Power Saving in WAP

A PROJECT

Submitted to

Oregon State University

Girish Potti

In Partial fulfillment of the requirement of

The requirements for the

Degree of

Master of Science in Computer Science

Presented June 1 ,2000

Commencement December 1,2000

An Abstract of the project of

Girish Potti for the degree of Master of Science in Computer Science
Presented on January 8th, 2001
Title: Power Saving in WAP

Wireless Application Protocol (WAP) is an industry standard aimed to bring the web to handheld devices. The handheld devices are constrained by battery life and it becomes important that power is conserved across web transactions. The power consumed by the handheld device is directly proportional to the time taken for a transaction and time can be minimized to conserve power. The WAP server gateway sits between handheld devices and web content providers. The turnaround time for transactions can be improved by optimizing the WAP server gateway. Load balancing of the requests between various server processes was implemented. A server cache on the WAP gateway server to "remember" previous requests and contents was implemented.

ACKNOWLEGEMENT

I would like to acknowledge and thank Dr.Suresh Singh for the guidance and computing facility support provided by him. My earnest thanks are extended to my Major Professor Dr.Timothy Budd, who provided direction and valuable insights through all phases of this project.Dr.Budd's attention to detail helped me refine my project report. I would also like to thank Dr.Prasad Tadepalli for the valuable time he spend going through my report. Finally, my deepest love to my wife Udayarani, who is always there to help. My gratitude extends far beyond the acknowledgment.

Table of Contents

Introduction

Technologies that were visualized by science fiction authors are made a reality by advances in science. The explosive growth of the Internet has taken the world by storm and people are using the "WEB" for all sorts of applications. Breakthroughs in the wireless technology has made hand held devices like cellular phones, PDAs (personal digital assistant) popular. The convergence of these two technologies has literally brought the Internet to everyone's fingertips.

Most of the communication technology being developed by the industry is designed for the powerful desktops and large machines and the communication link between desktops typically have high bandwidth and less data loss.

The small hand held devices present a much-constrained computing environment. These devices compared to desktops tend to have:

1) Less powerful CPUs
2) Less memory
3) Restricted power consumption
4) Small display
5) Different input devices (eg keypad)

Similarly the wireless data networks are limited by power, available spectrum, mobility and they compared to wired net typically tend to have :

1) Less bandwidth
2) More latency
3) Less connection stability
4) Less predictable availability

Mobile service providers would also like to extend some advanced services to the customers for attracting them. Mobile service providers can package lot of wireless telephony applications (WTA) such as call forwarding and news service in a user-friendly manner.

All these necessitate a new technology that builds on existing technology and narrows the limitations mentioned above. The Wapforum is an industry association consisting of lead players in wireless technology, they produced the de-facto world standard for wireless information and Telephony services on digital mobile phones and other wireless terminals. They came up with the

Wireless Application Protocol (WAP), an open global specification that makes browsing from the cell phones possible and also attempts to makes it an enjoyable experience.

For example, a hungry traveling man can find out the directions, relative to his location, to the nearest restaurant and can see their online menus and order them, to eat them hot when he reaches the restaurant. When munching he can get online stock quotes and do the trading while his urgent documents are mailed to him by his secretary. There are lots of other such conceivable applications.

WAP Architecture Overview

The Internet WWW (World Wide Web) architecture presents applications and content in standard data formats and are browsed by applications known as web browsers. The web browser sends requests for named data objects to a network server and the network server responds with the data encoding using the standard formats.

The WAP programming model is similar to the WWW programming model and has added on to the WWW model to match the characteristics of the wireless environment as mentioned below. The WAP model is shown in Fig 1. It gives the user the ability to use existing tools like cgi, xml, web server, a familiar programming model and a proven architecture. Optimizations and extensions were done to the original WWW model wherever possible as mentioned below.

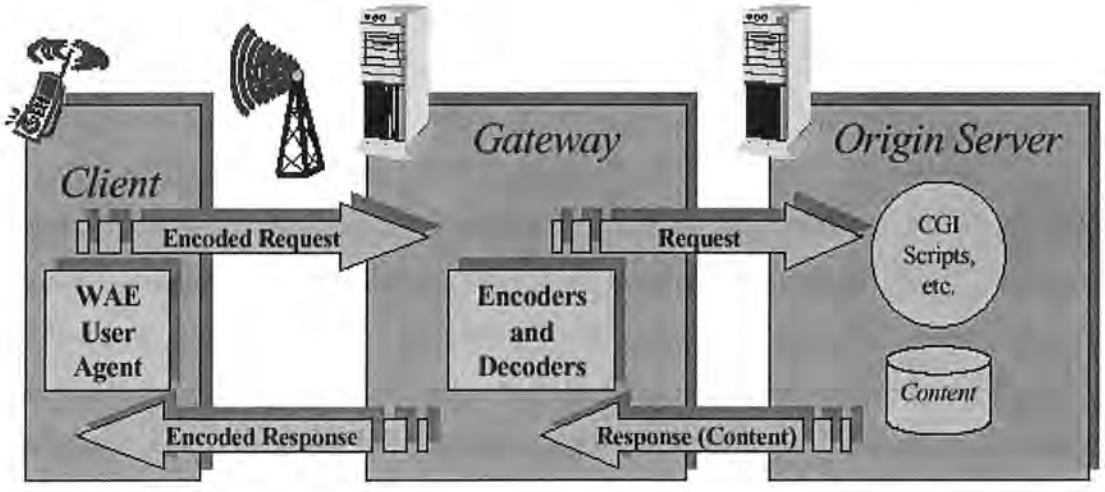


Fig-1

In the WAP model the clients (Mobile devices) send requests for contents to the WAP gateway using the WAP Protocol Stack.The WAP gateway uses the traditional WWW model to get the content from the Web Sever. The gateway encodes the content and sends it to the client.

- Content- The WAP contents are all binary encoded WML (wireless markup language), WMLscripts or WBMP (wireless BMP) format contents.
- The WAP Protocol Gateway – Translates requests from the WAP protocol stack (WSP, WTP, and WDP) into the WWW protocol stack (HTTP and TCP/IP) and vice versa.
- Content Encoders and Decoders – The content encoders and decoders translate the WAP content into Binary Format to reduce the size over the data network and also to keep the size within the range of the hand held devices.

If the request from the client is for a WML (wireless markup language) document it retrieves directly from the WML content servers. If the request is for an HTML document then the server should have a HTML filter that would convert the html to WML.

Mobile operators usually use the WTA server to push their services like call forwarding, update news and other telephony application to the client. The WTA server directly communicates with the client in WAP protocol stack and pushes their services.

