



# SRS Prototype - Review of Results

Version: 1.0  
Author(s): Andrew McMillan, Ewen McNeill  
Date: 23<sup>rd</sup> May 2002

**Commercial in Confidence**

# Table of Contents

|   |           |
|---|-----------|
| <b>1 INTRODUCTION</b>                     | <b>3</b>  |
| 1.1 BACKGROUND                            | 3         |
| 1.2 PROJECT SCOPE                         | 3         |
| 1.3 GENERAL CONCLUSIONS                   | 3         |
| 1.4 RECOMMENDATIONS                       | 3         |
| <b>2 SYSTEM OVERVIEW</b>                  | <b>4</b>  |
| 2.1 ENVIRONMENT                           | 4         |
| 2.1.1 Hardware and Operating Systems      | 4         |
| 2.1.2 Networking                          | 4         |
| 2.1.3 Installed Software                  | 4         |
| <b>3 TEST SCENARIOS</b>                   | <b>7</b>  |
| 3.1 PERFORMANCE TESTING                   | 7         |
| 3.1.1 Basic Performance                   | 7         |
| 3.1.2 Loaded Performance                  | 7         |
| 3.1.3 Application Operation               | 7         |
| 3.1.4 Batch Processing                    | 7         |
| 3.1.5 Processing Under Extreme Stress     | 7         |
| 3.2 REPLICATION                           | 7         |
| 3.2.1 Front-end Replication               | 7         |
| 3.2.2 Back-end Replication                | 8         |
| 3.2.3 Failover                            | 8         |
| 3.2.4 Compare Databases                   | 8         |
| 3.3 SECURITY                              | 8         |
| 3.3.1 Application                         | 8         |
| 3.3.2 Environment                         | 8         |
| <b>4 FINDINGS</b>                         | <b>9</b>  |
| 4.1 PERFORMANCE                           | 9         |
| 4.1.1 HTTP vs HTTPS                       | 9         |
| 4.1.2 Update vs Enquiry                   | 9         |
| 4.1.3 Database Size                       | 9         |
| 4.1.4 Response Size                       | 9         |
| 4.2 REPLICATION                           | 10        |
| 4.3 SECURITY                              | 10        |
| <b>5 DETAILED RESULTS</b>                 | <b>11</b> |
| 5.1 PERFORMANCE TESTING                   | 11        |
| 5.1.1 Basic Performance                   | 11        |
| 5.1.2 Loaded Performance                  | 11        |
| 5.1.3 Processing Under Extreme Stress     | 12        |
| 5.2 REPLICATION                           | 12        |
| 5.3 SECURITY                              | 13        |
| 5.3.1 Application                         | 13        |
| 5.3.2 Environment                         | 13        |
| <b>6 DETAILS OF TESTS</b>                 | <b>14</b> |
| 6.1 TESTING SETUP                         | 14        |
| 6.2 TESTING COMPONENTS                    | 14        |
| 6.3 DETAILED TEST RESULTS                 | 15        |
| 6.4 INSTALLATION NOTES FOR CRYPT::OPENPGP | 16        |

# 1 INTRODUCTION

## 1.1 BACKGROUND

InternetNZ have contracted Catalyst IT Ltd to prototype the SRS outlined in their Technical Architecture document.

## 1.2 PROJECT SCOPE

This document describes the findings of the prototyping of the Shared Registry System within an architecture based on the SRS Technical Architecture document.

## 1.3 GENERAL CONCLUSIONS

The general architecture outlined in the SRS Technical Architecture will support the SRS.

A number of issues were raised by the prototype, and many design issues were solved during the process. The most significant issue encountered is that of memory use on the backend server, which was higher than anticipated.

## 1.4 RECOMMENDATIONS

1. *The backend database / application servers for production use should be configured with 2GB RAM.*

Performance of mod\_perl applications tends to be limited by available RAM, and the SRS will be no different. Ensuring that there is plenty of RAM available to the SRS backend application servers will minimise the effort required to profile the application and minimise the RAM use.

