Message Digests for DNS Zones

Duane Wessels

DNS-OARC 26, Amsterdam

October 13, 2018

Overview & Outline

- Internet Draft proposing message digest over zone contents and a new ZONEMD RR type
  - Coauthors: P. Barber, W. Hardaker, W. Kumari, M. Weinberg
- Motivation
  - Channel security vs Data security
  - Alternatives considered
- How it works
  - Digest calculation algorithm
  - A simple example
  - Verifying the digest
- Implementation Experience
- Can it work with large, dynamic zones?

Motivation

Motivation

- Given a zone file, how can you tell if it's authentic?
- It should be possible to verify a zone's authenticity:
  - Independent from *how* it was received
  - Independent from *where* it was received
  - As a self-contained zone file
  - Using DNSSEC for strong security
  - *Before* being loaded into a name server
- What do we mean by *authentic*?
  - As published by the zone owner / operator
  - Complete
  - No records added, removed, or modified

Why?

- Historically, zone distribution was simple:
  - Master server
  - Small number of secondary servers
  - Transfer secured with TSIG (RFC 2845)
- Today's complexity
  - Multiple, third-party DNS providers
  - Wider distribution for anycast
  - RFC 7706 and "Hyperlocal Root"
- Other uses
  - RPZ – Response Policy Zones
  - CZDS – Centralized Zone Distribution System
  - Catalog Zones

Is This New?

- RFC 2065 (“DNSSEC v1” 1997) proposed a Zone Transfer (AXFR) SIG
- Later dropped from RFC 2535 (1999)
- Similar to this proposal
  - But only for zone transfer?
  - Any only for signed RRsets?
  - Less well specified

“SIG(AXFR) was rejected because it required putting the zone into canonical order and calculating the signature, in the case of dynamic update this is a real expensive operation, thus we got rid of it.” -- Olafur

Channel vs Data Security

Channel Security

- Protects data in transit
- Authenticates endpoints
- Places trust in a “server”
- Ephemeral

Data Security

- Protects data at rest
- Authenticates the data
- Places trust in “publisher”
- Independent of transport

Doesn't DNSSEC Already Solve This?

- Certainly DNSSEC protects clients from false data
- Does not protect consumers of zone files
  - Zone file consumers *could* validate all signatures and denial of existence records...
- Delegation records are unsigned
- DNSSEC does not prevent resolvers from sending queries to an incorrect (eavesdropping) name server
- Use of DNS data other than by users / validators
  - "control plane"
  - RPZ
  - SOA
  - uses not yet envisioned

Root Zone 2018080100	
10,773	RRsets
1,400	Signed
9,373	Unsigned

Why not just use...

- PGP
- S/MIME
- TLS/HTTPS
- TSIG
- SIG(0)
- DNSSEC signature over unsigned records
- One hash/digest per RRset

How It Works

New RR Type: ZONEMD

- At zone apex
- Four fields
  - Serial
  - Digest Type
  - Reserved
  - Digest

- Should ZONEMD Digest Type just use IANA protocol registry for DS digest types?
- More on “Reserved” later...

Digest Calculation Process

1. Add a “placeholder” ZONEMD record
2. Sort zone records using DNSSEC canonical ordering
    1. RRSets having same owner sorted by numeric RR type
3. Optionally sign zone
4. Calculate digest
    1. Over concatenation of sorted RRs in canonical on-the-wire format
    2. All records
    3. Exclude any extra SOA
    4. Include ZONEMD placeholder
    5. Exclude ZONEMD RRSIGs if zone is signed
5. Update Digest field of (placeholder) ZONEMD record
    1. Update ZONEMD signatures if zone is signed

A Simple Example

@	IN	SOA	2018040900 1800 900 604800 86400
@	IN	NS	ns1.other.zone
@	IN	NS	ns2.other.zone
www	IN	A	192.168.0.1
www	IN	AAAA	FC00:7F::1
...			

Start with a zone file

A Simple Example

```
@     IN    SOA    2018040900 1800 900 604800 86400
@     IN    NS     ns1.other.zone
@     IN    NS     ns2.other.zone
www   IN    A      192.168.0.1
www   IN    AAAA   FC00:7F::1
...
@     IN   ZONEMD 2018040900 1
          00000000000000000000000000000000000000000
```

Add placeholder ZONEMD record

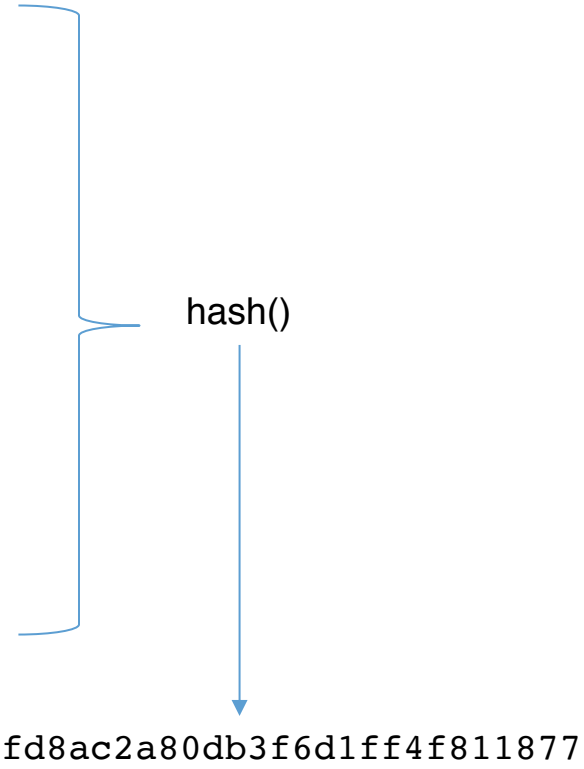
A Simple Example

@	IN	NS	ns1.other.zone
@	IN	NS	ns2.other.zone
@	IN	SOA	2018040900 1800 900 604800 86400
@	IN	ZONEMD	2018040900 1 0 00000000000000000000000000000000000000000000
www	IN	A	192.168.0.1
www	IN	AAAA	FC00:7F::1
...			

