands-on access to them.*

***Keyword:*** *Webserver, Information, Technology, Phone*

**Citation of this article**

Olalekan, O. B. and Adeola, A. (2017). Webserver, A Robust Approach To Information Communication Technology And Its Challenges. ***International Journal of Higher Education and Research***, 7(1), 58-80. www.ijher.com

## 1. INTRODUCTION

The primary advantage of using a dedicated server over a typical shared hosting account is the sheer amount of resources and control available to you, the customer. In many cases, the client is at liberty to install whatever software they desire, giving them greater flexibility and administrative options. Custom software can be installed whenever required, and also leaves less chance for a hacking attempt because only one site is being hosted and not several hundred, hence providing less targets. Dedicated server clients do not share resources, as those with shared hosting plans do; but rather, are at liberty to use all the resources available to them. This work aimed to enlighten and give core understanding on how to set up servers for remote access and how to manage a dedicated server.

## 2.0 MANAGED SERVERS AND UNMANAGED SERVERS

There are two types of dedicated servers available today: managed dedicated servers and unmanaged dedicated servers. An unmanaged dedicated server leaves nearly all the management duties of running a server in the purchaser's control. The customer in this case, updates software on their own, applies necessary patches, performs kernel compiles and operating system restores, installs software, and monitors security. With this type of dedicated server, the consumer is solely responsible for day-to-day operations and maintenance. The service provider, in turn, monitors the network, repairs hardware problems, and troubleshoots connectivity issues. Additionally, some service providers offer partial management of services, such as network monitoring, software upgrades and other services, but leave the general upkeep of the server in the hands of the client. An unmanaged dedicated server is best for someone with server management experience. (Kevin Airgrid 2009). A managed dedicated server is generally more proactively monitored and maintained on the part of the service provider. When renting or leasing a managed server, the service provider or host carries out the responsibility of software updates and patches, putting security measures in place, performing hardware replacements, and also monitoring the network and its connection for trouble. In other words, when utilizing a managed dedicated server, the host provider will perform both hardware and software operations. A managed dedication server solution works well for the customer with limited server management experience or limited time in being able to perform the duties necessary to keep a server running and online.

## 2.1 TECHNICAL ASPECTS IN CHOOSING A SERVER

When choosing a dedicated server, there are several things to consider: operating system, hardware options, space and bandwidth. The operating system of a server is similar to that on your own personal computer; once installed, the operating system enables one to perform tasks more simply. There are a bevy of server operating systems available today including linux-based and windows-based software. The operating system you choose should be directly relational to what operations your server will be performing, which types of software you'll need to install and also, what you're more comfortable with. Some common operating systems used for hosting at present are red hat enterprise Linux (rhel), freebsd, fedora (previously known as red hat linux), debian, and many other flavours of linux. Windows is not the most suitable platform for hosting due to its many security holes, and ease of unauthorized entry. There are several control panels which companies use for web hosting are: cpanel, plesk, ensim, and several others. A free alternative is webmin, but not very easy to use when working with a large number of sites**.** Hardware options are also something to consider when choosing a dedicated server. You'll need to pick a processor that's up to the task, the amount of memory you wish installed, firewall options, and the size of the hard drive. Celerons in general do not perform well when you start hosting several large powerful applications or web sites, pentium 4's (p4's), perform much better as they also have a larger cache, xeons on the other hand can perform much better, especially dual xeon processors. One should also take into consideration the amd option, dual (and single opterons) have been shown to outperform dual (or single) xeons by a margin of 30% or more in some cases. These are quite often cheaper to purchase, so therefore a more suitable solution. When looking at hard disks, you should remember that all hard disks fail at some point or another so it is a necessity in a server environment to have off-site backups or onsite backups of some sort. Having a second mirrored (raid-1) is a great solution. If you require fast access and high reliability and are willing to pay more it is a very good idea to go for scsi, they have much lower access time and much lower failure rate.

A certain amount of bandwidth is generally included when renting or leasing a dedicated server. Once you ascertained how much bandwidth you will require, you can adjust that limit with your service provider. The space you'll be given is generally directly relational to the size of your hard

drive. Some hosts also give clients the choice of uplink port speed (usually 10mbps/100mbps but can go up to 1000mbps). Bandwidth can be provided using the 95th percentile rule, or simply the via an allocated bandwidth (which is quite often around 1000gb/month). A router is also needed for the setting up of a remote server (C.H Hughes, 2000).

## 2.2. WEB SERVERS

The first step is to view the web server as a black box and ask the questions: how does it work; what can it achieve? It's a safe assumption that most internet users believe a web site's success or failure is due to its content and functionality rather than the server used to power it. However, the choice of the correct server, and understanding its capabilities and limitations is an important step on the road to success.

What does a web server do? It serves static content to a web browser at a basic level. This means that the web server receives a request for a web page such as Http://www.compengdepartment.com/index.html and maps uniform resource locator (url) to a local file on the host server.