2. *The Technical Design should consider processing the XML in more efficient ways on the backend server.*

The prototype used the Xerces XML libraries for XML processing. These libraries may use significantly more memory than alternatives, and the advantages of this library are primarily related to the input stream. It is possible that using an alternative strategy for output of the XML tree could significantly reduce memory pressure on the backend application servers.

3. *The Business Requirements should consider removing the requirement to place the record count at the start of the XML reply to the general query.*

This requirement imposes a need to fully process the response before starting to deliver it back to the front-end, and imposes a memory overhead as a result. Removal of this requirement will allow a broader range of choices within recommendation #2.

## 2 SYSTEM OVERVIEW

The SRS Prototype is an implementation of selected areas of the SRS Technical Architecture. Most notably, the following areas are included:

- Network architecture
- Replication architecture
- Operating system architecture
- Protocol Handling
- Encryption
- Software environment

The following areas of the application are not fully included within the prototype, as developed:

- Business Rules
- Human-oriented web interface
- Some registry interactions are not implemented
- Some functionality is missing from implemented actions
- Full audit trail.
- Support notifications

The guiding principle in implementing this prototype has been that it should validate the choices expressed within the Technical Architecture.

### 2.1 ENVIRONMENT

#### 2.1.1 Hardware and Operating Systems

The prototype environment consists of seven computer systems, grouped into "Wellington" and "Auckland" sets (although not actually geographically separated) designed to emulate the expected minimal production architecture. The "Wellington" group consisted of an OpenBSD firewall, a front-end webserver and two back-end application servers. The "Auckland" group was identical, but of lower hardware specifications and with only one back-end application server.

#### 2.1.2 Networking

The two firewall nodes operate a VPN between the "Wellington" and "Auckland" servers, controlling access between the network nodes. They also provide port forwarding of http and https traffic to each of the front-end webserver from the "public" addresses.

#### 2.1.3 Installed Software

##### Front End Software:

*Debian packages:*

apache 1.3.24-2.1

apache-common 1.3.24-2.1  
apache-dev 1.3.24-2.1  
apache-doc 1.3.24-2.1  
apache-perl 1.3.24-2-1.26-1-1  
libapache-mod-perl 1.26-3  
libapache-mod-ssl 2.8.8-0catalyst  
libapache-mod-ssl-doc 2.8.8-0catalyst  
libcrypt-ssleay-perl 0.23-2.1  
libdigest-md5-perl 2.13-2  
libmime-base64-perl 2.12-4  
libnet-perl 1.09.01-1  
libnet-ssleay-perl 1.08-1.1  
libperl5.6 5.6.1-7  
libstorable-perl 1.0.14-1  
libxerces1 1.7.0-1catalyst  
libxerces1-dev 1.7.0-1catalyst  
libxml-xerces-perl 1.7.0.0-1catalyst  
libwww-perl 5.64-1  
openssl 0.9.6c-2  
perl 5.6.1-7  
perl-5.005 6.3  
perl-5.005-base 6.3  
perl-5.005-doc 6.3  
perl-5.005-suid 6.3  
perl-base 5.6.1-7  
perl-doc 5.6.1-7  
perl-modules 5.6.1-7  
perl-suid 5.6.1-7  
ssh 3.0.2p1-9

*Other Perl Packages:*

DB\_File::Lock 0.04  
LWP::Parallel::UserAgent 2.54

**Back End Software:**

apache 1.3.24-2.1  
apache-common 1.3.24-2.1  
apache-dev 1.3.24-2.1  
apache-doc 1.3.24-2.1  
apache-perl 1.3.24-2-1.26-1-1  
gnupg 1.0.6-3  
libapache-dbi-perl 0.88-5  
libapache-mod-perl 1.26-3  
libapache-reload-perl 0.07-1  
libconvert-asn1-perl 0.11-1  
libcrypt-blowfish-perl 2.09-1  
libcrypt-des-perl 2.03-1  
libdate-calc-perl 5.0-2  
libdbd-pg-perl 1.01-3  
libdbi-perl 1.21-2  
libdigest-md2-perl 2.00-1  
libdigest-md5-perl 2.13-2  
libdigest-sha1-perl 2.00-1  
libmailtools-perl 1.44-1  
libmime-base64-perl 2.12-4  
libnet-perl 1.09.01-1

