sinodun INTERNET TECHNOLOGIES LTD

STABLES 4, SUITE 11 • HOWBERY PARK • WALLINGFORD • OXFORDSHIRE • OX10 8BA • UK

18 February 2014

Testing the Performance of OpenDNSSEC

Table of Contents

Introduction 1

Scope 1

Test Platform 1

Testing Criteria 1

Differences between 1.4 and 2.0 2

Test scripts 3

Testing Scenarios 4

Results for 1.4 5

Scenario 1 (MySQL, Single zone add) 5
Scenario 2 (MySQL, Bulk zone add) 7
Scenario 3 (SQLite, Single zone add) 9

Results for 2.0 11

Scenario 4 (MySQL, single zone add) 11
Scenario 5 (MySQL, single zone add, generate zonelist.xml every zone add) 14
Scenario 6 (MySQL, bulk zone add) 16
Scenario 7 (SQLite, single zone add) 18

Conclusion 2.0 vs 1.4 19

Specific performance issues identified 19
Basic Performance of 2.0 19
Comments on the 2.0 code 19
Design recommendations 20
Further work 20

Appendix 1: Source code for fake signer 21

Introduction

This document describes the approach to and results of performance testing OpenDNSSEC prior to the 2.0.0.a4 release. The primary aim of this work is to determine:

- What is the baseline performance of OpenDNSSEC-1.4
- Does OpenDNSSEC-2.0 provide the expected performance improvements

Scope

We can imagine two extreme types of customer, each with very different challenges:

- An ISP with lots of small zones
- A TLD operator with a small number of large zones

In the first case they may be limited by the enforcer component. In the latter they will be limited by the speed of the signer component. The signer component and the key generation task are most likely limited by the speed of the security module used.

In this report, the work is limited to simply measuring the performance of the enforcer component when using shared keys, which is the best case scenario.

No detailed code profiling has been done, just some investigation of the 2.0 code based on the empirical observations. However it forms the basis for future benchmarking where the test scripts could be extended and added to the automated test system, and profiling can be added.

The only numbers associated with performance requirements for 2.0 are that it can 'handle' 50,000 zones. It is hoped that this report can form the basis to refine the expected performance of the enforcer in 2.0 in terms of more detailed criteria.

It should also be noted that the 2.0 release is still under development so the results here are only initial observations serving to identify the worst bottlenecks in the system. An iterative approach will be needed for the system to reach its optimum performance.

Test Platform

A Dell PowerEdge platform kindly purchased and hosted by SURFNet was used for this testing, for technical details see: https://wiki.opendnssec.org/pages/viewpage.action?pageId=3211765. The system was running RHEL, no attempt has been made to further optimise the system, database or application performance.

Testing Criteria

Three main aspects of the enforcer have been tested in this work:

- Adding new zones to the database:

  - Using the `zone add` command (a single zone at a time)
  - Using the `zonelist import` command (a bulk import)
  - The impact of writing zonelist.xml out on every zone add
- 'Enforcing' new zones (i.e. generating signconf files)
- Performing a key roll-over on existing zones

In addition, some testing was performed on the effect of using MySQL instead of SQLite3 (the default)

Differences between 1.4 and 2.0

There are significant differences between how the enforcers in 1.4 and 2.0 work.

In 1.4 there is a separation between the command line utilities and the enforcer deamon itself. For example, in 1.4 the user can use `ods-ksmutil zone add` to add a single zone to the database and re-export the updated zonelist.xml file. This can be performed without running the enforcer itself.

A separate command to run (or notify) the enforcer (which sequentially processes all configured zones) can then be issued. This will perform key generation and process each zone to produced a signconf file, which can then be consumed by the signer.

In 2.0 by default a zonelist.xml file is not generated every time a zone is added. (There is a `--xml` flag to change this). However, in the current implementation an 'internal' zone list file is exported (which re-uses the zonelist.xml format) on every zone add, which is then intended for consumption by the signer.

Also, in 2.0 the ods-enforcerd daemon must be running to use the command line utilities. The command to add new zones is altered to `ods-enforcer zone add`.

