Pelican DHCP Automated Self-Registration System: Distributed Registration and Centralized Management

Robin Garner – Tufts University Network Operations

ABSTRACT

Pelican is an automated DHCP client self-registration tool. Unlike other popular self-registration tools, the Pelican registration system runs on an independent server with no access to the DHCP daemon's lease data or configuration files. It derives MAC addresses by SNMP-querying network devices and periodically generating and pushing updated configuration files to all participating DHCP servers. This allows Pelican to scale more effectively than most other existing automated registration systems. Pelican also tracks lease allocation on participating DHCP servers, providing administrators with a central repository of data for simplified administration and troubleshooting.

Introduction

Like most other higher education institutions, Tufts University is experiencing an explosive proliferation of mobile computing devices. Nomadic computing, the ability to move devices around and between campuses, is a self-evident requirement: people buy mobile computing devices because they are "mobile" and expect them to work more or less the same way throughout the University.

The Dynamic Host Configuration Protocol, or DHCP [12], is widely used to enable mobile as well as stationary computing. Some institutions enable nomadic computing by allowing anonymous access to their DHCP services but Tufts' IT governing committees have forbidden anonymous access to our networks; all devices must be registered with the DHCP server(s) in order to be granted a lease. We cannot depend on our users to figure out their device's MAC address and type it into a form ("Is that an 'O' or a zero ?"). Nor can we tell them they need to find a tech support person to help them register: our pool of support staff is small relative to the size of our user population and the proliferation of new technologies has put many additional demands on their time. Thus automated self-registration for DHCP service is absolutely necessary.

Given a fluctuating population of between 3,000 and 9,000 users, a network connectivity or registration trouble-report rate as low as 1% on any given day is still a substantial volume of calls. Support staff must have easy, fast access to aggregated DHCP server and registration system data (including robust logging) in order to provide effective and efficient support for self-registration services and nomadic computing.

Our Pelican DHCP registration tool advances the automated self-registration concepts pioneered by utilities like NetReg [1] and AutoReg [2] by decoupling the registration process from the operation of the DHCP server. By functioning independently, the Pelican server can coalesce all registration information and lease allocation data from many participating DHCP servers into a central repository. This repository can then be used to dynamically generate and distribute configuration files back to those servers. This enables complex correlation and reporting facilities that aid in troubleshooting and accounting.

Related Work

Before creating Pelican, we experimented with many other similar utilities. In the commercial sphere, we evaluated Cisco's Network Registrar [3] and Lucent's QIP [4] products. Both aim to provide comprehensive, integrated DHCP and DNS services with advanced administrative capabilities and their target markets appear to be large corporate installation and ISP's. In 1998 when Pelican was being designed, neither supported automated, externally authenticated self-registration and both were nefariously complicated to operate. The lack of registration facilities severely prejudiced us against both products long before we got around to contemplating pricing, supported architectures, and migration issues.

The non-commercial options that we considered included Southwestern's NetReg [1] and R.I.T.'s AutoReg [2]. These utilities combine automatic self-registration and DHCP system monitoring functionality. The target market for both is institutions of higher-education with large residential populations. Both are based on the same fundamental design concepts:

- Named wildcarding.
- Discrimination between known and unknown clients by the DHCP server and use of address

pools to distinguish between registered and unregistered devices.
- Co-location of the registration agent and the DHCP server on the same physical system.
- Direct parsing and manipulation of the DHCP configuration and lease files by the auto-registration system.
- Assumption of the universality of web browsers; administrators expect that all registrants have web browsers and use them frequently.
- Use of a web-based registration form that accepts username/password pairs and attempts to authenticate to a remote service.

In our trial deployment of NetReg, we found it to be stable and easy to use. It simplified the move-in and registration process dramatically, with the unintended consequence of putting half the residential support staff out of work.

The three main components of a NetReg architecture are a DHCP daemon, an HTTP daemon serving the NetReg scripts, and a specially configured nameserver called a "fake-root nameserver." A fake-root nameserver is one which resolves all name-resolution queries to a single address – the address of the NetReg system. This has the effect of directing all network service requests placed by unknown clients to NetReg. The DHCP daemon and HTTP daemon must be co-located on a single server while the nameserver can be located anywhere on the network, as shown in Figure 1.

