Development of the Domain Name System

Paul V. Mockapetris, Kevin J. Dunlap

Sigcomm Symposium, August, 1988

Presented by Steven Smith – CSC 7970

Outline

- Introduction – HOSTS.TXT and Reason to Move To DNS
- DNS Design
- Implementation Status at Writing
- Surprises, Successes, and Shortcomings
- Conclusions and Future Work
- Critique and Issues
- Take Aways

Introduction - Background

- DNS – Domain Name System
  - First designed in 1983.
  - Combination of Hierarchies, Caching, and Datagram Access.
  - Replacement for HOSTS.TXT mechanism
- HOSTS.TXT
  - Mapped hosts to names and addresses
  - Transferred to all hosts in the Internet via file transfers.

Introduction - Reasoning

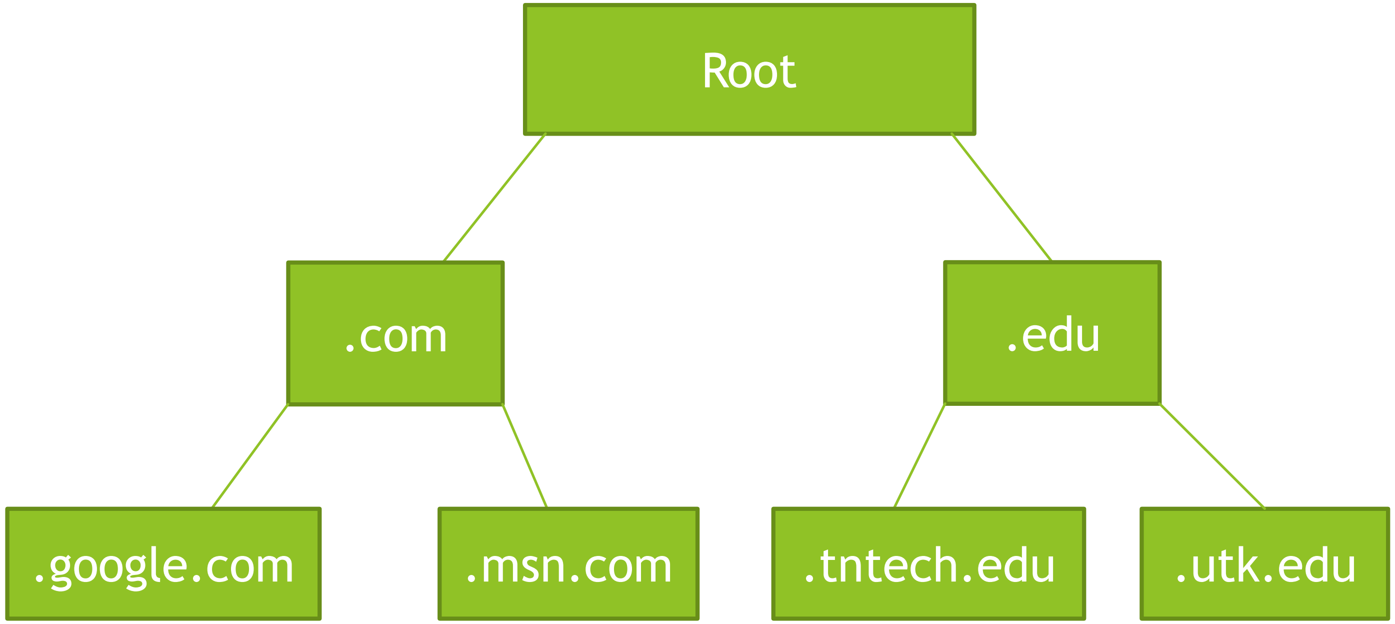
- HOSTS.TXT Issues
  - Problem – becoming too large leading to high distribution costs, and centralized management did not fit with the Internet's distributed nature.
  - Due to growth and evolution of community from ARPANET to the Internet.
  - Rapid changes and large size led to high costs.
- Wanted to allow for local control of network names and addresses.