Sort

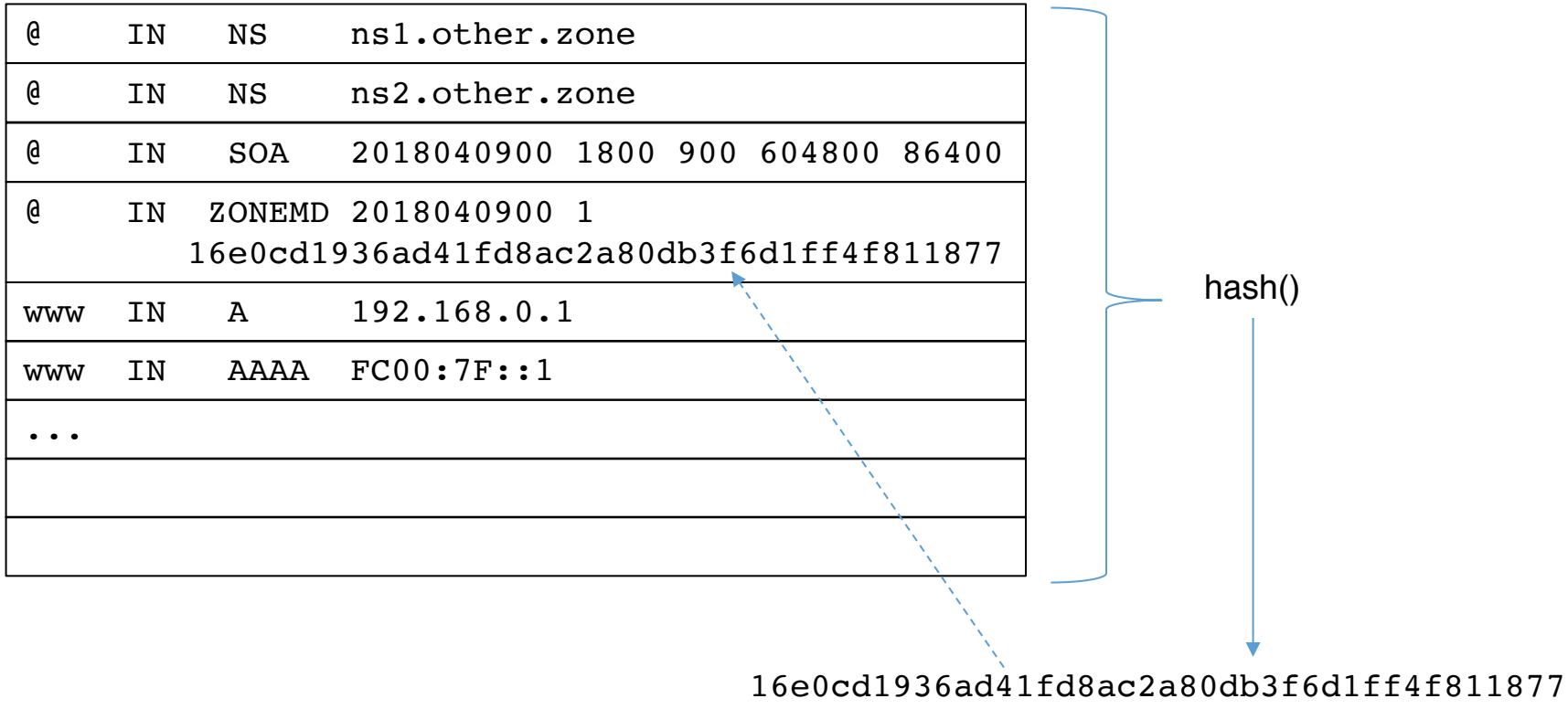
A Simple Example

@	IN	NS	ns1.other.zone
@	IN	NS	ns2.other.zone
@	IN	SOA	2018040900 1800 900 604800 86400
@	IN	ZONEMD	2018040900 1 00000000000000000000000000000000000000000000
www	IN	A	192.168.0.1
www	IN	AAAA	FC00:7F::1
...			



Calculate digest

A Simple Example



Update ZONEMD digest field

Digest Verification

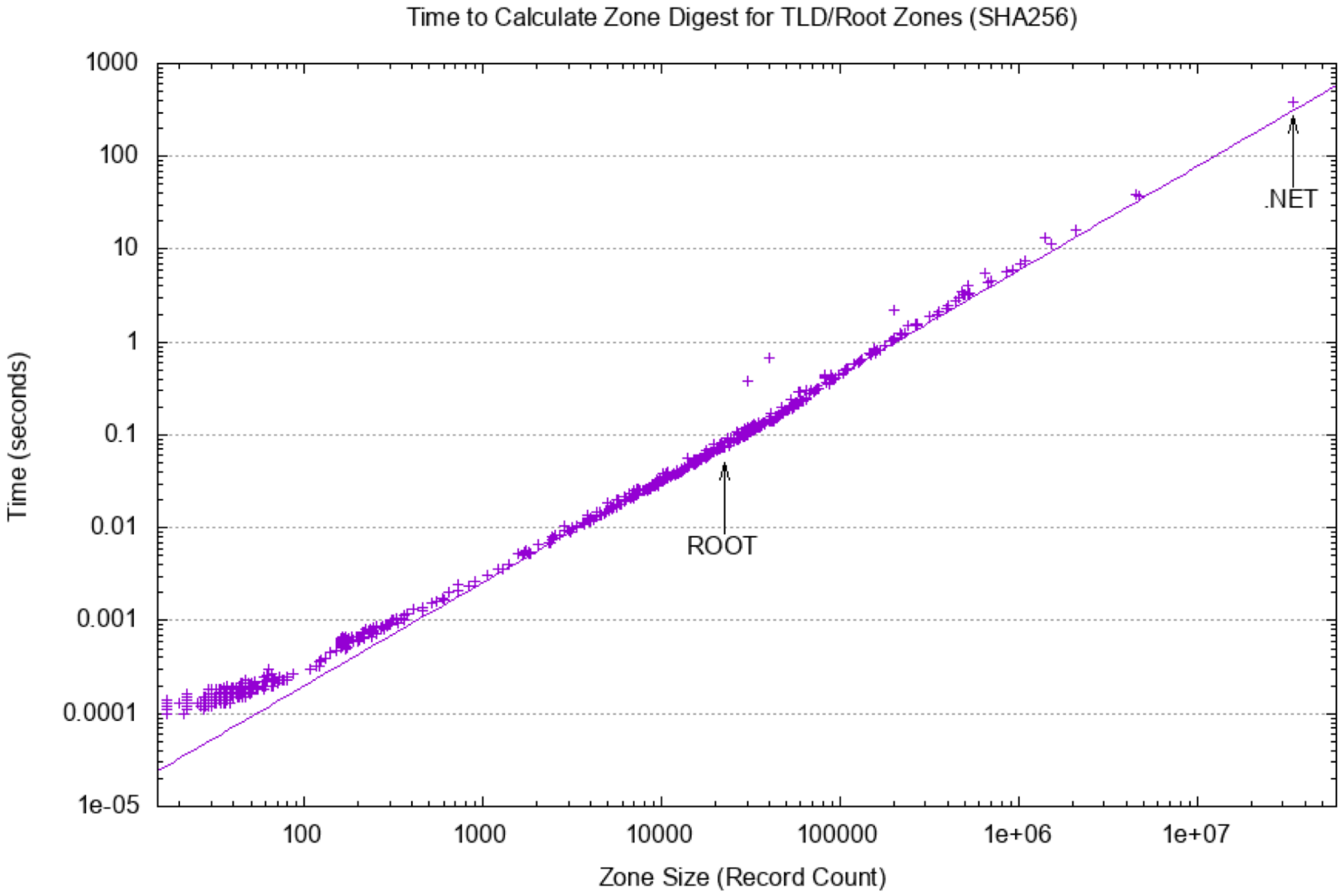
- Determine if zone should have DNSSEC signatures
- Determine if ZONEMD record provably does or does not exist if signed
- Validate SOA and ZONEMD signatures if signed
- Check for matching Serial in SOA and ZONEMD
- Check that ZONEMD digest type is supported
- Calculate zone digest and compare to ZONEMD digest field

