Appendix G
University of Zululand

The University of Zululand gained academic autonomy from the University of South Africa in 1970 although teaching on the campus started ten years earlier with 41 students and a staff of 14. Currently the university has 6600 students.

The university has three campuses: its main campus is located 20km south of Empangeni in the province of Kwazulu Natal, South Africa. The remote campuses are at Ulundi and the Cecil Renaud campus in the Durban metropolitan area.

1. Introduction

The University of Zululand has are around 750 computers, including 400 student lab workstations, serving 6600 students. At peak time, there are around 250 concurrent users.

The university is in a relatively poor area, and is not very well funded compared to some of South Africa's other universities.

The University has some on-line courses (using WebCt - www.webct.com). These courses can be viewed at the URL rdisat.uzulu.ac.za:8900/webct/public/show_courses.pl

2. The network

2.1 TENET

Current bandwidth to the Internet is 1152 Kbps, of which there is a 576 Kbps CIR for international traffic, burstable to 1152 Kbps. The connection to the Internet is via the Tertiary Education Network (TENET), which is an academic network similar in concept to the Janet network in the UK and the LEARN network in Sri Lanka. This approach is common to many countries. The TENET network (www.tenet.ac.za) gets its bandwidth via Telkom (www.telkom.co.za).

Tenet usage statistics, including that of the University of Zululand is available at www.tenet.ac.za/mrtg-new/graphs.asp

TENET "shapes" traffic to sites on 3 virtual circuits:

- International Traffic
- National Traffic not from TENET sites
- Traffic from other TENET sites (called Higher Education traffic).

The University's 1152 Kbps circuit is divided into

- Interational VC - CIR of 576 Kbps, BE of 1152 Kbps
- National VC – 192 Kbps CIR, 384 Kbps BE
- Inter-HE VC – 160 Kbps CIR, 320 Kbps BE

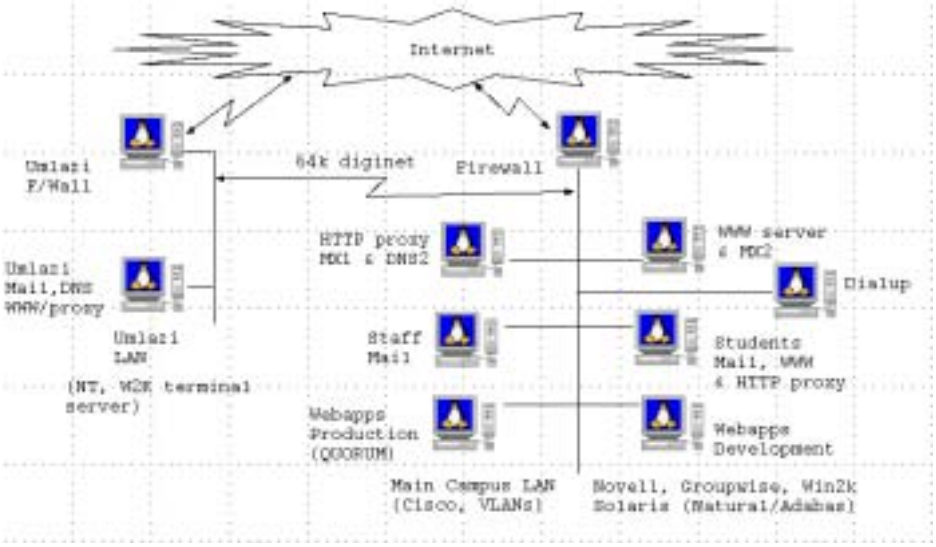
The university pays a monthly rate that is determined by the CIR of each of the VCs. The price for International CIR is about three times the price of National/Inter-HE CIR bandwidth.

2.2 The campus network

The university had Hewlett Packard Unix servers running HPUX. Maintenance of these machines cost R70 000 (around € 7500) per year. These were scrapped, and replaced by Linux servers; even the firewall runs Linux.

There are two proxy servers, one for students and one for staff.

Links to remote campuses are 64 Kbps leased line links.



2.3 Preventing P2P traffic

Only a few users who need to use protocols other than web, mail and FTP e get routable addresses, therefore bandwidth-intensive applications like streaming media or P2P (for example Kazaa) cannot be done.

2.4 Prepaid access

The university has implemented a system of quota based prepaid Internet access, because it was found that "a small proportion of users consumed disproportionate amounts of bandwidth." To limit these users, but still allow for high usage when necessary, it was felt that subsidised charging is the way forward. It was found that the principle of charging, along with good statistics, enabled the IT department to present a case for more bandwidth to management, because it gives them a sense that bandwidth issues are under control and money is well spent. Users are presented with a web page showing their student numbers, usage summary, number of users on-line and total credit remaining (see below), when they open their web browser.

The system is well described in an article by its developer. The article contains many useful points about optimisation, and is therefore included in full (with permission) in section 3 below. In its current state, other institutions wishing to implement this system should have the necessary skills to implement it (including programming skills). However, due to interest from other institutions, the system may be packaged for easy installation.

3. Implementing prepaid internet access at the University of Zululand

Soren Aalto <soren at pan.uzulu.ac.za>
Networking Services, University of Zululand, kwaDlangezwa, 3886

3.1 Introduction

The problems associated with unregulated Internet access are well known to the South African educational community. Given our high cost of Internet access bandwidth, unregulated demand will inevitably exceed the financial capacity of institutions to supply bandwidth to users, resulting in congested access circuits and poor overall quality-of-service.

This problem is especially acute in the context of historically disadvantaged institutions (HDIs) where IT facilities are usually not as well developed as at other educational institutions and IT budgets are severely constrained. Indeed, at many HDIs students do not have general access to the Internet, as these institutions are currently struggling with the financial and manpower requirements of providing adequate access for staff. This lack of student access to the resources of the Internet threatens to widen the quality-of-education divide that exists between HDIs and other tertiary institutions.

At the University of Zululand, we have faced the problem of providing Internet access in our student labs during a time of falling enrolment and ever-increasing pressure on the IT budget. Various considerations led us to a quota-managed/prepaid-usage model for managing use of the WWW and other Internet services. In effect:

- most Internet usage would continue to be subsidized by the university. But there would be a limit on the level of subsidized usage for users and groups.
- costs could be recovered for certain kinds of Internet usage such as personal/recreational usage or other types of usage that would otherwise be classified as abuse.

This paper outlines the motivation, design and implementation of the system we have developed at the University of Zululand for managing usage quotas and prepaid WWW access. This system uses our http caching proxy server together with user authentication to do bandwidth accounting and to enforce per-user and per-group bandwidth usage quotas in a flexible fashion. This system has been designed to do general resource accounting together with session/quota management in a fairly generic way, with a view to extending it to other services besides WWW access -- streaming media, email usage, dial-up access, printing, and perhaps even telephone usage.

3.2 Our dilemma

During 1999, we were faced with a huge latent demand for Internet access in our student labs by lecturers and students at the university. At the same time, student enrolment was declining, which had a direct effect on operational budgets and staffing for our department (networking services). Additionally, the causes of declining student enrolment needed to be addressed -- and it was clear that our failure to provide student Internet access contributed to a negative perception of the university, especially when compared to neighbouring institutions that did provide Internet access for their students.

However, the numbers were not encouraging. We were providing (unregulated but monitored) WWW access to approximately 350 staff members via a 128Kbps access to Uninet. This access circuit was heavily utilised during peak hours, although performance was usually acceptable.

It was clear that the student user base was potentially much larger than the staff population. An Internet access pilot for a single first-year computer applications course involved 730 additional WWW users, although this group was restricted to a single laboratory of 64 workstations. The number of active student users could easily top 3000 at the university, and these students would have access to 350 workstations by early in the 2000 academic year.