WAP Protocol Stack

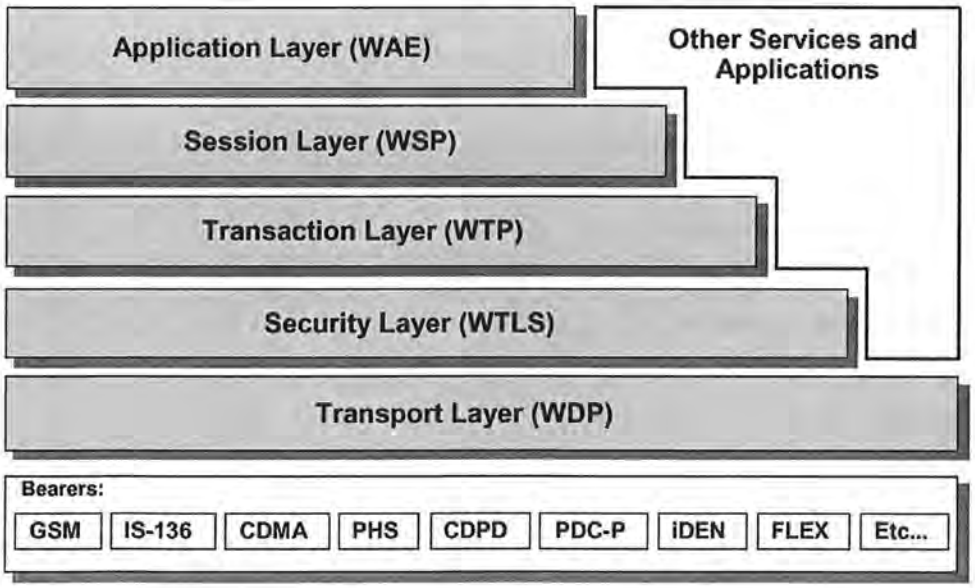


Fig-2

The WAP protocol stack consists of 5 layers over the underlying bearer connection as shown in Fig- 2. Each layer is accessible by the layer above as well as by any other application or service through the use of a well-defined service interface. There are well-defined request primitives by which each layer talks to it's adjacent layers.

Wireless Application Environment

The Wireless Application Environment (WAE) is comprised of a micro browser environment containing the following elements:

- WML – A markup language similar to HTML but optimized for use in hand held devices.
- WMLSCRIPT – Scripting language similar to Java script.
- WTA-Telephony services and programming interfaces.
- Content Formats: a set of well-defined data formats, including images, phone book records and calendar information

The WAE user agent has 2 basic functions: - to request for a url and to display the contents. The contents can be obtained either as a reply to a query or as content pushed in by the server. For the latter the server needs to know the capability of the phone. This is achieved through capability negotiation between WAE user agent and the server. WSP layer provides this functionality.

Wireless Session Protocol

WSP provides the application layer with two types of interfaces. One is a connection-oriented service on top of a transaction layer protocol. The second is connectionless secure or non-secure datagram service (WDP).

The services offered are

1) Long-lived session state.
2) Session suspend and resume with session migration.
3) A common facility for reliable and unreliable data push.
4) Protocol feature negotiation.

The protocol is optimized for low–bandwidth bearer network with relatively long latency.

A typical transaction through the WSP layer would look like Fig-3

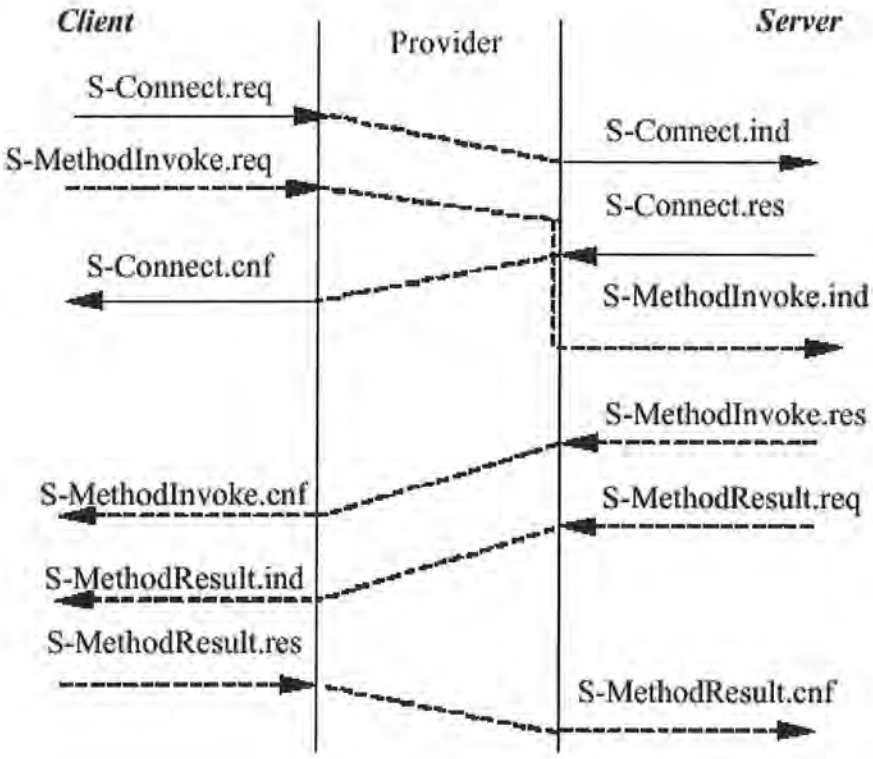


Fig-3

Each transaction would include a connect, content and disconnect request. The WAE in client passes a S-Connect.req primitive to WSP layer. The WSP layer contacts it's peer in the server and passes this request, which is forwarded up to the WAE in server as S-Connect.ind. The protocol requires a response and confirmation for all transactions.

Wireless Transaction protocol.

WTP layer lie over secure or non-secure datagram service in the protocol stack and offers the following functionalities

1) Three classes of transaction service.

   Unreliable one way requests

   Reliable one way requests

   Reliable two way request-reply transactions

2) PDU (protocol data unit) concatenation and delayed acknowledgements to reduce the number of messages sent.

   - WTP is responsible for concatenation (if possible) of multiple protocol data units into one transport service data unit.

3) Asynchronous transaction.
4) Reliability is achieved through unique identifiers, timers, retransmissions and duplicate removal.

   - Since datagrams are unreliable, WTP is required to perform re-transmissions and send acknowledgements in order to provide a reliable service to the WTP user.

5) No explicit connection set up in this layer in order to save overhead. The basic unit of "transfer" is a MSG (message) and not stream bytes. So unavailability of messages is indicated by abort messages.
6) Transaction layer caters to optional asynchronous transfers. Results are sent back when available.

   WTP is specified to run over a datagram transport service and the WTP protocol data unit is located in the data portion of the datagram.