Introduction – Existing Distributed Naming Systems

- DARPA IEN116
  - Too limited and host specific without enough benefits.
- XEROX Grapevine
  - Sophisticated
  - Used heavy replication with light caching.
  - Fixed number of hierarchy levels.
  - Would require support of its protocol.
  - Did not fit the Internet's distributed nature.

DNS Design

- Hierarchical name space with typed data at nodes.
- Database control delegated in hierarchical fashion.
- Intent for data types – Should be indefinitely extensible as new apps are added.



DNS Design – Base Assumptions

- Provide at least the same info as HOSTS.TXT.
- Allow the database to be maintained in a distributed manner.
- Have no obvious size limits for names, components, data, etc.
- Interoperate across the Internet and as many other environments as possible.
- Provide tolerable performance.
- Initial design assumed the necessity of striking a balance between a lean service and completely general distributed DB.
  - Lean service – Desirable for more implementation efforts and early availability.
  - General design – Reduce the cost of introduction, provide greater functionality, and increase the places DNS would be used.

DNS Design - Constraints

- Cost could only be justified if it provided extensible services.
  - Should be independent of network topology.
  - Capable of encapsulating other name spaces.
- To be universally acceptable, should avoid trying to force a single OS, architecture, or organizational style on users.
  - Avoid any constraints due to outside influences.
  - Permit as many different implementation structures as possible.

DNS Architecture- Active Components

- Name Servers – Repositories that answer queries with their possessed information.
- Resolvers – Interfaces to client applications, embody algorithms for finding name servers with the needed info.
- May be combined into one application or separated to suit needs.
- Often useful to centralize resolver function in one or more name servers.
  - Allows sharing the use of cached info.
  - Allow less capable hosts to rely on the resolving services of these servers without needing their own resolver.

DNS Design – Name Space

- Variable Depth Tree
  - Each node has an associated label – up to 63 octets in size. Case Insensitive.
  - Domain name of a node is the concatenation of all labels on the path from the node to the root. Up to 256 total octets to aid implementation.
  - Example: .scholar.google.com - Null(root)->.com->google.com->scholar.google.com
- Config files represent names as character strings separated by dots, but applications are free to do otherwise.
  - Example: Venera.Isi.Edu is a name with four labels – the null root is usually omitted.
  - Mailbox names typically encode the part before the @ in one label.
- Recommended to mirror the structure of the organization controlling the local domain.

DNS Design – Attached Data

- Does not constrain data that can attach to a name.
  - However, does specify some structures so replies to queries can be limited to relevant info.
- Data for each name is organized as a set of Resource Records (RRs).
  - Each contain a well-known type and class field followed by app data.
  - Multiple values of the same type are represented by separate RRs.
  - Provided space efficiency to reduce max RR size.
- Types – Represent abstract resources or functions.
  - Example: Host addresses.
- Class – Divides Database from type, and specifies protocol family.
  - Example: DARPA Internet

DNS Design – Database Distribution

- Provides 2 major mechanisms for transferring data from a source to destination.
  - Zones – Sections of the global database controlled by a specific organization
  - The organization is responsible for distributing copies to servers that make the zones available to the wider Internet.
  - Caching – Data acquired in response to a request can be locally stored for use with future requests.
  - Intent – Both mechanisms should be invisible to the user who will see a single DB without boundaries.

DNS Design - Zones

- Zones are a complete description of a contiguous section of the total tree name space, with some pointer info the other continguous zones.
  - Can be a single node or whole tree, but typically a subtree.
  - Data is typically maintained in a master file on the name server.
  - Authoritative data is stored by the server itself rather than retrieved or cached.
- Organizations gain control of a zone of the name space by requesting a parent organization delegate a subzone to them.
  - The subzone consists of a single node. The organization can then build this out without the original parent
  - Organization must maintain zone data and provide redundant servers for the zone.
  - Zone transfers require TCP for reliability.
- Goal – A parent organization should be able to have a domain name even if it lacks the communication or host resources for supporting the DNS service.