File from disk and serves it out across the network to user's web browsers. This entire exchange is meditated by the browser and server talking to each other using hypertext transfer protocol (HTTP). This workflow is shown in the figure below

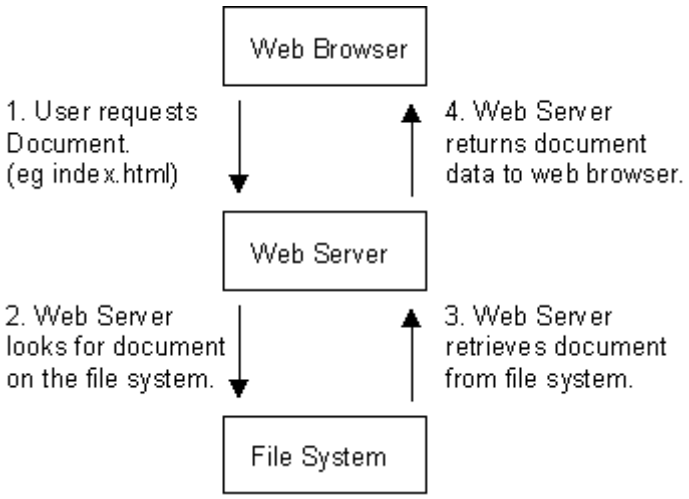


**Figure 1:** The Web Hierarchy

Because this simple arrangement, which allows the serving of static content such as hypertext markup language (html) and image files to a web browser was the initial concept behind what we now call the world wide web. The beauty of its simplicity is that it has led to much more

complex information exchanges being possible between browsers and web servers. Perhaps the most important expansion on this was the concept of dynamic content (i.e., web pages created in response to a user's input, whether directly or indirectly). The oldest and most used standard for doing this is common gateway interface (CGI) it basically defines how a web server should run programs locally and transmit their output through the web server to the user's web browser that is requesting the dynamic content. (Kevin Airgrid 2009). For all intents and purposes the user's web browser never really has to know that the content is dynamic because CGI is basically a web server extension protocol. The figure below shows what happens when a browser requests a page dynamically generated from a cgi program.

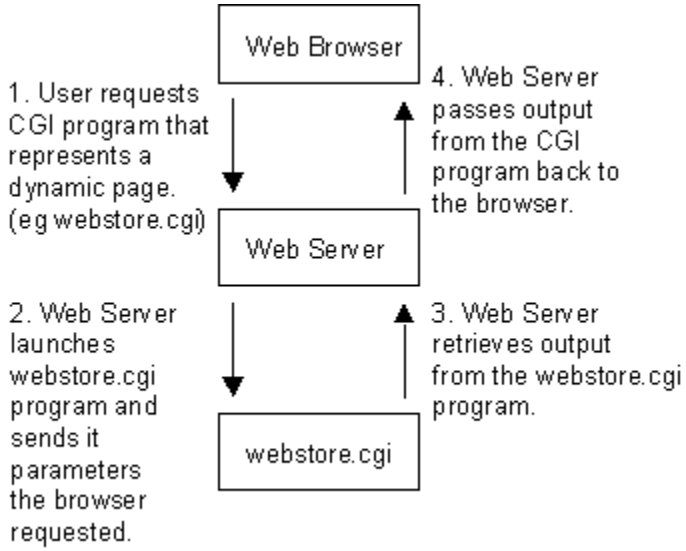


**Figure 2:** How the Wweb Funtions

The second important advance, and the one that makes e-commerce possible, was the introduction of hypertext transmission protocol, secure (HTTPS). This protocol allows secure communication to go on between the browser and web server. In a nutshell, this means that it is safe for user and server to transmit sensitive data to each another across what might be considered an insecure network. What happens when the data arrives at either end is another matter, however, and should not be ignored. We will discuss this a bit later.

The simplicity of the above arrangements is deceptive, and underestimating its complexities often leads to bad decisions being made about the design of a web-hosting infrastructure. It is too easy to focus on the design of the web pages themselves and the technologies used to create dynamic content, such as java, javascript, perl, c/c++, and asp, and to subsequently miss the fact

that each of these technologies can be aided, or hindered, by the platform on which they are to be run -- the web server itself..

Web servers are designed around a certain set of basic goals:

- Accept network connections from browsers.
- Retrieve content from disk.
- Run local cgi programs.
- Transmit data back to clients.
- Be as fast as possible.

Unfortunately, these goals are not totally compatible. For example, a simple web server could follow the logic below:
Accept connection.
Generate static or dynamic content and return to browser.
Close connection
Accept connection
Back to the start...

This would work just fine for the very simplest of web sites, but the server would start to encounter problems as soon as clients started hitting the site in numbers, or if a dynamic page took a long time to generate. For example, if a cgi program took 30 seconds to generate content (certainly not an ideal situation anywhere, but not completely unheard of), during this time the web server would be unable to serve any other pages. So although this model works, it would need to be redesigned to serve more users than just a few at a time. Web servers tend to take advantage of two different ways of handling this concurrency: multi-threading and multi-processing. Either they support the inetd module on unix (which is a form of multi-processing), multi-threading, multi-processing, or a hybrid of multi-processing and multi-threading. Early web servers used inetd to spawn a web server process that could handle each web browser request. They were fairly simple applications, and there was no expectation of them having to cope with a high number of hits, so this was a totally reasonable design decision to make at the time. Normally, a server process has to handle listening for and accepting TCP connections as

they are made. It must then make the choice to either juggle concurrent connections, or effectively block any new connections until the current one has been fully served and closed. The inetd daemon can do all this instead, by listening on the desired port (80, by default, for http requests), and running a web server process as it received each connection using this method also makes administration of the machine easier. On most unix machines, inetd is run by default, and is a very stable process. Web servers on the other hand, are more complex programs and can be prone to crashing or dying unexpectedly (although this has become less of a problem as these applications have matured). It also means that the administrator doesn't have to worry about starting and stopping the web server; as long as inetd is running, it will be automatically run each time an http request is received on the given port. [C.H Hughes]. On the downside, having a web server process run for each http request is expensive on the web host, and is completely impractical for modern popular web sites. These days, most web sites run a web server that supports either multi-processing or multi-threading, and are thus able to handle a much higher load.