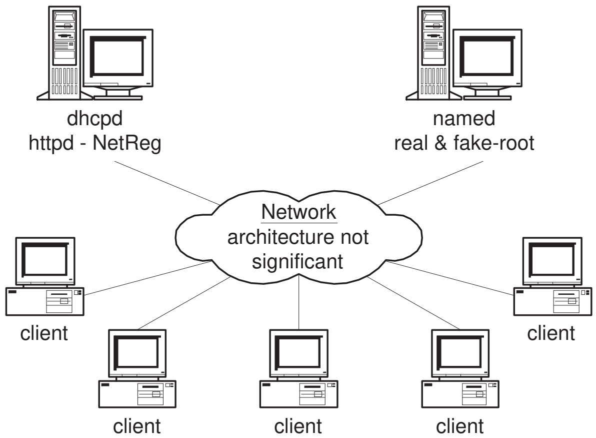

Figure 1: A typical NetReg architecture involves a single DHCP [server, a DNS fake-root server, an arbitrary network architecture, and clients.

```
## NetReg dhcpd.conf
## clients
host foo { hardware ethernet 00:00:00:aa:aa:aa; }
host bar { hardware ethernet 00:00:00:bb:bb:bb; }
host gub { hardware ethernet 00:00:00:cc:cc:cc; }
## subnets
shared-network music-library {
  # KNOWN clients
  #
  subnet 130.64.7.0 netmask 255.255.255.0 {
    option routers 130.64.7.1;
    pool {
      option domain-name-servers 130.64.5.20;
      max-lease-time 86400;
      default-lease-time 86400;
      range 130.64.7.10 130.64.7.240;
      deny unknown clients;
      }
    }
  # UNKNOWN clients
  subnet 10.0.7.0 netmask 255.255.255.0 {
    option routers 10.0.7.1;
    pool {
    option domain-name-servers 10.0.5.2;
      max-lease-time 120;
      default-lease-time 120;
      range 10.0.7.10 10.0.7.240;
      allow unknown clients;
      }
    }
  } # end music-library
# end dhcpd.conf
```

Figure 2: An example NetReg dhcpd.conf file.

Each subnet declaration in the DHCP daemon's dhcpd.conf contains two different pools of addresses: one for known clients and another for unknown clients. A client is known if there is a "host" entry for its MAC address in the dhcpd.conf file. The pool for unknown clients specifies the fake-root nameserver and a very short lease time (five minutes) while the pool for known clients specifies the normal nameserver and a much longer lease time (24 hours). When the DHCP daemon receives a request from a client, it checks to see if the client is known or unknown and allocates an address and nameserver from the appropriate pool.

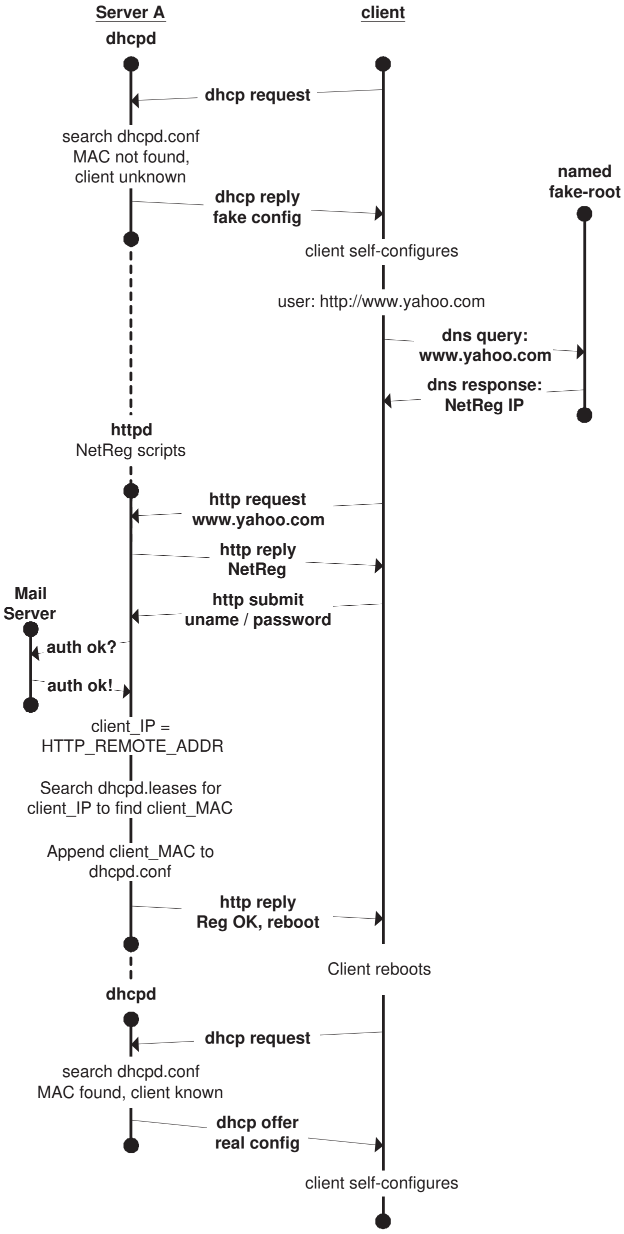

Figure 3: NetReg transaction graph.