Implementation Experience

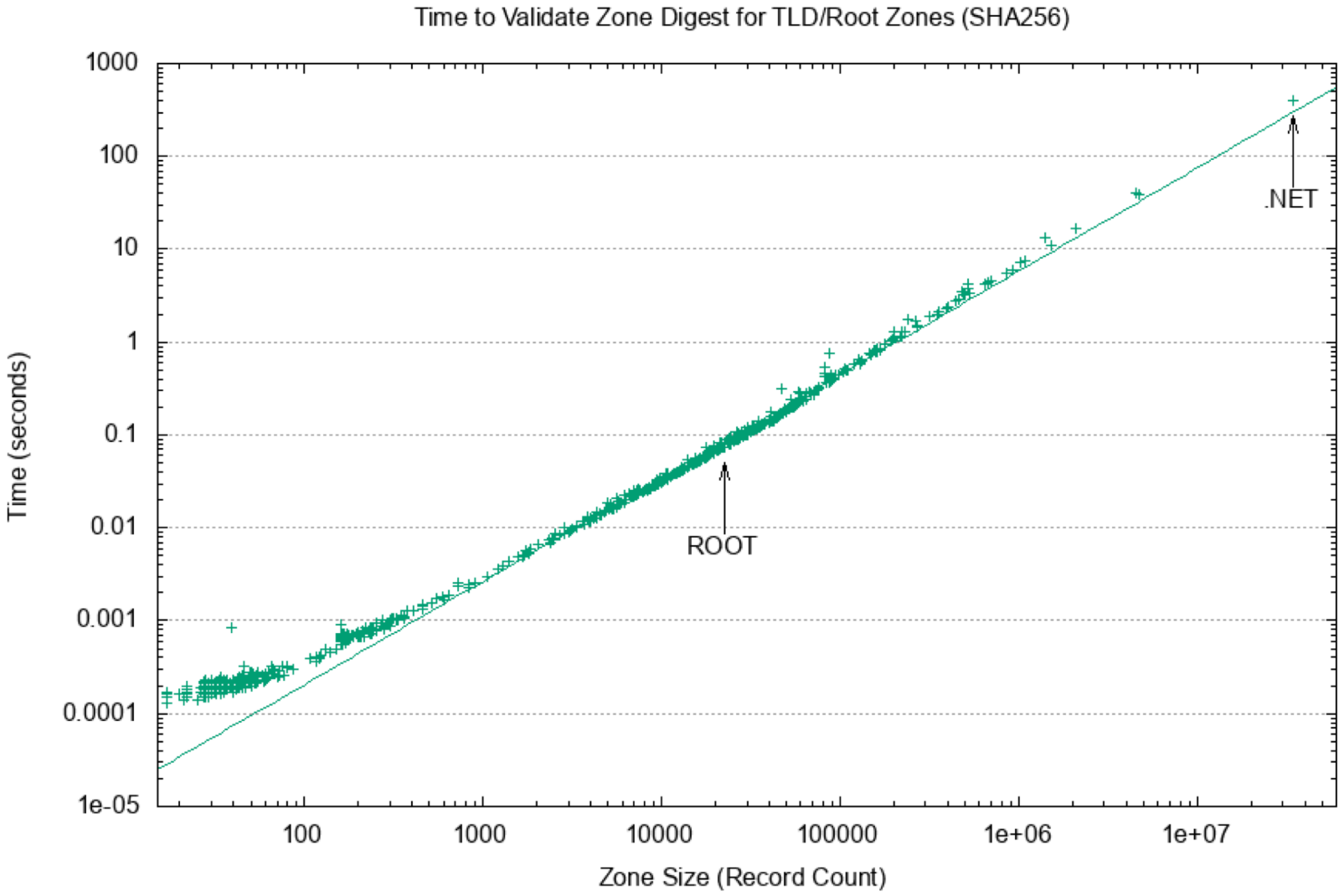
An Implementation

- Uses ldns (from NLNet Labs) for underlying RR manipulation
- Features implemented
  - Read zone file
  - Add ZONEMD placeholder
  - Compute digest and update ZONEMD
  - Re-compute ZONEMD signature
  - Verify digest from input file
- Thanks to Shane Kerr for another implementation
  - Led to discovery of a byte-order bug in my code

Basic Benchmarks



Basic Benchmarks



What About Dynamic Updates for Large Zones?

Large Zones & Dynamic Updates

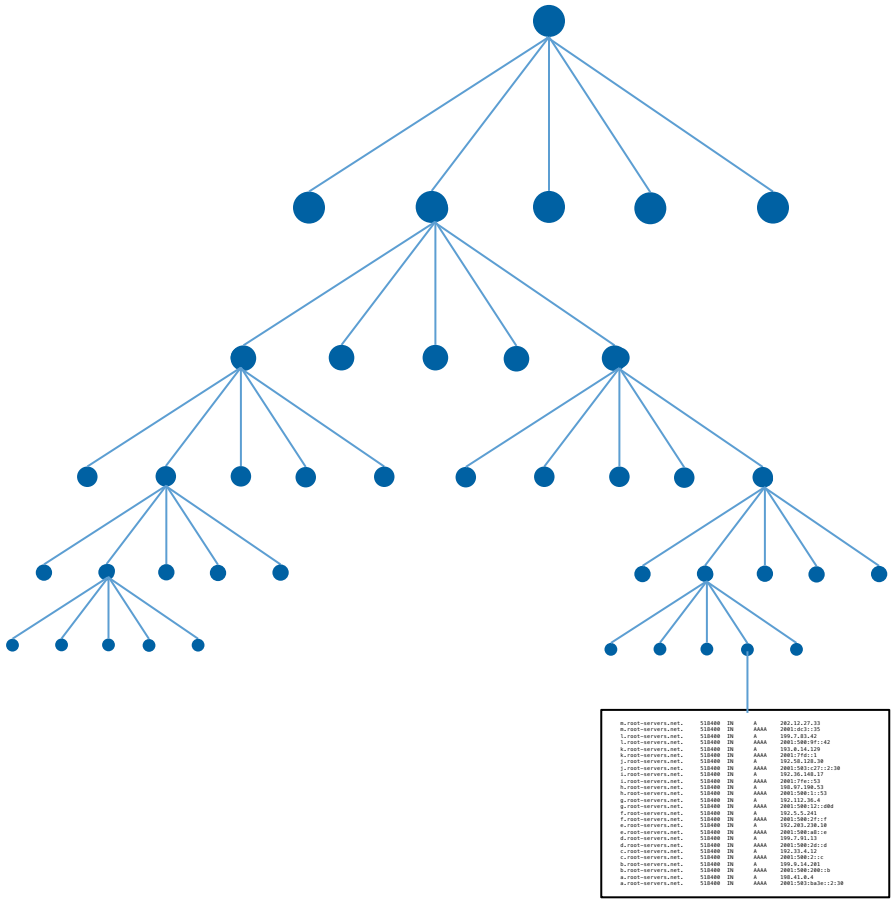
- Some of the dnsop list discussion has been about large zones and/or dynamic updates.
- Neither are a problem for zone digests, per se.
- It is possible to have a large zone but not need frequent digest calculation.
- It is possible to have dynamic zones where digest calculation is efficient.
- Taken together, large, dynamic zones could pose a challenge.