Wireless Transport Layer Security

The WTLS layer provides security for the transactions and also provide protection against accidental and malicious attacks on the server. They typically provide the following functionalities-

1) Data integrity – WTLS contains facilities to ensure that data sent between the terminal and an application server is unchanged and uncorrupted.
2) Privacy – WTLS contains facilities to ensure that data transmitted between the terminal and an application server is private and cannot be understood by any intermediate parties that may have intercepted the data stream.
3) Authentication – WTLS contains facilities to establish the authenticity of the terminal and application server.
4) Denial-of-service protection – WTLS contains facilities for detecting and rejecting data that is replayed or not successfully verified. WTLS makes many typical denial-of-service attacks

harder to accomplish and protects the upper protocol layers. It can be optionally turned on and off.

Wireless Datagram Protocol (WDP)

WDP layer is the WAP transport layer equivalent. It runs over the data capable bearer services, typically UDP is used in this layer.

The datagram transport is required to route an incoming datagram to the correct WDP user. Normally the WDP user is identified by a unique port number. The responsibility of WDP is to provide a datagram service to the WDP user, regardless of the capability of the bearer network type. Fortunately, datagram service is a common transport mechanism, and most bearer networks already provide such a service.

For example, all IP-based bearer's utilize UDP for this service.

Functions of WDP include-

1) Port number addressing
2) Segmentation and re-assembly (if provided)
3) Error detection (if provided)

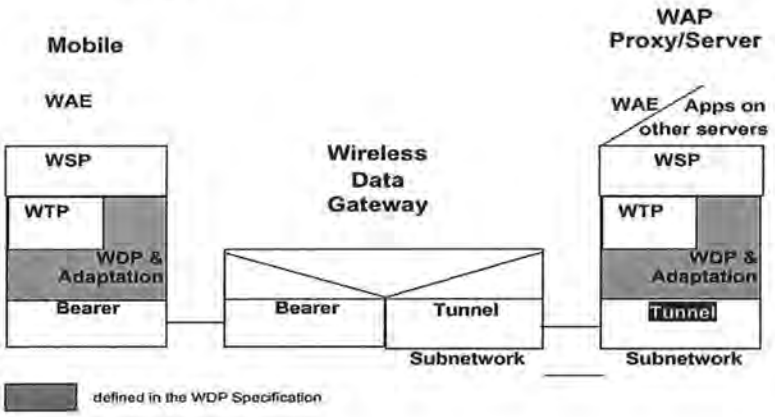


Fig-4

In Fig-4 the shaded areas are the layers of protocol to which the WDP Specification is specifically applicable. At the mobile client (phone) the WDP protocol consists of the common WDP elements shown by the layer labeled WDP. The Adaptation Layer is the layer of the WDP protocol that maps the WDP protocol functions directly onto a specific bearer. The Adaptation

Layer is different for each bearer and deals with the specific capabilities and characteristics of that bearer service.

The Bearer Layer is the bearer service such as GSM SMS, or USSD, or ANSI-136 R-Data, or CDMA Packet Data. At the Gateway the Adaptation Layer terminates and passes the WDP packets on to a WDP user.

The Tunneling protocol is the interface between the Gateway that supports the bearer service and the WAP Proxy/Server. For example if the bearer were GSM SMS, the Gateway would be a GSM SMSC and would support a specific protocol (the Tunnelling protocol) to interface the SMSC to WAP servers.

The SubNetwork is any common networking technology that can be used to connect two communicating devices, examples are wide-area networks based on TCP/IP or X.25, or LANs operating TCP/IP over Ethernet.

The WAP Proxy/Server may offer application content or may act as a gateway between the wireless WTP protocol suites and the wired Internet. If the phone goes out of range WDP is notified. WDP in turn notifies the relevant sessions. The session suspension facility can be used to rejuvenate the session once the phone comes in range.

Bearer

The WAP protocols are designed to operate over a variety of different bearers services, including short message, circuit-switched data, and packet data. The bearers offer differing levels of quality of service with respect to throughput, error rate, and delays. The WAP protocols are designed to compensate for or tolerate this varying level of service.

The bearer network is responsible for routing datagrams to the destination device. Addressing is different depending on the type of bearer network (IP addresses or phone numbers). In addition, some networks are using dynamic allocation of Addresses and a server has to be involved to find the current address for a specific device. Network addresses within the WAP stack may include the bearer type and the address (e.g. IP: 123.456.789.123).

Examples of Bearer : IP, GSM SMS/USSD, IS-136 GUTS

Bearer functions include

1) Routing
2) Device addressing (IP address)

3) Segmentation and re-assembly (if provided)
4) Error detection (if provided)

Power Saving in WAP

The constraints under which the cellular phone work and also the limitations of the wireless network LAN necessitates that data transfer overhead and connection time be minimized. The battery within the cellular phone has finite power capability and it becomes important that the client use it sparingly. Reducing the time it takes to process a request would reduce the time the battery is used. The direct proportionality is evident and can be optimized to conserve the battery energy. The WAP stack is optimized to have the bare necessary transactions only and so no saving on power can be made there. A careful analysis of the various implementations of gateways can help us fine-tune them so that the requests are processed and dispatched as soon as possible.

Kannel WAP Gateway Overview

The potential of the WAP market made all the leaders in cellular market take an active interest in getting the web to the handset. As a result companies like Nokia, Ericsson and others joined hands to form the Wapforum. They came up with the WAP protocol stack and the other specifications. The implementations were undertaken by different companies and as a result there were numerous Gateway implementations. The cellular providers chose a gateway provider and added on the WAP service to their existing service. Since all the gateways were from different companies they had problems communicating with each other. The flexibility obtainable with interoperability was lost in the nascent stage itself.

Kannel WAP Gateway is an open source WAP gateway from www.wapit.com as a result of their endeavor to bring uniformity in gateway servers.

Installing Kannel

Kannel is a WAP Gateway that can be installed on Red Hat Linux or HP UNIX readily. The source code can be obtained from www.kannel.org. Installing the xml library is a prerequisite for Kannel. Libxml can be obtained from xmlsoft.org.The Libxml (version 1.7 or above) have to be installed in the include directory from where Kannel detects it.

The commands for installing the kannel gateway include

Configure - This would check out the configuration were you are running the gateway
Make - would compile the source and install the gateway.

Simulators for WAP compatible phones are available free in the web. The one that is easy to install and use is from www.phone.com. The simulator can be installed only in windows and was installed on a Windows 95 machine. The simulator also requires a Java virtual machine being installed in the machine. The configuration file for running the gateway is provided in the appendix.

The phone simulator was made to request wml files from the gateway and the obtained pages were displayed in the phone simulator. Make sure that the simulator is run in non-secure mode because WTLS layer is not provided in Kannel. Running the simulator in secure mode produces unexpected results.

