

WAKE FOREST

UNIVERSITY

Winston-Salem, North Carolina, USA

LDAP for Systems and Network Administration

Wake Forest University needed over 1000 new DHCP scopes and DNS information for multiple network "views." By coupling custom LDAP schemas with Perl scripts, we edit data for our network once, rather than editing data for each service. Scripts fetch and reformat data for DHCP and DNS. Information is application-independent: new reports and applications use the same data, and import scripts for Excel spreadsheets and DNS zones change data globally.

Perl scripts pull LDAP information for configuration files. We create private DNS information through views. Hosts are public if their LDAP record has a public address or a static NAT address. Dynamic zones are created for DHCP networks. Web scripts let administrators add host names and IPs to LDAP. Tools verify that LDAP data isn't stale or malformed.

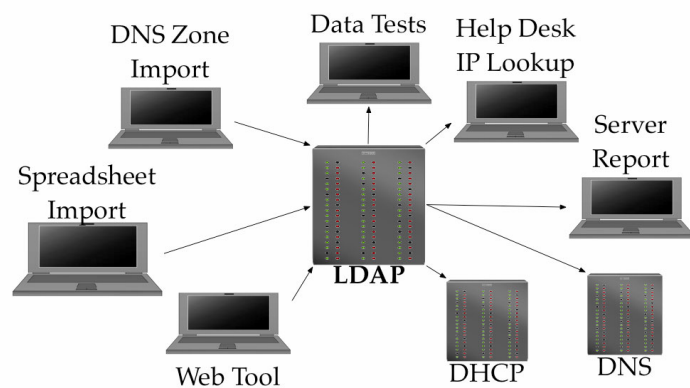
Our infrastructure can be further extended to solve new problems by creating additional schemas. We're planning to store our network ACLs in LDAP so that routers, firewalls, and servers can have identical port-level access rules. LDAP has proven highly flexible; it provides a simple, secure solution to the administrator's dilemma of redundant configuration information.

John Borwick is a systems administrator at Wake Forest University specializing in Perl, LDAP, DNS, and sendmail. His goal is to make administrators' lives easier by finding or creating tools to automate and standardize common processes.

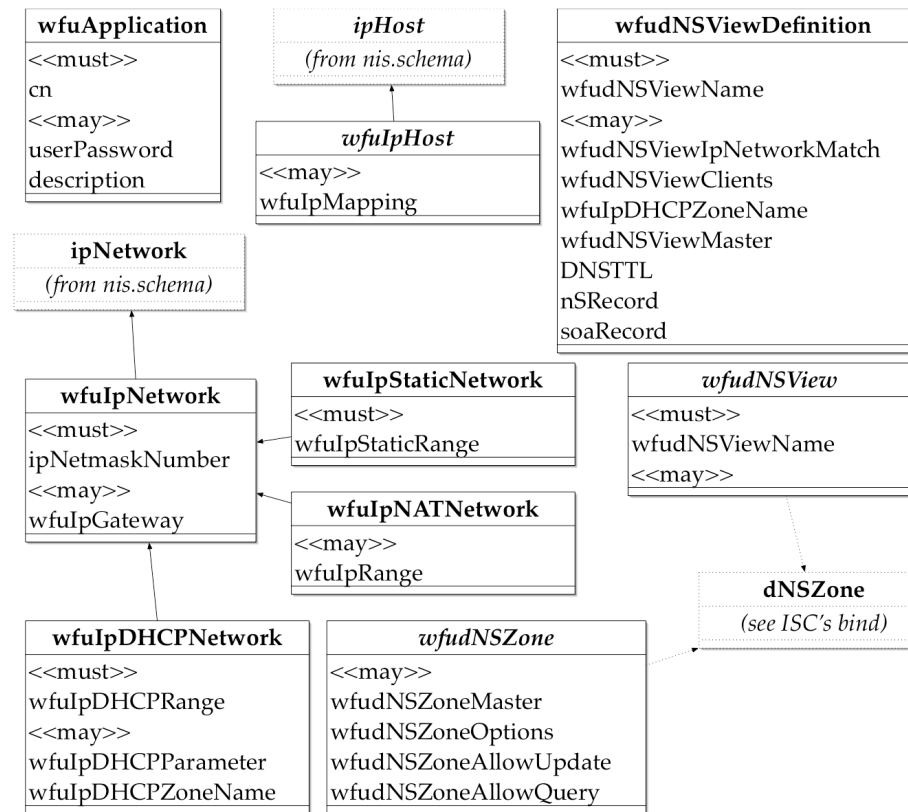
Deployment

So far, the principal technologies we use are OpenLDAP 2.2.15, Perl 5.8, Net::LDAP, and ISC's bind and named. The tools we have written include

