An Autonomic ENUM Implementation in Network Simulator-2 *

Sandoche BALAKRICHENAN
Institut National des Télécommunications
9, rue Charles Fourrier
91000 Evry, France
sandoche.balakrichenan@int-edu.eu

Monique BECKER
Institut National des Télécommunications
9, rue Charles Fourrier
91000 Evry, France
monique.becker@int-edu.eu

ABSTRACT

This paper presents the implementation of an autonomic ENUM simulation model which is validated with real measurements. It explains the approach used to build the model and integrate with NS-2. The main objective for this work is to use this simulation model to use different configurations or algorithms in ENUM system and come up with a promising solution to reduce the global response time for an ENUM query. For ENUM to be used for VoIP solutions, their response time should be as compared to the real time telecommunication systems; otherwise it is not going to be successful. We are convinced that this model could also be used to study different implementation models for a particular ENUM scenario and identify an optimized implementation.

Categories and Subject Descriptors

D.3.3 [**Model Validation and Analysis**]:

General Terms

Measurement, Performance, Verification

Keywords

DNS, HMM, Model, ENUM

1. INTRODUCTION

Personal communication identifiers (such as phone number, fax number, email address, instant messaging etc.) are normally stored and distributed on business cards. In today's anywhere/anytime communication world, it will be more convenient to have these identifiers on a public database on the Internet so that it can be accessible at all time without having the necessity of caring the business card or worrying losing it. Also if there are any changes in these identifiers, it can be updated without taking the trouble of reprinting the cards.

In such cases, one needs an *unique* identifier which may never change. All the other identifiers can be placed in a public database on the Internet, which can be queried by anyone having the unique identifier. Internet already has a large public distributed database; the Domain Name System (DNS), whose main functionality is to translate IP address to domain names and vice versa. DNS is a database of host information. If the same database could contain other information, it will be possible to add the personal communication identifiers to it.

The new IETF protocol, Electronic NUmber Mapping (ENUM) [1] is a technology that uses DNS to translate telephone numbers (telephone numbers are unique) into a set of information (email address, SIP phone number etc..), which are normally used as personal communication identifiers. The most important application of ENUM is that it enables the convergence of traditional telephony to IP telephony.

Though ENUM is a very innovative idea of convergence between the PSTN and the Internet, it has its own drawbacks which need to be studied. Most important of them are; security, privacy and the DNS Resolution Delay (DRD). DRD is the time taken by the DNS cache server to obtain a response for a query.

The ENUM architecture is similar to DNS, but a DNS study cannot be used for ENUM, since it has its own requirements for the DNS infrastructure. This is partly because of the volume and type of data and partly due to the service level expectation of customers used to PSTN performance. A ENUM deployment is successful only when the DNS services are capable of scalable performance, availability, reliability and security that is available to classical voice services. Furthermore the success of the ENUM will depend on the ability of the DNS to give response times similar to the ones given by the real time databases in the telecom world. A brief overview of ENUM is presented in section 2.

As indicated in [2],[3] real world systems are too complex to be studied automatically, and these models must be studied by means of simulation. We developed a simulation model which is based on an empirical French ENUM model. The measurements done on the empirical model are explained in section 3. The implementation of the model on an existing

*This work was supported by a French National Research Project : RNRT Numerobis

simulation tool, NS-2 (Network Simulator-2) [4] is explained in section 4. Any simulation model is considered erroneous if it is not close to the actual system [5]. We have validated the simulation model which is further explained in section 5 and finally in section 6 we give a brief idea of how the simulation model developed can be autonomous.

2. BACKGROUND

ENUM uses DNS to find available services for a given telephone number. These services are encoded in so called Naming Authority Pointer Resource Records (NAPTR RRs)[6]. The telephone number is converted to a Uniform Resource Identifier (URI) [7], which is used to retrieve the NAPTR RRs. The processing of ENUM query is as follows (Fig:1) .

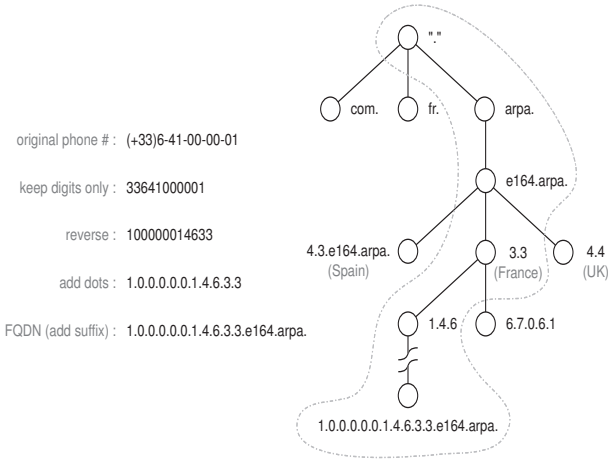

Figure 1: ENUM Tree

An ENUM compliant device such as phone or VoIP Private Branch Exchange (PBX) translates the telephone number into domain name by reversing the digits and putting periods between each. The resulting string is appended with *.e164.arpa.*, which is designed specially for providing ENUM services. For example the telephone number +33 641000001 becomes an Fully Qualified Domain Name (FQDN) 1.0.0.0.0. 0.1.4.6.3.3.e164.arpa. through this process. Next an inquiry about 1.0.0.0.0.0.1.4.6.3.3.e164.arpa. is submitted to the DNS. This inquiry retrieves the transcription and resolution rules, i.e. NAPTR RRs. The output is an URI for accessing IP-based applications. These addresses can be reached via Internet.