Efficiently Digesting Dynamic Updates

- Partition zone namespace and use non-binary Merkle Tree* hashing
- The ~~Reserved~~ Depth field defines depth of the hash tree
  - Depth = 0 means no tree and hash over whole zone
- Digests at leaf nodes over partitioned and sorted RRsets
- Digests at non-leaf nodes over child node digest values

*But NOT a blockchain

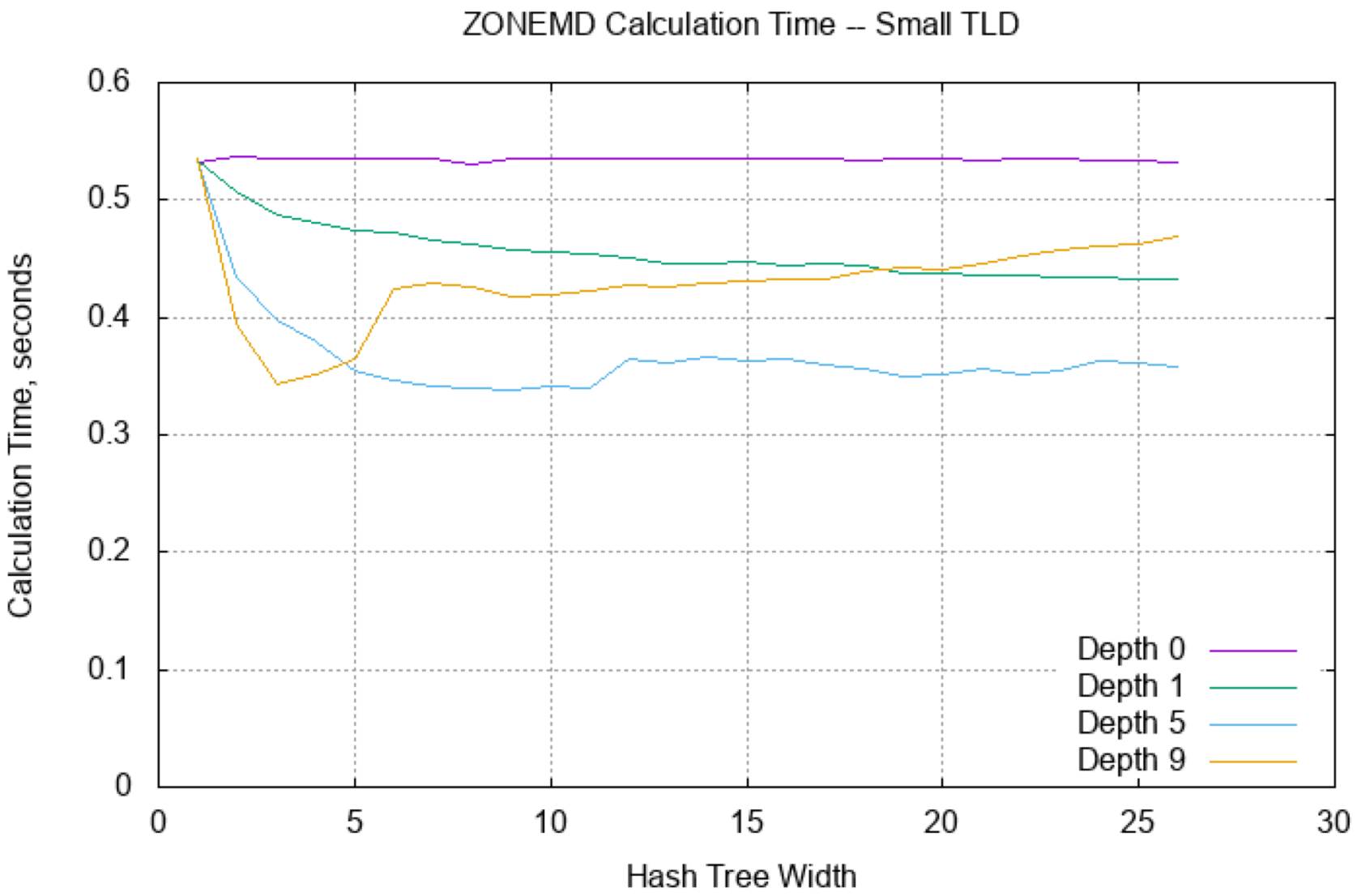
Hash Tree