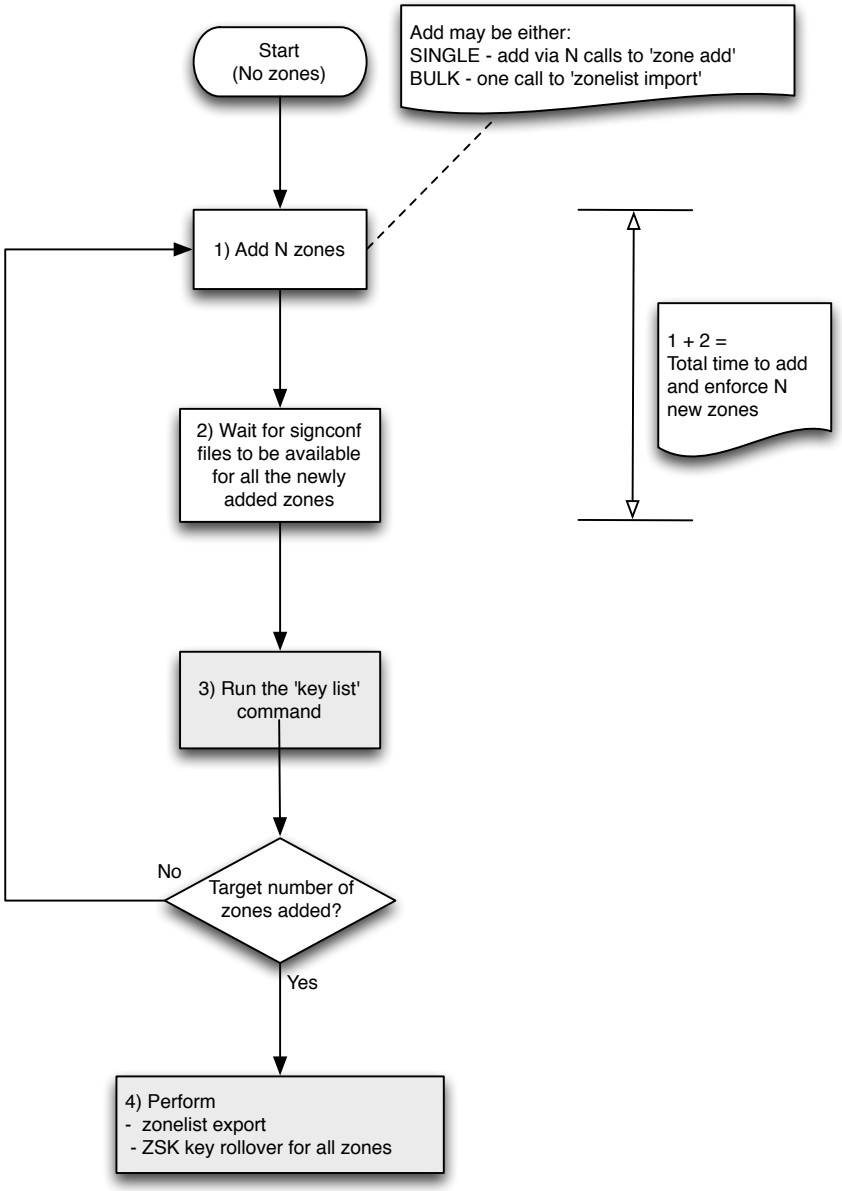
Every command issued with `ods-enforcer` is sent to the ods-enforcerd daemon and placed in a queue of tasks to be run at the daemon's discretion. The `zone add` command automatically triggers several tasks

- addition of the zone to the database
- key generation if required
- 'update' of the zone which processes the key logic
- export of the signconf file

This has the side effect that commands are issued and may return while the daemon is still to start working on the requested tasks and that therefore processing of zone adds and other tasks may be interleaved.

Test scripts

Several scripts were developed to perform various tests across the two enforcer releases. In general, the workflow tested is shown in the flow diagram below:



Notes:

1. The steps with a grey background were not run in all test as they were not necessarily required.
2. The aim of looping over multiple zone add/enforce steps is to simulate the addition of zones to an already running system.

3. In tests run on 1.4, the `–no-xml` flag was used to optimise the add, and therefore a separate '`zonelist export`' command was executed to update the zonelist.xml configuration file.
4. The time taken to add and enforcer N new zones is defined as: The interval between starting to add the first new zone and the point at which all the signconf files contain ZSK key locator information. This is the sum of steps 1 and 2, which are easily measured separately in 1.4 but not in 2.0.
5. A reasonable number for N and for the target number of zones were found by trial and error depending on how long the entire test took to run.
6. Various other administration commands were run in some tests, e.g. a zone delete after the rollover.
7. The 'ZSK key rollover' step described above is in fact only the first stage in a rollover i.e. publication of a new key.
8. In order not to pollute the logs with failed calls to the signer and to prevent the signer having any impact on the tests, a fake signer was created to receive the notifications, see Appendix 1.
9. All timings presented below are in seconds.
10. The scripts used can be found in in the OpenDNSSEC repository, test-cases-performance.d directory.

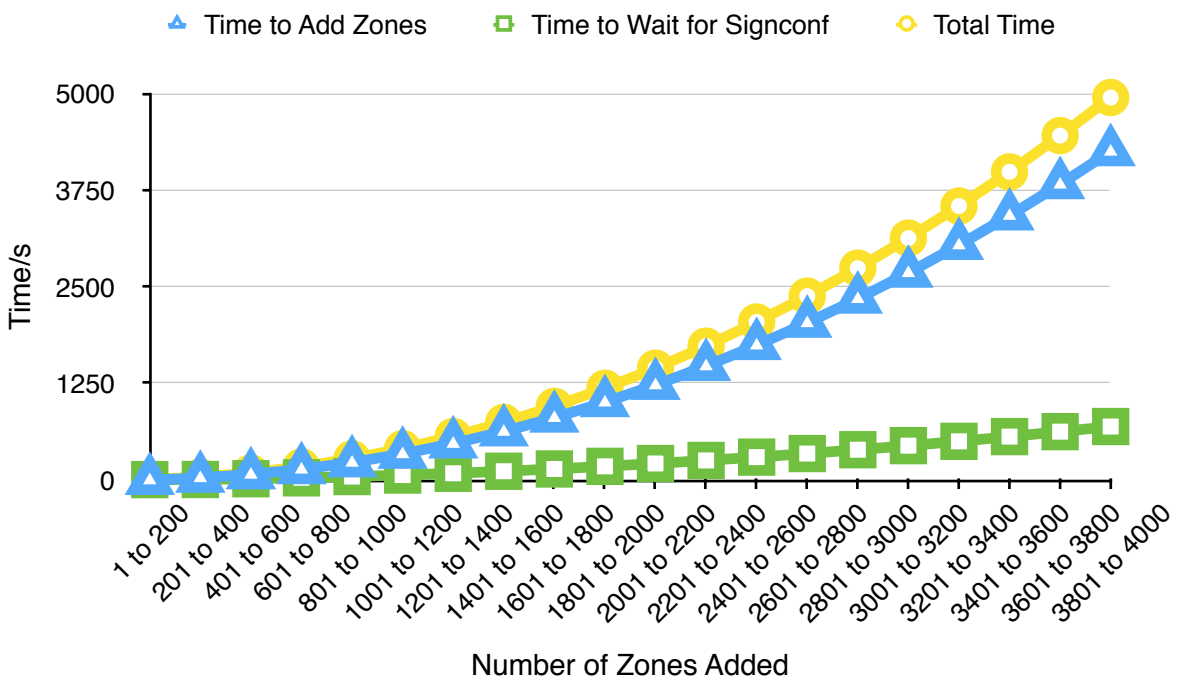
Testing Scenarios

All these scenarios were performed using shared keys (and the default policy)