3. EMPIRICAL MEASUREMENTS

Our research methodology has been structured around three phases; *analysis, simulation* and *optimization*. In this paper we explain only the first two phases. The analysis phase is explained briefly in the current section and further explanation of the analysis phase is directed to our previous paper [8]. Using tools (existing as well as developed) we collect packet level traces of measurements made on a French ENUM model and analyze them in detail. This analysis helps us to develop mathematical models based on the empirical measurements. Parameters are obtained from this mathematical model which are used in the *simulation* phase, such that the simulation model resembles the characteristics of the empirical model.

The French ENUM model taken for our measurements has a three tiered architecture. In this architecture, Tier0 corresponds to the base of the internet domain space that is designated for ENUM (i.e. e164.arpa). The main function of Tier0 is the administration and technical management of ENUM domain. Records at this level contain pointers to the ENUM Tier1 for an E.164 (E.164 is the international numbering plan for public telephone systems. E.164 is also used to indicate telephone number in this paper) Country Code. Tier1 function is to manage and operate ENUM in the country identified by the E.164 Country Code (CC), i.e. <CC>.e164.arpa. Records at this level contain pointers to the ENUM Tier2 for a full E.164 number. The French ENUM model has Tier2 chunk (which has information about a chunk of numbers; For example if numbers from +33-1-60-76-00-00 to +33-1-60-76-99-99 are assigned to operator1 and numbers from +33-6-41-00-00-00 to +33-6-41-99-99-99 are assigned to operator2, and if there is a query arriving with a value within the numbers in operator1, then the Tier2 chunk will redirect the query to the Tier2 number server of operator1) and Tier2 number servers. Records at the Tier2 number level contains the NAPTR RRs associated to an E.164 number.

Three types of real measurements were made on the French ENUM model (Fig: 2)in the *analysis* phase.

1. To measure and model the performance (loss rate and response time at each of the target servers at a given query rate and under a given DNS environment (Hardware, DNS software type and DNS database configuration))
2. To measure the global response time of the queries made on the French ENUM model (i.e. the global DRD Performance)
3. To measure and model the two important metrics which impacts the performance of the IP links (i.e. delay and loss) connecting the different nodes of the ENUM system.

3.1 Local DNS server measurements and model

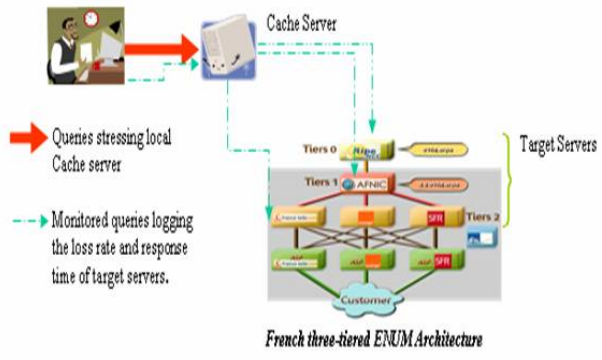

Figure 2: Empirical Measurement Set up

For the *first measurement*, we modified the tool Queryperf (which comes with BIND) and used it to stress the local cache server (Fig: 2). The stress test was done with a wide

range starting from a low (1000 - 2000) Queries per Second (QPS) to high (50000 - 80000) QPS rates. For each QPS rate average response time and loss rate of the target servers (Tier-0, Tier-1 and Tier-2) were measured. The results (Fig:3 and Fig:4) showed that each of these parameters increases with the query rate and we found that the best way to describe their evolution over the load for all the three target servers is to use a mathematical function defined in two parts. The first part of the function shows an *exponential growth*, whereas the second part is presented in the form of a *linear growth*.

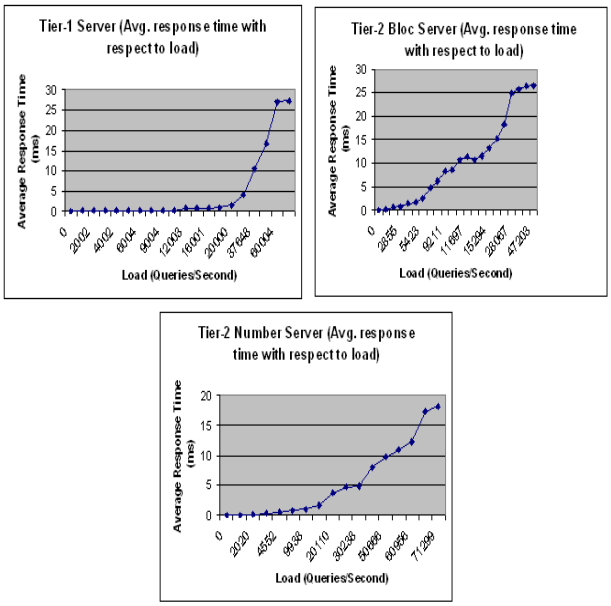

Figure 3: Average response time of Tier-1, Tier-2 bloc and Tier-2 number server

We used censoring techniques to cut the exponential and linear curve. For each exponential and linear curve, we used exponential and regression methods to identify how best the predicted curve fits the observed curve. The goodness of fit is estimated by the *R-square value*. The R-square value obtained for each approximation were found closer to "1". We identified parameters, which when applied into the mathematical model, best fit real world measurements. Thus parameters were calculated for all the three target servers to be used in the simulation tool.