Load Balancing Overview

The WDP layer is implemented as a software box called bearerbox and is multithreaded. All the other higher layer in the WAP protocol stacks are implemented as a software box called wapbox. The bearerbox is listening to the bearer network for any incoming content requests from clients and the bearerbox is also listening for requests by wapboxes for connection. Multiple wapboxes can connect to a bearerbox. In Kannel the bearer network is IP and the bearerbox is listening to an IP address, port number pair. Inside the bearerbox the programming language structure Boxc is used to keep the details of each individual wapbox.

The Boxc structure for a wapbox contains fields for

1) The logical wapbox id assigned to it
2) The load of the wapbox
3) The connection time
4) The ip address of the wapbox
5) The incoming request list for the Wapbox
6) The outgoing request list for the Wapbox

A list of attached wapboxes is maintained in wapbox_list structure. When a request for a connection (wapbox registers with the bearerbox) arrives from a new wapbox an id is assigned to it. An incoming and outgoing request lists are created for the wapbox. All the Boxc fields are assigned and the wapbox structure is added to the wapbox_list.

The wapboxes can be either idle or be active processing requests. If we are in a low load period the wapbox may be idle for quite some time. The bearerbox wouldn't be sure whether the wapbox is alive or has chosen to retire from service. Maintaining the records of a dead wapbox is a waste of resource and also can lead to undesirable routing problems. Also allocating some processing to an already dead wapbox can lead to incorrect results.

To avoid these, wapboxes periodically send heartbeat messages to the bearerbox saying "I am alive". The frequency of the heartbeat can be configured in the configuration file itself. Since the bearerbox is multithreaded the frequent interruption by wapboxes won't adversely effect its performance. The heartbeat is a good candidate for the wapboxes to tell the bearerbox about its

grievances-how much load it is having. The load of wapboxes can be piggybacked along with the "i am alive " message to the bearerbox.

Routing a message

The bearerbox sender module always listens to any incoming requests from "the outside world". The outside world can be anything-for example IP, GSM SMS/USSD, IS-136.
When a message comes from a client it is forwarded to the route_msg module to route to the proper wapbox. Each transaction would consist of several steps and would typically look like

A) Client -> Gateway
   WTP: Invoke PDU
   WSP: Connect PDU
B) Gateway -> Client
   WTP: Result PDU
   WSP: ConnectReply PDU
C) Client-> Gateway
   WTP: Ack PDU
D) Client -> Gateway
   WTP: Invoke PDU
   WSP: Get PDU (data: URL)
E) Gateway -> Client
   WTP: Result PDU (data: WML page)
   WSP: Reply PDU
F) Client -> Gateway
   WTP: Ack PDU
G) Client -> Gateway
   WTP: Invoke PDU
   WSP: Disconnect PDU

Packets A-C open a WAP session. Packets D-F fetch a WML page.
Packet G closes the session.

For each transaction the wapbox will be in different finite states depending on whether the message is connect request, result request or disconnect request It's quite imperative that all these messages comprising one transaction are routed to the same wapbox lest the wapboxes would find themselves in inconsistent states. The bearerbox maintains a list for associating transactions with wapboxes. The socket (IP address, port pair) from where requests come is alloted one wapbox and all subsequent requests from the same socket are routed to the same wapbox. When a request comes from a socket the history is checked to see whether any previous requests arrived from the same socket. If so the request is routed to the same wapbox it was routed to before. If not, a wapbox is selected at random and the request is send to that after updating the history list.

What exactly is the load to be distributed?

The overheads involved in a page request are

- The negotiating for connect and disconnect are facilitators for actual page request. The features of TCP necessitated a lot of overhead but guaranteed reliable, congestion free transfer in a lossy networking model. In the wirelesss environment reliability is built into the link layer and having them in higher layers is redundant. The bare necessary control transfers are built into the WAP protocol stack and is optimized by the wapforum.
- The communication delay in the wireless network from the client to the gateway and back. Only advances in the wireless arena can cement this and a lot of interest exists among the industry players for doing this.
- The communication delay in the wired net between the gateway and the http server. This also includes the amount of time the gateway takes to process requests. This could be substantial and can be varying depending upon the size of the page request and the load of the http (wml) server. This is a load that we can try to reduce for better power performance.

Factors in Load Balancing

When a request for a page comes, the bearerbox at present selects any wapbox in random and sends it the request. The wapbox in turn does the wml page request from the wml servers.

The processing speed of the machine that runs the wapbox determines the turnaround time of requests. If the wapbox were currently processing many requests it would be hard put to find resources for more requests.

There are 2 primary criteria's for load–balancing to take care of

1) If all the wapboxes have the same processing power then the load should be equally distributed among the wapboxes.

   If there were 2 identical wapboxes then a load mix of 50% each would be ideal.

2) If some wapbox have more resources at their disposal then they should be allotted more work since they would be able to handle those efficiently and fast.

Implementation of Load Balancing

(a) When should the Load Balancing Algorithm be called?

When a connect request packet comes from the client the load balancing algorithm should be called. The load balancing algorithm would pick an appropriate wapbox. The socket of client should be associated with this wapbox id and the pair maintained in the route_info list, route_info. All subsequent requests from the client should be send to the same wapbox by consulting the route_info list.

When a disconnect packet comes it should be routed to the same wapbox and then the socket,wapbox_id pair should be taken off from the route_info list.

Check_disconnect module

Given a message packet this module would convert the binary message into characters. Then it peruses the protocol data unit and checks whether it is a disconnect message and if so returns true. When a packet arrives and if it is a connect request a wapbox would be allocated (by calling the load balancing algorithm) and this information is maintained in the route_info list. If the check_disconnect module returns true then the information is flushed from the route_info list.

How is the load determined?

The WAP application module (wap-appl.c) takes care of fetching the actual wml document from the server. It converts the wml document to binary format and transfers it to lower layers. Whenever a new request comes for a page a fetch_thread is spawned and this thread takes care of the original fetching. A counter, count is maintained that reflects the number of active fetches

being undertaken by the wapbox. This activity, fetching the pages, is in fact the bottleneck and would give an idea of the load of the wapbox.

Whenever a fetch_thread is about to fetch a page it would increment the counter, count by 1.When the page has been fetched the counter count would be decremented. Since all these threads could change the counter simultaneously the count can be in an inconsistent state. The incrementing of the count is a critical section and access to it should be controlled. A count_mutex lock is maintained by the wapbox. Before any thread can operate on the counter it has to lock the count_mutex and release it once it is done. This way the count would always be in a consistent state.

(b) How is the wapbox load passed to the bearerbox?

The WAP application layer maintains a get_load() function that would return the counter, count (after locking and unlocking the mutex). The wapbox before sending a heart beat message would ask for this count from the application module. It piggybacks this load along with the heartbeat message to the bearerbox.