#	ODS version	KASP Database	Generation of zonelist.xml every zone add?	Zone Import Method	# of zones (N)	# of repeats	Key rollover
1	1.4	MySQL	No	Single	200	20	Yes
2	1.4	MySQL	No	Bulk	500	8	No
3	1.4	SQLite3	No	Single	200	10	Yes
4	2.0	MySQL	No	Single	200	20	Yes
5	2.0	MySQL	Yes	Single	200	20	Yes
6	2.0	MySQL	No	Bulk	500	8	No
7	2.0	SQLite3	No	Single	200	10	Yes

Results for 1.4

Scenario 1 (MySQL, Single zone add)



Number of Zones Added	Time to Add Zones	Time to Wait for Signconf	Total Time	Time for key list
1 to 200	3.898	9.932	13.830	0.166
201 to 400	32.186	11.060	43.246	0.283
401 to 600	76.525	20.744	97.269	0.394
601 to 800	142.506	33.191	175.697	0.511
801 to 1000	231.102	49.345	280.447	0.633
1001 to 1200	341.707	68.740	410.447	0.785
1201 to 1400	475.458	91.725	567.183	0.880
1401 to 1600	630.981	117.933	748.914	1.002
1601 to 1800	809.127	147.405	956.532	1.081
1801 to 2000	1008.938	181.283	1190.221	1.208
2001 to 2200	1231.800	217.873	1449.673	1.340
2201 to 2400	1477.003	257.443	1734.446	1.449
2401 to 2600	1746.040	300.444	2046.484	1.541
2601 to 2800	2034.202	346.487	2380.689	1.650
2801 to 3000	2345.811	395.933	2741.744	1.805
3001 to 3200	2681.213	448.572	3129.785	1.956
3201 to 3400	3040.167	504.378	3544.545	2.063
3401 to 3600	3420.779	565.337	3986.116	2.127
3601 to 3800	3824.478	629.347	4453.825	2.215
3801 to 4000	4250.294	695.624	4945.918	2.396
Totals	**29804.215**	**5092.796**	**34897.011**	**25.485**

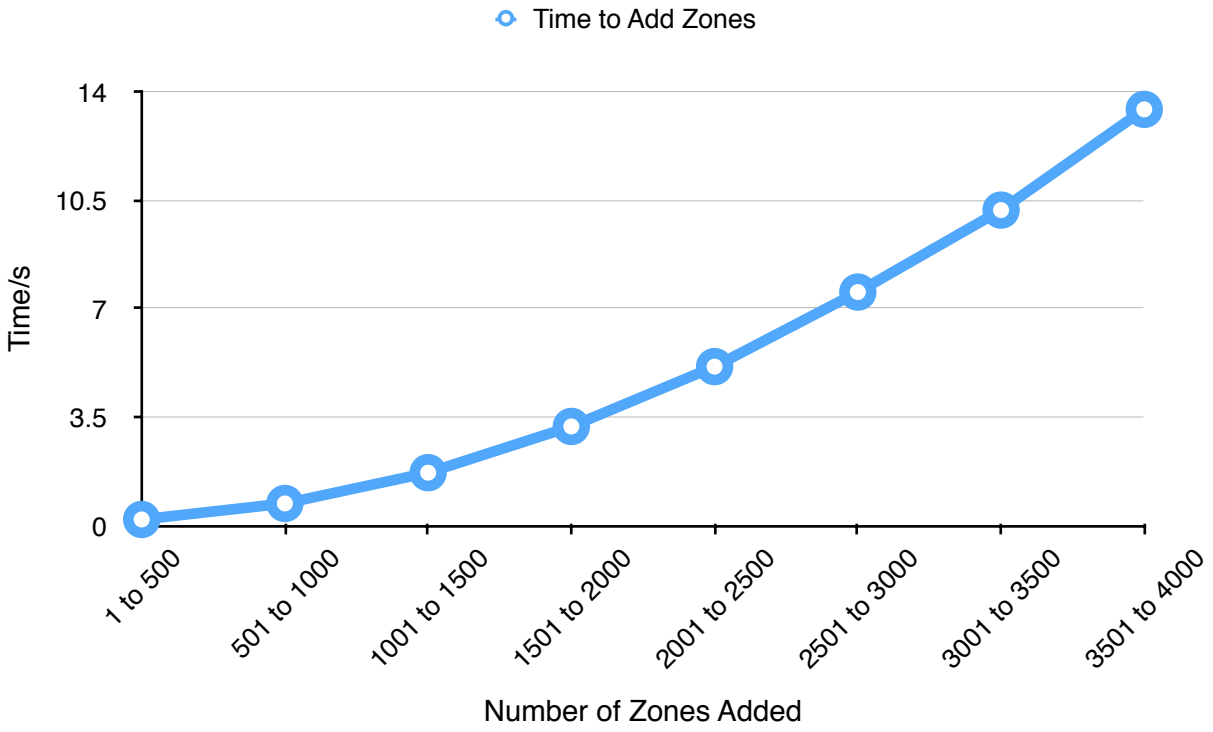
```
Sanity checks:                                  passed
Time to export zonelist:                        .056
Time to run ZSK 'key rollover' command:         98.521
Time to run enforcer following rollover:        23870.036
Time to run key list:                           3.755
Time to delete 1 zone:                          .430
```