3.2 Global response time measurements

While the local cache server was stressed, we also calculated the *global response time* for each query for the *second measurement*. The global response time is the sum of the response time of all the local DNS servers. We calculated the cumulative distributive function for a period of 5000 and 36000 different telephone numbers. For an experiment with 5000 telephone numbers, the cache server will be populated earlier and so the *hits* in the cache server will be more, which will result in lesser response time than with 36000 different telephone numbers. The resulting graph (has both simulation and empirical results) is in shown in section 5(Fig:10)

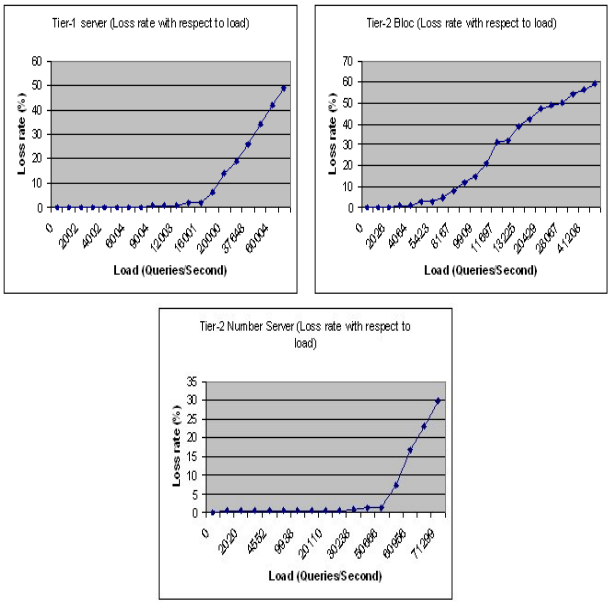

Figure 4: loss rate of Tier-1, Tier-2 bloc and Tier-2 number server

3.3 IP link measurements and model

For the *third measurement* we measured and modeled the real test set up for two metrics (loss and delay) on the links (connecting the user, the cache server and the target DNS servers).

Since DNS uses UDP, we used the same transport protocol for our measurements. A simple utility was developed and used for performing the active measurements. It comprises of a "Client" and a "Server" program. The client sends UDP datagrams at a fixed interval (Constant Bit Rate) or using an exponential (Poisson) distribution. Each packet contains a unique sequence number. When the packet is sent it is time stamped and this is recorded in a log file with the sequence number of the packet. The log file looks as in the table 1 When the server at the receiving end receives the datagrams

Sequence number	TimeStamp
61	1094728989.131322
62	1094728989.158886
63	1094728989.187941

Table 1: Packet Format in the Log file

sent by the client, it performs a time stamp of the received time with the sequence number of the packet in another log file. After the completion of the active measurements the log file of the server and the client are collected. The one-way delay and loss can be identified as follows:

Let us assign

Tc - Timestamp of the packet sent from the client

Ts - Timestamp of the packet received at the server

Ts - Tc of a unique packet sequence number will give the

time taken by the packet to travel from the client end to the sender end. i.e. the one-way delay. The one-way loss is also obtained from these log files. We used the libpcap and libnet libraries for the measures to be more accurate. The errors due to the famous NTP synchronization problems had to be eliminated from the obtained trace to get appropriate value. The explanation of clearing the trace does not come within the scope of this paper.

Due to space constraints we give a very brief overview of how we modeled the loss and delay of the IP links and finally how we validated the model.

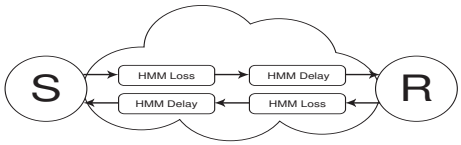

Figure 5: IP Link Model

The model developed should reflect the features of an IP link. So we designed an asymmetric Internet cloud model (client-server nature of the Internet), where the delay and loss are not correlated. Since we want the internet cloud to reflect the dynamicity and long-term dependence of the internet traffic we decided to use Hidden Markov Models (HMMs). Our model of an IP link has four HMMs (Fig:5), one for the delay and one for loss on both direction of the IP link.

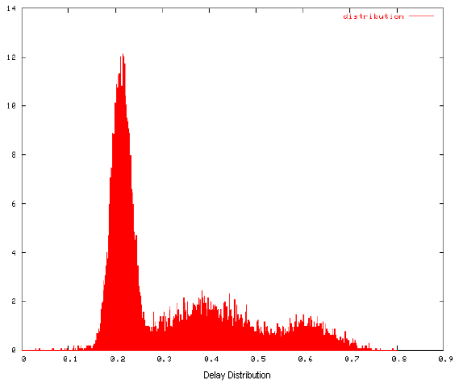

Figure 6: Delay distribution using Simulation

The delay is represented by a Continuous Hidden Markov process. Two states are used to model this event. A truncated gaussian with average μ and standard deviation σ is associated with each state of the continuous hidden markov chain. Since delay cannot be negative the distribution is truncated to obtain only positive values. We validated the delay model by simulation in NS-2. The results obtained by simulation also confirms that the delay distribution approaches a gaussian. The hidden markov chain comprised of three states and the three gaussian values were centered around 0.6, 0.2 and 0.375 seconds (Fig:6).

The loss is modeled by MMPP+M/M/1/N process. The loss rate measured is segmented on a 5 second window for the complete trace period. On applying the Expectation Maximization algorithm procedure [9] for the estimation of HMM we obtain the following values: where p is the observation matrix, Γ is the transition matrix and π being the stationary distribution of the markov chain.

$$p = \begin{pmatrix} 0.95 & 0.206 & O.07 \end{pmatrix}.$$

$$\Gamma = \begin{pmatrix} 0.937 & 0.0623 & 0.0006 \\ 0.0026 & 0.9973 & 0.0002 \\ 0.0000 & 0.0004 & 0.9996 \end{pmatrix}.$$

$$\pi = \begin{pmatrix} 0.0267 & 0.6581 & 0.3152 \end{pmatrix}.$$

Simulation of the loss model was done in NS-2. On applying the same procedure used for real trace, estimation was also done on the simulated trace, the values thus obtained are as follows:

$$p = \begin{pmatrix} 0.94 & 0.204 & 0.07 \end{pmatrix}.$$

$$\Gamma = \begin{pmatrix} 0.9431 & 0.0568 & 0.0000 \\ 0.0022 & 0.9976 & 0.0002 \\ 0.0000 & 0.0008 & 0.9992 \end{pmatrix}.$$

By comparing the real and simulated values; which seem to be identical, the loss model was validated.