In our example configuration in Figure 2, the two pools are drawn from different logical subnets, one a normal publicly routable subnet and the other an RFC1918 private non-routable subnet. Though this is not necessary (both pools can be assigned from a single, routable subnet) we have found that the use of private addresses for unknown clients makes them distinctive and easily identifiable which greatly simplifies troubleshooting. (Supporting two logical subnets on a single LAN requires making a minor configuration change on the routers: a secondary address must be added to each interface, a feature supported by all major vendors.)

A transaction diagram of a typical NetReg registration is shown in Figure 3, with some DHCP-specific steps omitted for clarity. When an unknown client connects, it receives an IP address from the unknown pool, 10.0.7.0/24, and the fake-root nameserver, 10.0.5.2. Any http connections initiated by the client are referred by the fake-root nameserver to the NetReg HTTP daemon, which displays a registration form requesting proof of the user's authenticity. Having verified the user's authentication, NetReg searches the DHCP daemon's dhcpd.leases file for the client's MAC address (keying on the IP address of the HTTP "REMOTE_IP" variable) and appends it to the dhcpd.conf file as a "host" statement. NetReg then restarts the DHCP daemon and displays a web page instructing the user to reboot the client. On reboot, or after five minutes when its lease expires, the client will request a new address from the DHCP daemon and will be granted an address from the known pool, 130.64.7.0/24, and the normal nameserver, 130.64.5.20.

NetReg's simplicity, however, severely limits its scalability: it works very well in a single-server environment but has no mechanism for sharing registration information in a multi-server environment. Our DHCP architecture spans three campuses and involves six servers with overlapping service zones. If we deployed NetReg, we would find that as our users wandered between service zones, they would have to register at each new server to which they connected. Users and their support personnel would find this confusing and inconvenient, particularly given the stunning regularity with which our users forget their passwords. Tracking lease allocation for individual clients would require querying each server individually and collating responses into a useful format would be tricky. These limitations are inherent to the design decision that depends on direct access to the DHCP server's files; they apply not just to NetReg but to all utilities that function in a similar fashion.

Design Specifications

Based in part on our experiences with NetReg and QIP, we formulated the following criteria for a successful automated self-registration system:

- Support for multiple, potentially independent, external authorization and authentication mechanisms.

- Support for more than one DHCP server.
- Maximum ease-of-use: user should need only a web browser, a login ID, and a password on one of the University's authentication systems.
- Automatic prompting to register un-registered devices. No education or tech support intervention necessary.
- Web-based administration interface.
- Alternative supervised registration mechanism ("proxy registration") to support users, visitors, etc. who may not be included in the known authorization scheme or who are unable to authenticate for some reason.
- Support for client data collection that includes user affiliation (for trend analysis, cost accounting, selective culling, etc.), registration date, and expiration date.
- Easily customizable open-source implementation.
- Support for centralized DHCP server lease data collection and aggregation.
- Support for automatic culling of registrations after a configurable interval to prevent the accumulation of stale data without making the user responsible for "de-registering" devices (which would never happen).
- Support for DHCP-style class designations.
- Robust logging.

We analyzed our environment to determine the capacity we would need to accommodate:

- Between 3,000 and 9,000 active registrations at any time during the year.
- An annual influx of roughly 4,500 new registrations over the course of ten to fourteen days each fall.
- Close to a million individual leases, assuming 24-hour lease allocations and a two week data retention rate.

Achieving Scalability: Discovering MAC Addresses

Pelican achieves a degree of scalability impossible in a NetReg-style registration system by decoupling the registration process from the operation of the DHCP system. This allows Pelican to operate in an environment that may include multiple DHCP servers, multiple registration servers, DHCP fail-over, and load-balancing so that, in principle, Pelican can accommodate any size of network and an arbitrarily large client base (Figure 4).

Though a DHCP server's lease file is the most convenient repository of IP to MAC address associations, it is not the only one available. All network devices maintain IP-MAC associations in their ARP tables. Given an IP address, industry-standard SNMP queries to network devices can be used to derive the MAC address associated with the IP. Tufts' network employs routers from three major vendors, all of which support and respond to these queries the same way.