## 2.3 RUNNING OF APPLICATIONS ON WEB SERVERS

A discussion of application servers and web servers would not be complete without a brief introduction to some of the technologies used to allow a web server to serve dynamic data. In a way, these technologies are what led people to realize that web servers can also serve application data such as xml instead of merely serving up html documents. Although these technologies may appear to be close in definition to an application server, they are different. Web application technologies are primarily focused on delivering dynamically generated html documents to a user's web browser while they interact with a web site. The pure application servers do not format data for humans to read. Rather they act as an engine that processes data for another program to read and interpret on behalf of the user.

The first such technology is common gateway interface, commonly referred to as CGI. As previously discussed in the section that focused on what a web server does, CGI programs serve html data dynamically based on input passed to them. Many people associate CGI with a language called Perl because perl has been used to create a majority of the cgi scripts that currently in use. However, CGI is not language specific -- it is merely a protocol for allowing the web server to communicate with a program. CGI can be written in any language, and common

choices, in addition to perl, include c, python, and TCL. The disadvantage of cgi is that it tends to be slow because each request for a dynamic document relies on a new program being launched. Starting new processes on a web server adds extra overhead. Soon after cgi came into being, other technologies quickly followed to solve this performance issue. Microsoft active server pages (asp) technology consists of embedding a vbscript interpreter into the microsoft internet information server. On the java front, servlets and java server pages connects a perpetually running java virtual machine to a web server. Servlets have an additional advantage over asps in that they become cached in the java virtual machine after their first execution. Vbscript pages are reinterpreted each time they are hit. In the open source community, PHP (http://www.php.net/). It is similar to jsp and asp technology in that php consists of a set of additional code tags placed inside of existing html documents. The interesting part about php is that it is a language developed purely to serve web pages rather than being based off of an existing language such as perl, python, visual basic, or java. This makes php-written applications very succinct compared to equivalent vbscript or jsp applications. While all of this was going on, the perl community did not rest on its laurels. All of today's major web servers have perl acceleration solutions available to it. Apache has a free solution called mod perl, which embeds the perl interpreter inside of apache. Not only does this speed up perl scripts, but the scripts themselves are also cached by mod_perl, providing further performance boosts. Mod_perl also hooks tightly into apache so a perl developer can literally change the behavior of how the apache web server works. Previous to mod_perl, this type of control over a web server belonged solely to the realm of c programmers coding to the low-level apache api. Commercial solutions from activestate and binary evolution also exist to accelerate perl in a similar way to mod_perl. Activestate's product, perlex, accelerates cgi/perl scripts for internet information server while binary evolution's product exists for netscape, apache, and internet information server on both unix and nt platforms. With all these fast solutions around, it may seem like it's a bit hard to choose a language to in which to develop when making web servers serve up dynamic pages. Actually, it's quite easy. Unless you have a really large hit rate where squeezing every performance bit out helps, you should go with whatever language is comfortable for you or your organization to do its job effectively

## 3.0 THE MOBILE WEB SERVER

While it has been possible for quite some time to access webpages on the Internet from mobile phones, reaching web servers on mobile phones has been neither possible in the general case nor in demand. First of all, almost all people percieve the mobile phone solely in the role of a service consumer, and not a service provider, for instance, while applications where the mobile phone is a content provider are almost non-existing (ComVu. Pocketcaster.) (Tapuz Mobile.). Consequently, operators, intent on protecting and optimizing their networks, neither allow autonomous downlink traffic (from Internet to phone) nor optimize for significant uplink traffic (from phone to Internet) today. With the ever spreading adoption of Service Oriented Architecture (Hoa He, 2003) (Jia Zhang,et al, 2004) , solutions for desktop applications involving locally run web services, and even special purpose electronic appliances incorporating web servers (Masahide Nakamura, 2004) we expect that HTTP access to mobile phones will soon be essential service to mobile users. Furthermore, these devices are becoming ever more powerful in terms of processing and storage capacity, and have already bypassed those of the early web servers when the web was young. A necessary first step towards such a goal is putting a working HTTP server implementation on the mobile phone. This readily makes web pages and web services available to web browsers accessing mobile content locally or via some proximity radio technology such as Bluetooth, WLAN, etc. However, for ubiquituous and unified access, a web server should be reachable via cellular IP from the public Internet, a service generally not available in most operator networks. This paper discusses obstacles in today's operator networks one has to overcome when providing connectivity to mobile web servers, describes a solution we have developed at Nokia Research Center, and finally suggests possible future directions.