4. IMPLEMENTATION OF THE SIMULATION TOOL

Now we move into the second phase the *simulation phase*. To design an ENUM simulation model we need information about the topology, the different modules used and also its configuration files.

4.1 Simulation Environment

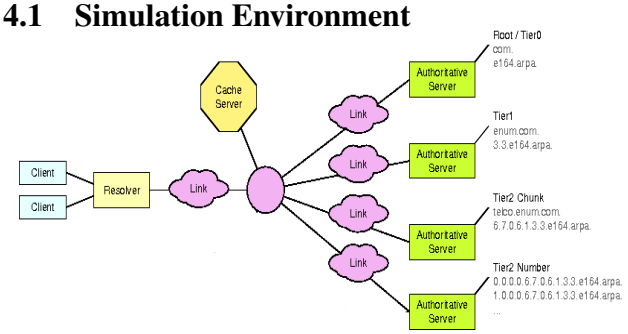

Figure 7: Simulation Topology

The topology (Fig: 7) resembles the empirical French model that we have used to make the real measurements. Here we give a brief explanation of all the modules used in the topology.

ENUM Clients. In the topology we have two ENUM clients. The functionality of this module is to generate and send queries of format (1.0.0.0.0.0.1.4.6.3.3.e164.arpa.) to the resolver.

Resolver. The resolver module receives requests from ENUM client and creates a DNS query type packet with the data obtained from the client. The resolver makes recursive query to the cache server. The burden of finding the answer to the query from the resolver is placed on the local cache server.

Cache Server. Most of the DNS resolution is processed by the cache server module. It receives recursive queries from the resolver. If the information asked by the query is present in the cache database, then there's a *cache hit* and it can answer directly to the resolver; otherwise in case of a *cache miss* it has to send iterative queries to the authoritative server before it can answer to the resolver. Since it knows only the name and place of the root server(s) in the beginning, it has to send a few iterative queries until it finds the response for the resolve's query (which yields high number of hops in the beginning). But the process is shortened (fewer hops) when the cache get populated over time as data is stored in the cache database for a TTL(Time To Live) period. For a deeper understanding of DNS cache server concepts we refer to book [10]

Authoritative Server. The authoritative server(s) receives the query from the cache server and searches for the query in its database. If it has the response, then it replies, otherwise it responds with information about the server which can possibly have the requested information.

IP link Module. This module is used to introduce loss and delay parameters to the different links in the simulation environment.

4.2 Configuration Files

In this subsection we explain about the data structure that is used to generate, transfer, process or look up information. In order to have a simulation model which could be modified easily, we designed all the data structure to be obtained from configuration files. This will help any future users to use the simulation by just modifying the configuration files for any scenario without understanding the complexity of the simulation code. The DNS software that we have used for the empirical measurements (explained in section 3) is BIND [11]. The BIND DNS parameters is the one that is followed in the configuration files to develop the ENUM simulation model. Here we give a brief idea of the configuration files used by different modules in the topology.

ENUM client. In our initial simulation implementation we have two enum clients. Each of the enum client will retrieve input values from the respective configuration files "enum1.conf" and "enum2.conf" to generate traffic. Each of them will generate different ENUM queries to simulate different users.

Resolver. As we already explained, *resolver* represent the module which formulates a DNS query packet type and send it to the local DNS cache server. The configuration file "resolv.conf" file contains information about the cache servers connected to the resolver. The resolv.conf for our simulation topology has two cache servers ($n(6) and $n(7) which represents nodes in NS-2) connected to the resolver. The algorithm used to identify which cache server to connect is explained in subsection 4.6

Cache Name Server. In real BIND set up it is a process called "named" in the cache name server which answers queries from resolvers. This application reads its data from a configuration file called "named.conf" which in turn gets its information from the zone files. Several zone files can exist but one zone file in particular keeps a database of records that supply the named process with most of its answers.

In our simulation set up, when the cache name server gets a query from the resolver, it searches the named.conf file. The named.conf file at the cache server looks like following:

```
zone "." in {
type hint;
file "db.cache";
};
```

According to the named.conf configuration file above, the cache server is of type *hint.* This type of servers usually will store a local cache of host name and address mappings. If a requested address or host name is not in its cache, the hint server will contact the master name server (in our case it is Tier-0), get the resolution information and add it to its cache. The zone file "db.cache" has information on how to contact only the root server. So with this configuration at the beginning all queries are redirected to the root server.

The zone statement identifies the location of the hints files which contains the name and address of the root servers on the internet. The zone file db.cache contains the following lines:

```
. 360000 NS                          A.ROOT-SERVERS.NET.
A.ROOT-SERVERS.NET. 360000   A $n(2)
```

The first line indicates the Name Server (NS) resource records. This record indicates that there is one NS for the root domain "." i.e. A.ROOT-SERVERS.NET. The second line represents the name-to-address mappings. "A" stands for address and the resource record maps the name (A.ROOT-SERVERS.NET.) to node address ($n(2)). Node address is used in place of IP address in NS-2 simulations. The real value of cache name server comes only after it build up its cache. Each time it queries an authoritative name server and receives an answer, it caches the records in the answer. Over time, the cache will grow to include the information most often requested by the resolvers querying the cache name server.