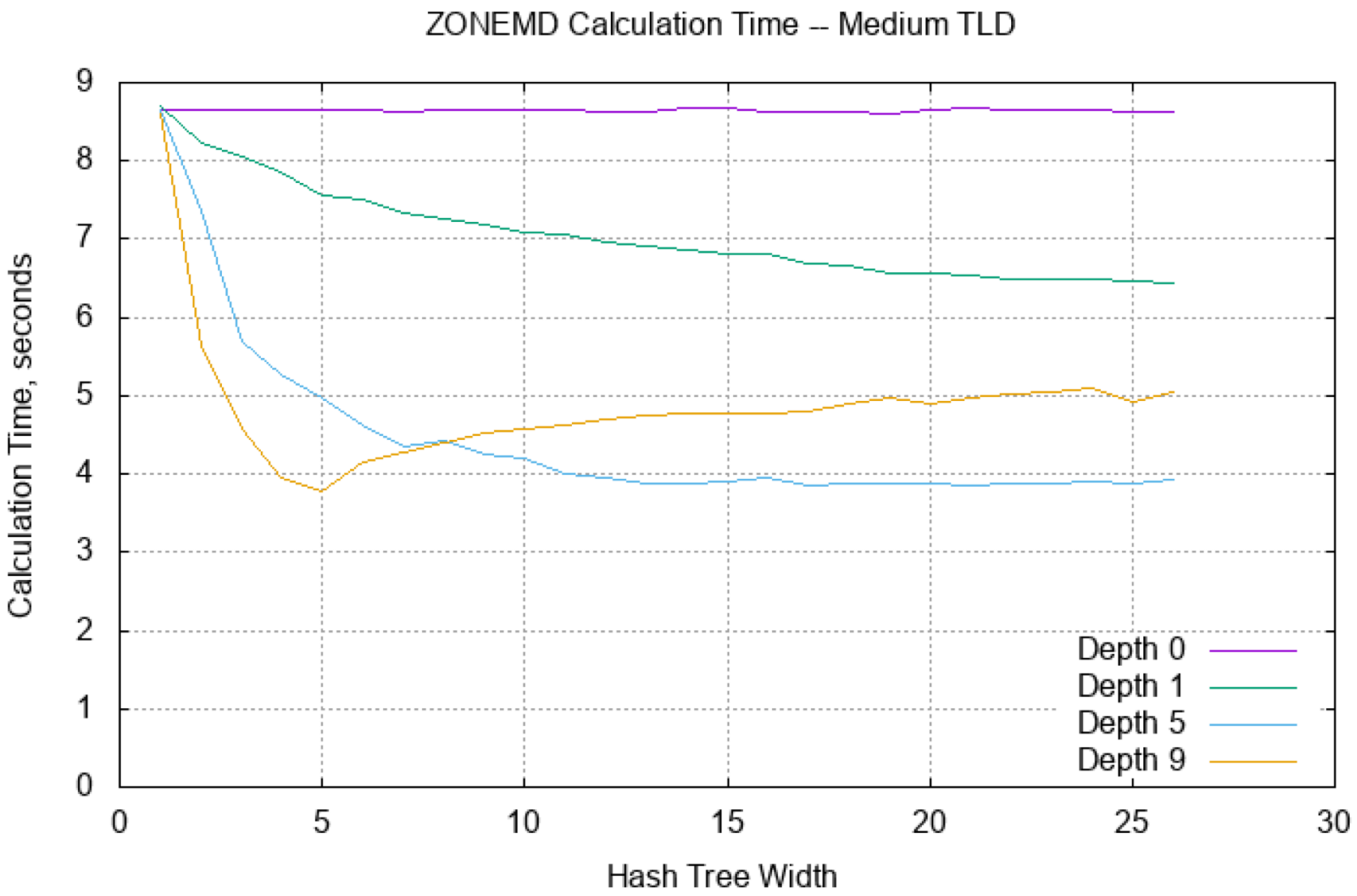
- Of some depth *D* and width *W*
- Total possible # partitions = W^D
- Deterministic partition function, e.g. based on depth and owner name

```
get_branch(name, depth) {
  len = strlen(name);
  if (len == 0)
     return 0;
  pos = depth % len;
  branch = *(name+pos) % max_width;
  return branch;
}
```

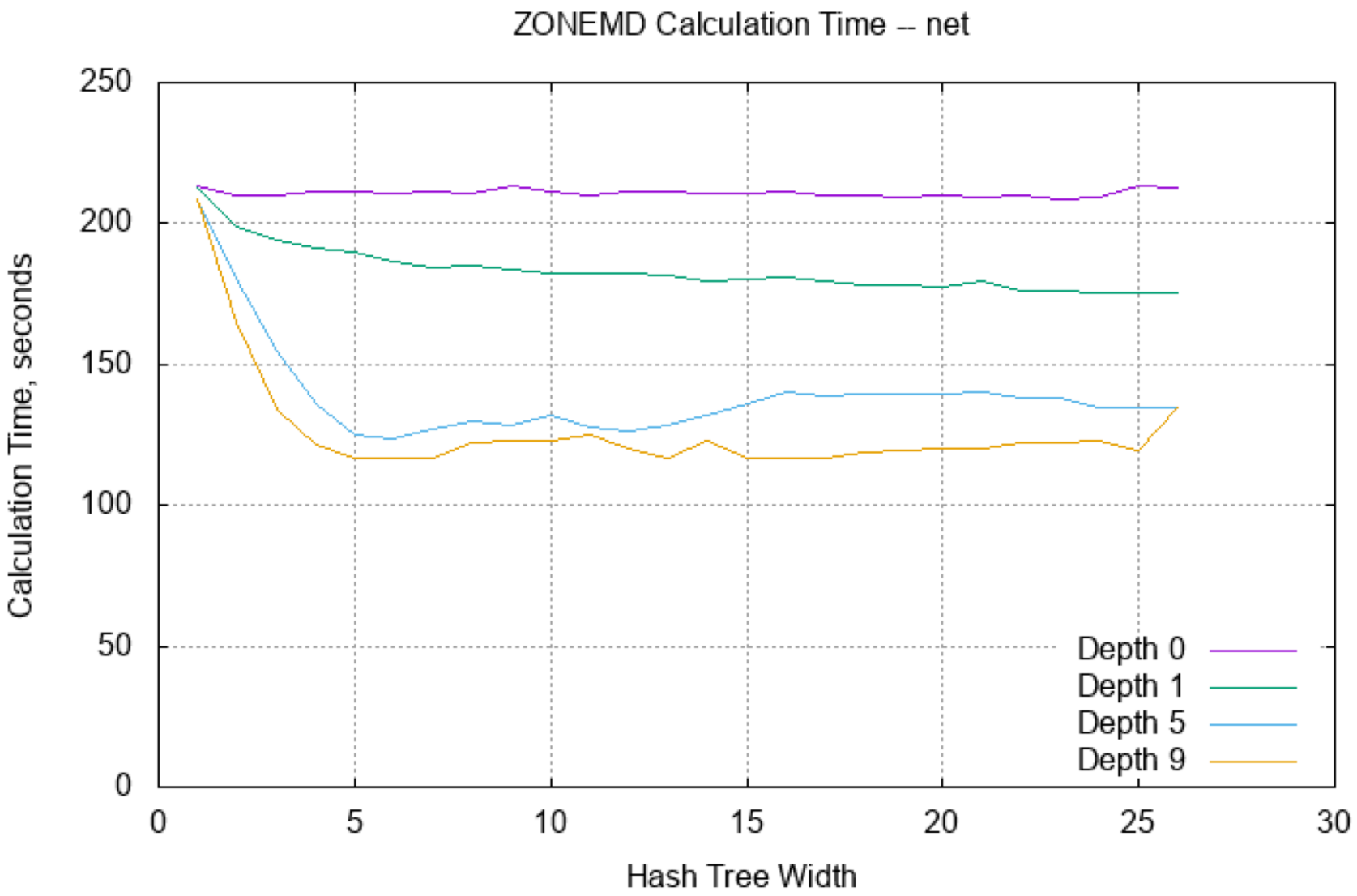
Tree Depth/Width vs Digest Calculation Time?