Clearly the potential demand from the student user base could be an order of magnitude larger than the current staff demand, which was already saturating our current access circuit capacity

during peak usage periods. However, we were in no position to finance a doubling or tripling of our access bandwidth to provide for this demand. Unregulated student access would clearly lead to appalling quality of service. In crude terms, this left us with the following alternatives:

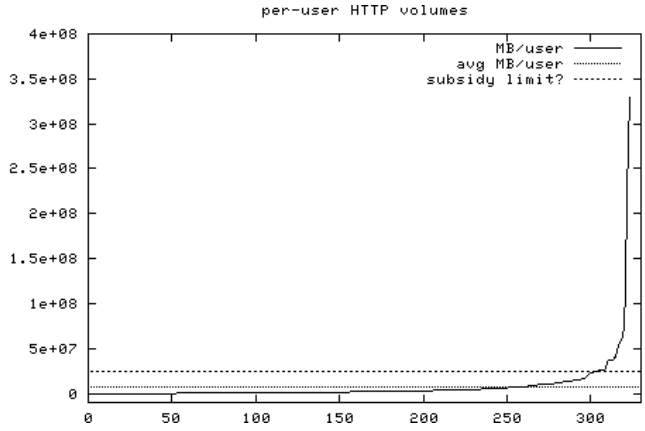
- reduce the demand by limiting usage
- raise money to pay for increasing our access bandwidth

The prepayment/quota model we have adopted allows us the flexibility to do both of these.

3.3 Observations on internet usage

There is no such thing as a "typical user." For a large population of WWW users, you will find a wide variation in how many Mbytes of traffic (HTML pages, images, data files, etc.) different users download in a given period. The distribution of traffic/user is typically very skewed, with most users' traffic tallies being well under the average for the population and a small number of users being way above the average.

The following graph shows the per-user traffic, in Mbytes, for the 324 active users on our campus during the week of 13-19 Feb., 2000:

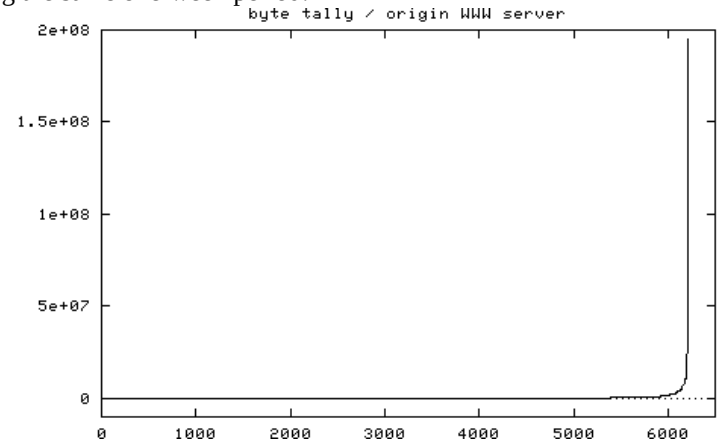


This graph shows is that a small number of users are responsible for a significant fraction of the bandwidth used. In the example above, the top 5% of users account for 50% of the total traffic (2425 Mbytes) for the week. The average volume/user was 7.56MB, with the highest volume user downloading 324 Mbytes, approximately 43 times the average.

The traffic typically associated with abuse corresponds to the area above the dashed line in the above graph.

If the top 5% of users were limited to 25% of the total traffic volume (i.e., no more than five times the current average traffic value), then we would cut our total traffic volume by 25%. Another alternative is that these users would have to pay for usage above five times the current average. This would correspond to a subsidized usage quota for users.

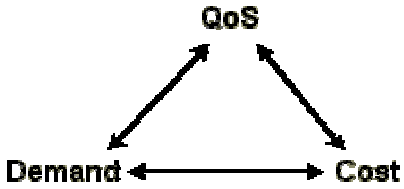
Here is a graph showing how much traffic came from each of the 6214 different hosts accessed during the same one-week period.



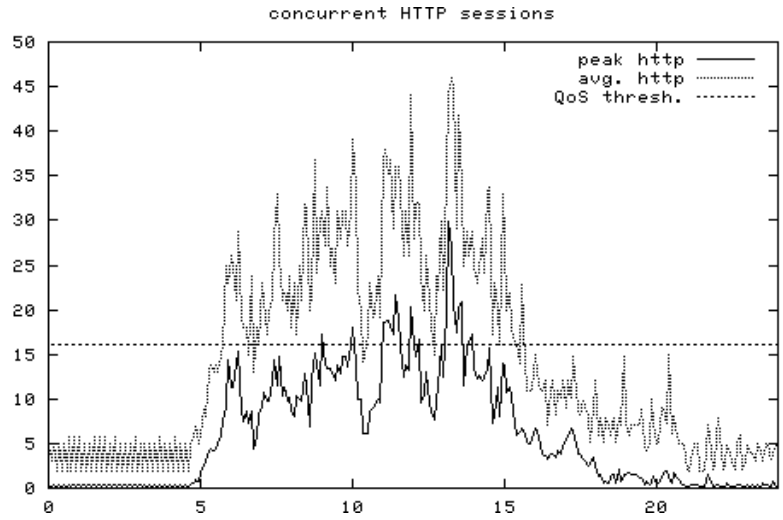
This traffic volume distribution is even more skewed than the per-user distribution, with the

top 5% of hosts accounting for 75% of the total traffic volume. Contrary to our expectations (and popular belief), very few of the high volume sites were identifiable as "inappropriate content," (e.g. pornography) Only 5 of the top 100 sites by volume were obvious pornography sites, which surprised us -- our guesstimates would have been higher. There is a substantial amount of "recreational use" traffic among the high volume sites -- gaming sites feature frequently, as do software downloads, WWW-based email/chat services, news and shopping services. The whole list of 6214 hosts does contain a fairly large number of identifiably pornographic domain names, but these are scattered throughout the demand curve.

Finally, we must bear in mind the relationship between costs, demand and quality-of-service (QoS). Any demand can be accommodated at a fixed cost, if we allow the QoS to deteriorate. In fact, eventually this limits user demand as the service becomes frustrating to use.



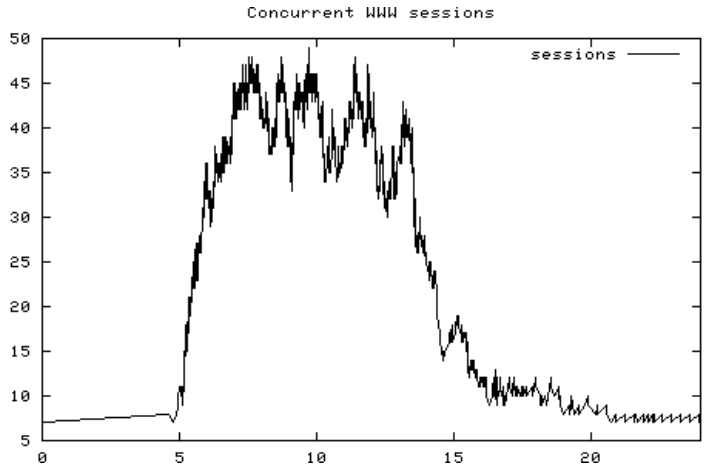
The challenge is to choose an acceptable minimum QoS level and to try and meet this target at minimum cost given the constraints of user demand. As a rough QoS heuristic, we aim for midday peak usage performance where most http connections still exceed 1kbyte/sec (which some might consider poor). In terms of congestion on our access circuit, our 128Kbps circuit can deliver 16 Kbytes/sec of incoming data. Our QoS heuristic would translate into a peak period user demand of 16 or fewer simultaneous http connections on average. The graph below shows the number of simultaneous http connections at any instant through our proxy server during a one day (Wednesday) of the same week depicted above:



The two curves above are plots of the time averaged and peak number of simultaneous HTTP connections computed over successive 5 minute snapshots during the day. We can see that we are mostly meeting our desired QoS threshold of having fewer than 16 simultaneous HTTP connections on average, but that during most of the day there are frequent short periods where this threshold is violated.