Authoritative Name Server. The Authoritative Name Server in the simulation environment represents Tier0, Tier1, Tier2 chunk and Tier2 number server of the French ENUM model. The configuration file in this category has either only partly information (such as the configuration file "db.0.0.1.4.6.3.3.e164.arpa", which contains information on which server to contact if the first seven E.164 number of the query maps with .0.0.1.4.6.3.3.e164.arpa.) or complete information (such as NAPTR RRs in case of the configuration file db.0.0.0.0.0.0.1.4.6.3.3.e164.arpa).

We will take the example of Tier-1 server for understanding the configuration structure of the authoritative servers. Similar to the cache server it is the named application which reads data from "named.conf" file, which in turn gets its information from the zone files. The named.conf file at the authoritative server looks like following:

zone ”numerobis.prd.fr” in {
type master;
file ”db.numerobis.prd.fr”;
};

This servers is of type ”master”. They are the authoritative servers for their particular zone. They read the data from the file on its host. For example the file ”db.numerobis.prd.fr” contains information for all the FQDN which ends with ”numerobis.prd.fr”. The content of the file db.numerobis.prd.fr is as follows:

```
$TTL    10800
@ IN    SOA ns1.numerobis.prd.fr. admin.ns1.numerobis.prd.fr. (
        1 ; serial
        10800 ; Refresh after 3 hours
        3600 ; Retry after 1 hour
        1w ; Expire after 1 week
        1h ; Negative caching TTL of 1 hour
)
```

Their explanation is like this. $TTL 10800 line states that records looked up and cached in a caching server from this file have a TTL (Time To Live) of three hours. The cached entry expires after three hours and is removed from the cache when that much time has passed. The ”@” sign in the line refers to the ”origin” for this zone file which is ”numerobis.prd.fr.”. ”IN” stands for Internet. SOA refers to ”Start of Authority”. SOA indicates that this name server is the best source of information for the data within this domain. The first name after SOA(ns1.numerobis.prd.fr.) is the primary name server of the numerobis.prd.fr zone. The second name (admin.ns1.numerobis.prd.fr.) is the mail address of the person in charge of the zone. The NS records for the file ”db.numerobis.prd.fr” is as follows:

@	IN	NS	ns1
sfr	IN	NS	bloc.sfr

These records indicate that there are two name servers for the zone numerobis.prd.fr. The name servers are the hosts ns1.numerobis.prd.fr and bloc.sfr.numerobis.prd.fr. Now we look into the name-to-address mappings of the file ”db.numerobis.prd.fr”.

ns1	IN	A	$n(3)
bloc.sfr	IN	A	$n(4)

ns1.numerobis.prd.fr is mapped to node $n(3) and bloc.sfr.numerobis.prd.fr is mapped to node $n(4). So the query which comes under each of these zones are redirected to the respective nodes. This process is followed until the query gets the full NAPTR RRs.

4.3 Hash Tables

Packet classification is one of the essential tasks of network processing. In our simulation environment all the incoming packets have to be processed. For example the cache server has to separate the header and data from the packet that it has received from the resolver. The header is used to identify from which client it has received the query from and also the packet ID. In turn the data i.e. the domain name is used to look up in a hash table to check whether the response for the query is in its cache. Such look ups are done at the resolver, cache and authoritative servers in our simulation. The job of look ups had to be done in line speed to decrease the processing delay.

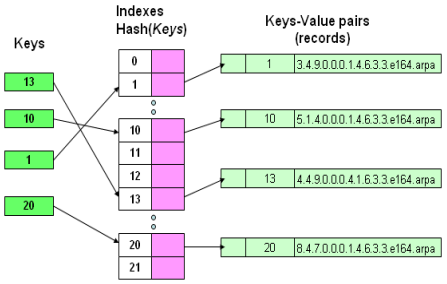

Figure 8: Hash Table

The primary idea behind a hash table is to establish a mapping between a set of all possible keys and positions in the array using a *hash function*. A hash function accepts a key and returns its *hash coding* or *hash value*. Keys vary in type but hash codings are always integers. The hash tables that we have developed in our simulator is based on the STL(Standard Template Library) in C++. It uses a *chained hash table* technique. The chained hash table fundamentally consists of an array of linked lists. Each list forms a *bucket* in which we place all elements hashing to a specific position in the array as seen in the figure 8. The bucket number is computed by taking the hash value modulo the number of buckets. To insert an element, we first pass its key to a hash function. This tells us in which bucket the element belongs to. In figure (Hash Table) hash value of the packet ID 1 (assuming the hash value is same as that of the ID) is stored in bucket 1(Hash value of 1 modulus 21 buckets = bucket 1). To look up or to remove an element, we hash its key again to find its bucket, then traverse the appropriate list until we find the element that we are looking for. In this case since each bucket is a linked list, the hash table is not fixed to a number of elements.

4.4 Modifications to NS-2

In NS-2 the procedure is as follows:

- The user creates the topology of the network by means of an OTcl based interpreter.
- The Internet links are specified in terms of bandwidth and of scheduling discipline.
- The routing strategy is also specified, through traditional or user-specific algorithms.
- Traffic generators are also provided or user-specific traffic generation could be used.