Because the SNMP queries are subnet-based, Pelican must maintain the network address and mask (130.64.7.0/24) of each subnet on which it operates in a database table. This allows Pelican to operate on non-classful subnets like /25's or /18's. Because registration on any particular subnet can be restricted based on user class, the minimum and maximum addresses of each unknown pool are also stored in the database, along with the types of users, if any, that are allowed to register on that subnet. This information is manually entered via web forms.

Pelican does *not* need to know about the network topology; it derives topological information dynamically based on responses to SNMP queries. Pelican must only be given the IP address of one SNMP-enabled router (the "root router") which has routing information for the rest of the network and all routers must be able to respond to SNMP queries from the Pelican server.

To resolve a client's IP address to a MAC address, Pelican determines the client's subnet by comparing its IP address to the Pelican database. It then SNMP-queries the root router and asks if the router has no route, an indirect route, or a direct route to the subnet. A "no route" reply generates an error log message and indicates either a network service interruption or nefarious behavior. An "indirect" response causes Pelican to request the next-hop address on the route. In the manner of traceroute, Pelican follows next-hop addresses through routers until it finds a router that reports back as having a "direct" route, or until it exceeds a configurable maximum number of hops. Pelican queries the router for the interface associated with the route and then for the MAC address associated with the IP address in the interface's ARP cache.

Pelican Components

Implementing Pelican required first making several platform decisions about the software environment in which Pelican would execute.

All Pelican development and operation occurs on UNIX systems here at Tufts, though I imagine that there is no reason it could not be ported fairly easily to a Microsoft platform. All our production servers are Suns running Solaris (v.6 through 8, depending on the system). All are meticulously patched and wrapped by a nocturnal red-haired BOFH.

Pelican uses an SQL-based relational database for data storage. We chose to write Pelican in PHP rather than Perl, not for any important technical reasons, but because we just generally prefer PHP. We chose relatively standard servers: Apache's httpd, ISC's dhcpd, and ISC's named. We briefly flirted with an alternate web server but found that Apache was more reliable. Incidental necessary utilities include awk, ssh, snmp, wget, and cron.

Component Configuration

Pelican was designed as a series of different components which can be run on separate physical systems (Figure 4), but Pelican can also operate on a single machine for small scale deployments and budgets.

Pelican, like NetReg, requires a DHCP daemon, an HTTP daemon, and a fake-root nameserver. Unlike NetReg, each of these may run on separate machines. Additionally, Pelican requires an MySQL database which may also run on an independent machine.

In Pelican, known clients are subcategorized into administrator-defined classes. At Tufts, we assign known clients to either the "student" or "staff" classes based on the registrant's entry in the university's LDAP system, but a default class can be specified as a configuration option.

Pelican's dhcpd.conf file is similar to NetReg's in its use of address pools but there are two key differences. The first is that Pelican includes class specifications that instruct Pelican to match clients to classes based on their MAC addresses. The second is that known clients are identified in NetReg with "host"

```
## Pelican dhcpd.conf

## class definitions

class "staff" {
  match hardware;
  }

class "student" {
  match hardware;
  }

## clients

subclass "staff"   1:00:b0:d0:29:3b:4e;
subclass "staff"   1:00:05:02:01:ea:20;
subclass "staff"   1:00:05:02:00:ff:b5;
subclass "student" 1:00:50:da:09:7f:f7;

## subnets

shared-network music-library {

  # KNOWN clients

  subnet 130.64.7.0 netmask 255.255.255.0 {
    option routers 130.64.7.1;
    option domain-name-servers nameserver.university.edu,
    max-lease-time 86400;
    default-lease-time 86400;
    pool {
      range 130.64.7.200 130.64.7.250;
      allow members of "staff";
      allow members of "student";
      }
    }

  # UNKNOWN clients

  subnet 10.0.7.0 netmask 255.255.255.0 {
    option routers 10.0.7.1;
    pool {
      option domain-name-servers regserver.university.edu;
      max-lease-time 300;
      default-lease-time 300;
      range 10.0.7.11 10.0.7.250;
      allow unknown clients;
      deny known clients;
      deny members of "staff";
      deny members of "student";
      }
    }

  } # end music-library

# end dhcpd.conf
```

Figure 5: An example Pelican dhcpd.conf file.

statements, whereas in Pelican they are identified with "subclass" statements (see Figure 5). MAC addresses specified in subclass statements must have a "1:" prepended onto them.