(c) What load balancing algorithm to use?

The bearerbox when it encounters a new message has to decide which wapbox to send the message to. It has at its disposal a list of wapboxes with each entry having the details of the wapbox (id, outgoing and incoming list) along with its load.

We have to select the wapbox with the minimum load from the list. A binary search or a linear search would suffice. We are typically looking at the number of wapboxes in the range of 50-150. A liner search or binary search doesn't have much performance difference in this case. Moreover binary search requires us to have the list already sorted. The overhead involved in maintaining a sorted list balances any inefficiency of linear search.

A linear search of the wap_list was undertaken to find the box with the minimum load. The message is send to that wapbox for processing.

(d) Is the implementation of load balancing justifiable time wise?

Does it take care of the criteria's for load balancing mentioned above?

[See factors in load balancing]

The two factors for load balancing mentioned above are taken care of here. If all the wapboxes have equal processing power the load should be evenly distributed. At a given time instance the counter would reflect the load the wapbox is servicing at that time. If some other wapbox is having less load, the incoming request is passed on to it since the counter for that wapbox will be smaller. Thus equalizing the load. If some wapbox has more processing power they will service the request more quickly and this would be reflected in the load counter (The load counter will decrease faster here). Thus depending on the counter and thus load, more messages will be passed to the less loaded machine.

Server Cache Overview

The client phone communicates with the WAP gateway for getting the contents of a url. The gateway does the original http request, converts the wml document into binary and sends it back to the client phone. The client in fact is not aware of the presence of the gateway and sees it as a part of the service provider. The client can cache the url and it's content, if the client browser allows that. Then if the client wants to access the url again it can get it from its cache instead of doing the expensive get from the web. The cache control header and the time to live field in the page can be used to determine the time the client should keep it in it's cache. The server on its part usually at peak time sees the same url being requested by different clients. The existing implementation does a get from the original server for all incoming url requests. There are sites that get hits of the order of thousands/day. These sites are accessed frequently by different customers. Sites like Hotmail, CNN, Yahoo are just few of these. Lot of time is wasted in accessing the same page from the same server again and again. The time to finish a transaction is directly proportional to the power consumed by the client phone. We could conserve the power by minimizing the time to get the pages. The server maintains a cache which stores the recently accessed urls and it's contents. When a subsequent request for the same page comes, instead of getting the page from the original server the gateway would take it out from it's cache and return it. The page is converted to binary before it is cached. That way the time taken for converting pages is also conserved.

Server Cache Implementation

A cache is maintained in the gateway that would hold the contents requested by the clients for a time period. The cache- control header in the page would allow determining whether to cache the page or not. If the time to live field is specified in the header the page is stored for the specified time else it is stored for a server-specified time.

The cache is implemented as a hash table hashed using the url of the page. The hashing would make storing and retrieving the content from the cache an easy operation.

Hashing

Hashing involves two basic operations. Each page should be mapped to one slot of the table consistently so that the storing and retrieving operation becomes trivial. A hashing function is used to map each page into a unique integer. The hash function used was to sum up the integer value of each character in the url.

The size of the hash table is implementation specific and can be set to reflect the load of the gateway. The integer generated by the hashing function need not be less than the size of the hash table. It has to be tuned to be within the hash table size. Most of the hash table implementations would have different hashing functions but the second operation is mostly the same which is- the integer obtained from the hashing function is divided by the hash table size, CACHE_SIZE and the remainder is taken (modulo operation) as the index into the hash table. This way we are guaranteed of an index within the hash table limits. But many urls can now map to the same hash slot and can lead to confusion. This phenomenon is called collision. To avoid collision each slot is made out to be a bucket that can hold several pages at the same time. Each hash table slot holds a linked list of elements that have the same hash index. The linked list cannot be allowed to grow indefinitely since it might lead to memory exhaustion. The size of each list is restricted by MAX_ELMNTS_IN_SLOT value, which can be set by each installation according to their resource availability.

Cache

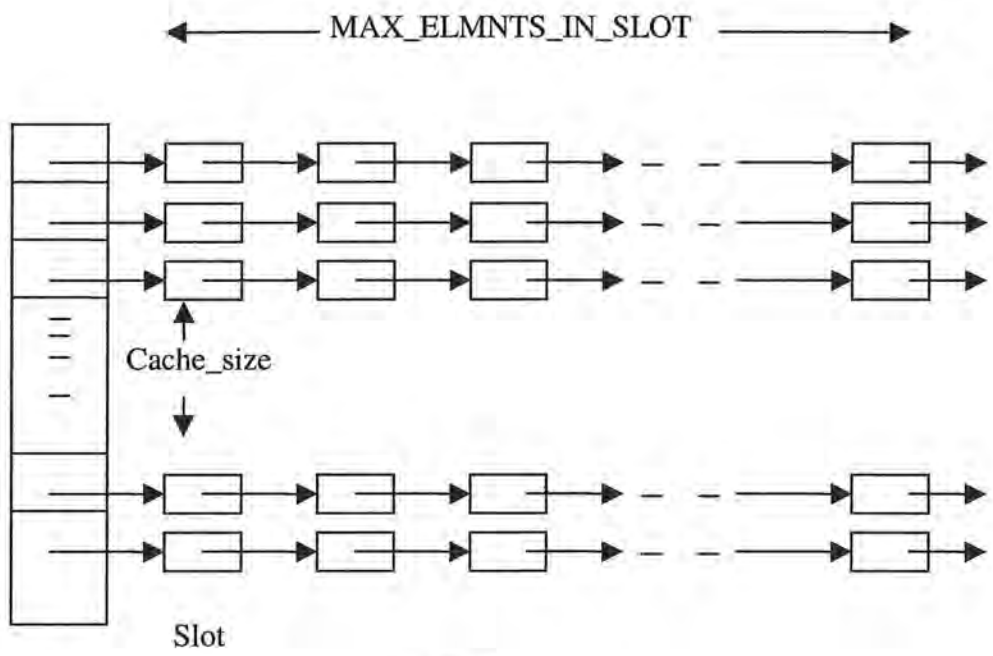


Fig-5

Structure of the cache

The hash table, cache is an array of hash_element (a data structure instance)

Struct hash_element cache[CACHE_SIZE];

Where each hash_element is

struct hash_element{

List *element;

};