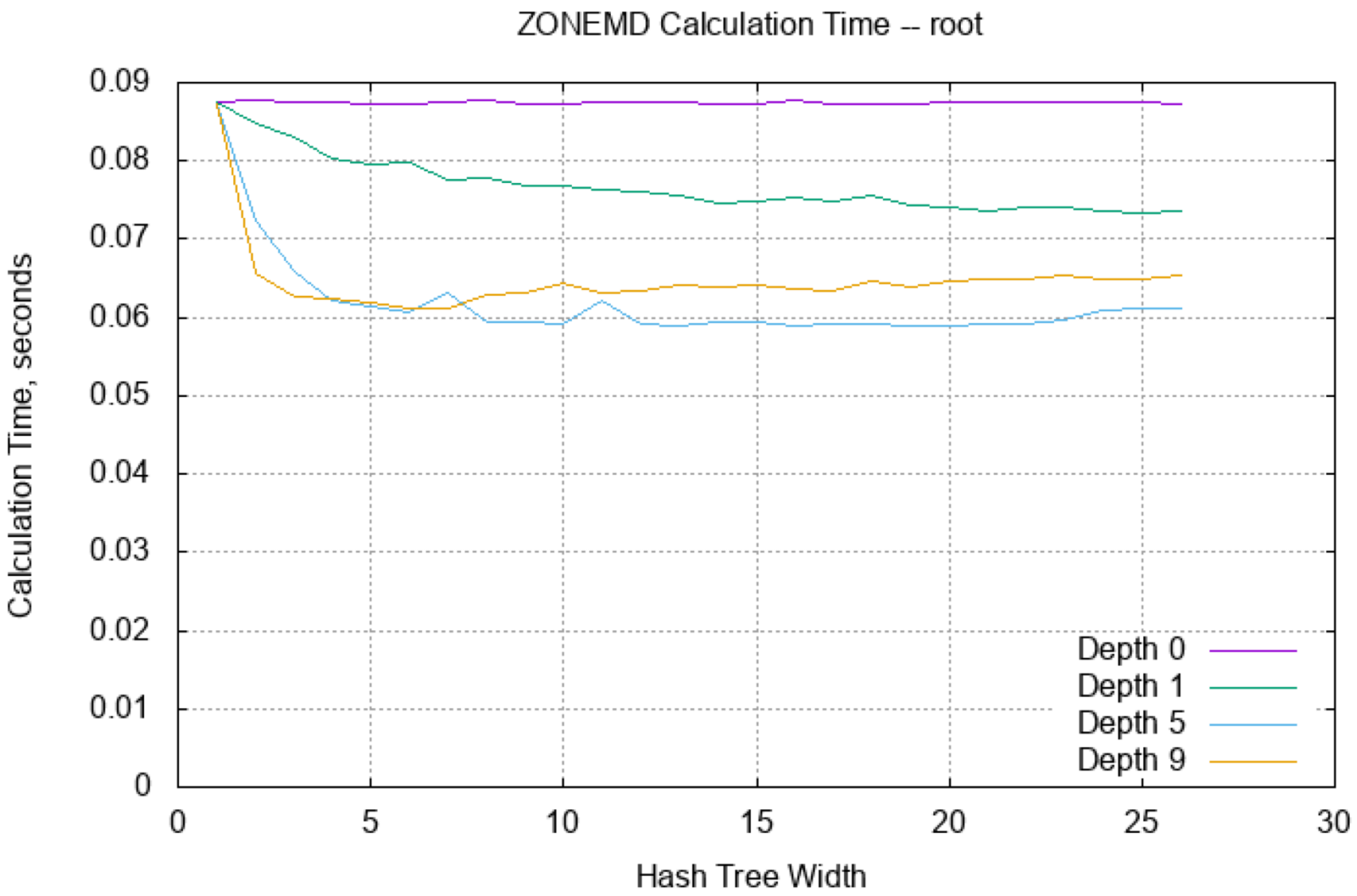
Tree Depth/Width vs Digest Calculation Time?



Tree Depth/Width vs Digest Calculation Time?



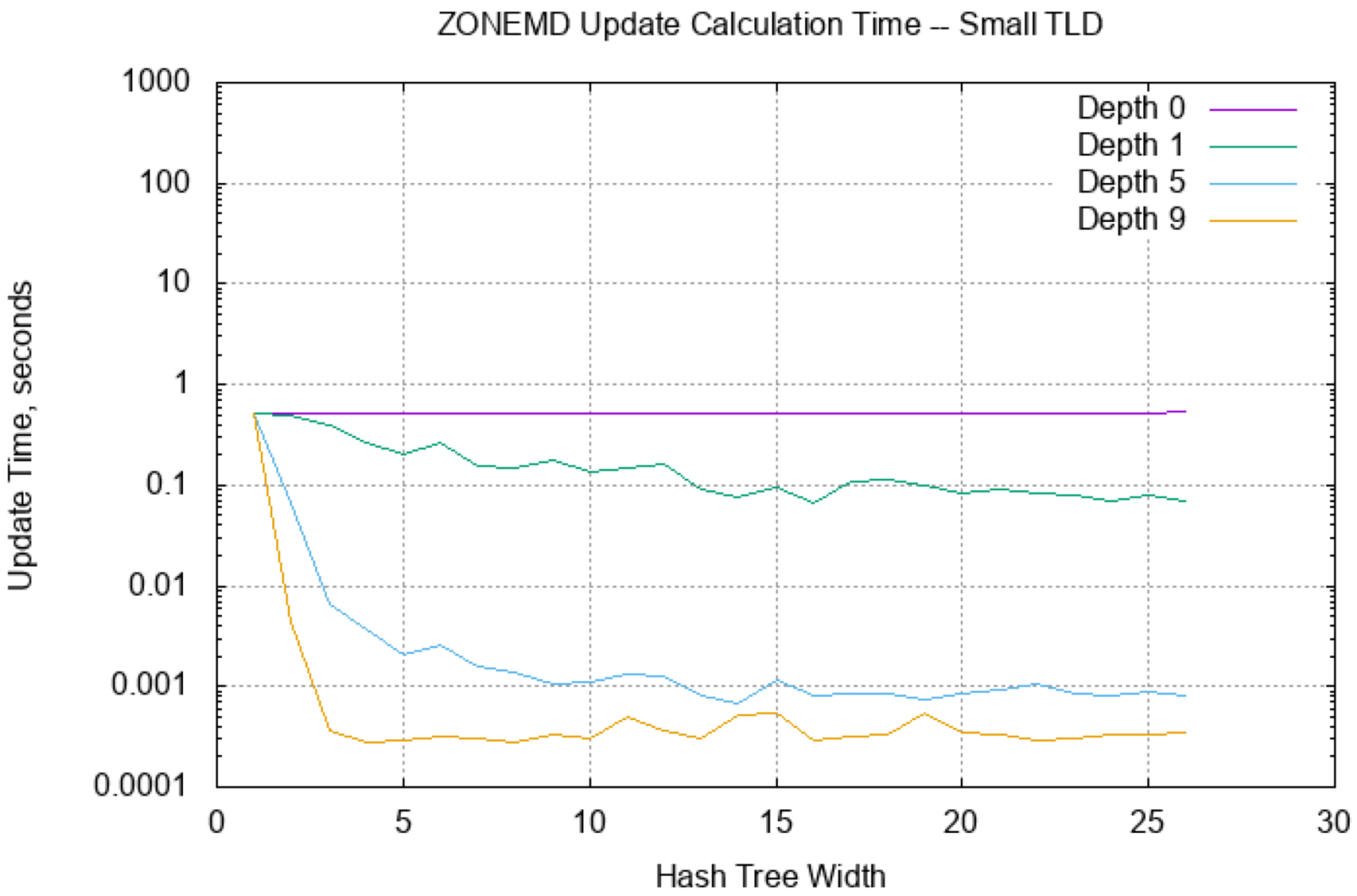
Tree Depth/Width vs Digest Calculation Time?