DNS Design - Caching

- DNS resolvers cache responses for use by later queries.
  - Controlled by TTL field in units of seconds. Set in each RR.
  - Low TTL reduces periods of inconsistency while high TTL minimizes traffic load and allows for periods of unavailability.
  - Example: Recommended TTL value for host names is 2 days.

Implementation Status at Time of Writing

- DNS in use throughout DARPA Internet.
- HOSTS.TXT still in use by older hosts, but DNS is recommended.
- At the time, Domain space partitioned into around 30 Top Level Domains (TLDs).
  - Example: .com, .edu.
  - Currently, there are over 1,000 TLDs.
  - SRI-NIC manages domains for all non-country TLDs and delegates subdomains to organizations that wish to maintain their own name space.

Implementation Status at Time of Writing – Root Servers

- Search algorithm for DNS allows a resolver to search downward from domains it can access.
- Root server and TLDs supported by 7 redundant name servers.
  - Typical traffic 1 query per second.
- Vast majority of queries are 4 types
  - All info, host name to address, address to host, and mail info (MX).
  - 10-15% of all queries referred to servers for lower level domains.
- Berkeley University provided Unix Support for DNS with BIND.
  - First organization to depend solely on DNS for host/address resolution.

Surprises

- Refinement of Semantics
  - Initial assumption was the form and content of information was well understood.
  - Did not pan out as even common concepts such as IP host addresses caused issues.
  - Supporting multiple addresses for single hosts caused huge discussions about if addresses should be ordered and how.
- Performance
  - Worse than originally planned.
  - Heavy load, network growth, slow speed links led to multiple queries from the same sources, causing further delays.
  - Difficulty in measuring performance due to being swamped by unrelated effects such as gateway changes and DNS software releases.
  - Caching had better performance than expected, exceeding HOSTS.txt due to anticipating large database sizes.

Surprises

- Negative Caching
  - DNS provides 2 negative responses to queries – Name does not exists, and name exists but requested data does not.
    - Examples: Misspelled name, Host type of mailbox that is not set.
  - Happened much more often than expected.
  - Up to 60% error rate on root servers.
  - Caused by program using old-style host names or names from other mail services such as UUCP.
  - Recommendations to reduce these helped, but negative responses were still 25-50%.
  - Should have cached negative responses as well to improve performance.

Successes – Variable Depth Hierarchy

- Variability in tree depth.
- Growth of Workstation and Local Networks meant organizations were finding a need to organize within their own networks.
- Vastly different organization sizes led to need for different depths needed in the hierarchy.
- Made it possible to encapsulate any fixed level or variable level system.
- Example: UK's name service NRS and DNS were able to mutually encapsulate each other's name space.

Successes – Organizational Structuring of Names

- Naming structure’s independence from network, topology, etc.
  - Very popular and prolific.
- However, TLD structure is controversial.
  - Authors stat they could change this however as DNS is flexible enough to accommodate almost any tree-based structural choice.
  - Requires a consensus to be reached.

Successes – Datagram Access

- UDP Datagram used as main method of communication (outside of TCP-based zone transfers).
  - Maximum size of 512-bytes did not lead to issues and helped reduce resource usage.
  - Seemed to be essential due to poor performance of the Internet.
- Drawback
  - Need development and refinement of retransmission strategies that are already well developed for TCP.
  - Lead to much unnecessary traffic caused by repeated queries.

Successes – Additional Section Processing and Caching

- Additional Section Processing - Allows name servers to provide additional info that fits in the datagram beyond the answer to a query.
  - Allows the server to anticipate additional requests.
  - Example: When a root server passes the name of a host, they will include its address as it is assumed it is needed for use.
  - Experiments show this was estimated to cut query traffic in half.
- Caching – essential to the poor performance of the Internet.
  - Problem – DNS Admin strategies can make it less reliable or useful.
  - Example: Admins assigning short TTLs to RR nodes that rarely change.