libperl5.6 5.6.1-7  
libpgperl 7.2.1-2  
libpgsql2 7.2.1-2  
libssl-dev 0.9.6c-2  
libssl0.9.6 0.9.6c-2  
libstatistics-descriptive-perl 2.4-2  
libtime-hires-perl 1.20-4  
libtimedate-perl 1.11-1  
liburi-perl 1.18-1  
libwww-perl 5.64-1  
libxerces1 1.7.0-1catalyst  
libxerces1-dev 1.7.0-1catalyst  
libxml-xerces-perl 1.7.0.0-1catalyst  
perl 5.6.1-7  
perl-5.005 6.3  
perl-5.005-base 6.3  
perl-5.005-doc 6.3  
perl-5.005-suid 6.3  
perl-base 5.6.1-7  
perl-doc 5.6.1-7  
perl-modules 5.6.1-7  
perl-suid 5.6.1-7  
postgresql 7.2.1-2  
postgresql-client 7.2.1-2  
postgresql-contrib 7.2.1-2

**Other Perl Packages:**

Class::Loader 2.02  
Convert::ASCII Armour-1.4  
Convert::PEM 0.06  
Crypt::CBC 2.02  
Crypt::DES\_EDE3 0.01  
Crypt::DSA 0.12  
Crypt::OpenPGP 1.00  
Crypt::Primes 0.49  
Crypt::RIPEMD160 0.04  
Crypt::RSA 1.48  
Crypt::Random 1.11  
Data::Buffer 0.04  
Math::Pari 2.010303  
Sort::Versions 1.4  
String::Random 0.198  
Tie::EncryptedHash 1.1

**Packages on "Registrar" client machines:**

The packages on the registrar machines match the Perl packages on the backend machines. No other special software was used on the registrar machines; they make their requests with the standard Perl LWP modules.

## 3 TEST SCENARIOS

### 3.1 PERFORMANCE TESTING

#### 3.1.1 Basic Performance

test query and update performance with existing database (110,000)

test query and update performance with larger database (550,000)

requests through HTTP

requests through HTTPS

#### 3.1.2 Loaded Performance

script to run multiple requests in parallel (evenly spread through the second), up to 9 requests in parallel.

run on three "registrar" client machines for higher loads to ensure that registrar load does not affect results.

requests through HTTP

requests through HTTPS

#### 3.1.3 Application Operation

multitable update: "update" actions (multiple actions in one request)

simple query: "status"

large complex query: "query" (general register query)

#### 3.1.4 Batch Processing

DNS export timings

#### 3.1.5 Processing Under Extreme Stress

Run "Loaded Performance" and "Batch Processing" simultaneously, (batch processing will run in a loop). Measure differences in performance from single tests.

### 3.2 REPLICATION

#### 3.2.1 Front-end Replication

feed requests to all backends and ensure they stay in sync (by comparing databases)

### 3.2.2 Back-end Replication

feed requests to 2 backends, wait for backend replication to bring the other backend into sync (measure time taken)

single requests

### 3.2.3 Failover

While making "performance" tests:

unplug srs-w-app2 - observe

unplug srs-a-app - observe

unplug srs-w-app - observe

replug srs-a-app - observe

replug srs-w-app - observe

replug srs-w-app2 - observe

manually switch to srs-a-web, and process some transactions

### 3.2.4 Compare Databases

The contents of databases on all three systems will be compared after replication tests to ensure they are the same.

## 3.3 SECURITY

### 3.3.1 Application

invalid registrar number (not known)

wrong registrar number (not matching signature)

invalid signature (eg, of wrong request)

invalid XML (ie, improperly formatted)

invalid request (eg, missing important parts of HTTP request)

### 3.3.2 Environment

attempt logins to machines directly (will fail)

use nmap to ensure only HTTP available web->app

use nmap to ensure only HTTP/HTTPS available outside->web

use nmap to ensure nothing available outside->app

check door is locked :-)

# 4 FINDINGS

## 4.1 PERFORMANCE

Target figures for average performance were achieved in all cases. In many cases the maximum result was under the targetted average.