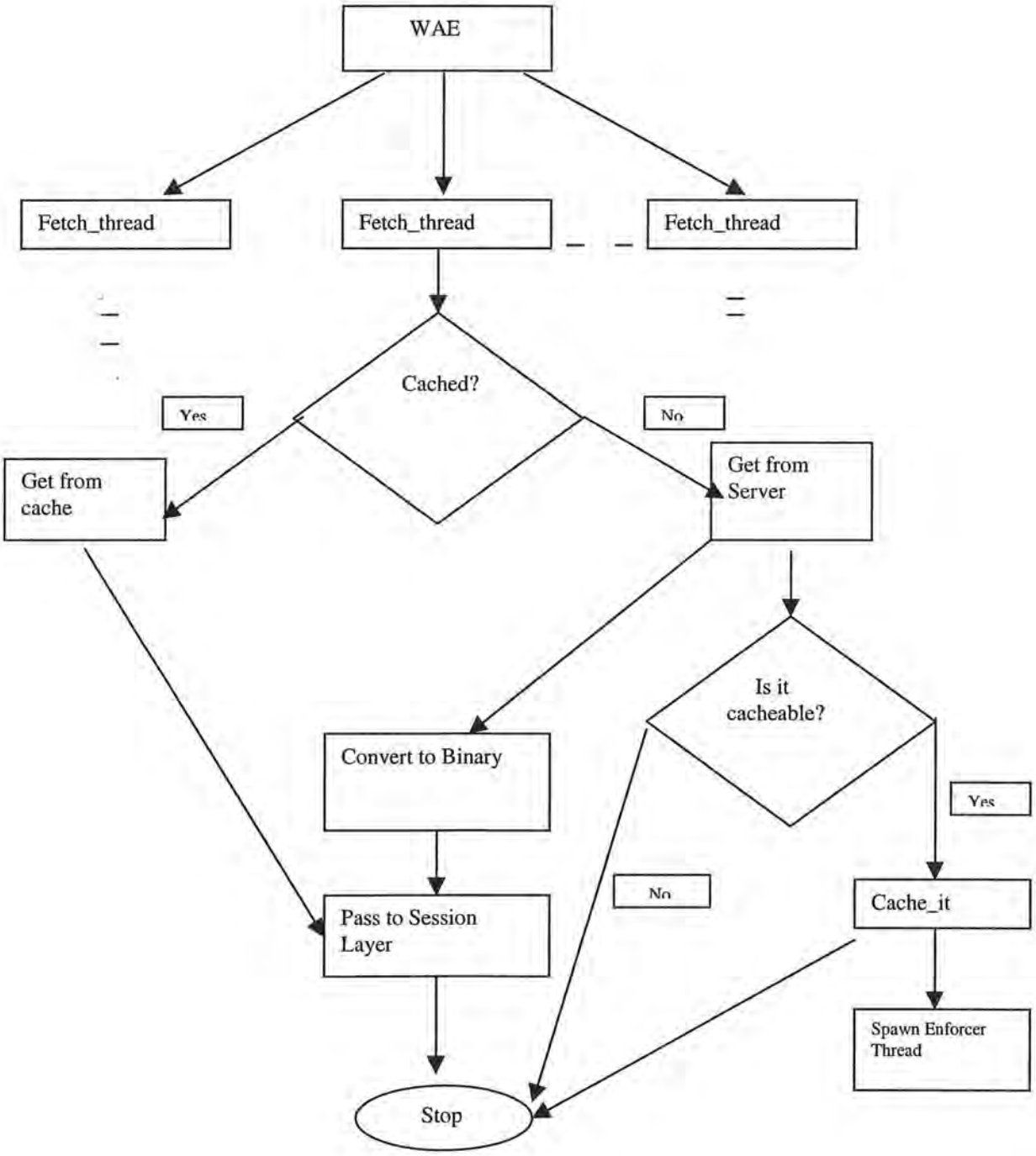
Each slot in the array, hash_element holds a linked list of contents that maps to the cache slot.

The linked list, element, contains elements of the type slot

```
slot {
    struct content *slot_content;
    time_t ins_time;
    };
```

Each slot contains the contents of the page and also the time it was entered into the cache

Flow Chart Diagram for Server Cache



All the fetch_threads being spawned and also the enforcers being spawned may access the cache simultaneously. Protecting the cache by a lock becomes necessary.

Cache_mutex

The cache table is a critical section, which could be modified by different threads at the same time. This could lead to inconsistency in the information extended by cache .To protect against this a lock is maintained, cache_mutex that has to be locked by threads before they access the cache and unlock it when they are through. To see the potential problem consider the following scenario

Thread 1 could be looking for a particular url and found it at the xth slot in the yth hash_element and was about to access it.

Thread 2 is deleting the xth slot in the yth hash_element at the same time.

Thread1 would now access a wrong element or could raise some exceptions.

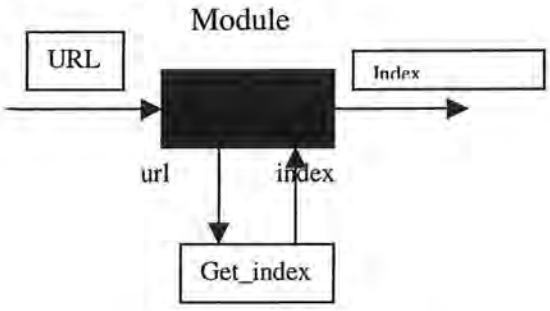
Program Flow for Server Cache

When the WAP application layer receives a request for page from lower layers it spawns off a fetch_thread that is responsible for the fetching of the page from server. The fetch_thread would initialize proper headers and does the page get. The content is converted to binary format and forwarded to the session layer.

I have introduced the server cache in between so that the get from the server would not be necessary if a hit is found in the cache.

The modules introduced for implementing the server cache are

1) am_i_cached module



The fetch thread before doing the http server "get" would call the am_I_cached module with the specific url. The module would do the following steps.

Step1: get hash index from get_index module

Step2: get the linked list at the specified index from the hash table

Step3: get the url from the contents of each slot of the linked list

Step4: convert the url to characters, url1

Step5: compare url1 with the argument url and return the index inside the list where the content is, else return MAX_ELMNTS_IN_SLOT+1