This user demand is a function of the number of simultaneous users. The graph below shows the number of concurrent user WWW "sessions" for the same period.

These graphs are disturbing as they suggest that our ability to provide our desired QoS is already strained with a daytime peak of approximately 40-45 simultaneous users. Our worry was that the potential demand from our student user base could easily exceed this by a considerable amount.



3.4 Approaches to managing internet usage

The first question is whether we should manage Internet usage, and if so, why?

Our main motivation for bandwidth and cost management was the financial pressure on our institution and our department. We needed to be able to establish the level of "true" user demand, and be able to meet this demand at an acceptable QoS level. And we had to do this while spending as little money as possible.

Perhaps a very real motivation for implementing bandwidth/cost management was essentially political -- to show management at our university that our Internet resources are being managed in a "businesslike" fashion and that we could correlate "value" with "costs." The simple ability to produce usage information, measure demand and limit abuse would put our department in a strong position when we motivated for future bandwidth (and so cost-of-service) increases. This motivation prejudiced us toward considering our management problem as a fundamentally economic problem, so we were convinced from the outset that some kind of billing system would be implemented.

3.4.1 Content filtering

Some institutions have attempted to combat the abuse problem by using content filtering software to reduce the level of inappropriate usage, such as downloading pornography. Popular content filtering software are packages are Cyber Patrol, Net Nanny and others.

The observations in the previous section show that it would be impractical to try and implement content filtering by hand -- there are simply too many different hosts that would need to be examined, and inappropriate content will be distributed over a large number of these. Content filtering software use central lists of "blocked" sites that are classified by the company that produces the software. One objection to the use of content filtering is that it relies on an external agency to decide what constitutes inappropriate content.

Content filtering is a popular approach to combating Internet abuse because it is the only off-the-shelf software solution for managing WWW access. However, these packages are primarily aimed at the home user market where they are used to keep children from being able to access pornography and other undesirable material.

We were not interested in the content filtering approach as:

- It is not (usually) appropriate to treat our users like children
- Several cases of bandwidth abuse on our campus have not involved undesirable material, and would not have been prevented through the use of content filtering software.
- To our surprise, it didn't appear that inappropriate material is a large contribution to our overall bandwidth demand.
- We have reservations about ceding the responsibility for content control to outside agencies, especially in light of recent legal actions by Mattel, Inc., the producers of CyberPatrol, against two students who published various details about the operation and vulnerabilities of this software. It is clear that content filtering software is intrinsically "closed" in approach, but the hostility of Mattel, Inc. to open source approaches is

worrying.

We acknowledge that content filtering is appropriate for many situations. We would like to offer it as a service, however we would also like to see an "open" approach to content filtering where we could implement our own filters based on content classification information that was available from a central site. This would enable us, for example, to examine log files to see what percentages of existing traffic could be classified as pornography, recreational and other kinds of usage without actively having to block such usage.

3.4.2 Charging for access

The problem with content filtering is that it confuses an ethical issue (inappropriate use) with a purely economic one - how to fund and control the cost of providing Internet access. The real problem is that the demand for bandwidth will inevitably exceed supply as long as the cost of bandwidth to the end-user is zero. The costs of providing Internet access must be passed on to the end user in order to bring demand back to realistic levels. Charging for usage will also recover some of the cost of providing access bandwidth. The funds generated can then be used to improve the overall quality of service by purchasing additional bandwidth.

There are several "charging models" that can be used to recover costs:

flat rate charging

All users are charged a fixed amount, typically monthly. This amount will be based on the average usage.
This model is popular with commercial ISPs for their dial-up customers, largely due to its administrative simplicity. However, it has the following disadvantages:

- most users use less than the average amount of bandwidth. These users end up subsidizing the minority of high-bandwidth users. This tends to make the entry cost of the service a barrier to low-end users.

  In particular, in the HDI environment, many users are very new to the Internet. Our fear was that flat rate charges would exclude many of these users.

- the fee tends to legitimize abuse as the service has now been "paid for," in spite of the fact that high-end users are being subsidized by low-end users.

Commercial ISPs have the advantage that their dial-up customers need to worry about the metered cost of their dial-up connections on their Telkom bills, and this factor tends to limit extreme abuse.

bill for bandwidth used

The alternative to flat-rate charging is to charge users on the amount of traffic they consume, so that users pay for what they use. This is fairly simple to implement, as a proxy server keeps a log of all WWW accesses, and these can be tallied for each user, and a charge computed based on a cost/Mbyte for WWW traffic.

While this approach is undoubtedly fair, it has the following drawbacks:

- there is an impractically large administrative overhead of billing a large number of users and collecting accounts due. Most users will have small accounts that are barely worth the administrative effort of collecting.

- there is a risk of non-payment/bad debt. Our university already has a substantial problem with non-payment of outstanding tuition. There is clearly a risk of runaway Internet usage from students who have no intention of paying for tuition, let alone for Internet bandwidth.

prepaid access

The administrative problems associated with billing are largely alleviated by changing to prepaid access. Users buy "bandwidth" in fixed amounts. These amounts are added to a per-user usage quota that is stored in an on-line database/directory, and the system enforces these usage quotas by not allowing

the user to access the WWW once the user has accumulated bandwidth charges that exceed the user's quota.

Prepayment systems have been successful in other industries, specifically in expanding the customer base to include higher credit risk users of services such as telephones, cell phones, electricity, etc. Prepayment eliminates the problems of debt collection and bad debt risk while eliminating the administrative overhead of collecting accounts. *Users are empowered as they buy as much of the service as they need when they need it.*

However, a prepaid usage system requires two things that are not needed in an ordinary billing system:

- a method of controlling user access so that access is no longer possible once accumulated charges exceed a user's quota limit.
- a secure *token* that can be sold to the user and used to increase the user's quota limit.

3.4.3 Our choice

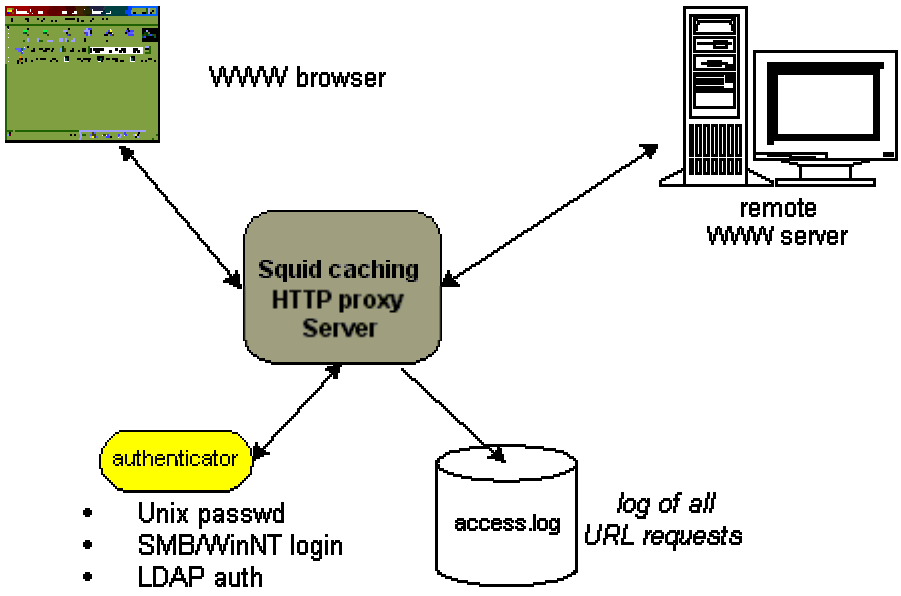
We chose a prepaid usage with partial subsidization model. While all Internet access will be billed for, most normal Internet usage will continue to be subsidized by the university through the provision of quota allotments to departments, research units, courses, etc. Additional usage can be purchased on a prepaid cash purchase basis.