### 3.1 OPERATOR NETWORKS

In most operator networks, an HTTP request originating from the public Internet cannot reach a web server that is running on a phone inside the operator network for several reasons. 1. Operator firewalls are usually configured to prevent all traffic that is not initiated from inside the operator network. This is a simple measure to make operator IP networks resistant to most forms of attacks, with the assumption that traffic initiated only from inside is legal, and from outside only responses are allowed. This means that an HTTP request cannot reach a mobile web server inside most operator networks. While TCP connectivity is supported in operator networks, it usually has some special traits that makes it subtly different to conventional wired TCP connections. In addition to being initiated only from inside the operator network, they are prone

to become stale. Problems induced by such behavior do not manifest themselves in current use of cellular IP networks, for instance, during web browsing, as HTTP requests sent from inside are quickly responded to by Internet web servers. Therefore a solution is needed to provide true always on connectivity in an environment that is currently not used and therefore not tuned for traffic patterns involving prompt responses to asynchronous requests from outside. Mobile phones usually have dynamically allocated IP addresses, since current applications using cellular IP involve mobile phones as clients, where the IP address of the client hardly matters. There are operators providing services with fixed dynamic addresses, but such support is far from general, and certainly not scalable. Without a solution to this problem, there is no user friendly way to determine the actual IP address of a mobile web server, hence it cannot be easily reached. Network Address Translation (NAT) is commonly used for security reasons as well as to alleviate IPv4 address depletion problem, preventing an HTTP request from outside the operator network to find the mobile web server. Among these, there are two closely related but different types of obstacles: accessibility and addressability. The first two are related to accessibility, as in those cases the traffic itself is either impossible or breaks down. While addressability assumes accessibility solved, it is a subtly different problem that makes a system impracticable; even if traffic itself is possible, a user must have an easy way to address the mobile web server. For instance, forcing people to figure out and manually type in dynamic IP addresses is not viable option.

An additional question is that of latency and speed of operator IP networks, and the fact that these are nowhere near to what one expects when browsing to powerful servers over the wired Internet. However, this seeming show stopper argument stems from a wrong although natural assumption, namely, that mobile web servers are nothing but web servers, and that the very same content types would be served from a mobile phone as served from stationary web servers. In fact, we envision exactly the contrary; mobile phones are not meant to host huge digital libraries, vast databases to support e-business, popular portals, or news sites. Mobile web servers will be best performing as sources of dynamic, context-dependent, personal data streams (webapplications and web services). While some of these will undoubtly be of multimedia types (audio, video, image), textual and XML-based data will play a significant role, where speed does not matter that much. Also, while a system could easily be built where response latency can be reduced significantly by uploading, synchronizing, mobile data with content servers, such

solution would simply trade low latency for scalability inherent to the distributed datasource system represented by a mesh of individual mobile web servers, freshness of mobile data, and potential applications involving on demand content generation. Furthermore, anticipated increase in cellular IP throughput, web caches, and separation between static/ non personal and dynamic/personal data are all expected to help closing the gap between wireless and wired web experience. Finally, operators will optimize their networks for significant uplink traffic once it is demanded widely, just like they do it for current traffic patterns today.

### 3.2 SOLUTION ASPECTS

When setting out to create HTTP connectivity to mobile web servers, we have determined a number of important goals an ideal solution should meet. The solution must not invoke any changes in browsers. Meeting this goal ensures that all existing browsers can be used to access pages, web applications originated from a handset, and existing web services client frameworks can be readily put into use in the mobile context as well. • The HTTP connectivity must be transparent to the HTTP server, that is, it should not know whether an HTTP request came from a local web browser, a nearby device via some proximity radio technology, or from a remote client via the cellular. The solution ideally should necessitate as little operator involvement as possible, and should be independent of any particular operator network**.**

### 3.3 OVERVIEW OF CONNECTIVITY

HTTP is designed to allow intermediaries -proxies, gateways, and tunnels -between the client and the origin server (Network Working Group.). The targeted URL and HTTP header ‘Host’ are the two most important data in an HTTP request that allow an intermediary to determine where to send an HTTP request forward. The internet gateway, as shown in Figure 1, providing the HTTP connectivity to the mobile phone is simply a gateway in HTTP/1.1, or reverse proxy in Apache-httpd parlance. The communication between a browser/web client and the gateway is un modified HTTP. The problems of addressability boils down to what type of URL the web client needs to use and how the internet gateway dispatches an incoming HTTP request to the intended mobile phone. The proprietary protocol between the gateway and the mobile phone has to overcome the accessibility problems.

### 3.4 ACCESSIBILITY

The HTTP server running on the mobile phone creates a server socket and starts listening to it, just like HTTP servers generally do. If the HTTP server were on the Internet – with no firewalls or other obstacles in between a simple HTTP connection from the browser would suffice, and that is conceptually what we want to achieve in a transparent manner. This is accomplished by adding a level of indirection: a TCP connection between mobile connector and gateway connector. Mobile connector is a process running on the mobile phone that establishes a TCP connection to the gateway —or more accurately to its peer component, the gateway connector, after which the mobile phone is considered online. This way, the connection between the mobile and gateway connector is established from inside the operator network, a legal direction. Once the link between the two connectors is alive, the gateway connector can dispatch HTTP requests any time to the mobile phone. Whenever the mobile connector receives an HTTP request from the gateway connector, it creates locally a socket connection to the port the HTTP server is listening to. This means that from the HTTP server point of view, the mobile connector is just another HTTP client. The protocol between the two connectors is not HTTP. In HTTP, the one creating the connection and sending the request is the same endpoint: the client. The proprietary protocol, on the other hand, is initiated by the mobile connector, but HTTP requests are sent by the gateway connector. Furthermore, this protocol supports more than mere HTTP traffic relaying; it can authenticate the mobile connector and keep the connection alive by a regular stream of mostly empty messages sent by the gateway connector.