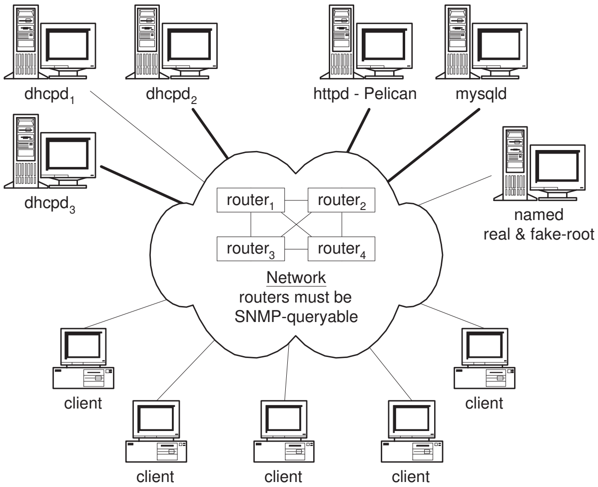

Figure 4: A typical Pelican architecture.

Operation

Pelican has three different and distinct operational processes:

- Client registration.
- Configuration generation and distribution.
- Data collection.

These processes run independently and asynchronously, as shown in Figure 6. A client is properly registered only after processes 1 and 2 have completed one cycle.

Client Registration

Figure 6 illustrates the transactions in a Pelican registration. Unregistered clients making dhcp requests are granted IP addresses from the unknown pool and the fake-root nameserver. The fake-root nameserver resolves all queries to the address of the Pelican server. Clients connecting to the Pelican server are presented with a form inviting them to enter their username and password on any one of three email servers. They are instructed to read the Acceptable Use Policy and indicate their willingness to abide by it by hitting 'Submit' and proceeding with registration.

After verifying that the given username and password correspond to a valid email or shell account, Pelican extracts authorization information from the University LDAP system. The authorization information is used to derive the user's DHCP class ("student" or "staff") and affiliation (department, school, etc.). Pelican confirms that this class of user is permitted to register on the client's subnet by comparing the client's IP address to its dhcp_pool database table. Then the authentication and authorization data is MD5 encrypted and passed back to the client as a cookie

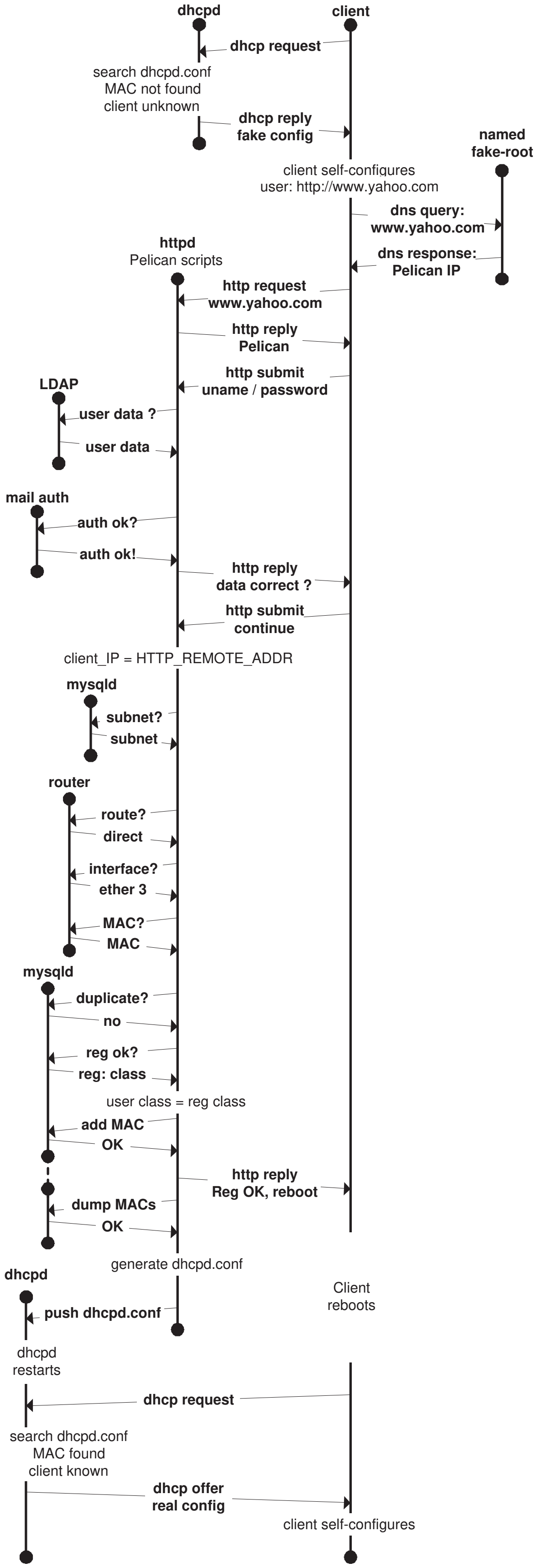

Figure 6: Pelican transactions.

which is used both for session tracking and data storage. The integrity of the cookies is checked at each step of the registration process. The user is presented with a web form displaying the registration data and asking the user to confirm its accuracy by hitting 'submit' to continue the registration process.