### 4.1.1 HTTP vs HTTPS

In the low-latency environment of our LAN, HTTPS had only a very small effect on performance of both enquiry and update transactions. As load increased the effect of the protocol overhead on the front-end webserver became increasingly apparent although in percentage terms the increase was relatively constant.

Using a connection with a latency of around 100 ms (bouncing requests through a system at the other end of a cable modem) showed a noticeable increase – the additional overhead was around 4.5 \* latency. In reality, registrar's production systems with significant levels of registry activity are likely to have comparatively low latency connections to the SRS.

In general, the performance of HTTP vs HTTPS suggests that the performance overhead of HTTPS may be acceptable in production for use by all queries, although at peak load testing the protocol overhead had a significant impact on the front-end webserver.

### 4.1.2 Update vs Enquiry

Performance of 'update domain' transactions was almost identical to that of 'whois' transactions under low load conditions, although there was a greater variability in response times for 'update domain', most likely due to database checkpointing.

Under increasing load the additional disk flushing required by the update has its effect and we see noticeably greater timing for the updates.

### 4.1.3 Database Size

Increasing the number of domains in the database to 550,000 had no significant effect on performance of either update or simple enquiry. Domains were increased by copying existing records and appending "--a", "--b", "--c" or "--d" to the 3rd component of the domain name ([catalyst.net.nz](http://catalyst.net.nz) became [catalyst--a.net.nz](http://catalyst--a.net.nz)).

DNS Export performance was slightly worse than linear, increasing from 52s at 110,000 domains, up to 295s at 550,000 domains. Both of these results were significantly lower than target.

### 4.1.4 Response Size

The results show a substantial performance hit when the number of results increases. A query returning 100 records is around 10-20 times as slow as one retrieving only 1 record. The overhead is not related to database performance - the query in question executes sub-second. Detailed measurement of the performance of a particular query returning around 550 records indicates:

Action processing: 20 seconds  
Serialising: 9 seconds

This involves building an XML tree of the 550 domains and serialising it into a response of around 1MB which is then returned to the frontend.

This processing appears to be putting considerable memory pressure on the backend server. Apache client processes, which are using 48M for the 2000 domain query, seem likely to grow to 160M on a 10,000 result query.

## 4.2 REPLICATION

Replication performed as intended, with a latency of around 5 seconds from the last update transaction before all servers agreed they were synchronised. Since all requests were submitted to all active servers, this process was in general a process of discovery, rather than of replicating the actual result.

During tests where one server was excluded from the front-end processing, the latency to full replication increased to around 10 seconds (approximately 5 seconds replication plus 5 seconds of discovery and agreement).

The replication architecture, as implemented in the prototype, goes significantly beyond what was proposed in the technical architecture, and what was planned to be implemented during the prototype. Additional capabilities include:

- Some automatic recovery of synchronisation.

- Some automatic evaluation of trustworthiness.

Replication was implemented in a modular fashion, as a middleware layer operating on the front-end webserver, receiving requests and passing them on to all active backends.

The technical design should include a review of the state transitions for the replication.

## 4.3 SECURITY

The prototype has not necessitated any redesign of network features which might compromise the security in any way. The security features of the planned architecture appear to work as specified, without interfering with the functioning of the application.

The point of the prototype (in relation to security) has been to confirm that the security features of the proposed architecture do not compromise the operation or functionality of the application. The prototype has been completely successful in showing this.