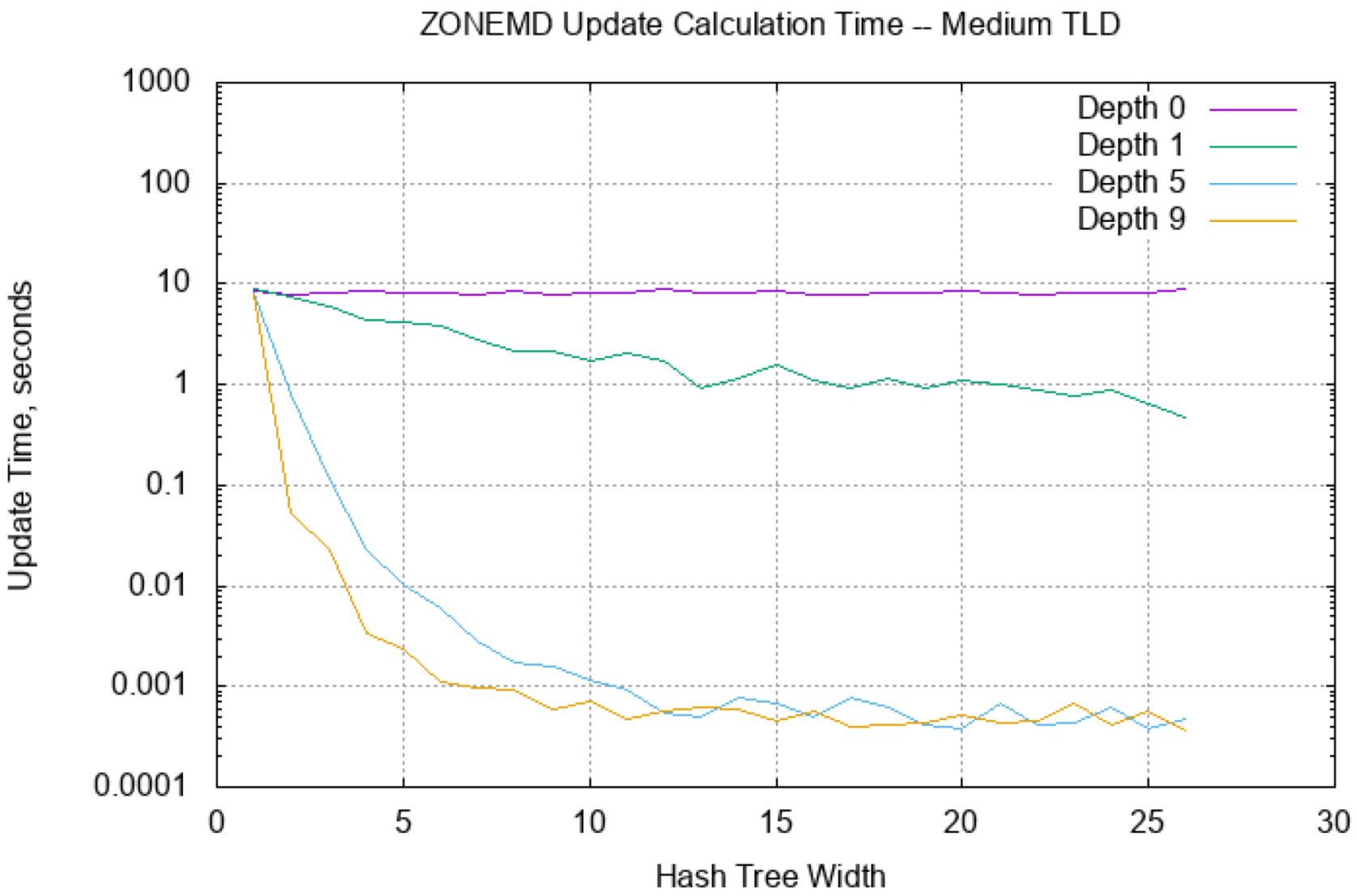
Hash Tree Enables Efficient Updates

- Only need to sort RRs in leaf nodes with updates
- Re-hash leaf node and its intermediate nodes leading up to the top
- Significantly faster
- Increased complexity
- Imposes a new / different zone data structure