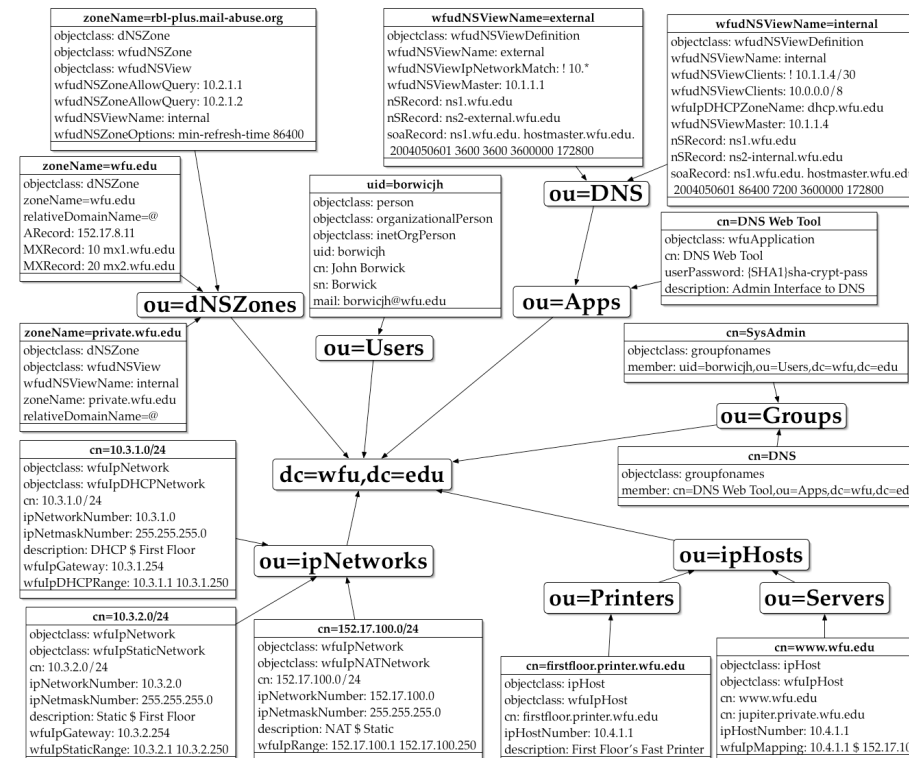
- Spreadsheet Import: Networking's IP address scheme was stored in a 100 column Excel document. Perl scripts parsed the document each time the design changed.
- Web Tool: Administrators can find subnets, add new machines to DNS, classify machines for report purposes, edit DNS entries, and search for arbitrary machine data.
- DNS Zone Import: Departments that could not support multiple DNS views needed us to integrate their zones. Perl scripts parse DNS files to add machine data.
- Data Tests: LDAP only has minimal support for validating data. Test suites can process LDAP entries to check their content and structure.
- Help Desk IP Lookup: Debugging a network with 1000 subnets is hard. This CGI script returns the gateway, netmask, and location for IP addresses.
- Server Report: A CGI script returns an Excel-compatible CSV file for all machines classified as "servers", suitable for distribution to our managers.



Schema



Structure



Net::LDAP

Net::LDAP is an excellent Perl interface to LDAP written by Graham Barr. Its OO design resembles the DBI module. Net::LDAP supports everything that we've needed to do with LDAP, and appears capable of much more—including server-side sorting, virtual list views, SSL, and SASL.