# 5 DETAILED RESULTS

## 5.1 PERFORMANCE TESTING

### 5.1.1 Basic Performance

| Test   | Result            | Target | Notes  |
|--|-------------------|--------|--|
| Whois query (HTTP)                           | 0.62s (0.89s max) | 1.5s   | This base scenario is the starting point for evaluating the effect of most scenario changes. The 'whois' query is used.                          |
| Whois query (HTTP over higher latency link)  | 0.99s (3.8s max)  | 2s     | Using a remote cable connection to connect through gave a latency of around 50mS each way for a total of 100mS overhead.                         |
| Whois query (HTTPS)                          | 0.84s (1.13s max) | 2s     | 0.5s - 1s overhead expected for https.   |
| Whois query (HTTPS over higher latency link) | 1.46s (4.3s max)  | 2.5s   |  |
| Update domain (HTTP)                         | 0.66s (1.81s max) | 2s     | Using domain update requests.  |
| Update domain (HTTPS)                        | 1.02s (5.73s max) | 2.5s   |  |
| 100 result query (HTTP)                      | 8.0s (14.0s max)  | 10s    | The bulk of this time is in the multiple frontend and backend layers of XML processing.  |
| 100 result query (HTTPS)                     | 8.3s (9.7s max)   | 11s    | The longer connection means that HTTPS has little cumulative effect.   |
| 500 result query (HTTP)                      | 40s (52s max)     | 50s    | The bulk of this time is in the multiple frontend and backend layers of XML processing.  |
| 2000 result query (HTTPS)                    | 84s (97s max)     | 100s   | These result sizes introduced significant memory pressure on the backend application server with less memory.                                    |
| Update domain (500k, HTTP)                   | 0.69s (1.12s max) | 3s     |  |
| Whois query (500k, HTTP)                     | 0.63s (1.08s max) | 2s     |  |
| Update domain (500k, HTTPS)                  | 0.89s (1.97s max) | 3.5s   |  |
| Whois query (500k, HTTPS)                    | 0.97s (2.17s max) | 2.5s   |  |
| DNS Export                                   | 52s               | 200s   | The full DNS export process involves several further steps after creation of the zone files, so performance will not be this good in production. |
| DNS Export (500k)                            | 295s              | 500s   |  |

### 5.1.2 Loaded Performance

| Test                    | Results           | Target | Notes |
|-------------------------|-------------------|--------|-------|
| Update domain (6, HTTP) | 2.47s (10.28 max) | 3.5s   |       |
| Whois query (6, HTTP)   | 1.86s (6.74 max)  | 3s     |       |

| Test                    | Results            | Target | Notes |
|-------------------------|--------------------|--------|-------|
| Whois query (6, HTTPS)  | 2.27s (7.09s max)  | 3.5s   |       |
| Update domain (9, HTTP) | 4.03s (15.79s max) | 7s     |       |
| Whois query (9, HTTP)   | 2.65s (12.87s max) | 6s     |       |
| Whois query (9, HTTPS)  | 3.18s (12.74s max) | 7s     |       |

### 5.1.3 Processing Under Extreme Stress

| Test                  | Results            | Target     | Notes |
|-----------------------|--------------------|------------|-------|
| Whois query (9, HTTP) | 2.88s (13.89s max) | 10s        |       |
| DNS Export            | 4:16s then 1:48    | 10 minutes |       |

## 5.2 REPLICATION

| Test                                  | Results  | Target   | Notes   |
|---------------------------------------|----------|--|---|
| Normal operation                      | Working  | Working  | Latency of around 5 seconds.  |
| Replicate to 3 <sup>rd</sup> backend  | Working  | Working  |   |
| Replicate to 3 <sup>rd</sup> (loaded) | Working  | Working  | The third server was underpowered and took some time to catch up. Don't use a P200 in production.       |
| Unplug srs-w-app2                     | Correct  | Responses served by w-app and a-app.   | Initially the system switched to read-only mode for a short period, while srs-a-app (a P200) caught up. |
| Unplug srs-a-app                      | Correct  | Read-only mode served by w-app.  |   |
| Unplug srs-w-app                      | Correct  | Timeout error to client. F/E marks all as untrusted.   | Consider changing this behaviour for production.  |
| Replug srs-a-app                      | Correct  | Read-only mode served by a-app, after manually enabling it as 'trusted'.   |   |
| Replug srs-w-app                      | Correct  | Initially read-only mode served by a-app, until replication catches up and w-app is automatically upgraded to trusted. |   |
| Replug srs-w-app2                     | Correct  | Initially untrusted, replication should catch up and the server become upgraded to trusted.                            |   |
| Process through srs-a-web             | Working. | Transactions can come through srs-a-web interleaved with ones through srs-w-web.                                       |   |
| Compare databases at low level        | Correct  | domain, dns and domain_contact tables contain the same information   |   |