Key observations

1. Time taken to add the new zone dominates over the time to enforce the zone
2. Time taken to add the zone and also enforcer the zone increases non-linearly
3. The time taken to export a zonelist.xml file is negligible (a experimental run with the `–no-xml` option added to the zone add command confirmed this).
4. For a system with 4000 zones (using shared keys):

Criteria	1.4 timing (s)
Average time to add 1 new zone	~21
Time to delete 1 zone	<0.5
Average time taken to enforce a new zone	~3.4
Average time taken to perform ZSK rollover per zone	~ 6 seconds per zone (6.6hrs in total)
Key list	< 4 seconds

Scenario 2 (MySQL, Bulk zone add)



Number of Zones Added	Time to Add Zones
1 to 500	0.214
501 to 1000	0.730
1001 to 1500	1.722
1501 to 2000	3.212
2001 to 2500	5.141
2501 to 3000	7.543
3001 to 3500	10.185
3501 to 4000	13.432
Total	**42.179**

Notes

1. The various times taken to run the enforcer were not measured in this test. This is because under the test conditions (where the enforcer daemon is not running while the zones are added) these timings are completely independent of how the zones were added to the system.

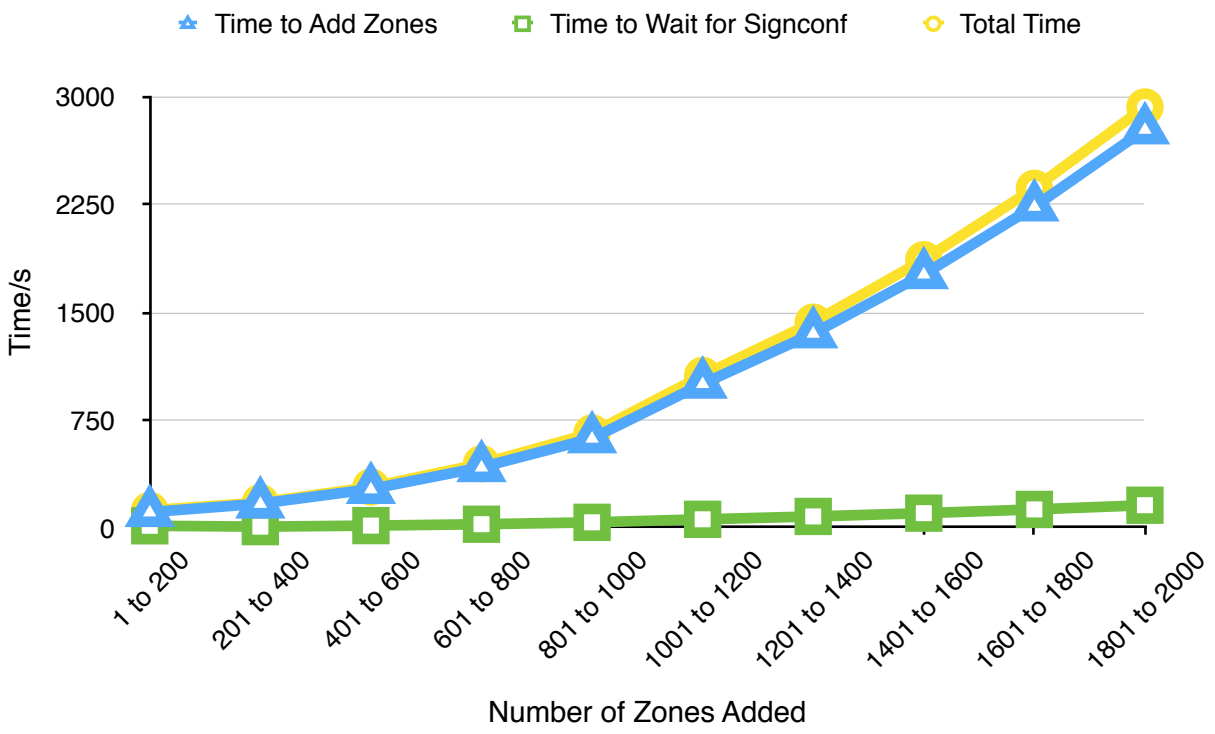
2. Note that due to the details of the implementation of the ‘zonelist import’ command the process of adding zones in this way results in a mixture of zone updates (for existing zones) and zone adds (for new zones).

Key observations

1. Bulk add is clearly significantly faster than single zone add, so in this case the time taken to run the enforcer would dominate (see scenario 1).

2. For a system of 4000 zones, a bulk zone add takes ~ 0.03 seconds per zone.

Scenario 3 (SQLite, Single zone add)




```
Sanity checks:                                  passed
Time to export zonelist:                        .024
Time to run ZSK 'key rollover' command:         19.673
Time to run enforcer following rollover:        14093.842
```

Number of Zones Added	Time to Add Zones	Time to Wait for Signconf	Total Time	Time for key list
1 to 200	108.129	13.525	121.654	0.169
201 to 400	167.486	7.515	175.001	0.408
401 to 600	268.709	14.637	283.346	0.743
601 to 800	420.604	24.403	445.007	1.165
801 to 1000	621.182	37.718	658.900	1.857
1001 to 1200	1000.989	58.567	1059.556	2.543
1201 to 1400	1351.288	78.528	1429.816	3.333
1401 to 1600	1763.076	101.387	1864.463	4.205
1601 to 1800	2234.001	127.323	2361.324	5.226
1801 to 2000	2770.076	156.672	2926.748	6.470
Totals	**10705.540**	**620.275**	**11325.815**	**26.119**