2) Get_index module

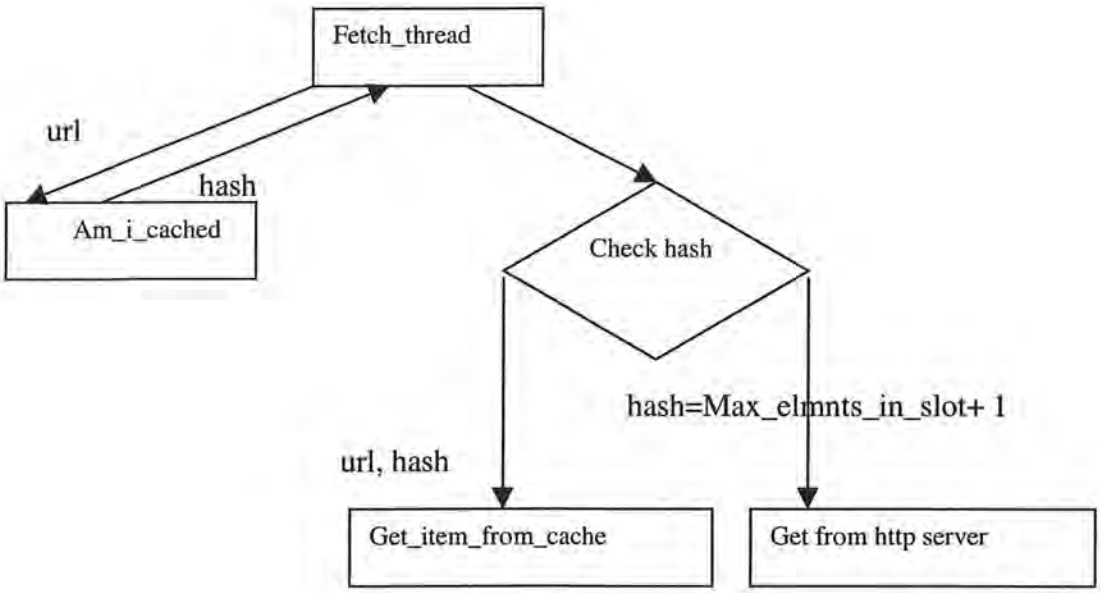
This module implements the hash function for the cache table. It adds up the ascii values of all the characters in the url. The integer obtained may be a huge number greater than the size of the table. The integer is subjected to modulo operation with Cache_size.

For example:

www.abc.com would map to 1062 (w==119,a=67 and so on)

Now if Cache_size were 100 the hash_index obtained would be 62.

3) Get_item_from_cache Module



Get_item-from_cache module would call the get_index module to get the hash_index. The parameter hash would have the index into the linked list where the content is. With both these information the module retrieves the contents and returns it to the fetch_thread. Now the module

would guess that since this content was used now there is a more than fair chance it would be used again. This principle is called the temporal locality. The content is taken off from its present position in the linked list and inserted into the top of the linked list. We should make sure that some other threads do not use the cache across the am_i_cached and get_it_from_cache module. Since we want the index returned by the am_I_cached module to be usable in get_it_from_cache module we can lock the cache across the modules.

We can also make use of the spatial locality and bring to fore all the pages associated with this url to the front of the list.

For example: if you have both www.abc.com and also www.abc.com/def and if you had a hit to www.abc.com you can guess that www.abc.com/def would be referenced in the near future and bring it also to the front of the linked list (I haven't implemented spatial locality since I was not sure about it's advisability).

4) The cache_it Module

If the content is not cached we need to get it from the http server. After getting it from the http server we need to see whether the page is cacheable or not. Parsing the cache control header would tell us whether to cache it or not. If it can be cached we convert it to binary and then call the cache_it module to cache it.

Step1 : call get_index module to get the index into the hash table.

Step2: Is there a linked list already present in this hash_element

If yes go to step4

Step3: Create a new Linked list. Insert the content to the list.

Go to step6

Step4: Insert the element as the first element in the list

Step5: Check whether list length is greater than MAX_ELMNTS_IN_SLOT

If so delete the last element.

Step6: return

In step4 we are inserting the element as the first element to preserve the temporal locality.

The content along with the time it was inserted is encapsulated inside a slot and inserted into the list. Thus a slot has the information regarding when it was inserted. The cache_it module also has information regarding how long to keep the content in the cache. The module can look at the Time-to-Live (TOL) field of the content to ascertain this. If the TOL is not

specified the content should be kept in cache only for a server specified time. The content shouldn't be allowed to stay inside the cache for more time since that is not desired by the content provider. If the content provider has not chosen to dictate how long it should be kept in the cache it is upto the gateway installation to come up with the time. During peak load time the content would be automatically be taken off from the cache when the number of elements exceeds the MAX_ELMNTS_IN_SLOT. At other times the server should take off the content from the cache to guarantee consistent contents for the clients.

5) Enforcer Module

The Cache_it module spawns off an enforcer thread to ensure that the cache does not contain outdated contents. The cache_it module maintains a data structure, eraseme

```
struct eraseme{
 int hash;
 int time_left;
 };
```

It contains the details of the content to be erased. The hash field in this structure holds the hash_index of the list where the content would be. We can't maintain the index into the list where the content can be since the content can be anywhere in the list over a period of time. The time_left field indicates the time after which the content is to be erased from the cache. This time could reflect the time to live field in the header of the page or could be an installation-specified time. I have chosen a default value of 30 minutes.

When the enforcer is spawned by the cache_it module, for each content it caches it is passed some information about the content. The information is, which linked list to operate on (hash_index) and the time after which to be active and erase the content. The enforcer sleeps for the specified time. After it wakes up from sleep it has at its disposal the hash_index where to find the linked list. The content could be anywhere within this list.

The enforcer looks at each content to determine when it was cached. It also determines the current time and calculates the elapsed time of each content in the cache.

Elapsed = current time- time in slot

The enforcer checks whether this time is greater than the time_left field inside the eraser and if so this content is deleted from the list.

Thus the enforcer makes sure that no content stays inside the cache and becomes outdated.

Test Results

Kannel Gateway without Load Balancing and Server Cache

No of WapBoxes	No of msgs	No of Threads	Time taken(s)	Rate (msg/s)
1	10	1	7.0	1.4
		5	3.0	3.3
		10	2.0	5.0
	20	1	14.0	1.4
		10	2.0	10.0
		20	3.0	10.0
	50	1	34.0	1.5
		10	6.0	8.3
		25	5.0	10.0
		50	5.0	10.0
	100	1	69.0	1.4
		25	11.0	9.1
		50	12.0	8.3
		100	14.0	7.1
	200	1	139.0	1.4
		10	22.0	9.1
		50	21.0	9.5
		100	21.0	9.5
	500	1	360.0	1.5
		25	53.0	9.4
		50	54.0	9.3
		100	56.0	8.9
	1000	1	700.0	1.4
		10	103.0	9.7
		25	108.0	9.3
		50	104.0	9.6

No of WapBoxes	No of msgs	No of Threads	Time taken(s)	Rate (msg/s)
2	10	1	7.0	1.4
		5	2.0	5.0
		10	2.0	5.0
	20	1	14.0	1.4
		10	2.0	10.0
		20	2.0	10.0
	50	1	34.0	1.5
		10	5.0	10.0
		25	5.0	10.0
		50	5.0	10.0
	100	1	69.0	1.4
		25	10.0	10.1
		50	10.0	10.0
		100	11.0	9.1
	200	1	139.0	1.4
		10	21.0	9.5
		50	21.0	9.5
		100	25.0	8.0
	500	1	360.0	1.5
		25	61.0	8.2
		50	49.0	10.2
		100	50.0	10.0
	1000	1	700.0	1.4
		10	120.0	8.3
		25	96.0	10.4
		50	98.0	10.2