To the right you can see an example based on Net::LDAP. It searches through our directory for all machines with a "wfuIpMapping" attribute, which describes how internal IPs translate to external NATs in our firewall. It then prints out the mappings. This script could be used to verify that all firewall mappings are correct.

```
use Net::LDAP;
our $LDAP = Net::LDAP->
  new( 'ldap.wfu.edu' );
$LDAP->bind( 'cn=Manager,dc=wfu,dc=edu',
  password => 'PASSWORD' );
my $s = $LDAP->search(
  base => 'ou=ipHosts,dc=wfu,dc=edu',
  filter => '(wfuIpMapping=*)',
);
while( my $e = $s->shift_entry() ) {
  my @names = $e->get_value('cn');
  my %external_ip = map { split /\s*/ }
    $e->get_value('wfuIpMapping');
  print "Mappings for @names:\n";
  print "INTERNAL $e ->";
  foreach ( keys %external_ip ) {
    print "\n";
  }
}
```

DHCP

It was easy to generate dynamic DHCP configuration files: we simply iterate through all our DHCP networks and print out data as we find it.

"ipNetworkNumber", "ipNetmaskNumber", "wfuIpDHCPRange", and "wfuIpGateway" attributes are formatted and inserted in "dhcpd.conf". The corresponding "in-addr.arpa" DNS zones are added by expanding subnets into a set of class C addresses and entering each class C as a zone.

We store the DDNS keys in LDAP to make sure they are synchronized between the DNS and DHCP configuration files. We store the OMAPI keys in case we need remote DHCP control.

```
# configuration file defaults
preamble();
# OMAPI keys are stored in LDAP...
omapi();
# as well as DDNS keys.
my ($key) = get_keys( 'DHCP' );
process_key( $key ) if $key;

my $default_zone = default_dns_zone();
print qq(ddns-domainname "$default_zone";\n\n);

# add "zone" statements for all DNS zone names
foreach my $zone ( $default_zone, dns_zones() ) {
  process_zone( $zone, $key );
}

# add "subnet" and "zone" statements for scopes
foreach my $subnet ( dhcp_subnets() ) {
  process_subnet( $subnet, $key );
}

# expand /22's, etc. into class C dynamic zones
foreach my $class_c
  ( class_c_networks($subnet->[0], $subnet->[1]) ) {
  process_zone( ip_to_dns_zone( $class_c,
    '255.255.255.0' ), $key );
}
}
```

DNS

The DNS configuration file generator must create "named.conf" as well as all zone files. It must determine whether a zone should be presented in a DNS view, and whether individual records should be added to a view's zone. Simple A and PTR records come from "ou=ipHosts"; more complex records like MX and DNAME come from "ou=dNSZones". Zones without SOA records receive the default from "ou=DNS,ou=Apps". DNS also generates dynamic zones as needed by DHCP.

```
# adds "key" statements to named.conf:
my @keys = find_keys( 'DHCP' );
process_key($_) foreach (@keys);

# goes to "ou=DNS,ou=Apps,dc=wfu,dc=edu":
foreach my $view ( find_views() ) {
  # searches through "ou=dNSZones"
  my @view_zones =
    view_zones( view => $view, dhcp_keys => \@keys );
  # put configuration in named.conf:
  view_write_config( view => $view,
    zones => \@view_zones,
  );
  # create an actual zone file from "ou=ipHosts"
  # AND "ou=dNSZones"
  populate_view_zone_files( zones => \@view_zones,
    view => $view,
  );
}
```

Reports

By making LDAP the authoritative store of configuration information, we can run accurate reports about our system configuration. To demonstrate the power of standardizing on LDAP, a CGI-based Excel report for management was created. The report lists all of the IP addresses and names for each server, along with an optional description, location, and server manager. A separate tool was developed for the Help Desk to determine an IP address's network and return the information we know about that network.

Future network and host information can be incorporated through auxiliary schema to the same record that holds DNS information. For example, an "wfuIpHostHardware" objectclass could be created to describe the model and serial number of a given "wfuIpHost".

```
print <<"_HEAD_";
Content-type: application/vnd.ms-excel
Content-disposition: filename="stats.csv"
_HEAD_

my $s = $LDAP->
  search( base => $BASE,
    filter => '(objectclass=device)',
  );

print delimited("Names", "Ips", "Description" );
while( my $e = $s->shift_entry() ) {
  my @names = $e->get_value('cn');
  my $ips = $e->get_value('ipHostNumber');
  my $description = $e->get_value('description');
  print delimited( ( join ' ', @names ),
    ( join ' ', @ips ),
    $description,
  );
}
```