Key observations

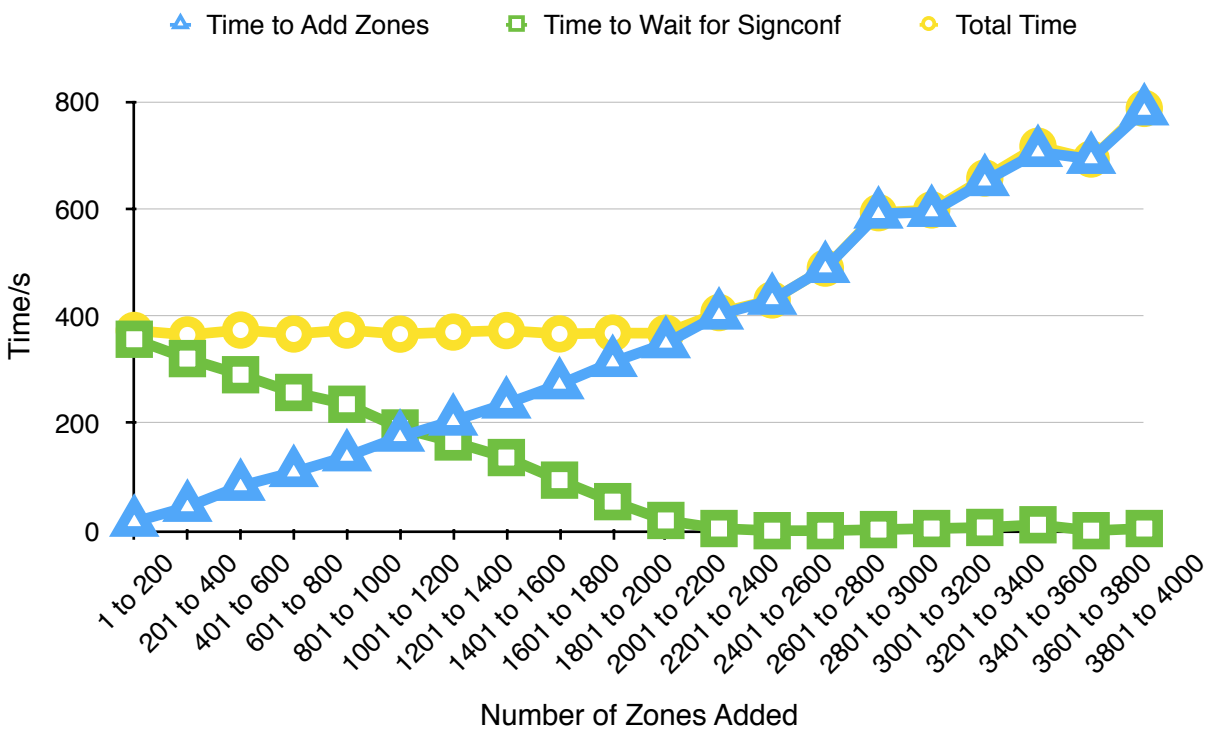
1. Again, the time taken to add the zone dominates over the time taken to enforce the zone, and increases non-linearly.

2. Total time taken to add 2000 zones is almost 3 times greater than that taken with MySQL.

3. The total time taken for the enforcer to run is ~ 14% less that than on MySQL.
4. With both MySQL and SQLite3 the key list times increase as more zones are added, however the slow down is significantly faster with SQLite3.
5. Again, the time taken to export a zonelist.xml file is negligible.
6. Unfortunately the time for the rollover cannot be directly compared with that from scenario 1 (MySQL) as it is for 2000 rather than 4000 zones, but is 7 seconds per zone. This compares with 6 seconds per zone in scenario 1.

Results for 2.0

Scenario 4 (MySQL, single zone add)



Number of Zones Added	Time to Add Zones	Time to Wait for Signconf	Total Time	Time for key list
1 to 200	16.354	356.781	373.135	0.201
201 to 400	44.155	320.703	364.858	0.396
401 to 600	83.277	290.637	373.914	0.593
601 to 800	109.001	257.566	366.567	0.786
801 to 1000	138.252	235.624	373.876	0.989
1001 to 1200	174.905	191.416	366.321	1.181
1201 to 1400	204.805	165.369	370.174	1.380
1401 to 1600	236.670	136.299	372.969	1.577
1601 to 1800	272.529	94.210	366.739	1.765
1801 to 2000	314.123	54.121	368.244	1.962
2001 to 2200	349.013	19.044	368.057	1.623
2201 to 2400	402.437	4.011	406.448	2.016
2401 to 2600	430.045	0.002	430.047	2.348
2601 to 2800	489.351	0.002	489.353	2.528
2801 to 3000	590.618	2.006	592.624	2.943
3001 to 3200	593.837	4.011	597.848	2.338
3201 to 3400	651.224	6.016	657.240	3.330
3401 to 3600	705.489	11.026	716.515	3.527
3601 to 3800	692.642	0.002	692.644	3.683

Number of Zones Added	Time to Add Zones	Time to Wait for Signconf	Total Time	Time for key list
3801 to 4000	783.001	4.011	787.012	3.246
Totals	**7281.728**	**2152.857**	**9434.585**	**38.412**

```
Time to export zonelist:          3.227
Sanity checks:                    passed
Time to run ZSK rollover:         3905.137
Time to delete 1 zone:            5.208
```

Key observations

1. Compared with the 1.4 results, the timings show a very different progression as zones are added. Only after looking at the code and logs was it possible to determine the reason for this.

    A. The time taken to add each batch of 200 zones up to a total of 1000 zones is dominated decreasingly by the time taken to wait for the enforcerd to create signconf files containing key locator information.

    B. From 1000 to 2000 total zones the time taken is dominated increasingly by the time needed to add the zones.

    C. For the first 2000 zones the total time taken to add each batch of 200 zones is roughly constant. This is a result of the slowing zone adds leaving gaps for the enforcerd to generate signconf files containing key locator information.