Both the authentication and authorization mechanisms are highly configurable: all variables set by the authorization system can be replaced by default values in an environment where an authorization system is either unavailable or considered redundant.

After confirming authorization, authentication, and accuracy, Pelican derives the MAC address of the client and checks the database to ensure that it is not a duplicate registration. The MAC is then inserted into the database along with user's login name, class, and affiliation, and a timestamp and expiration date. The default expiration date is 365 days. Frequently, large areas or groups of clients are registered at the same time (during a cutover, for instance). In order to avoid the inconvenience of having them all expire at the same time, a random number of days between one and 30 is added to each expiration date.

Having successfully added the client, Pelican displays a page asking the user to wait ten minutes and reboot their computer. The client is now registered. Every few minutes, Pelican informs the DHCP servers of all new registrations. On reboot, or after five minutes when its registration lease expires, the client sends an address renewal request to the DHCP server. The DHCP server now recognizes the client as a registered client and declines to renew the original lease. Subsequently, the client requests a new address and is granted a lease for a real, routable address.

In the event that a user is not able to self-register (forgotten password, short-term visitor, etc.), Pelican allows specific users to be designated as having ''proxy privileges'': the right to register a computer on behalf of someone else. In a proxy registration, the proxy registrant may select an expiration interval: one week, one month, three months, six months, or a year.

Operation

The DHCP servers must periodically be informed of +all new registrations in order to distinguish registered from unregistered clients. Every few minutes (every ten minutes at Tufts), a script is run from cron that dumps the Pelican database into an appropriately formatted DHCP class list. The class list is combined with one or more DHCP server-specific dhcpd.conf files and distributed via ssh to participating DHCP servers. The servers are subsequently restarted.

Data Collection

Pelican collates lease information from many different DHCP servers into its database to allow simple centralized searching. DHCP lease file format is not inherently suitable to be read into a database, so we wrote an awk script that reads the lease file and converts each lease entry into a single line of comma-delimited fields that include the FQDN of the dhcp server, the MAC address of the client, start time, end time, and the uid and name that the client supplied when requesting the lease. This script is installed on each DHCP server, along with an instance of Apache's httpd. The httpd is secured and configured to only accept connections from the Pelican server and only allow access to the lease-parsing script. Every few minutes (again, every ten minutes at Tufts), the Pelican server's crond runs a script that uses wget to connect to the httpd on the DHCP server, activate the awk script, and retrieve its output. The leases are read into the database and can then be easily searched and correlated to individual MAC addresses.

Early in development we used Perl to parse the lease file, but found that awk was substantially faster.

Miscellaneous

Automated client expiration prevents the accumulation of stale data in the database. Every week a cron-initiated script is run which deletes any client whose registration has expired.

A web-based administrative interface allows privileged users to view and search the contents of the Pelican database.

Our DHCP servers are configured to grant ''real'' address leases for 24 hours and ''registration'' leases for five minutes. Once the leases have been gathered from the DHCP servers into the database, entries for real addresses are maintained for two weeks while entries for registration addresses are purged after only three days. The purge script is run from cron every night.

Performance

Pelican is ubiquitous at Tufts. It was designed to co-exist with other DHCP infrastructures in order to avoid the necessity of a forklift deployment. We were able to deploy Pelican on a per-subnet or per-building basis, depending on the preferences of front-line support staff.

Deployments

Most of the deployments occurred during the summer or winter breaks when the user population was lowest. Generally, this limited the potential number of registrants to a maximum of 500 at any given time On average, each cutover brought in between 50 and 120 registrants. The vast majority of registrations occurred between 9:00 am and 11:00 am. The highest single-day volume of registrations was 334, with the highest single-hour volume totaling 80 clients (10:00 am to 11:00 am, Jan 12). These numbers are more reflective of human behavior than they are of system performance: registration occurred at the users' time and pace. Determining the maximum effective registration rate would require scripted testing which we haven't done yet, but we do see multiple registrations per second fairly regularly.