3.5 Implementation - initial idea

Our initial requirement was to provide a method for restricting a user's WWW access based on a user's accumulated charges. We realised that it would be simple to modify the user authentication program used with our WWW proxy to deny user logins when a user had exceeded his/her quota.

3.5.1 The function of a WWW proxy

Like most other institutions, all WWW browsers on our campus are required to fetch WWW pages through a caching proxy server. Rather than a WWW browser contacting a remote server directly to ask for a page, the browser asks the on-campus proxy server to fetch the page on its behalf as shown in here:



The main reason for using a caching proxy server is to reduce the demand for bandwidth by caching downloaded objects on the server. If a user requests an object that has previously been requested by another user at our site, then the proxy server can deliver the object directly to the

user without having to fetch it from the remote server. Eliminating redundant downloads of WWW objects can save anything from 10% to 30% on the level of HTTP traffic into a site. These savings translate directly either into

- cost savings - the site can buy less access bandwidth, or
- improved QoS - the site effectively has an additional 10% to 30% available HTTP bandwidth on average.

Like most sites who have implemented HTTP caching, we did this in order to make more efficient use of their access bandwidth. However, we soon realised that the proxy server provided a valuable central point of administrative control for WWW access as:

- Each URL fetched through the proxy server is recorded in a log file. This log file records the time of access, the size in bytes of the object, the elapsed time required to transfer the object, whether the object was fetched from the remote server or delivered from the local cache, the URL fetched, the IP address of the workstation making the request.

  This information allows one to construct an accurate picture of user demand, usage patterns, etc. It also allows one to easily compute billing information based on bandwidth usage.

- the proxy server can be configured to require users to identify themselves via username/password authentication for WWW access.

We use Squid as our HTTP caching software running on Linux on Dell PowerEdge servers (PII/350, 256MB RAM, 4x4GB F/W SCSI drives). Squid is a popular open-source program for HTTP/FTP caching that runs on most Unix and Unix-like systems.

3.5.2 Controlling access through user authentication

Squid provides for user authentication by allowing the administrator to configure an external program that verifies username/password combinations. Several authentication programs exist for use with squid that authenticate users against Unix/NCSA password files, Windows NT domain logons, LDAP directory services and other methods.

As all the authentication programs for Squid are available as source code under the GNU Public License, we realised that it would be straightforward to modify these to check that a user had not exceeded his/her quota when authenticating a username/password pair. Once a user exceeded his/her quota, subsequent authentication would fail and they would no longer be able to retrieve pages from the WWW through the proxy server.

The handling of usernames/passwords by an HTTP proxy server is somewhat confusing. Once a user's browser has authenticated to a proxy server, it sends the username/password pair on every HTTP request. The proxy server caches these credentials to avoid having to verify them on each request. The proxy is configured to re-verify the username/password credentials after a set interval, typically every 5 minutes, by passing the username/password to the authentication helper program.

What this means, in terms of behaviour is:

- The user is prompted for a username/password when the browser first accesses a page through the proxy server. After that, the browser remembers the username/password and the user doesn't see the authentication dialog again until the browser is shut down and restarted. The user has to log on once per session.
- If a user exceeds his/her configured quota, then within the next five minutes, the proxy server will reject the user's authentication credentials. The user will be unable to retrieve WWW pages from the proxy until his/her quota is increased.

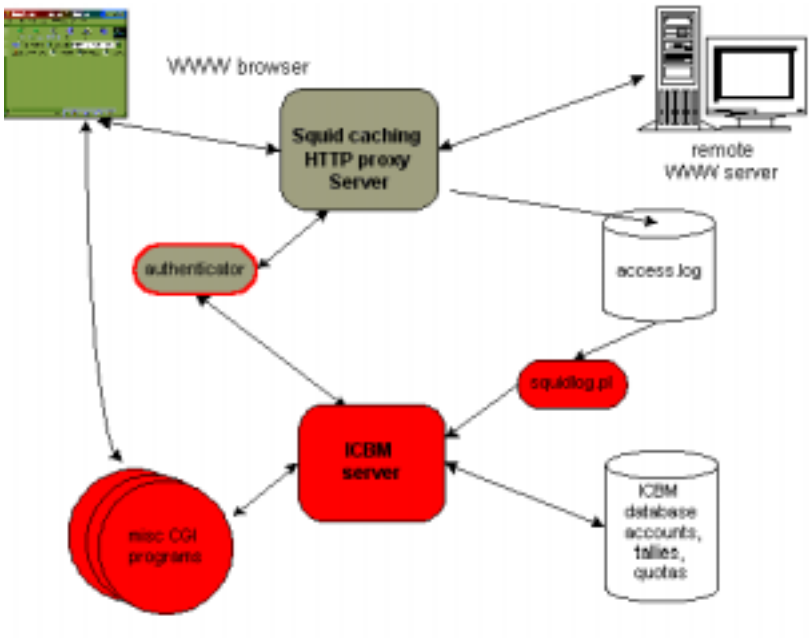
3.6 Prototype development

3.6.1 Initial prototype

An initial prototype of the WWW quota management system was written in C using a DBM database that stored user account, tally and quota information. This prototype consisted of a server that accepted request messages and returned responses over TCP connections. Accounting information was fed into the server from a Perl script that parsed the squid log file as it was produced. The authentication program would then query the server to check whether

the quota for a particular user had been exceeded.

The interactions of these entities are shown in the following data flow diagram:



This prototype was tested on a small group of students, declared a qualified success as a proof-of-concept prototype, and then immediately abandoned. While it did work, we found that:

- Administration was difficult due to problems with the DBM database libraries which didn't allow the database to be updated by another program while the server was running. This necessitated shutting the system down to add new users, for example.
- Billing information was presented to users via their browsers using CGI programs that interrogated the server. The resulting CGI programs were awkward, and necessitated adding more and more request messages to the server's message API, which was a potential source of security problems.

3.6.2 Rewrite in Java

After the initial prototype, we decided on a rewrite with the following requirements:

- We should use a standard SQL database to store account/tally/quota information
- The server should have an embedded HTTP server for the various user reports, billing information and administrative/management interface. The CGI programs of the previous prototype would all be embedded in the server.
- The server should be written in an object-oriented programming language.

C++ was considered as an implementation language, however the above requirements left Java as a much more suitable choice of implementation language.

3.6.3 Why Java?

Java is a small, well designed, object-oriented language with a rich standard class library that satisfied all of the above requirements. It has a well defined interface to SQL databases -- JDBC. Additionally, there is a well defined framework for extending a Java-based HTTP server with servlet classes. In a Java-based HTTP server, servlets play a role similar to that of CGI programs in an ordinary HTTP server in that they are executable entities that produce dynamically generated content. But servlets are far more powerful than CGI programs as they are an integral part of the server and have access to state information in the server that is maintained across invocations.