### 3.5 CONNECTION ESTABLISHMENT

In order to be online, the owner of the mobile web server has to register on the gateway, so that it can associate between the credentials supplied by the mobile connector upon connection establishment, and the URL prefix of the user, for example john.doe. Conceptually there are two types of communication channels between mobile connector and gateway connector: control channel and data channel. The TCP connection the mobile connector creates when going online is in fact the physical manifestation of the control channel - this is the essential part that is kept alive, and is used to initiate HTTP relay sessions. Whenever there is a need, the gateway connector asks the mobile connector to create data channels, which are currently implemented as separate TCP connections to the same port. These data channels can be reused for the next HTTP request. The problem of dynamic IP address and NATs in between the two connector peers do no

matter to the gateway connector, it simply associates the control channel (TCP socket connection) to the URL prefix of the owner john.doe—of the mobile phone.

### 3.6 KEEPING THE CONNECTION ALIVE

As mentioned above, the gateway connector has the responsibility to generate regular keep-alive traffic by which the control channel is kept usable. If the period between two keep-alive messages or HTTP request is less than the period after which a TCP connection would become stale, the connection will be always permeable from the Internet end. There would have been a good reason to give the mobile connector the task of generating regular keep-alive messages: according to our experience, a message sent over a stale TCP connection from inside the operator network revives the connection, and this way the protocol could have been made self-correcting naturally. On the other hand, this would have resulted a significantly more complex state machine of both endpoints of the protocol. In the current demo implementation, once the connection is established, it's only the gateway connector that can send a message autonomously over the control channel: either a request for opening a data channel, or a dummy keepalive message if there is nothing else to send. The mobile connector only waits for messages after the initial handshake. If the gateway cannot send a keepalive message, it considers the mobile phone offline, and generates a "Mobsite offline" page to any web client requesting content from the mobile phone. There is, of course, the question of how regular the keepalive messages have to be. To this end, we have defined and implemented a protocol that can employ one of the following three strategies. The mobile connector can tell the gateway connector not to generate a keepalive stream. This feature is not only useful when testing the system, but also allows for the keepalive mechanism to be turned off when it is known that it is not needed, as is the case when the connection goes, for instance, over USB. By default, the gateway connector tries to discover a nearoptimal value for the keepalive period, which is why keepalive messages of our protocol are more than empty messages: they specify when the next keepalive message will be due. If the mobile connector does not receive a keepalive message (or a data channel request) by the time the next message is due, it will consider the connectivity broken, and will try to fix it by establishing a new control channel.

Meanwhile, the gateway connector is trying to discover a sufficiently good keepalive period by attempting to use different values for it — a non-viable keepalive period is signaled by a new

control channel from the mobile connector that did not receive its due keepalive message. In such case, the gateway selects the last known good value for the keepalive period. Our current implementation of the discovery algorithm tries safe but non-optimal values first (short keepalive period), and then tries incrementally pushing them towards less safe but more optimal values (longer keepalive periods). The discovery algorithm the gateway connector employs is heuristical and a likely subject to improvements, but the conceptual algorithm, its implementation, or any of its running parameters can be centrally changed without modifications to any existing mobile connectors, as they only care about is how much time to wait before trying to re-connect. It is also possible to have keepalive messages of fixed keepalive period without dynamic discovery by explicitly setting the keepalive parameters in the settings of the mobile connector software

## 4.0 IMPLICATIONS OF MOBILE WEBSERVER PERSONAL

A regular website can be personal but the personal content is largely explicitly created and has to be manually uploaded, because the website does not reside on a personal device. Contrary to that, a mobile phone is personal and both contain and collect personal data that implicitly and immediately are available. Contact entries, logs of dialed and received calls, sent and received SMSs, taken pictures and video clips are all data that easily can be used for (semi)automatically creating a mobile home page. If you wanted to reach the similar kind of personal level on a regular web site you would have to continuously upload data. It would be possible, but at some point it becomes easier to simply share the data from the location where it is already stored.

## INTERACTIVE

Currently the content a website returns depends largely on the input parameters of the request and sometimes on the identity of the one browsing. The content may be dynamic to its nature but there is no explicit human participation in the content generation. When the website is personal and resides on a device that is carried along for most of the time, the situation changes, as it is possible to involve the owner of the phone in the content generation.

## 4.1 DYNAMIC

The website on a mobile phone can be dynamic in the sense that the content can change depending on implicit state changes on the phone. For instance, the background color of a page

or even the style sheet could change depending on the mode — general, meeting, silent, etc. —of the phone. This possibility could also be utilized so that when you intend to contact someone, you would first browse to his or her mobile home page. This would, of course, not necessarily require explicit actions but occur automatically when you select an entry from the contacts on your phone. Based on the state of the phone the default way for contacting — call, SMS, email, etc. — could be presented on the page. It would even be possible to take into account the identity of the one browsing. Thus, it could be the one being contacted who would decide the way the contacting is made, not the one who is contacting.