No of WapBoxes	No of msgs	No of Threads	Time taken(s)	Rate (msg/s)
3	10	1	7.0	1.4
		5	2.0	5.0
		10	2.0	5.0
	20	1	14.0	1.4
		10	3.0	6.7
		20	3.0	6.7
	50	1	34.0	1.5
		10	6.0	8.3
		25	6.0	8.3
		50	6.0	8.3
	100	1	69.0	1.4
		25	15.0	6.7
		50	12.0	8.3
		100	13.0	7.7
	200	1	139.0	1.4
		10	20.0	10.0
		50	18.0	11.1
		100	18.0	11.1
	500	1	360.0	1.5
		25	43.0	11.6
		50	45.0	11.1
		100	45.0	11.1
	1000	1	700.0	1.4
		10	117.0	8.5
		25	114.0	8.8
		50	94.0	10.6

Kannel Gateway with LoadBalanacing

No of WapBoxes	No of msgs	No of Threads	Time taken(s)	Rate (msg/s)
2	10	1	8.0	1.4
		5	2.0	5.0
		10	1.0	10.0
	20	1	15.0	1.4
		10	4.0	5.0
		20	4.0	5.0
	50	1	35.0	1.5
		10	5.0	10.0
		25	5.0	10.0
		50	5.0	10.0
	100	1	76.0	1.9
		25	11.0	8.6
		50	11.0	8.6
		100	10.0	97.7
	200	1	142.0	1.9
		10	21.0	8.0
		50	21.0	8.9
		100	19.0	9.8
	500	1	356.0	1.4
		25	81.0	6.2
		50	47.0	10.6
		100	45.0	10.9
	1000	1	698.0	1.4
		10	96.0	8.2
		25	96.0	8.2
		50	89.0	11.2

No of WapBoxes	No of msgs	No of Threads	Time taken(s)	Rate (msg/s)
3	10	1	7.0	1.4
		5	2.0	5.0
		10	2.0	5.0
	20	1	14.0	1.4
		10	3.0	6.7
		20	2.0	10.0
	50	1	34.0	1.5
		10	5.0	10.0
		25	5.0	10.0
		50	5.0	10.0
	100	1	69.0	1.4
		25	10.0	10.1
		50	9.0	11.1
		100	9.0	11.1
	200	1	139.0	1.4
		10	23.0	8.7
		50	18.0	11.1
		100	16.0	11.8
	500	1	360.0	1.5
		25	44.0	11.4
		50	42.0	11.4
		100	38.0	11.9
	1000	1	700.0	1.4
		10	94.0	10.6
		25	92.0	10.8
		50	92.0	10.8

Kannel Gateway with Server cache

No of WapBoxes	No of msgs	No of Threads	Time taken(s)	Rate (msg/s)
1	10	1	6.0	1.7
		5	2.0	5.0
		10	2.0	5.0
	20	1	11.0	1.8
		10	2.0	10.0
		20	2.0	10.0
	50	1	28.0	1.8
		10	5.0	10.0
		25	3.0	16.7
		50	3.0	16.7
	100	1	53.0	1.9
		25	6.0	16.7
		50	7.0	12.5
		100	7.0	14.3
	200	1	107.0	1.9
		10	15.0	13.3
		50	12.0	16.7
		100	12.0	16.7
	500	1	264.0	1.9
		25	22.0	22.7
		50	34.0	14.7
		100	33.0	15.2
	1000	1	529.0	1.9
		10	53.0	18.9
		25	40.0	25.0
		50	39.0	25.6

Kannel Gateway with Server cache and Load balancing

No of WapBoxes	No of msgs	No of Threads	Time taken(s)	Rate (msg/s)
2	10	1	6.0	1.7
		5	2.0	5.0
		10	1.0	10.0
	20	1	11.0	1.8
		10	2.0	10.0
		20	2.0	10.0
	50	1	27.0	1.9
		10	3.0	16.7
		25	3.0	16.7
		50	2.0	25.0
	100	1	54.0	1.9
		25	5.0	20.0
		50	5.0	20.0
		100	7.0	14.3
	200	1	109.0	1.8
		10	12.0	16.7
		50	11.0	18.2
		100	10.0	20.0
	500	1	265.0	1.9
		25	24.0	20.8
		50	19.0	26.3
		100	18.0	27.8
	1000	1	532.0	1.9
		10	52.0	17.2
		25	36.0	27.8
		50	39.0	24.4

No of WapBoxes	No of msgs	No of Threads	Time taken(s)	Rate (msg/s)
3	10	1	5.0	2.0
		5	2.0	5.0
		10	1.0	10.0
	20	1	12.0	1.7
		10	2.0	10.0
		20	2.0	10.0
	50	1	27.0	1.9
		10	3.0	16.7
		25	3.0	16.7
		50	2.0	25.0
	100	1	54.0	1.9
		25	5.0	20.0
		50	5.0	20.0
		100	7.0	14.3
	200	1	109.0	1.8
		10	12.0	16.7
		50	10.0	20.2
		100	10.0	20.0
	500	1	265.0	1.9
		25	20.0	25.0
		50	17.0	29.4
		100	16.0	30.0
	1000	1	532.0	1.9
		10	54.0	17.0
		25	37.0	27.0
		50	35.0	28.6

Test Results Inference

- Kannel gateway with Load Balancing performs better on high peak periods and with more Wapboxes than the gateway without Load Balancing.
- Kannel gateway with Server Cache performs 50% more efficiently than the gateway without Server Cache.
- Kannel gateway with both Load Balancing and Server Cache performs better than the gateway with just Server Cache on peak periods and with more Wapboxes.

Conclusion

WAP is an exciting technology breakthrough and the full potential is not utilized due to such constraints as clumsy input mechanism, power limitations of the cell phone. The power saving features suggested would extend the battery life considerably. Load Balancing and Server Cache were implemented in Kannel gateway and the gateway tested. Appreciable performance enhancement in the gateway was obtained. Careful implementation of the gateway with an eye on saving power, thereby increasing efficiency, can lead to less computing and hence more battery power.

Future Work

Careful study of implementations can bring up more power saving candidates and those can be implemented. The server cache can have limitations when implemented in real world due to the limitation of memory to hold the cache. The cache can be designed so that part of it would be stored in hard disk (secondary storage) and the other part in memory. When needed the part stored in disk can be brought to the memory. Careful study needs to be done to get a proper mix. Getting from disk is a costly operation and it should be made sure that it doesn't prove to be a bottleneck.

References

Operating Systems Concepts– Galvin and Silberschatz

Introduction to Algorithms – Cormen ,Leiserson,Rivest

www.kannel.org

www.wapforum.org

www.phone.com