## 5.3 SECURITY

### 5.3.1 Application

| Test                | Result | Notes   |
|---------------------|--------|---|
| Invalid Registrar   | Found  | This check was removed during update testing for ease of testing. |
| Incorrect signature | Found  |   |
| Invalid signature   | Found  |   |
| Invalid XML         | Found  |   |
| Invalid request     | Found  |   |

### 5.3.2 Environment

| Test                             | Result                     | Notes |
|----------------------------------|----------------------------|-------|
| Attempt direct logins            | Fails                      |       |
| Nmap from srs-w-web to srs-w-app | Shows ports 22, 80 and 443 |       |
| Nmap from external to web        | Shows ports 22, 80, 443    |       |
| Nmap from external to app        | Shows ports 22, 80         |       |
| Check door                       | Locked.                    |       |

# 6 DETAILS OF TESTS

## 6.1 TESTING SETUP

The servers were configured with persistent database connections enabled, and with script recompilation off for all tests. These will be normal settings in production but are set differently during development. All perl modules used by the application were configured to be preloaded.

The rsync process was run in standalone daemon mode on all backend application servers, rather than from inetd. The replication backend was operating as a background daemon during all tests.

The PostgreSQL installation was optimised slightly to increase the buffers from the default 128 to 512.

The equipment used for these tests was as follows:

| System     | Specification   | Notes  |
|------------|---|--|
| srs-wgtn   | Pentium II, 233MHz, 64MB RAM                          | OpenBSD router/firewall/VPN                                      |
| srs-w-app  | Celeron, 1.1GHz, 512MB RAM, ATA-100 (35MB/s transfer) | Primary Linux App/DB Server                                      |
| srs-w-app2 | Celeron 950MHz, 256MB RAM, SCSI-1 (4MB/s transfer)    | Secondary Linux App/DB Server                                    |
| srs-w-web  | Celeron, 1.1GHz, 256MB RAM, ATA-100                   | Primary Linux web server   |
| srs-akld   | Pentium II, 233MHz, 64MB RAM                          | OpenBSD router/firewall/VPN                                      |
| srs-a-app  | Pentium II, 233MHz, 224MB RAM                         | Tertiary Linux App/DB Server (not included in performance tests) |
| srs-a-web  | Pentium II, 233MHz, 64MB RAM                          | Secondary Linux web server (not included in performance tests)   |

The 'Auckland' servers were provided for testing of replication and failover conditions and were not used for performance testing as they were significantly underpowered. In tests of even 100 requests from a single client, one after the other, srs-a-app fell sufficiently behind as to have a load average of around 7 at the end of the test run. When accidentally included in a four request test we believe the load average exceeded 80, but it was quicker to reset the machine than to shut it down. By comparison, peak loads on the primary and secondary application servers did not exceed 10, even with nine clients in parallel while running DNS export continuously.

## 6.2 TESTING COMPONENTS

| Test type | Description   |
|-----------|---|
| 'whois'   | A 'whois' request was made to the front-end webserver. The domain used for each 'whois' was different for all parallel clients, across each test run. Each client used a separate set of 100 domains, taken from a randomly ordered list of 2,000 domains.<br>For the 'best case' tests the same domain name was used repeatedly. |

| Test type     | Description   |
|---------------|---|
| 'update'      | An 'update' request was made to the front-end webserver. Validation of the source registrar ID was disabled for the update tests, as it made the test setup unnecessarily complicated (all parsing and processing was retained, with only the final numerical comparison removed).  |
| 'DNS Export'  | The DNS export program was run, producing DNS zone files suitable for export to the DNS servers. Timings were done with the unix 'time' command.  |
| 'Replication' | The replication processes were operating during all testing. During 'whois' and DNS processing the background components of these were quiescent, since no change was occurring within the database. Processing of 'whois' and 'update' passed through the replication layer, with agreeing results received from a minimum of two application servers before returning the answer to the client. |

For transactions, the measured time includes building the XML at the client-end and signing the request, prior to submitting to the webserver, all processing to produce the response (submission and response from two application servers in parallel), delivery back to the client, parsing of the XML and validation of the received signature.