    D. After 2000 zones the zone add totally dominates and the enforcerd appears to take no time at all to generate signconf files.

2. The total time to add 4000 zones is 9434 seconds ~ 3.6 times faster than in OpenDNSSEC 1.4

3. For a system with 4000 zones (using shared keys):

Criteria	1.4 timing (s)	2.0 timings (s)
Average time to add 1 new zone	~21	~4
Time to delete 1 zone	<0.5	~5
Average time taken to enforce a new zone	~3.4	[~0.02]
Average time taken to perform ZSK rollover per zone	~6 (6.6hrs in total)	~1 (1hr in total)
Key list	< 4 seconds	~4

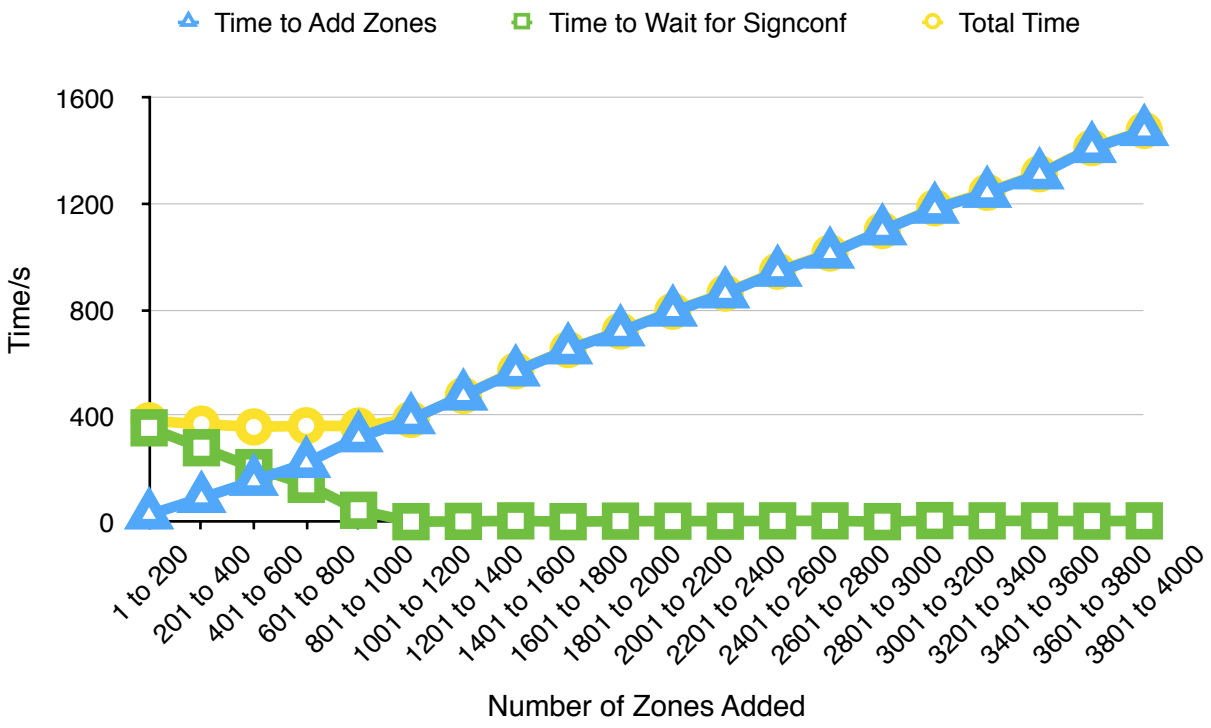
    A. The time taken to export a zonelist.xml file is ~ 3 seconds. Over 50 times slower than in OpenDNSSEC 1.4. After investigation it could be seen that this was due to a basic implementation of the export that could be optimised. This work is underway.

    B. The time to add a single new zone is ~4 seconds. After investigation it could be seen that this is dominated by the time taken to export the 'internal' zone list file, as described above.

C. The time taken to enforce a new zone appears to be 0.02 seconds per zone. However, this is because of the dominating zone add time. Further investigation is needed to change the test script to separate out these effects.

D. The time taken to perform a rollover is much faster than 1.4, however this limited set of results do not provide any information on how this number scales. ***In the best case scenario, if this figures is independent of the number of zones configured, a projection to a system of 50,000 zones produces a figure of 14 hours to roll the (shared) ZSKs of all the zones.***

Scenario 5 (MySQL, single zone add, generate zonelist.xml every zone add)