The Java-based prototype of the server was based on a class library, Acme.Serve, which provided an embedded HTTP server that could be extended with servlet objects and also supported CGI programs. The server used JDBC to talk to an open-source SQL database, MySQL, running on a Linux host.

The main selling point of the Java/servlet approach was that the skeleton of the server, with the basic message API and servlets that displayed information from an SQL database was developed from freely available components in under two days. This initial success completely sold us on the Java servlet/JDBC framework for developing the server.

3.6.4 Basic design

The basic design of the server has two main components. The HTTP server and various servlet classes process all HTTP requests. There is also message-based interface, where request messages from clients and responses from the server are sent over TCP connections to port 3178 on the server. The request/response message format is loosely designed on the URL encoding for CGI programs, which is easy to debug and free format. Each client connection to the server is handled by a separate thread.

Both the message API and all the servlets access three main data structures:

- a list of current sessions
- a cache of active account information from the database
- a cache of active account tally information from the database

Updates to account tally information are made to the cached data which is flushed back to the database at periodic intervals. This alleviates potential bottlenecks that can arise while updating the database, as the hierarchical account structure and structured billing (discussed below) can mean that a single URL can cause the update of perhaps 20-25 separate tallies.

3.6.5 A name for the project

One of the more difficult parts in the early prototype stages of this process was that the server being developed didn't have a catchy name. The first prototype was jokingly christenened ICBM - the Internet Cache Bandwidth Manager, which had the appeal of being some ways from politically correct. After the rewrite in Java, we followed the tradition of adding a 'J' at the beginning of the acronym, giving us the name JCBM. This name seems to have stuck, in the absence of anything better.

3.6.6 Using URL redirection to control access

Another shortcoming of the original prototype/approach was that using user authentication to enforce user quotas was fairly user-unfriendly. A user who had exceeded their quota would have their login refused without any indication of why. A substantial portion of our student user base will be very confused by this and assume that they have either forgotten or mistyped their username/password.

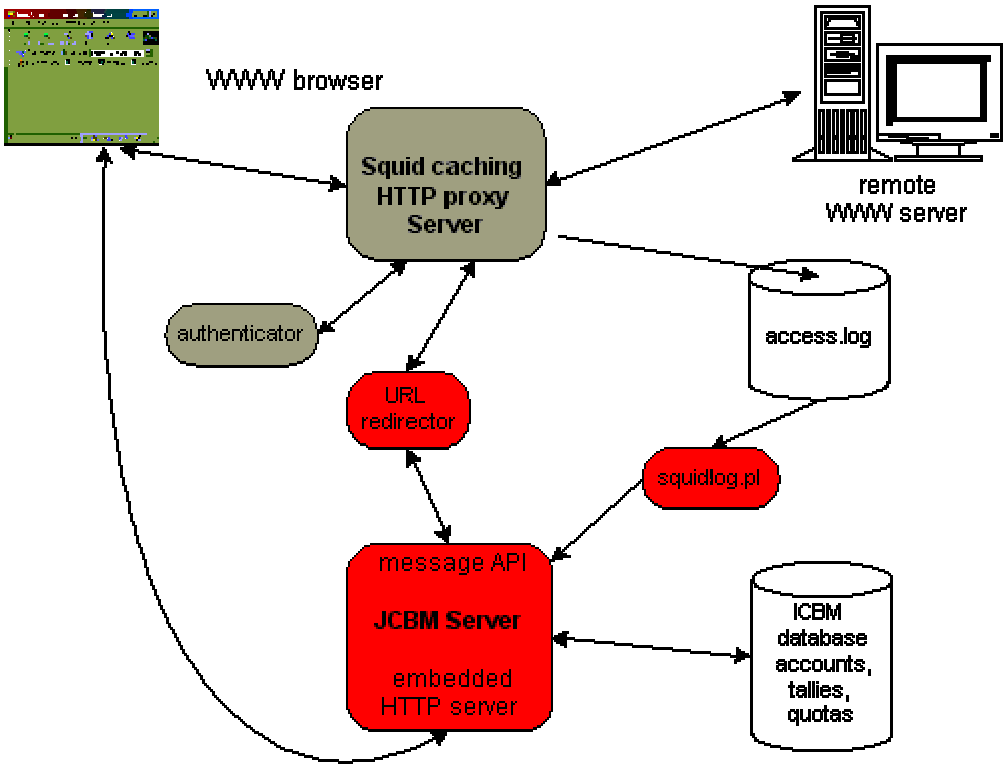
Fortunately, Squid supports another external interface for URL redirection. Squid can be configured to pass each URL requested to an external program that can do one of two things:

- pass the URL back unchanged, in which case the proxy fetches the requested URL normally
- pass back a different URL, in which case the original URL request is redirected to the new URL.

The redirector interface is intended for implementing content filtering -- a URL redirector would typically maintain a database of undesirable URLs, and requests that were found in the database would be redirected to a WWW page containing a notice telling the user that the requested URL has been classified as undesirable.

We realised that URL redirection could be used in a much more powerful way. We made a modification to squid that passes the username as well as the client IP address to the redirector. Then we wrote a redirector that keeps a cache of active session information which is retrieved from the JCBM server. If a particular session exceeds the quota for the account it is being billed to, the URL redirector will discover this when it next refreshes the cached session status. At this point, the URL redirector will redirect all requests to a servlet that informs the user that their current balance has exceeded their quota.

The following diagram shows all the components in this design:

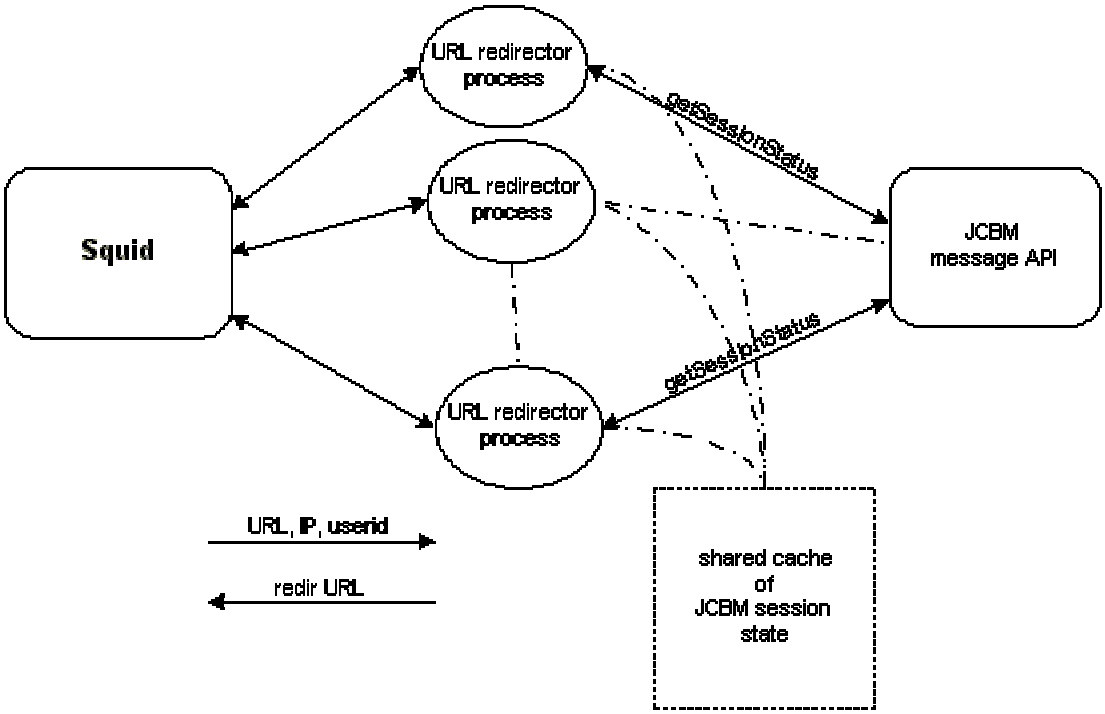


Using URL redirection allows us to handle various aspects of the user interface in a user-friendly way. For example, when opening a billing session, the user may have to choose one of several available accounts to which the session is to be billed. Using the URL redirector, the users URL request is redirected to a page that displays a list of accounts available to the user.