The figure (9) illustrates the files that were modified (left side) and added (the right side) in NS-2 to integrate the

ENUM application. The modifications are done both in C++ data functions and OTcl control functions and the additions were done only in C++. The modifications made are as follows:

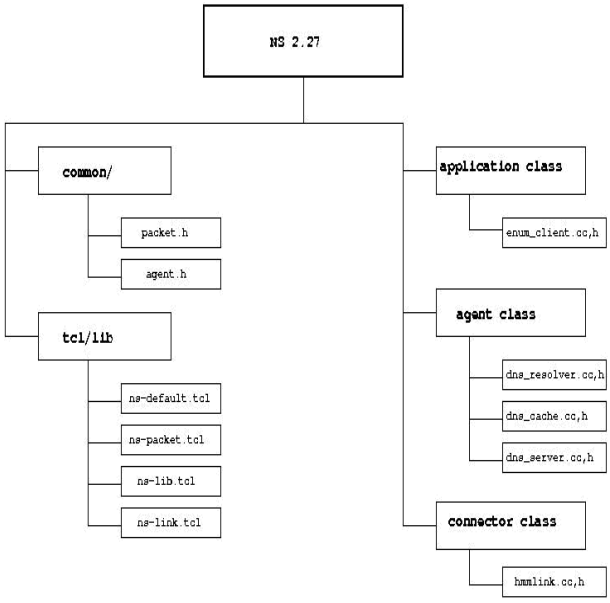

Figure 9: Module Implementation in NS-2

common/packet.cc,h. The class "packet" defines the structure of a packet and provides member functions to handle a free list for objects of this type. Since we introduced ENUM query/response type packet, we had to describe the corresponding packet format in this class.

common/agent.cc,h. The class "agent" supports packet generation and reception. To have a new application an agent running with the NS-2 distribution the agent class is modified to support the member functions of the new agents.

tcl/lib/ns-default.tcl. Default values for member variables, those visible in OTcl only and those linked between OTcl and C++ with bind are initialized in the ns-default.tcl file. Default values for the newly introduced configurable parameters are introduced in ns-default.tcl.

tcl/lib/ns-packet.tcl. To enable the new packet format used by ENUM we had to add the new packet types to ns-packet.tcl. This file is executed when the simulator initializes and creates an array containing a mapping between the class names and the names of the currently active packet header class.

tcl/lib/ns-lib.tcl. This file interprets node configurations specified in NS-2 simulation script. In orde to introduce losses at the node we added an extension to accommodate them

tcl/lib/ns-link.tcl. The modifications in the link that is introducing loss and delay parameters in the queue for the IP link is done here.

In the forthcoming subsections we will see how new modules were integrated to NS-2.

4.5 ENUM Client Implementation

The Enum client application inherits the NS-2 *Application* class. There are two basic types of applications in NS-2: *traffic generators* and *simulated applications.* The Enum client falls into the traffic generators category. This module accesses the configuration file (enum.*.conf) to generate a sequence of ENUM type queries. The configuration file is of the format "336 41000000 5000 500" where

- 336 41000000 represents Starting Number
- 5000 represents Size and
- 500 represents Average Inter arrival Time

From the "starting number" enum client generates 5000 different E.164 queries and add e164.arpa with each query. Finally "." are introduced between each number such that a complete FQDN is generated. A packet is created to add this generated data. Since, we did not have real traces of ENUM, these requests were sent to the resolver at a random interval. At present this random interval is fixed to the same interval that we used in the testing of real ENUM model. The distribution chosen for traffic generation in the simulated model may not quite adhere to the real ENUM scenarios of traffic generation.

4.6 Resolver Implementation

In NS-2, an application is attached to an agent and the application sends data through the underlying agent or by calling a set of methods defined in the agent. Enum client is the application which generates the traffic and it needs an agent to carry the traffic to its appropriate destination as well as to receive data. Since DNS uses UDP, the simulation model also uses the same as the transport agent. UDP is already present as a library routine in NS-2.

To perform the resolver operations, a new *dns_resolver agent* is created as the sub class of the UDP agent. dns_resolver agent is necessary to create a DNS type query from the data received from the application (enum_client) and also facilitate in receiving a DNS-type response from the local cache DNS server. It is necessary to explain the different hash tables used for this module before going into the implementation details.

The hash tables are necessary to store and do the look up of corresponding information to the queries. Two different hash tables were used in this module:

- The first hash table (htable1) contains the packet sequence number as "key" and complete DNS query packet information as "value". The value information is packed as a structure which contains the following fields:
  - The timeout value (At what time the same query will be called again)
  - Type of the packet (Whether it is a query or response)

  - The string (e.164 string format)
  - The number of iteration (explained in the "Timer algorithm")

- The second hash table (htable2) contains the packet sequence number as "key" and the time when the packet is send to the cache server as "value".

The procedure that is followed at the resolver agent is as follows:

A new Packet arrives. On receiving a packet, the type of the packet is identified to check, whether it is a "response type" or "query type" packet.

Response type packet. In this case the packet contains a response for the query it requested. The sequence number of the packet is parsed from the received packet and the sequence number is searched in the htable1.

- If it is found then the global response time is calculated from subtracting the value of the received time - send time. The send time data is obtained by looking at the second hash table (htable2) with the sequence number of the received packet.
- If the sequence number is not found, packet is ignored since the response has been received already and the values are deleted from the hash tables.

Query type. If the packet received is not response type, then it is from the enum_client. In this case, a new packet is formulated with the packet id as increasing sequence number (which is unique), the ENUM query and a field for calculating the number of hops. The hop value increases with every new node the particular packet traverses. The time that the packet is sent is updated in htable2. The formulated packet is sent to the corresponding cache server.

While instantiating the dns_resolver object, the number of cache servers (nb_cacheservers) that is accessible by the resolver is obtained from the resolv.conf configuration file. If a new cache server is added, the resolv.conf file is updated.

Once the query is sent to the cache server a timeout is calculated for the query (in case the packet gets lost or delayed). After the timeout if a response for the query is not received, the query is resent to the same or different cache server. The timeout is calculated by the Timer Algorithm.