**CONTEXT AND LOCATION DEPENDENT**

In traditional websites the geographical location of the actual web- server lacks meaning since it never changes and it has no impact on the returned content. With a mobsite this is no longer the case as the returned content may depend on the geographical location and surrounding context.

**OFF LINE**

With ordinary websites administrators go to great lengths in order to ensure that the website stays up. With mobile websites, the assumption can no longer be that the website is always available, as people occasionally simply turn off their mobile phones. Instead of considering this as evidence that a mobile phone is unsuitable for hosting a website, we view this as a inherent property of mobile websites that simply has to be taken into account. If the mobile phone were directly addressable it would be difficult to deal with this in an end-user friendly way, since, when the phone is turned off, it would simply disappear and web browsers would report it in some browser specific way. The owner of the mobile phone could not provide any additional information. The presence of the gateway provides means for dealing with this issue in a more end-user friendly way. Since the requests intended for some mobile phone will always go through the gateway, it is possible to explicitly report that the mobsite is offline. For instance, currently when turning off the webserver on the phone, the user is asked whether he wants to leave an out–of–site message. If he chooses to, the text he enters is uploaded to the gateway and if someone tries to browse to the mobile website while it is offline, the gateway will display the customized message.

**CHALLENGES**

That mobile phones are capable of running a web server and that it is possible to make a mobile website visible in the operator networks of today is only a part of the story. There are numerous challenges that need to be addressed before mobsites can become ubiquitous.

**COST**

Currently many operators still charge by byte and rates, quite high. However, the situation is not quite as bleak as might initially be assumed:

The prices for data transfer are decreasing and there already are operators offering flat rates, although they may not yet have thought of the implications brought by websites being hosted on mobile phones. Mobile phones are being equipped with WLAN, which means that for much of the time, the wireless cost would be zero. The presence of the gateway means that there is a point where caching can be employed. For instance, in the case of a picture gallery it can easily be arranged so that a picture is downloaded at most once. Another aspect is that even if access control is employed at the mobile website in order to control who can access content and thus cause costs, it is still possible for a hostile user to circumvent that by simply over and over again trying and being denied access. If access control is employed only in the mobile phone there is no alternative but to transfer the request there and that alone already causes cost. To prevent cost from being generated, the access control must take place before the traffic enters the cellular connection. So, from a cost perspective, as long as there are no true cellular flat rates, the access control should take place already at the gateway.

**BATTERY CONSUMPTION**

The battery consumption is a problematic issue. Currently all activities that increase the battery consumption for instance, browsing the web or talking are quite explicit for phone user and thus will not surprise him. If a web server, or any other server for that matter, is running on the mobile phone, an external party can increase the battery consumption without it being obvious to the owner of the phone. To prevent surprises, some means for controlling the access based on the state of the battery may be needed. But as in the cost case, any activity for reducing the battery consumption should take place before a traffic reaches the phone. Again it turns out to be benefitial to have a gateway between the browser and the phone. For instance, the gateway could allow anonymous browsing to the mobile website only when the battery is fully charged or the phone is attached to a charger.

**4.2 SECURITY AND PRIVACY**

Because the phone contains a fair amount of personal information, it is important that the phone owner can control who can access what. That is a challenge because traditional means such as accounts and passwords may not work that well in the context of a personal website. Applying access control in a straightforward manner could easily imply that a phone user would become

an administrator who explicitly manages accounts and passwords and who answers “support requests” when someone has forgotten his. And from an other perspective, the amount of passwords a person would have to remember could easily grow unwieldy if mobile websites become popular. There is also a more subtle aspect that stems from the fact that mobile websites are context dependent. Traditionally, access control on websites is essentially a function of the identity of the one browsing and the content he tries to access. With mobile websites that alone will not always suffice when deciding whether or not to allow access. Consider for instance the earlier mentioned case of being able to find out other mobile websites in the surrounding. You may want to allow that depending not only on the identity of the one browsing but also on the context and even your mood. That can be arranged by utilizing the interactive aspects of the personal website. When someone tries to access some link, traditional means for access control can first be applied. Then, since the identity of the person at that point is known, it is possible to show a dialog with the question Allow access to X for Y. That is, for some content the phone owner can explicitly decide on a case by case basis who can access it .

**BACKUP AND RECOVERY**

We envision that the mobile phone could in many cases turn into the primary device of a person. All relevant personal data are stored there and only there, and accessed from other devices using a browser. But mobile phones are lost and stolen, and, as other hardware, also break down at times. If all your personal data is on the device, it will be imperative that it is not irrevocably lost in cases like that. Thus, efficient means for backing up and recovering the data of a mobile website are needed.

### 4.3 SECURITY

One concern about having a web server on a mobile phone is whether it will be a target for the kind of attacks stationary web servers are usually target of, such as Denial Of Service, IP spoofing attacks, HTTP messages exploiting known bugs, and theft of private content, to name a few. While the primary function of the gateway described in this paper is to provide connectivity in current operator networks, it can be used to fight these problems as well. Indeed, corporates and news sites employ web gateways or reverse proxies not only for load balancing, but in order to be able to secure their private networks by controlling IP traffic. Similarly, a gateway can be put to the same task. In fact, if operators were to allow HTTP access to mobile phones, they would be likely to do so via web gateways controlling access and cost; the only practical difference between an ordinary web gateway as part of an operator firewall and a gateway is that

the latter employs a proprietary protocol instead of simple HTTP proxying of the former, because we set out to implement HTTP connectivity without any operator involvement. Finally, since web servers on the mobile phone cannot be reached unless they go online first, a control method checking the version of an HTTP server implementation on the phone can be easily implemented, and old versions with known security bugs could be automatically, or with minimal user interaction, upgraded.

