CS424 Assignment 3

Jacob Honoroff, Benny Tsai

December 5, 2005

1 Cryptographic Benchmarking

- Program: Benchmark.java
- Script: bench.script

We generated tests for each HMac for SHA1 and MD5, AES encrypt/decrypt, RSA encrypt, decrypt, and key generation , and DSA sign/verify and key generation. We used a single 16-byte block for the symmetric algorithms, and a single block for each of the assymetric algorithms as well by encrypting/signing a 16-byte array, and decrypting/verifying the result. We did 10000 trials for each of the encrypt/decrypt/sign/verify computations, and 10 trials for each of the key generation since they were much slower and too many trials would have been infeasible. The following table shows the mean time for each computation:

Computation	Avg. time in ms
AES enc/dec	0.0029
HmacSha1	0.0209
HmacMD5	0.0109
DSA key generation	5659.7
DSA sign	3.849
DSA Vrfy	7.3338
RSA key generation	201.1
RSA Encrypt	0.4984
RSA Decrypt	5.3101

2 Comparison between DNS and PK-DNSSEC

- Simnet Programs: DNSSECEval.java, DNSEval.java
- Simnet Scripts: dnsseceval.java, dnseval.java
- Perl scripts: parseeval.pl, getsizes.pl, gettimes.pl

DNSSECEval.java and DNSEval have the same functionality, the only difference is that DNSSECEval extends DNSSEC and DNSEval extends DNS. The both have a function that can be called from the simnet UI, `evaldns`. This function spawns a thread that generates the DNS requests.

The Eval classes are run from 19 nodes on the network that act as stubs. The time between startup for each of the nodes is assumed to be negligible.

2.1 Request Generation

To model real-world traffic scenarios, 10 intervals are used where within each interval, the rate of packets is proportional to a sample on a sine wave. So the packet rate peaks during intervals 4-5. The packet rate ranges from approximtely 20 requests ms to 200 requests per second.

To follow the correct request distribution, several random numbers are choses using Math.random(). The first is used to generate a failed request 15% of the time and otherwise generate a proper request. The second is used to get an NS query 5% of the time and otherwise get an A query. The last is used to get the correct breakdown of different top-lovel, same top-level, and same second-level requests.

Three Vectors are stored for each of A queries and NS queries where each vector holds requests that fall into the correct top-level / second-level category. These vectors are computed at the beginning of the `evaldns` call based on the node's own name, which is computed using a lookup of its IP address into hard-coded tables of IP addresses and node names. A valid request is thus generated by using the correct vector given which domain the request should be in and whether it is an A record or an NS record. A request that will fail is generated by appending the given domain suffix to a random string.

2.2 Results

Note: We do not believe we obtained good results for the DNSSEC part. In our run, we received NullPointerExceptions from what we figure must be a bug in our implementation. We also get output in our run that the EDU-NS and COM-NS DNS Processor threads get exceptions and finish. This seems not good.

Nonetheless we tried to make whatever observations we could given the results we got:

- The computational time and bandwidth is high at the beginning of the run for DNSSEC. We believe this is because the resolvers' caches are not yet populated so more of the crypto routines have to be run.

  Observe the following data from interval 0 for DNS

  ```
  Interval 0
  Num Packets: 617
  Total packet size: 42050
  ```

```
Total time: 182
Num Requests: 190
Avg time per packet: 0.294975688816856
Avg request size: 221.315789473684
Avg packet size: 68.1523500810373
```

and for DNSSEC

```
Interval 0
Num Packets: 636
Total packet size: 240390
Total time: 3742
Num Requests: 190
Avg time per packet: 5.88364779874214
Avg request size: 1265.21052631579
Avg packet size: 377.971698113208
```

In this interval, where the caches are not yet populated, we have a much larger average time per packet and average request size for DNSSEC than DNS, which is what we would expect.