Timer Algorithm. Every time a request is sent from the resolver to the cache server, it is calculated as iteration (nb_iteration). If there are two cache servers and when a new query is generated from the resolver, it is the first iteration. Once the same query is sent to all the different accessible cache servers it is calculated as one cycle (nb_cycle). We used the BIND configurations for fixing the number of maximum servers (NB_MAX_SERVERS) to 3 and number of maximum cycles (NB_MAX_CYCLES) to 4.

By using the algorithm, timeouts and the target server to which the query packet is to be sent are calculated depending on the number of cache servers, iterations and number of cycles. After every time the timer algorithm is called the nb_iteration is updated in the hash table (htable1) for the particular packet.

input : TIMEOUT_SINGLE = 5 (in seconds);
TIMEOUT_MULTIPLE = 10 (in seconds);
NB_MAX_CYCLES = 3;
NB_MAX_SERVERS = 4;
output: The TTL value and the target local cache server to send the data

nb_cycle = iteration/nb_cacheservers (*Convert the result to Integer Value*);
if *nb_cycle == 0* **then**
    Timeout = TIMEOUT_SINGLE;
    Target Server = Iteration *mod* nb_cacheservers;
if *nb_cycles > NB_MAX_CYCLES* **then**
    The packet is considered lost;
if *nb_cycle > 0* **then**
    if *nb_cacheservers == 1* **then**
        TIMEOUT = 2∗nb_cycle*TIMEOUT_SINGLE;
        Target Server = Iteration *mod* nb_cacheservers;
    else
        TIMEOUT =
        nb_cycle*TIMEOUT_MULTI*nb_cacheservers;
        Target Server = Iteration *mod* nb_cacheservers;
    end

Algorithm 1: Algorithm for calculating Timeout

4.7 Cache Server Implementation

As discussed in the previous subsection here also it is necessary to explain the hash tables present in this module before going into the implementation part. There are three hash tables used by the cache server module:

- Name server (NS) Hash table - to give information about a segment of the database
- Authoritative server (AS) Hash table - to store the information pertaining to a particular domain, exclusive of any sub domains that have been delegated to their own authoritative servers and
- NAPTR hash table - to store NAPTR RRs corresponding to E.164 number.

The cache agent module is also developed as the sub class of the NS-2 UDP agent. On instantiation of the cache object, it updates all the three hash tables with the data from the configuration file "db.cache". The cache agent on receiving the query packet from the resolver, have to follow successive referrals querying the different authoritative server until it receives a response for the query. The cache Agent module works as follows:

A Packet Arrives. In this case the cache node can either receive packets from the resolver (as query) or from the server (as response).

Query type packet. When it is a query type packet, the cache server looks up its NAPTR hash table (which contains RRs)

to verify whether it has response for the particular query. If it is a *cache hit*, then it updates the hash table, formulates the response type packet and sends it back to the resolver. On the contrary if it is a *cache miss*, then it searches in the NS hash table to find the authoritative server. Finally from the authoritative server hash table it gets the address of the AS node which can provide some information about the received request. For every data processing done, the corresponding hash tables have to be updated.

Response type packet. In this case the cache server receives packet from any of the AS. If the packet that has been received contains the NAPTR value then it means that the response for the query is obtained. Then the response packet is formulated and send it back to the resolver. On the contrary if the packet contains information about another AS which might have information about the query, the cache server ask the AS for the address for the query and this process is continued until it gets an answer. For every transaction done in the cache server, the stipulated hash table have to be updated.

4.8 Authoritative Server Implementation

Similar to the cache agent here also three hash tables are used:

- NS Hash table
- AS Hash table and
- NAPTR hash table

The server agent is also created as the sub class of the UDP agent.

A Packet Arrives. When the AS receives the packet, it uses a mechanism to determine whether the packet should be lost or not. The loss rate value for determining this process is got from the empirical measurements(from the analysis phase) and the agent retrieves this data from the configuration file.

If the packet is not lost. Here a processing delay is introduced into the server. This information is also obtained from the empirical measurements (from the analysis phase). A Gaussian distribution function is used to calculate the delay. Then it looks into its hash table either to send the server address for the query or a response which will be used by the cache server to identify the destination which can either give more details or the address of the server.

4.9 IP link Model Implementation

In NS-2, Connector class is used to link two nodes (for e.g. a resolver and the cache server). In our simulation model we created a new class called *HMMLink* which inherits the Connector class. This class contains the attributes for the observation and transition matrix. The methods in this class enables to make the hidden Markov chain transitions. Two class inherits the HMMLink class: *HMMdelay* and *HMMloss*. Usage of this module is in two stages; the first stage is to create the HMMLink object and then bind the object with the existing link object present between two nodes in NS-2. This object thus instantiated is placed between the objects *head_* and *enqT_* of the *link class* in Ns-2

A packet arrives. When a packet arrives in the link, the method "recv" in the HMMLink class uses the configuration parameters explained below to determine whether the received packet is lost or not, and if not, how much the delay is.

HMMdelay. This object calculates the delay for the packet using a configuration file as shown below:
1
10 5
1000
Poisson
The first line indicates the number of states for the transition matrix while the parameters in the second line are for average and standard deviation delay. The third line is the time between each state in milliseconds and final line is used for identifying the distribution type.

HMMloss. This object will calculate whether an arriving packet should be lost or not. The loss configuration file is as shown below:
1
0
1000
CBR
All the parameters in the loss configuration do the same operation as that of loss, except the second line which indicates the loss rate.

5. VALIDATION

As explained in section 3 our research methodology had three phases namely: *analysis, simulation* and *optimization.* The goal of the simulator is not to stop within this three phases. The promising solutions obtained from the simulation should be used in a fourth phase which is *implementation.* The results obtained from the simulation models should be implemented as real solutions in real world ENUM scenarios.