3.6.7 Technical aspects of URL redirector

The current URL redirector was written in C and communicates with the JCBM server over a standard sockets-interface TCP connection. The only complication in implementing the URL redirector is that Squid usually starts up several copies of the redirector program for performance considerations. All these copies of the URL redirector must share a single copy of the cached session information. This was done using SVR4 IPC, namely shared memory segments to store the session info cache and a single semaphore to guarantee atomic update of the session list.

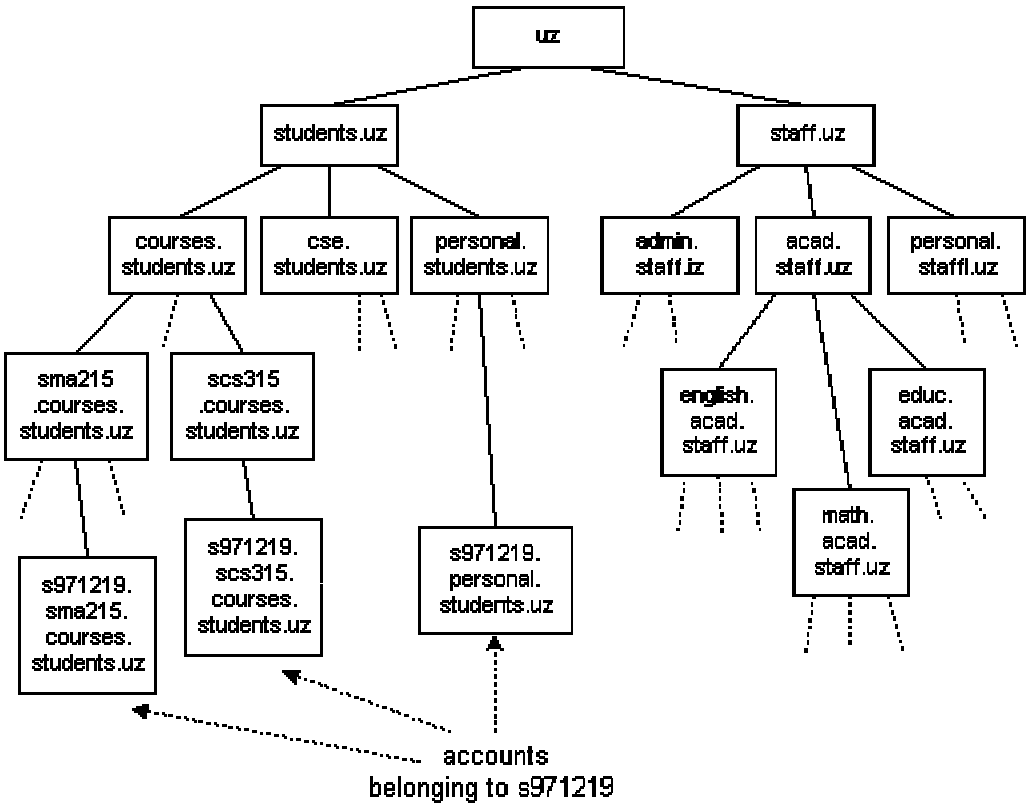
The following diagram shows the several URL redirector processes sharing a common cache of session status information. The Squid proxy server sends requested URLs to the URL redirectors, and these processes may in term request updated session status information from the JCBM server if the cached information for the session making the URL request is stale.



3.7 Current system features

3.7.1 Hierarchical accounts

Billing accounts for the system are organised into a hierarchical structure. In a real implementation, this hierarchical structure is intended to reflect the hierarchical group structure of users. For example, at our university, our users would be divided into student and staff users. The staff users would be divided into administrative staff and academic staff, and these groups would be further subdivided into faculties and departments. Student users would be subdivided into courses.



With this hierarchical structure, a usage quota can be attached at any point in the hierarchy. So individuals can have a usage quota, but each group can also have a usage quota.

In the current implementation it is also possible to administratively enable/disable any part of the tree. So, as an example, it would be possible to disable all student accounts by simply disabling the account uz.students in the above hierarchy. When a portion of the hierarchy is disabled, then any all sessions associated with accounts in this disabled portion of the hierarchy will be marked as blocked. These sessions will be unable to fetch URLs through the proxy server as the URL redirector will then redirect all URL requests for these sessions to a servlet that tells the user that their account is currently disabled.

In fact, the current implementation has a notion of "overriding" the behaviour of a parent account, so for example, while all student accounts could be disabled, one particular course could be selectively enabled. This enables quite fine level administrative control over which accounts have access.

A good example of the application of this kind of access control would be a policy where general student access is not permitted during daytime hours (8am-4pm), but each course is permitted access during its scheduled class times. This might be necessary when bandwidth resources are at a premium to guarantee an acceptable QoS for users in scheduled classes.

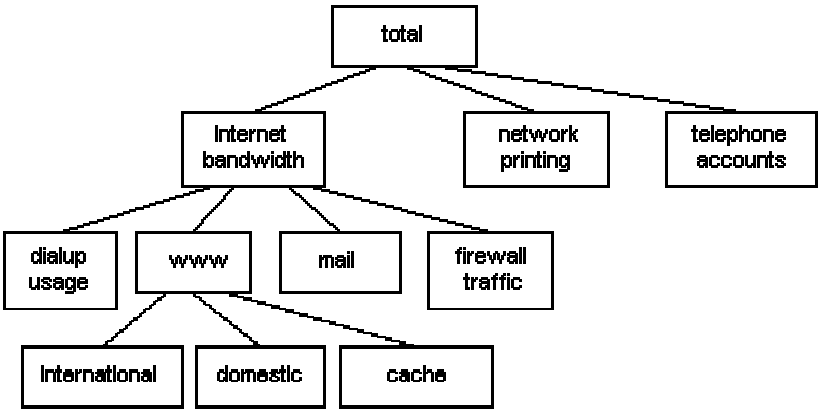
An improvement we would like to make to the current server is to extend the semantics of restrictions on accounts so that these also have time and address restrictions. This would make it possible to attach a restriction to the group account for a particular course so that the account was governed by the general restrictions on student accounts, but was otherwise enabled for access during scheduled class times for the course for workstations in a particular lab.

Each user may have more than one account -- in fact, this is typically the case. In our current implementation, a user will be identified by his/her login on the Linux server that hosts the user's email account. The user's login ID will be the first component of the fully qualified name of each of the user's accounts. For example, a student user might be registered in two courses and additionally have a personal usage quota. The list of accounts for this student would look like:

- s971219.scs315.courses.students.uz
- s971219.sma215.courses.students.uz
- s971219.personal.students.uz

3.7.2 "Structured" billing

A user account can contain billing information for several different kinds of services. These services form a natural hierarchy as well. For example, Internet charges can arise from WWW usage, Email usage, dialup charges and other traffic that comes directly through the border firewall (secure HTTP, streaming media clients, etc.). Each of these categories may be further subdivided. In the diagram below, WWW traffic is divided into International, Domestic and Cache traffic. Each of these types of traffic may be charged at a different rate/Mbyte.