- We expected to see high computation times during times of high traffic for DNSSEC, but we did not get this result. Possible explanantions include that we did not use a high enough traffic rate, or that the bug that caused the null exceptions made it so that there was not the expected load on the servers.
- There is a strange dropoff in the diversity of packet sizes partway through the run of DNSSEC. This is consistent with a lack of RSA Decryptions after a certain point early on in the run that was noticed when gathering crypto for the SK-DNSSEC evaluation. Again, this may be due to the bug that killed off some of our DNS server threads

3 SK-DNSSEC Evaluation

- Program: Modified SimnetCryptoEngine.java

We modified SimnetCryptoEngine.java to print output whenever a crypto routine was called during a run of our evaulation program. Here are the results:

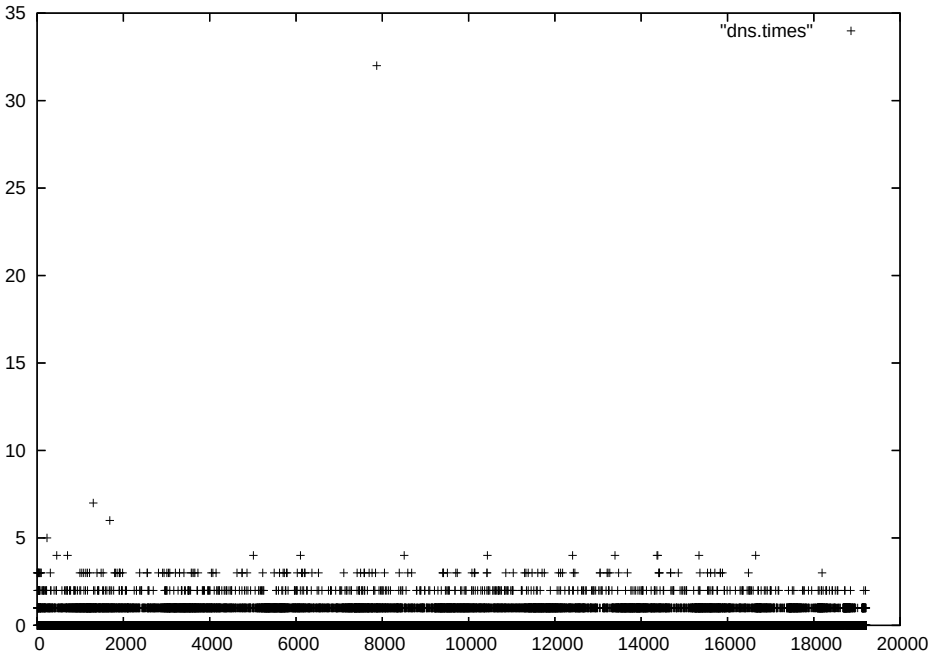

Figure 1: DNS computational load. X-axis processing times, Y-axis sample number

Routine	Amount
DSA Verification	12186
DSA Sign	3171
HMac	16763
RSA Decrypt	19
RSA Encrypt	13529

Once again, we have strange data regarding the amount of RSA decrypts which we believe is due to the thread killing bug. By using SK-DNSSEC we would save on all of the above computations according to our results from the crypto benchmarking.

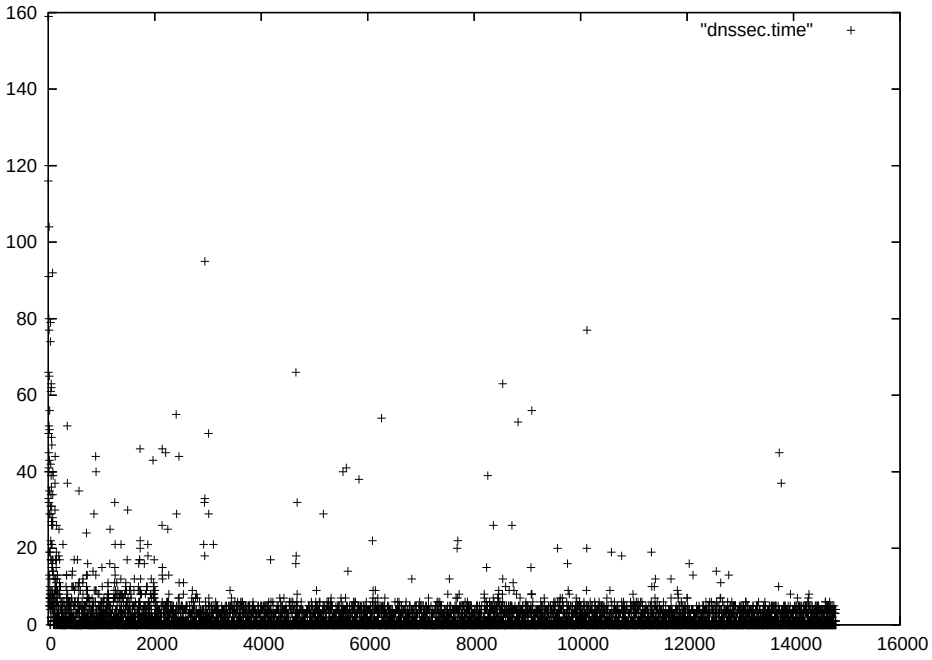

Figure 2: DNSSEC computational load. X-axis processing times, Y-axis sample number