## 4.4 COST CONTROL

Another concern of having a web server on the mobile phone is whether it will cause huge phone bills. One of our assumption is that operators will switch to flat-fee payment models — a trend already perceptible. With flat fee rates, the only type of cost is reaching the monthly limit of byte transfer after which the owner has to pay extra fee. The best place to control cost is at the gateway, since an HTTP request does not yet generate cost for the owner of the mobsite at that point. It is possible to build a system where mobsite owners can ration their monthly byte limit for different uses, like certain amount for private, family, friends, colleauges, and the public Internet.

The only prerequisite to this is to have an access control in place where the gateway plays an active role, that is, a scheme where the credentials of the web client and access rights to mobile content are managed by the gateway transparently. While such a mobile access control mechanism assisted by the gateway sounds

like an extra requirement, it is, in fact, necessary by itself: an HTTP request challenged by the mobile web server instead of the gateway will have already generated cellular cost to the owner regardless of whether it can pass the authentication challenge or not. Also, the fact, that operators currently do not have specific offers for ‘uplink HTTP traffic generators’, does not mean that they won’t have such in the future, when mobile web applications and web services will be commonalities. Just as today one can select from a multitude of plans with different benefits for different phone usage profiles, there will surely be a plethora of service types for different private, social, entrepreneur, enterprise -mobile content providers.

## 4.5 WEBSITE ON A MOBILE PHONE

From the fact that it is possible to run a web server on a mobile phone does not follow that it would be meaningful to have any kind of websites on mobile phones. The processing power, the

amount of memory and the available network bandwidth of a mobile phone will, for the foreseeable future, be inferior to that of regular computers. Furthermore, since mobile phones still occasionally are turned off, a mobile website—or mobsite, as we call it— will not always be accessible. In addition, the cost for the cellular connectivity is likely to remain rather high for still some time. Consequently, it is probably a mistake to simply move what could be characterized as a "regular" website to a mobile phone. However, a phone equipped with a web server is very different from a phone without one and a website on a mobile phone is very different from a regular website, and if these specific aspects are taken into account, a number of new possibilities emerge. In the following sections we will explore some of these in greater detail.

### 4.6 A PHONE WITH A WEB SERVER IS DIFFERENT Web UI

Even though the user interfaces of mobile phones are quite user friendly, they are still rather constrained compared with the possibilities offered by the big screens and proper keyboards of regular PCs. Considering that many users of mobile phones have access to PCs for a significant fraction of the day, at work and at home, it is a restriction that also in that context the only UI available for a mobile phone is the native one. If a web interface is created for the core applications of a mobile phone, it would mean that whenever you have access to a PC- any PC, not necessarily your own-with Internet connectivity you would be able to access the functionality of your phone using a proper keyboard and a big display. Reading and answering SMSs could take place from within a regular Internet browser and you could even arrange for an RSS feed for received messages. It could even turn out that some users would more frequently use the web interface than the native UI of the phone. A web interface to the an application on a mobile phone could obviously also be used remotely. Every mobile phone user is likely to have forgotten his phone at home at some time or another. With a mobile web server on the phone it is easy to browse to it remotely and check, for instance, whether someone has called or sent an SMS, and even answer SMSs. If you have a web server running on the mobile phone then you can have web service providers as well. That opens up new possibilities for making mobile phones even more general purpose than they currently are. Referring to the calendar example of the previous subsection, an electronic calendar does not really become useful before it can be made accessible to other people and it can be used for setting up meetings. In practice that has meant that you need to have a centralized server where the calendar data is held, which has

largely implied a corporate environment. With a web service interface to the calendar it would be easy to create a distributed calendar application without the presence of any centralized server. Consequently, the calendar application on a mobile phone could become quite useful for ordinary users and perhaps even sufficient as such for small enterprises. New Messaging Concepts Currently the means for communicating with the phone are basi- cally defined by standards, phone manufacturers and operators. In practice, you can call a phone or send it an SMS or MMS message. ranted, there are, for instance, implementations of instant messaging and push-to-talk, but common to these is that they are clients to a server somewhere else and in order to use the functionality you need to have the appropriate client installed. With a generic HTTP connectivity to the phone it is possible to create new messaging concepts without a-priori standardization or support from the operator. For instance, we have created a web application that allows you to send a message to the phone from a regular web browser. The message is either shown directly on the screen as an instant message or placed in the inbox of the phone along with the "real" SMSs and MMSs.

From a cost perspective this is also quite interesting. Provided you already have 2.5G/3G access for instance, for browsing the web functionality similar to SMS can be provided essentially for free as the amount of data in a textual message is negligible compared to the amount of data transferred during a regular browsing session. Pull Instead of Push To disseminate any data from a mobile phone currently requires that the phone owner actively, for instance, sends an SMS or an MMS, or uploads a picture to a website. With a web server on the mobile phone we can change that. For instance, instead of sending MMSs to your friends when you are on a vacation, you could provide them with a pointer to a picture gallery on your phone that they can browse at their leisure. It obviously would also be possible to allow people browsing the gallery to leave comments and write blog entries. However, a public webgallery on the mobile phone is likely to be a realistic option only as ong as the number of people accessing is remains fairly low.