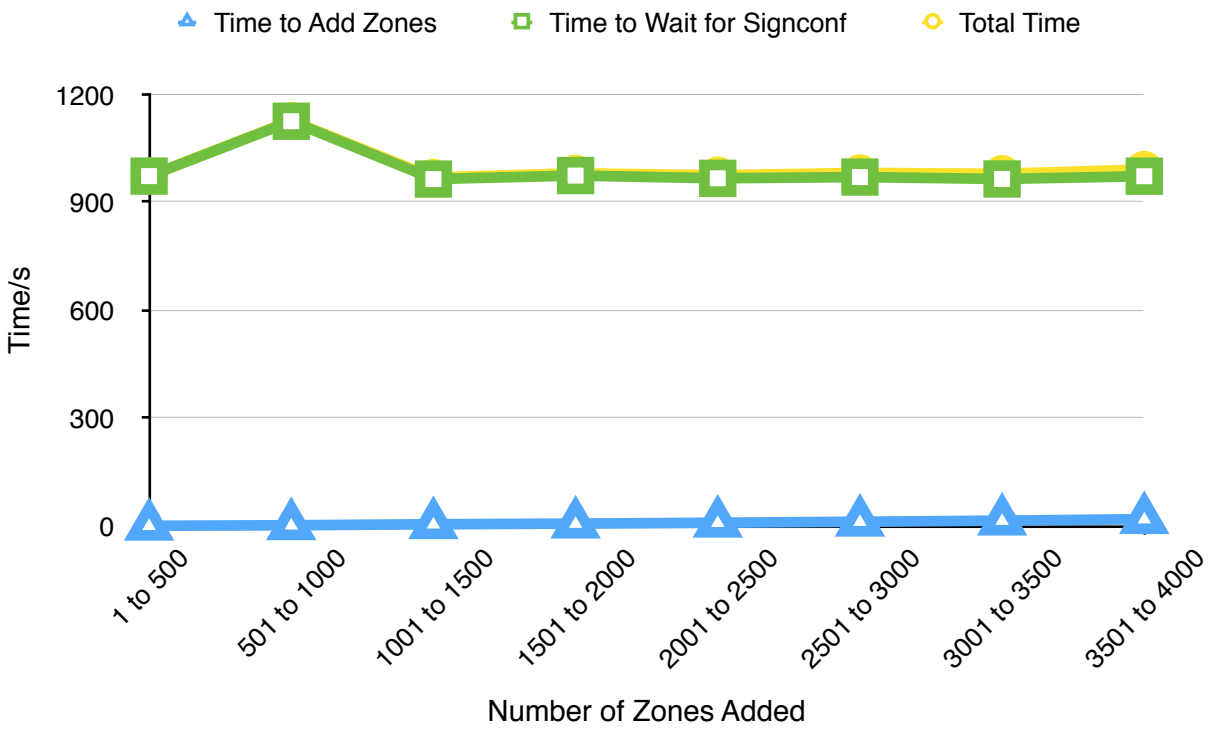
Number of Zones Added	Time to Add Zones	Time to Wait for Signconf	Time for key list	Total Time
1 to 200	26.559	354.822	381.381	0.200
201 to 400	89.159	279.598	368.757	0.398
401 to 600	153.714	204.455	358.169	0.590
601 to 800	221.179	141.300	362.479	0.786
801 to 1000	318.365	43.118	361.483	0.983
1001 to 1200	386.635	0.002	386.637	1.225
1201 to 1400	476.043	1.004	477.047	1.375
1401 to 1600	567.344	3.009	570.353	1.168
1601 to 1800	650.969	0.002	650.971	1.723
1801 to 2000	719.070	2.006	721.076	1.471
2001 to 2200	793.685	2.006	795.691	1.618
2201 to 2400	862.485	2.006	864.491	1.768
2401 to 2600	944.393	3.009	947.402	2.537
2601 to 2800	1012.899	3.008	1015.907	2.059
2801 to 3000	1100.974	0.002	1100.976	3.060
3001 to 3200	1182.692	4.011	1186.703	3.121
3201 to 3400	1242.776	3.009	1245.785	3.332
3401 to 3600	1312.644	3.008	1315.652	3.513
3601 to 3800	1412.178	2.006	1414.184	3.095
3801 to 4000	1476.453	3.009	1479.462	3.462
Totals	**14950.216**	**1054.390**	**16004.606**	**37.484**

Key observations

1. Adding a zonelist.xml export to every zone add command has changed the cross over point between the zone add and enforcer to occur earlier at ~ 600 zones.

2. The total time to add 4000 zones has approximately doubled, confirming the issue with the zonelist.xml export seen above.

Scenario 6 (MySQL, bulk zone add)



Number of Zones Added	Time to Add Zones	Time to Wait for Signconf	Total Time
1 to 500	0.307	973.239	973.546
501 to 1000	1.713	1125.978	1127.691
1001 to 1500	4.556	965.060	969.616
1501 to 2000	6.329	974.727	981.056
2001 to 2500	8.773	967.175	975.948
2501 to 3000	11.553	970.555	982.108
3001 to 3500	14.521	965.322	979.843
3501 to 4000	18.692	973.003	991.695
Totals	**66.444**	**7915.059**	**7981.503**