This is handled by having a hierarchical structure of cost codes. Each cost code has a charging rate associated with it, and additionally a discount parameter that can be used to effect off-peak hour usage discounts.

The total charges for a particular cost code will be the total for all of the "children" of that cost code. In the example above, the WWW cost code will be the total of the cost codes for

International, Domestic and Cache WWW traffic.

3.7.3 How items are tallied

When a tallyItem request is sent to the server, it contains a cost code and a quantity (e.g. size of downloaded URL in bytes). The request also contains information that will bind the request to a current session (e.g. workstation IP address and user ID).

The cost code and quantity are used to compute a charge and discount for the request. This charge is added to the current tally for that cost code and account.

The charge then propagates up the hierarchy of cost codes. For example, if the item being tallied was a URL that was classified as International WWW traffic, then the charge would be added to the following tallies:

- International WWW traffic charges
- total WWW traffic charges
- total Internet bandwidth charges
- total charges

The tally is also propagated up the account hierarchy. For example, if the tally was for a session being charged to to the account s971219.scs315.course.students.uz, then the tally would be added to the tally hierarchy for each of these accounts:

- s971219.scs315.courses.students.uz
- scs315.courses.students.uz
- courses.students.uz
- students.uz
- uz

Each time the tally is added to an account, it is propagated up the hierarchy of cost codes for the tallies for the account. In this particular example, this would mean that a single tallyItem request would affect 20 tally items (4 cost codes for 5 accounts) in the database of all tallies. All of these updates take place in the tally cache in the server, and the updates are written back to the database periodically.

3.7.4 HTTP user/management interfaces

Users can see the status (current tallies, charges and quotas) of any of their accounts on-line. These reports are generated directly by servlets in the JCBM server. Since these reports are generated by servlets embedded in the server, the servlets have access to the currently cached tallies and session status in the server. This means that the reports the user sees are a real-time reflection of his/her current charges/tallies/quotas.

There are several servlets that handle the submission of HTML forms for:

- browsing the account hierarchy
- editing account parameters -- administrative enable/disable accounts
- listing all current sessions

3.8 Administrative model

3.8.1 Student usage

Each student user will have several accounts with a separate usage quota for each account:

- For each course that a student is registered in that requires Internet access, the student will have an account and quota for the course. The per-user quota amount will be negotiated with the lecturer before user accounts are created. The quota amount will depend on the usage requirements of the course.
- The lecturer in the course will be able to adjust the quotas for accounts in the group for the course. This will give the lecturer discretion to allocate additional usage for students.
- There will be a usage quota for the group account for the course. The total usage for all accounts in the group may not exceed the group quota. If it does, then all the accounts in the course group are unable to use the WWW.
- Additionally, each student will have a personal use account. Quotas for these accounts will be issued on a prepaid cash basis only.

This provides a fairly flexible model for subsidizing course-related usage, while curtailing

compulsive runaway usage/abuse. It also gives users the freedom to fund their own usage if their needs exceed their quota amounts, or if they are not registered in courses with an Internet access quota.

3.8.2 Staff usage

Staff usage will similarly to student access. Each department will be given a usage quota for all members of the department. The department will nominate an administrator who can assign individual quotas to department members.

As in the case of students, staff will have the option of buying additional quota amounts for their personal accounts, or for "topping up" their departmental quotas.

An important aspect of personal accounts for staff is that the usage details of these accounts will be kept private. Effectively, our appropriate usage policy will be "he who pays the piper calls the tune." Users paying for their own access will be free to access whatever they wish. For departmental usage quotas, we hope that devolving the administration of quotas to a member of the department will make abuse problems easily spotted and peer controlled.

3.8.3 Prepayment

While most quota usage is subsidized as course-specific or departmental usage, there is the option for users to purchase additional quota amounts for cash.

In order to minimise the administrative difficulties of buying additional quota amounts, we chose to implement a voucher-based system similar to that used by prepaid cell phone users. In our system, we print vouchers in fixed denominations (R5.00, R20.00). Each voucher has a serial number and a secret number printed on it and is sealed in an envelope. A user buys a voucher, goes to a WWW browser, and enters the serial number and secret number on the voucher into a WWW form and submits it.

On the server side, there is a database that contains a record of every voucher that has been printed. The database records the serial number of each voucher together with a cryptographic hash of the secret number printed on the voucher. Using these values, a CGI program on the server can verify the authenticity of the voucher without actually knowing the secret number on the voucher. This means that the vouchers cannot be compromised without them actually being stolen. And if vouchers are stolen, then the serial numbers of the stolen vouchers can easily be removed from the database so that they can no longer be redeemed. (Additionally, any stolen vouchers that are redeemed will be traceable as there will be a record of who redeemed them and what accounts they were "deposited to.")

Once a voucher is redeemed, the Rand amount of the voucher is added to the quota for the user's current account and the voucher is marked as redeemed in the database so that it cannot be reused.

For staff users, prepayment amounts can be issued against payslip deductions rather than requiring staff users to buy vouchers for cash. In this case, the prepaid access system still has advantages over billed access in that the user is consciously aware of how much he/she is spending -- as the quota amounts have to be bought in advance. This eliminates possible problems from a compulsive user accumulating a large debt against his/her next payslip without realising it.

3.8.4 Charging models

In our current implementation, the charge for a tallied item depends on two things:

- the cost code for the tally. In the case of a URL, there might be different WWW cost codes depending on whether the URL is retrieved from an overseas server, a domestic server, or the case where the URL is retrieved directly from the on-site cache.
- the current discount in effect for the cost code. The discount can be increased during off-peak usage times to provide an incentive for users to use the WWW during periods where there is more available bandwidth, thereby spreading the demand load more evenly around the clock.

We are still unresolved on the issue of how to compute the charges and time-of-day discounts for different kinds of traffic. By creating differences in various charges, we can provide incentives for end users to shift their demand to less expensive services (download from

domestic mirror sites), or moving to off-peak usage (download large files after hours when the access circuit is uncongested).

3.9 Implementation issues

3.9.1 Browser configuration and rollout

We chose to deploy Netscape communicator in our student labs as it is a single program solution for our student Internet access needs -- WWW browsing and email/news. Netscape configuration, customisation and rollout is well documented and we have a lot of control of the interface our students see.

Deploying Netscape in the student labs required a non-standard installation/configuration of the software. Each user needs his/her own particular user profile, which contains various configuration information such as the user's email address, mail/news servers, http proxy settings and other per-user files such as the user's bookmarks, cookies, certificates, etc. We configured Netscape to look for the user's Netscape profile directory on his/her home directory on the file server. Unfortunately, this particular aspect of the configuration was not documented anywhere we could find -- we relied on a combination of hints from people on mailing lists and comparing times/dates on snapshots of all the files on a machine before and after changing the configuration. Netscape looks for initial configuration information in the file C:\WINDOWS\SYSTEM\NSREG.DAT. This file cannot be marked read-only. If it is, Netscape will not start.