During its first twelve months of operation at Tufts, the entire Pelican apparatus (web server, database, management scripts, user interface) ran on a single multi-purpose Sparc10 with dual 60 Mhz processors. In the middle of student move-in week this past September, load jumped from a daily maximum of 334 registrations to 546. While the Pelican system wasn't directly affected, the DNS fake-root had an unintended consequence: our Sparc10 crashed twice due to the load spike that resulted from the web server trying to service all the automated http queries to microsoft.com generated by machines that were connected to the network but not yet registered. At one point the httpd logged 11,000 bogus requests in fewer than six hours of operation. This volume was generated by fewer than 500 clients, some of which queried at one-second intervals. We were forced to do an emergency migration, splitting parts of the Pelican system between two different machines and allowing the web service to be handled by an Ultra250 with dual 300 Mhz processors. The database, scripting, and administrative interface remained on the original Sparc10. We have not experienced any further performance issues.

During the interval between August 5th and September 5th, we recorded 4,106 registrations on our main campus: 1,418 staff and 2,684 students. Across the three campuses, there are a total of 8,000 registered clients.

Naturally, Pelican performance is highly dependent on the environment: SNMP performance is affected by the architecture and load of the routers in the network. SQl database transactions are affected by the load and configuration of the server. Our SQL server runs on a multi-purpose dual 60 Mhz processor Sparc10. Our network is a heterogeneous collection of Cisco and Foundry routers, some of which do no traffic shaping/filtering and others of which are well occupied in that respect. Packets-per-second loads on the routers also vary widely. There are several wide-area links running at speeds between 1.54 Mb and 10 Mb, multiple gigabit backbones, and LANs generally operate at 100 Mb. Consequently, SNMP statistics tracking the speed with which MAC addresses can be determined are highly context dependent.

With this in mind, these are the statistics we've collected:

SQL insertion speed:

```
High:  43ms
Low:    5ms
Median: 5ms
Avg:    6ms
```

High insertion lags correspond to times when the insertion overlapped with a cron-initiated database dump.

The number of routers traversed is not as significant an indicator of performance as the cpu utilization on those routers. At Tufts, the longest times for MAC determination arise from registrations that traverse two particular routers that are also heavily occupied with filtering and policy enforcement. The shortest times for MAC determination arise during registrations traversing four other routers that do nothing except move packets.

Average total time of execution for the script is under 150 ms, most of which may be attributed to the MAC determination process.

85% of the errors logged by Pelican are due to browsers not having cookies enabled. Though Pelican notifies users of the problem, it would appear that not all users understand what it means to "enable cookie support" or at least are not able to do so without assistance. We logged 410 instances of disabled cookies during 2891 registrations, most of which were corrected by users after a single notification.

10% of the errors were duplicate registrations: people who tried to register their client twice. This may occur when a user fails to reboot their computer after registering and waiting the recommended interval or because the client fails to relinquish network settings when it receives a new DHCP configuration after registering.

The remaining 5% of errors (109 during three weeks of operation) were an assortment that included expired cookies, access denied by the central directory for administrative reasons, and access denied based on user class.

We have seen no evidence of users attempting to tamper with the authentication and authorization data in the cookies.

On average, with 8,000 operating clients and a 2-week retention, our lease table contains 550,000 entries. Retrieving 3600 records from this table based on a partial-string comparison of an IP address ("172.16.237.%") takes 7.5 seconds on a moderately loaded Sparc10.

Operation and Impact

Getting Apache, MySQL, and PHP compiled properly (with LDAP support if you want to do directory lookups is the most challenging aspect of installing and maintaining Pelican. We note with distress that the process does not appear to be getting easier with time or new package revisions.

Nevertheless, Pelican has been running very reliably at Tufts for thirteen months. Network administrators love it: the number of static address requests has dropped from eight per day to eight per week. Because all lease and client data are stored in database tables, we have the ability to cross-join against our ARP log database to identify stolen IP addresses. The ability for users to trivially move around campus has allowed us to enable more advanced network services, including wireless installations.

It has been very well received by end-users as well: they no longer need to get support assistance to connect a new computer to the network and they can travel easily in and between campuses.

Front-line support staff have found the transition from static addressing to DHCP and class-based access control challenging. Most have absorbed the technical concepts reasonably quickly, though about 5% continue to have difficulty. Most seem to feel that the additional challenge is mitigated by the reduced burden of address requests. Some resent the loss of control and security now that users can roam without their explicit acquiescence.