Each client machine involved in these tests (there were three) was configured as a different registrar, but all requests from that machine were 'from' the one registrar.

## 6.3 DETAILED TEST RESULTS

| Test                                    | Requests | Total Time | Maximum | Average | SD   |
|---|----------|------------|---------|---------|------|
| whois-100k-1machine-1query              | 100      | 62.4       | 0.89    | 0.62    | 0.1  |
| whois-100k-https-1machine-1query        | 100      | 84.33      | 1.13    | 0.84    | 0.09 |
| update-100k-1machine-1query             | 100      | 65.95      | 1.81    | 0.66    | 0.13 |
| update-100k-https-1machine-1query       | 100      | 101.82     | 5.73    | 1.02    | 0.53 |
| whois-loadtest-1machine-1query          | 100      | 65.31      | 0.89    | 0.65    | 0.04 |
| whois-loadtest-1machine-2queries        | 200      | 175.41     | 2.25    | 0.88    | 0.32 |
| whois-loadtest-1machine-3queries        | 300      | 350.66     | 2.69    | 1.17    | 0.25 |
| whois-loadtest-1machine-4queries        | 400      | 642.58     | 5.04    | 1.61    | 0.58 |
| update-500k-1machine-1query             | 100      | 68.78      | 1.12    | 0.69    | 0.11 |
| whois-500k-https                        | 100      | 87.62      | 1.36    | 0.88    | 0.11 |
| whois-500k-1machine-1query              | 100      | 62.57      | 1.08    | 0.63    | 0.1  |
| update-500k-https-1machine-1query       | 100      | 89         | 1.97    | 0.89    | 0.17 |
| whois-500k-https-1machine-1query        | 100      | 97.21      | 2.17    | 0.97    | 0.3  |
| whois-loadtest-2machines-4queries       | 400      | 577.91     | 6.47    | 1.45    | 0.6  |
| whois-loadtest-2machines-6queries       | 600      | 1116.28    | 6.74    | 1.86    | 0.71 |
| whois-loadtest-3machines-9queries       | 900      | 2380.97    | 12.87   | 2.65    | 2.3  |
| whois-loadtest-https-2machines-6queries | 600      | 1363.54    | 7.09    | 2.27    | 0.76 |
| whois-loadtest-https-3machines-9queries | 900      | 2862.36    | 12.74   | 3.18    | 1.74 |
| whois-loadtest-dns-3machines-9queries   | 900      | 2588.34    | 13.89   | 2.88    | 2.48 |
| update-loadtest-3machines-9queries      | 900      | 3627.08    | 15.79   | 4.03    | 2.13 |
| update-loadtest-2machines-6queries      | 600      | 1481.34    | 10.28   | 2.47    | 1.25 |

| Test  | Requests | Total Time | Maximum | Average | SD   |
|---|----------|------------|---------|---------|------|
| whois-remote-cable-bounce-1machine-1query       | 100      | 98.64      | 3.8     | 0.99    | 0.33 |
| whois-remote-https-cable-bounce-1machine-1query | 100      | 145.96     | 4.3     | 1.46    | 0.34 |
| querycustid-1x400-http                          | 400      | 3211.01    | 13.98   | 8.03    | 0.78 |
| querycustid-1x400-https                         | 400      | 3305.69    | 9.68    | 8.26    | 0.30 |
| query-1x80-large-http                           | 80       | 3133.38    | 51.89   | 39.17   | 1.88 |
| query-huge-http                                 | 13       | 1092.52    | 86.05   | 84.04   | 1.48 |

## 6.4 INSTALLATION NOTES FOR CRYPT::OPENPGP

The Perl Crypt::OpenPGP package provides a complete, RFC2440 (OpenPGP) compliant OpenPGP implementation, in "pure" perl (it makes extensive use of XS packages which call compiled (C) code directly).