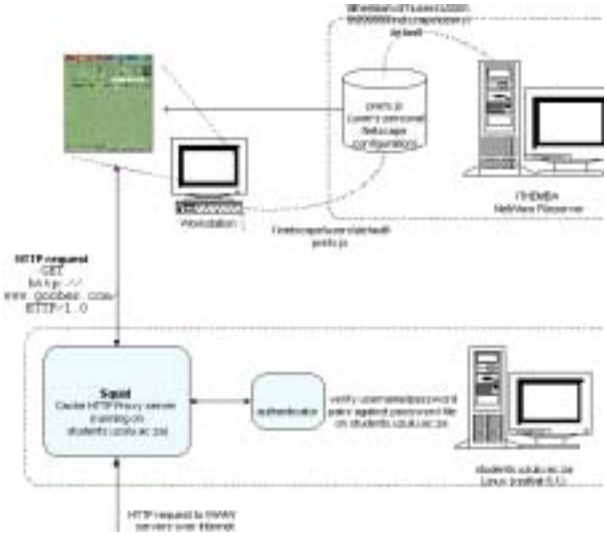
We used ZenWorks/NAL (Netware Application Launcher) to provide an easy way to install the Netscape software distribution at a workstation. After the software is installed on the workstation, we have an ActivePerl script that builds a customised user configuration in the user's home directory. The script picks up the user's student number from an environment variable containing the student's login ID on the workstation, looks up the student's details in a database and generates a customised user profile for that user.

We have replaced the desktop shortcuts and menu items for Netscape so that these point to an ActivePerl script that checks timestamps on files in the user's profile against the template profile for all users. If changes have been made to the template profile, these are incorporated in the user's profile the next time the user launches Netscape. We also removed many troublesome options from the default Netscape installation -- RealPlayer currently will not work through our firewall configuration (as we don't currently have sufficient bandwidth to support RealPlayer), so it not available to students. Neither are several of the utilities that allow students to edit/create/destroy user profiles.

3.9.2 Our particular environment

In our current implementation, all student users have two accounts - a LAN account (Novell Netware) which gives them access to the workstations in the student lab, and a Unix account which is used for IMAP mail and for authentication for the HTTP proxy server.

This is shown in the following diagram



3.9.3 Managing user accounts/passwords

One of the most time consuming (and unplanned for) issues was the issuing of user accounts/passwords. The existing system for student LAN (Novell login) accounts was that all registered students were given an account. These accounts usually had either a null password or a standard password which the students were expected to change when they first logged in. Which they almost never do.

Clearly such a system is open to abuse, and it is clear to us that there are existing problems with students "borrowing" accounts, unregistered students using other students accounts.

However, the incentives for abuse are limited as all students have pretty much the same access to a standard suite of Office applications. This all changes when Internet access is introduced as we have now associated a monetary value with an account. Even in the case of per-course quotas, users will want to be sure that other students can not simply login to their account and exhaust their quota -- which compulsive Internet users will certainly wish to do.

This implied the need for a secure system of issuing account passwords.

Under the current system, each student user has a Novell login which is used to gain access to the workstation. The user then has a second account on the Linux server that handles student email (IMAP) and http proxy authentication and accounting services. Access to either mail or the http proxy (or shell access) will require this username and password.

All registered students have accounts on the Linux server. However, the current system for issuing passwords for these accounts involves the following steps:

- When the student launches Netscape for the first time, he/she is taken to a CGI script on the server that usually generates the student Web home page. However, this CGI script will note that the student does not yet have a password and will redirect the student to a form asking them to open their account.
- This form will ask them to supply their student number and name (for purposes of redundancy -- we already know these details) and other sundry information. When the form is submitted, the CGI script that handles it will pick the next unassigned clear text password from a database table and set this password as the initial password for the account. The CGI script will then return the reference number of this password to the user.
- The user will then take this reference number, together with his/her student card to the networking helpdesk.
- Upon verification of the student card, the helpdesk person will retrieve a printed form with the matching reference number that contains the clear text of the initial password.
- The user then takes this printed form, launches Netscape and is taken to a form that asks the user to change the initial password to one of the user's choosing.

While this system does have potential security vulnerabilities to brute force type attacks, it is a major improvement on the existing system and in principle supplies an initial password to each user without the password being handled in clear text by any of the system administrators. This is necessary to reassure the users that their accounts and usage quotas cannot be tampered with by other users.

3.10 Future development directions

There are several areas of future possible development that have been in our minds since the early stages of development of the system. Some of these possibilities have influenced the existing design of the system.

3.10.1 Security issues

The security of password handling for HTTP proxy authentication is acknowledged to be poor. Usernames and passwords are sent base64 encoded, effectively in the clear. And username/password credentials are sent with every single URL request from the browser to the proxy.

In the long term, this is unacceptable, and one of the advantages of separating user authentication from the mechanism for quota enforcement will makes it easier to adopt a different method for user authentication.

User authentication is part of a broader architectural issue. We are looking at integrating user authentication from our LAN servers (primarily NDS based) and our Unix systems.

3.10.2 Integrating other services

It was the original intent of the design of this system that it would be a general accounting service for a range of different network services. The reasons for focussing on WWW traffic were:

- it comprises the bulk of our Internet traffic
- the cache server produces a detailed and easy-to-parse log file of all WWW activity.

The general design of the JCBM server makes it easy to add accounting and quota management for any service that produces a parseable log file. In the short term we plan to introduce accounting of email usage and dialup usage by writing scripts that parse the log files that are produced by these services. Services that pass directly through the firewall will also be subject to accounting, and log files on the firewall can also be parsed to account for secure HTTP traffic and other applications like streaming media. While we do not currently have the bandwidth resources to offer access to streaming media, these applications will be of increasing importance and we will need to add these into the accounting infrastructure.

We are looking at ways to integrate accounting for network printing, and if possible telephone usage by extracting the information from PABX log file records.

3.10.3 Platform issues

The Acme.Serve class library that the current JCBM server is based on is not a current Java servlet implementation. We are researching more mainstream Java server/servlet implementations with a view to rehosting the current JCBM server on one of these. In particular, we would like to be able to use Java server pages (JSPs) to speed up the development of some of the servlets. Some of the dynamically generated output in the current implementation is still being done with Perl/CGI as this is quicker to write than straight Java servlet code.

3.10.4 Intranet applications

One of the interesting possibilities of using the URL redirector is the implementation of a messaging system in the server. Bulletins to users can be delivered through the browser interface by redirecting URL requests to a servlet that returns queued messages for a particular user or group.

3.10.5 Directory service integration

The hierarchical structure of user accounts duplicates one of the functions of a typical directory service, such as NDS, LDAP or Active Directory. We would like to integrate the management of user accounts into an existing directory service. This is part of a longer-term initiative to integrate the management of aspects of our Linux-based services into a standard directory service.

In order to do this, we need to understand how to extend the directory schema of a standard directory service infrastructure. We are currently researching how to do this with OpenLDAP as the documentation and development tools are freely available for it. The hope is that LDAP integration will enable us to interoperate with the other two major commercially provided directory services, Novell's NDS and MicroSoft's ActiveDirectory as both of these claim to offer LDAP access to their directory information.

The directory service can contain information about what services are allowed for a particular user and which services are blocked. Certain classes of user may have access to secure HTTP or RealAudio, where others may not.

3.10.6 QoS support

Linux has well developed, if not yet well documented QoS support, including traffic shaping and class-based queuing (CBQ). Eventually we would like to be able to offer different QoS levels to different classes of users in the account directory. Users could be billed a different rates that corresponded to different QoS levels.

Directory-enabled QoS support is being promised by the industry. We think that it will be important to be able to offer differential QoS for Internet services at the boundary of the network.

Streaming media and other real-time data streams will be an important area for QoS support. For a site like ours, with limited access bandwidth, audio/video streams will require the reservation of bandwidth at the border firewall. Connection requests that require a guaranteed QoS will need to be accepted/rejected based on (i) whether the user has access to the service in question and (ii) whether sufficient bandwidth can be reserved to guarantee the requested QoS can be met.

High QoS services will need to be billed slightly differently than ordinary data flows.