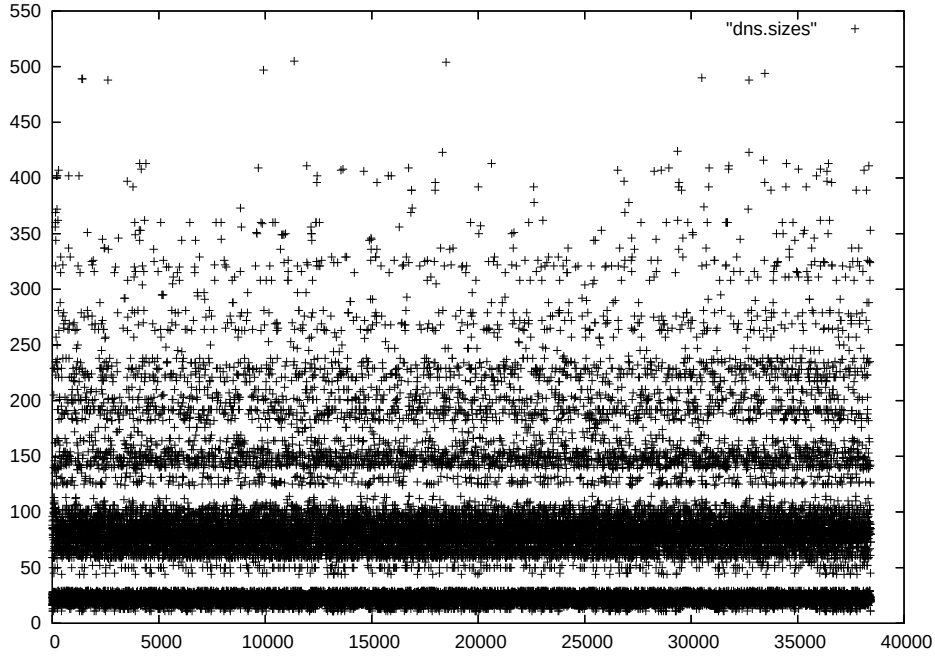

Figure 3: DNS bandwidth load. X-axis packet size, Y-axis packet number

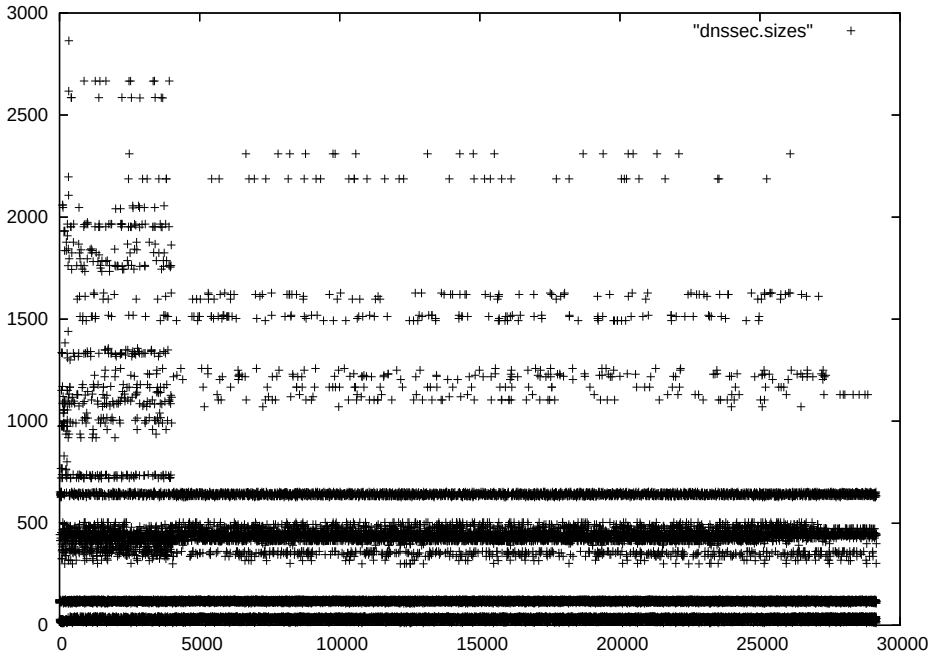

Figure 4: DNSSEC bandwidth load. X-axis packet size, Y-axis packet number