Crypt::OpenPGP depends on a lot of other packages, directly and indirectly, which are not installed on a Debian system by default, many of which are not packaged by default. All but one of these can be retrieved from CPAN; the remaining package (Math::Pari) is best retrieved from the authors site, as the version needs to match the version of the underlying C math library used.

Crypt::OpenPGP depends on:

|                  |   |
|------------------|---|
| MIME::Base64     | (debian pkg: libmime-base64-perl)   |
| Compress::Zlib   | (debian pkg: libcompress-zlib-perl)   |
| Data::Buffer     | (CPAN: Data-Buffer-0.04.tar.gz)   |
| Math::Pari       | (Math-Pari-2.010303.tar.gz, author's site via CPAN link; depends on pari-2.1.3.tgz from ftp://megrez.math.u-bordeaux.fr/pub/pari/unix/) |
| Crypt::Blowfish  | (debian pkg: libcrypt-blowfish-perl)  |
| Crypt::DES       | (debian pkg: libcrypt-des-perl)   |
| Crypt::DES_EDE3  | (CPAN: Crypt-DES_EDE3-0.01.tar.gz)  |
| Crypt::RIPEMD160 | (CPAN: Crypt-RIPEMD160-0.04.tar.gz)   |
| Digest::MD2      | (debian pkg: libdigest-md2-perl)  |
| Digest::MD5      | (debian pkg: libdigest-md5-perl)  |
| Digest::SHA1     | (debian pkg: libdigest-sha1-perl)   |
| Crypt::DSA       | (CPAN: Crypt-DSA-0.12.tar.gz; many deps)  |
| Crypt::RSA       | (CPAN: Crypt-RSA-1.48.tar.gz; many deps)  |

Crypt::DSA depends on (in addition to packages already listed above):

|               |                                    |
|---------------|------------------------------------|
| Class::Loader | (CPAN: Class-Loader-2.02.tar.gz)   |
| Crypt::Random | (CPAN: Crypt-Random-1.11.tar.gz)   |
| Convert::ASN1 | (debian pkg: libconvert-asn1-perl) |
| Convert::PEM  | (CPAN: Convert-PEM-0.06.tar.gz)    |

Crypt::RSA depends on (in addition to packages already listed above):

|                       |   |
|-----------------------|---|
| Convert::ASCII:Armour | (CPAN: Convert-ASCII-Armour-1.4.tar.gz) |
|-----------------------|---|

|                    |  |
|--------------------|--|
| Crypt::CBC         | (CPAN: Crypt-CBC-2.02.tar.gz)  |
| Sort::Versions     | (CPAN: Sort-Versions-1.4.tar.gz)   |
| Tie::EncryptedHash | (CPAN: Tie-EncryptedHash-1.1.tar.gz; requires three minor changes to run without warnings) |
| Crypt::Primes      | (CPAN: Crypt-Primes-0.49.tar.gz)   |

It is best to install the dependencies (in the order listed above), then Crypt::DSA and Crypt::RSA and then Crypt::OpenPGP. The CPAN module can be used to automate some of this installation, but it is best to install the Debian packages of things first.

Done manually the whole install takes about 30 minutes, including writing these notes. Done automatically (through CPAN module ) it would probably take less time, but Math::Pari would have to be installed manually FIRST, as the package in CPAN doesn't work with the current pari-2.1.3.tgz and many other tests run by the CPAN module will fail. Fortunately Math::Pari has no other dependencies, and thus could be installed first.

Another OpenPGP-compatible package will be required (such as GnuPG) to generate the signature keys to use.

Debian packages used include:

- libdigest-md2-perl
- libcompress-zlib-perl
- libcrypt-blowfish-perl
- libcrypt-ssleay-perl
- libcrypt-des-perl
- libdate-calc-perl
- libxerces1
- libxml-xerces-perl