Additionally, because the University has a distributed support infrastructure, it can be tricky to figure out who is responsible for providing support to users when they are roaming: does support have geographical constraints? How does a support person from one organization and area of campus become familiar with the network landscape elsewhere in order to assist a roaming user? Who, if anyone, provides support for public ports (where Pelican registration is not supported, but roaming for registered clients is) ? How is support handled in areas where students (who have a separate support infrastructure) and staff commingle ? How do users identify public ports? Though we anticipated some of these questions when we initiated development, others caught us by surprise. Our answers are still being developed. Ultimately, by forcing us to reconsider our support model and social landscape, Pelican's impact on Tufts may go far beyond its technical footprint.

Future Work

One of our design assumptions about Pelican was that we would be interested in only a fairly minimal set of data about each client: its MAC address, to whom and by whom it was registered, when it was registered, and when it would expire. Once deployed, it quickly became clear that transactional information about the client was also interesting: its IP address at the time of registration, when it was re-registered, when its user or responsible party was administratively changed, and when it was deleted. Support for retaining some of this information was quickly glommed onto the existing schema, but a more efficient schema would involved two separate tables: one with the minimal client data necessary to construct a dhcp configuration (MAC address and class), and a second table that stored related information about the client (who registered it, when, etc.). Future work will explore the feasibility of the type of schema as well as the possibility of abstracting the database calls in order to support other database engines.

Also, during the initial development of Pelican the decision was made to use cookies to store some of the data necessary to complete registrations because that required the least time to implement. We believe that moving to server-side session tracking will give us more flexibility and reduce our exposure to security risks by hiding more operational details.

Related future work will tie Pelican in with our ARP monitoring system to allow us to identify miscreants who steal addresses without registering. If we get really fancy, we may attempt to implement SNMP-set statements to automatically disable switch ports or block MAC address with selective filters.

Finally, we hope to see Tufts move to an integrated, directory-based authentication/authorization mechanism sometime this year – maybe even with support for digital certificates. Pelican's authentication and authorization code is modular and will be modified as necessary to adapt to advances in related University systems.

Availability

The Pelican scripts, along with our notes on compilation and installation, are available at http://www.net.tufts.edu .

Acknowledgments

Pelican was produced by the author and Peter Radcliffe with invaluable input from Marc Jimenez and Keith Malvetti. All are members of the Tufts University Network Engineering team. Special thanks to Alva Couch for his editing and advice and to Judi Rennie and her team, who helped us troubleshoot under pressure. Thanks also to Mark Mason for his assistance and patience.

Author Information

Robin Garner has been working in the networking field since 1988 and has been a Network Engineer at Tufts University since 1998. Robin can be reached via email at robin@net.tufts.edu or by U. S. mail at Tufts University; 169 Holland St.; TAB 303; Somerville, MA 02144

References

[1] Valian, Peter & Todd Watson, ‘‘NetReg: An Automated DHCP Registration System,’’ *Proceedings LISA XIII*, Usenix Assoc., 1999.

[2] Campbell, Matt, ‘‘Automatic DHCP Registration,’’ http://www.rit.edu/~mrcsys/dhcp/dhcp98, 1998.

[3] ‘‘Cisco Network Registrar,’’ http://www.cisco.com/warp/public/cc/pd/nemnsw/nerr/index.shtml .

[4] ‘‘Lucent QIP,’’ http://www.lucent.com/products/solution/0,,CTID+2011-STID+10016-SOID+767-LOCL+1,00.html

[5] Valian, Peter, ‘‘NetReg 1.2,’’ http://www.southwestern.edu/ITS/netreg/ , 1999, 2000.

[6] Graves, Rich, ‘‘Review of DHCP Registration Systems,’’ http://www.unet.brandeis.edu/~rcgraves/dhcp.html , 1999.

[7] Dromas, Ralph & Ted Lemon, *The DHCP Handbook: Understanding, Deploying, and Managing Automated Configuration Services*, Macmillan Technical Publishing, ISBN 1-57879-137-6, 1999.

[8] Ablitz, Paul & Cricket Liu, *DNS and BIND, Third Edition*, O'Reilly & Associates, ISBN 1-56592-512-2. 1998.

[9] Langfeldt, Nicolai, *The Concise Guide to DNS and BIND*, Que Corporation, ISBN 0-7897-2273-9, 2001.

[10] Stevens, Richard, *TCP/IP Illustrated, Vol. 1*, Addison-Wesley, ISBN 0-201-63346-9, 1994.

[11] Beck, Robert, ''Dealing With Public Ethernet Jacks – Switches, Gateways, and Authentication,'' *Proceedings LISA XIII*, Usenix Assoc., 1999

[12] ''Internet Software Consortium Dynamic Host Configuration Protocol (DHCP),'' http://www.isc.org/products/DHCP/ .