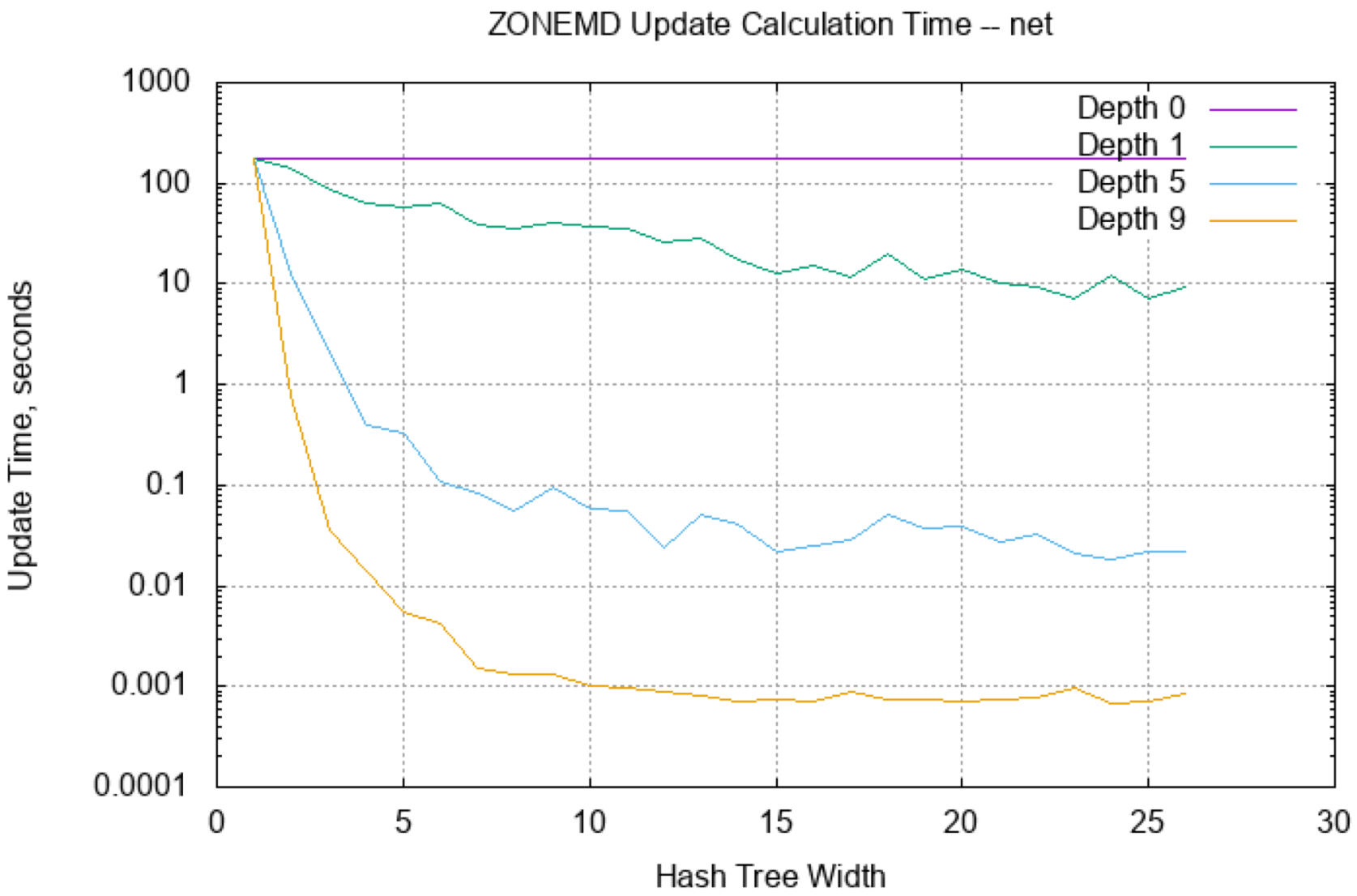
Digest Update Time vs Depth



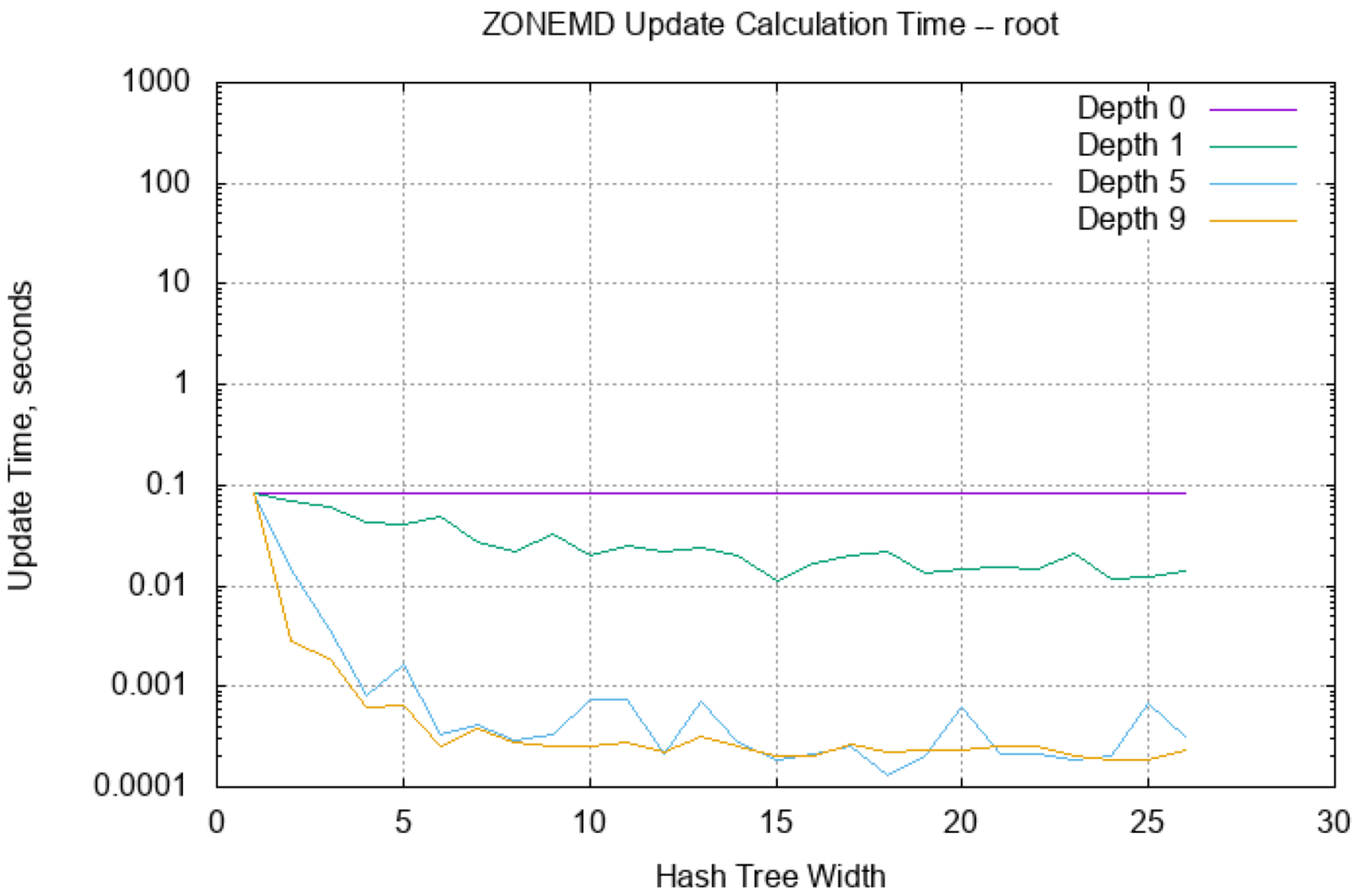
Digest Update Time vs Depth



Digest Update Time vs Depth



Digest Update Time vs Depth



Conclusions, Questions, Links

- Efficient support for large dynamic zones is possible with hash trees
  - Even appears to improve performance for large static zones
  - But is the complexity worth it?
- Digest useful without DNSSEC?
  - Should DNSSEC be required?
- Draft: draft-wessels-dns-zone-digest-03
  - Proposed Standard → Experimental
  - Depth > 0 not described
- Implementation:
  - https://github.com/verisign/ldns-zone-digest

VERISIGN®