## 5.0 CONCLUSION AND RECOMMENDATION

Many challenges were faced during the construction and implementation of this project which includes

1. Getting the appropriate server to use for the hosting of the website on the computer. Many servers which include apache web server, Microsoft IIS (internet information services), abyss web server, and purple nova were tried, but access to all these server on the local area network could not be gained but not the internet.

2. Resolving the phone IP address to a domain name was a bit.

It is suggested that further studies on this project should include how to overcome this limitation and make it perfect. In conclusion, setting up a dedicated server are very useful for remote access and hosting of files to be accessed remotely, also designing a website and hosting it for remote access. The needs for websites are important in this modern age so that people can access data anywhere around the world.

Security measures on the server make it secure and free from hackers. Dedicated servers are also managed via several software that makes it easy to implement, manage and monitor. In a nutshell, a virtual server account resides on a computer that also houses other virtual server accounts, whereas the dedicated server is all yours. With a virtual server the computer's hardware resources and associated bandwidth are shared among multiple sites. A virtual server is an excellent solution for small to medium sized businesses that want to create a powerful internet presence. A dedicated server is an industrial-strength system of hardware, operating system software, and web server software which you control and to which you have exclusive access. All of the server's powerful resources and bandwidth are allocated entirely to your internet site, enabling you to offer features and service levels to your customers, employees, and suppliers that go beyond the standard capabilities of a virtual server.

It is possible to provide HTTP access to web servers running on mobile phones on today's operator networks without any special arrangement with operators. The current implementation is suitable enough to provide HTTP accessibility to a fair amount of mobile users, ready for use in university research projects and small businesses willing to maintain a gateway on their own. For an industrial use, scalability, security, access and cost control has to be provided as well, but none of these problems are conceptually difficult or even novel, a combination of off-the-shelf tools and techniques in webhosting and website management will likely to go a long way in developing a full-fledged solution. It is recommended that this project should be improved upon. Issues like resolving the IP address of the server residing on the mobile phone should be solver

and a proper DNS server should be set up on the mobile phone as well. The mobile web server should also be made available over the internet - beyond the local area network.

**REFRENCES**


Apache Software Foundation. Apache http server project. httpd.apache.org.

Apache Software Foundation. Apache tomcat. tomcat.apache.org.

Belle L. Tseng, Ching-Yung Lin, and John R. Smith. Video summarization and personalization for pervasive mobile devices. In Storage and Retrieval for Media Databases, January 2002.

Business object enterprise xi release 2 http://www.philbrickarchive.org/

C.H Hughes“server tutorials”september 1, 2000 http//:www.serverwatch.com

Carlifoniastate university sacramento 2005 http//:www.educationaltrainers.com

ComVu. Pocketcaster. www.comvu.com

G.M. Virginia “remote desktop tutorial” 2008. http://en.wikipedia.org/wiki/Special:BookSources/9780130829870

Hao He. What is service-oriented architecture. webservices.xml.com/pub/a/ws/2003/09/30/soa.html.

Jia Zhang, Jen-Yao Chung, and Carl K. Chang. Migration to web services oriented architecture - a case study. In Proceedings of the 2004 ACM symposium on Applied computing. ACM Press, March 2004.

Jonathan Lemon. Kqueue: A generic and scalable event notification facility. people.freebsd.org/ jlemon/papers/kqueue.pdf.

Chawla, K., Zhimei Jiang, Xiaoxin Qiu, and Amy Reibman. Transmission of streaming video over EGPRS wireless network. In Proceedings of First IEEE International Conference on Multimedia and Expo, July 2000.

Kevin Airgrid 2009“web designer’s success guide” 2009 http://ecow.engr.wisc.edu/cgi-bin/get/ece/342/schowalter/notes/chapter.

Loaf Zimmermann, Sven Milinski, Michael Craes, and Frank Oellermann. Second generation web services-oriented architecture in production in the finance industry. In Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languagres, and applications. ACM Press, October 2004.

Masahide Nakamura, Hiroshi Igaki, Haruaki Tamada, and Ken ichi

Matsumoto. Implementing integrated services of networked home appliances using service oriented architecture. In Proceedings of the 2nd international conference on Service Oriented Computing. ACM Press, November 2004.

Network Working Group. Hypertext transfer protocol – HTTP/1.1. http://www.rfc-editor.org/rfc/rfc2616.txt.

Source Forge. Mobile web server. http://sourceforge.net/projects/raccoon.

Server configuration file http//:www.maxpc.co.uk

Tapuz Mobile. Blogtv. http://www.tapuzmobile.com/products.asp.

Timo Ojala, Jani Korhonen, Tiia Sutinen, Pekka Parhi, and Lauri Aalto. Mobile k arp at – a case study in wireless personal area networking In Proceedings of the 3rd international conference on Mobile and ubiquituous multimedia. ACM Press, October 2004.

Wikman Johan Dosa Ferenc, Tarkiainen Mikko. Personal website on a mobile phone. http://research.nokia.com/tr/NRC-TR-2006-004.pdf.

W. Richard Stevens. Advanced Programming In The Unix Environment. Addison-Wesley, 1993.

Webhosting school UK. Http//:www.w3schools.co.uk