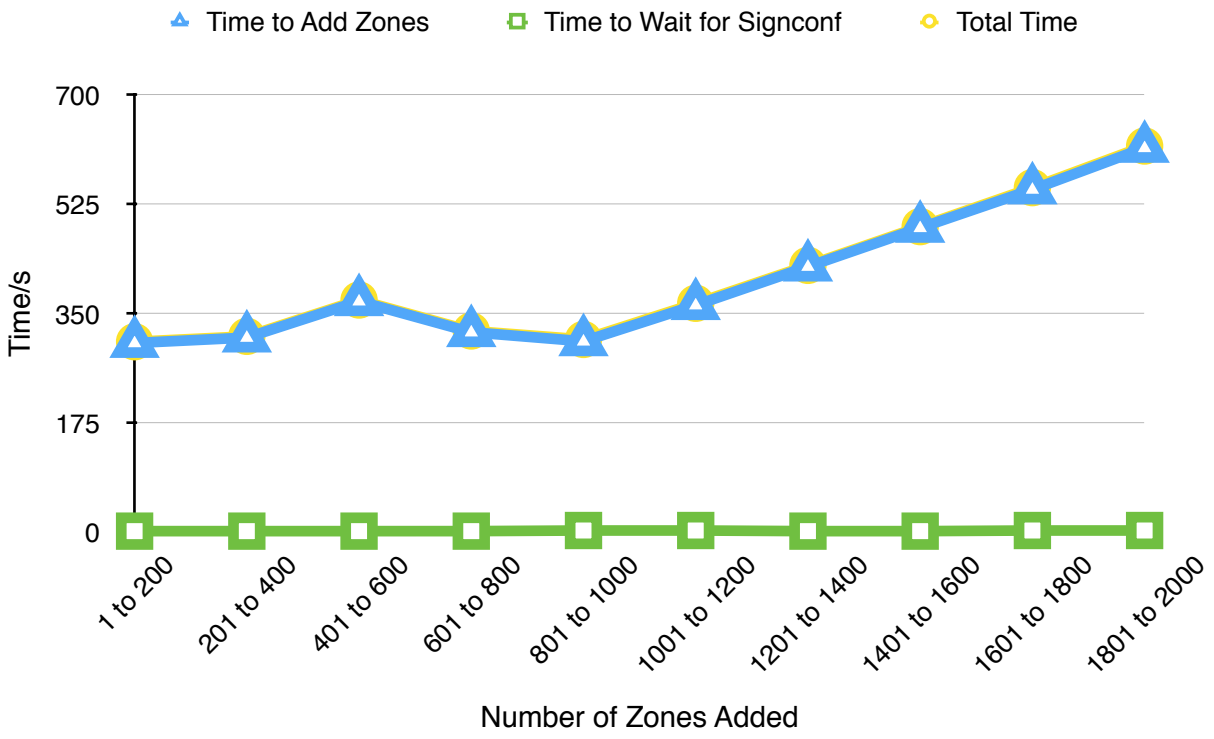
Key observations

1. The bulk zone add command is significantly faster than the single zone add.
2. In this case the time taken to enforce the zone completely dominates:
    1. On a system of 4000 zones, the approximate time to enforce each new zone is ~2 seconds per zone, compared with ~3.4 seconds in 1.4.
    2. It is not immediately clear why the enforcement of a new zone is twice as slow as the key rollover measured in scenario 4, which in principle a similar operation in this test. This warrants further investigation.

3. However importantly, this time is flat with the number of zones added, not increasing as in 1.4

4. **A projection to a system of 50,000 zones produces a figure of 28 hours to enforce all the zones in the system.**

Scenario 7 (SQLite, single zone add)



Number of Zones Added	Time to Add Zones	Time to Wait for Signconf	Total Time	Time for key list
1 to 200	303.038	2.006	305.044	0.072
201 to 400	311.693	2.006	313.699	0.140
401 to 600	369.840	2.006	371.846	0.207
601 to 800	320.297	2.016	322.313	0.279
801 to 1000	305.718	3.008	308.726	0.351
1001 to 1200	363.569	3.008	366.577	0.424
1201 to 1400	425.258	2.006	427.264	0.483
1401 to 1600	487.244	2.006	489.250	0.549
1601 to 1800	548.638	3.008	551.646	0.619
1801 to 2000	615.206	3.008	618.214	0.690
Totals	**4050.501**	**24.078**	**4074.579**	**3.814**

```
Time to export zonelist:          .720
Sanity checks:                    passed
```

Key observations

1. A brief test was done with SQLite, which will be re-run once the underlying issues found with the zone add are fixed. Clearly the `zone add` command completely dominates in this instance, being very roughly twice as slow as the comparable results from scenario 4 (MySQL, single add).

Conclusion 2.0 vs 1.4

Specific performance issues identified

1. Clearly, the zone add functionality needs to be optimised and no real conclusions about the performance of this command can be drawn until this is done.

2. It was also observed that there are repeated, unnecessary, update commands being sent to the signer (As an example - during the zone add commands in scenario 4 update messages are sent for a zone even when the signconf file for that zone doesn't contain any key locator information). At first sight these unneeded messages appear to be due to way the database layer and the schedular are implemented and used. This may have a significant impact on the signer performance.

Both the above issues have been raised with the developers and fixes are in progress.

Basic Performance of 2.0

In terms of the performance of purely enforcing zones, the key comparisons for systems with 4000 zones, MySQL, shared keys are:

Criteria	1.4 timing (s)	2.0 timings (s)
Average time taken to enforce a new zone	~3.4	~2
Average time taken to perform ZSK rollover per zone	~6 (6.6hrs in total)	~1 (1hr in total)

2.0 is clearly faster than 1.4 and also the time taken to enforce a new zone is flat up to 4000 zones, providing improved scalability over 1.4.

With the current 2.0 implementation a system with 50,000 zones, MySQL, shared keys would take 14 hours to perform the first stage of a key rollover on all the configured zones.

Comments on the 2.0 code

As part of this work a lot of investigation of the code for 2.0 was performed. This was needed in order to understand what was actually being measured. This investigation raised a few comments on the code that warrant further examination:

1. Despite being able to specify the number of worker threads in the configuration 2.0 doesn't make significant use of threads because there is only ever one item in the scheduler queue.

2. The scheduler appears to have been dropped in as a straight copy from the signer. As a result of this there are many unused code paths that are likely to make debugging and maintenance difficult (For example there is code for a totally unused FIFO queue). They may also introduce security risks.

3. The database ORM layer is very complex, it has no documentation and no active developer who understands it.

Design recommendations

1. The unneeded code in the enforcerd version of the scheduler should be removed.
2. The scheduler should be re-worked to make more efficient use of threads.
3. The database layer should either be documented by the person who wrote it or completely rewritten.

Further work

Further work is recommended in the following areas:

- Re-asses `zone add` after the implementation is optimised
- Profile the zone enforcement and key rollover to better understand the implementation
- Perform benchmarking 2.0 up to 10s of 1000s of zones, and profile how the rollover time scales with increased number of zones.
- Perform some tests with pre-generated keys to ensure no degradation of performance
- Combine these results with performance data for HSMs and SoftHSM to understand which will realistically limit the performance criteria that depend on key generation.

Appendix 1: Source code for fake signer

```
#!/bin/bash
while [ 1 ] ; do
  rm ../../../../root/local-test/var/run/opendnssec/engine.sock
  cat /dev/null | /usr/bin/nc -nlU ../../../../root/local-test/var/run/opendnssec/
engine.sock >> /tmp/fakesigner.log
done
```