In order to use the simulation solutions in the implementation phase one should be confident enough that the obtained simulation results are accurate and meaningful. Validation of a simulation model with real measurements should be done to understand how accurately does the simulation model reflect the operations of a real system [12],[5].

As discussed in this book by Harry Perros [12] one of the most powerful validation methods is called *output validity.* If actual data are available regarding the system under study (empirical measurements), then these data can be compared with the output obtained from the simulation model. Obviously is it true that if they do not compare well, the simulation model is not valid.

In order to satisfy the *output validity* validation method, we first compared the global response time of the real and the simulation model. For the validation purpose we again state the reference [12] which explains about the need for a method called *Relationship Validity.* It defines that in order to have the structure of a system under study, fully

reflected down to its very detail in a simulation model, the models assumptions should be credible.

Assumptions like topology, bandwidth, generation of ENUM queries and DNS database have been verified by logical checks and also by the NS-2 visualizing tool NAM. Other parameters like the ENUM servers (Tier-0,Tier-1, Tier-2 and Tier-3) loss rate and response time has been taken as input from the real measurements. Also the IP link delay and loss values are obtained from real measurements. So to our knowledge *Relationship Validity* has been done.

For output validity, we used parameters obtained from real measurements into the simulator and compared two kinds of results; for a lower number of request (5000) and a higher number of requests (36000). We compared the cumulative frequency distribution of real measurements and simulated results for validating the simulation model as shown in Fig: 10.

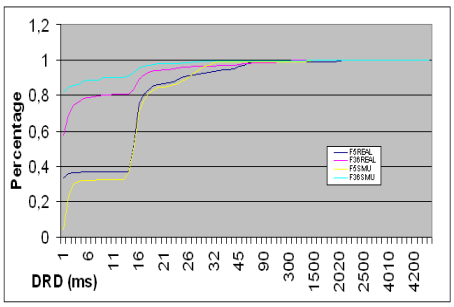

Figure 10: Comparison of real and simulation results

In this paper [5] Averil M. Law suggests that a null hypothesis of saying that the real system and the simulation model are same is clearly false, since the model is only an approximation of the real system. So the question we have to ask is whether the differences between the model and the real system are significant enough to affect any conclusions derived from the model. To answer the question, statistical procedures such as confidence intervals can be used.

The confidence interval at 95% interval can be calculated by the formula: $Mean \pm (2 * \frac{StandardDeviation}{\sqrt{N}})$ The standard rate error i.e. $\frac{StandardDeviation}{\sqrt{N}}$ is calculated as 1.047226. So in order to have the confidence interval at 95% interval the mean global response time of the simulation results should should be $\pm$ 2*1.047226(i.e.2.0944452) of the real global response time measurements. The mean global response time of real measurements for 5000 queries estimated at the local cache server is 10.53116667. The global response time calculated for 5000 queries by simulation is 10.717. So this proves that the conclusions arrived from the simulation model can be significant in real implementations.

6. HOW THE SIMULATION MODEL IS AUTONOMOUS

If systems can manage themselves given high level objectives from administrators they are considered as *autonomous*. As discussed previously (subsection 4.2) the simulation model is designed to get all its values from the configuration files. At its present state the simulation model do not have integrated into it the two of the four requirements of autonomic computing envisioned by IBM [13] namely; self-healing and self-protecting. The other two requirements namely; Self-Configuring (adaptive TTL at the cache server on the basis of the global response time) and Self-Optimizing (load balancing of the cache servers) are implemented. The case studies are not explained here due to space constraints.

7. CONCLUSION

A new approach is followed to build an autonomous simulation tool. The tool is validated with real measurements. The simulator tool that we have developed can be used to study an optimized delegation model for a particular scenario. The tool can also be used to study the feasibility of different ENUM enabled services with minimal modifications. We are convinced that our work could influence different metrics (such as Time to Live for different types of Resource Records in Tiers-(0...2) DNS Servers)) which will lead to recommendations of the new ENUM protocol to achieve better performance.

8. REFERENCES

[1] P. Faltstrom. E.164 number and dns.rfc-2916, September 2000.

[2] Monique Becker and André-Luc Beylot. *Simulation des Réseaux*. Hermés, 2006.

[3] Averill M. Law and David Kelton. *Simulation Modeling and Analysis*. Mc-Graw Hill, 1991.

[4] Vint : Virtual internetwork testbed, http://www.isi.edu/nsnam/vint/index.html.

[5] Averil M. Law. How to build valid and credible simulation models. In *Proceedings of the Winter Simulation Conference*, 2005.

[6] M. Mealling and R. Daniel. The naming authority pointer (naptr) dns resource record. rfc-2915, September 2000.

[7] T. Berners-Lee. Uniform resource identifiers (uri): Generic syntax. rfc-2396, August 1998.

[8] Thomas BUGNAZET Sandoche BALAKRICHENAN and Monique BECKER. Studying enum performance with modeling and simulation. In *Proceeding os the Asian Modeling Symposium*, 2007.

[9] Kavé Salamatian and Sandrine Vaton. Hidden markov modeling for network communication channels. In *ACM SIGMETRICS Performance Evaluation Review*, 2001.

[10] C. Liui and P. Albitz. *DNS and BIND*. Oreilly, 4th edition, April 2001.

[11] Bind (berkeley internet name domain) a dns server implementation from isc (internet systems consortium) http://www.isc.org/index.pl?/sw/bind/.

[12] Harry Perros. *Computer Simulation Techniques : The definitive introduction!* NC State University, 2007.

[13] J. Kephart and D. Chess. The vision of autonomic computing. *Computer Magazine*, IEEE:41–50, 2003.