Successes – Mail Address Cooperation

- Agreement between representatives of different communities to use organizationally structured domain names for mail addressing and routing.
  - CSNET, BITNET, UUCP, and DARPA Internet among these.
- Provided a good opportunity to clean up mail addresses.

Shortcomings – Type and Class Growth

- Initially there was great demand to increase the size of type and class specifiers to allow for many additions.
  - Only 2 type and classes added over 5 years, and 2 types were dropped.
  - Either demand was misunderstood or new types and classes were too difficult to create.
- Problem – Almost all existing software regarded DNS types and classes as compile-time restraints, which meant they must be recompiled to deal with changes.
- Methodology and guidelines for adding these are needed, but the problem is this involves the design of special name space sections, TTL selections, etc.
- Different members of the Internet have different views – No Consensus.

Shortcomings – Easy Upgrading of Applications

- Conversion of network applications to use DNS was difficult.
  - Problem – Needed to handle Transient failure, where a distributed naming system would have periods where it cannot access certain info.
  - Access to naming system also needed to be better integrated into Operating Systems, so application designers did not have to add functionality themselves.
  - Example: Adding access as the shell level.

Shortcomings – Distribution of Control vs. Distribution of Expertise or Responsibility

- Distribution of authority does not distribute expertise.
  - This means maintainers will fix things till they work, not till they work well.
  - Leads to problems with consistency in name server design.
  - System designers should try to compensate for this.
  - Initial policy was to delegate a domain to any organization which filled out a form listing its redundant servers and other requirements.
  - Should have made them demonstrate their redundant servers had real data before delegating the domain and assure they were on different networks to prevent a Single Point of Failure.
  - Examples in documentation were designed for narration, not for actual use. Sample TTL values of 1 hour instead of the recommended days.
  - Debugging of systems made difficult due to lack of the protocol providing a method to retrieve version.

Conclusion

- Need for distributed functionality and new opportunities for future services justify the creation of DNS.
- Modifications to HOSTS.TXT could have postponed a new system, but would have still led to large issues.
- Considerations that would have improved DNS:
  - The usefulness of negative response caching.
  - More difficult to remove functions than add new ones due to the need for the community to all convert to a new version of the service.
  - Implementers will often lose interest once a level of performance they expect is achieved.
  - Distributed software should including a version number and parameters that are easily viewed.
  - Variations in the implementation structure is a good idea, but service variations cause issues.

Directions for Future Work

- DNS in production so changes are difficult.
- Research in other naming systems could provide useful additions.
- Data description techniques from ISO could provide a better mechanism for adding data types, while DNS infrastructure could speed ISO prototyping.
- Develoipng an approach to structure the total tasks into layers depending on the situation could be valuable.
  - Example: A system for managing file names on a local disk vs. host names.
- Technical and/or political solutions to the growing complexity of naming.

Critique

- Good background information provided
- Fairly thorough explanation of reasoning
- Not much information on how improvements would or could be actually done.
- Production system – Difficult to change as mentioned.
- Not much explanation of other Distributed Naming Systems.
- Security and Privacy not considered or addressed.

Potential Improvements

- Designing for Availability
  - Initial designs were good for the beginning.
  - Root servers could and did lead to some traffic issues.
  - Much criticism at the time about root servers as well.
- Improving Performance
  - Potential for better caching via distribution of most popular queries at regular intervals to subdomains.
  - Example: Google.com is one of the most queried sites, so distribute its information regularly to subdomain resolvers.

Potential Improvements

- Authentication and Data Integrity – DNSSEC
  - If a host points to a malicious name server, there is nothing in the initial design to prevent them from getting bad data.
  - Use of Digital Signatures (hashed result information encrypted with private keys) with Public Key Infrastructure to verify the authenticity of result data.
  - DNSSEC can result in leakage of zone information due to keys being assigned by zone and NSEC responses.
    - Example: Querying for b.google.com would return an NSEC response stating no domain exists between a.google.com and z.google.com if they existed.
  - RFC 4470 proposes listing no domain exists between two lexically close domains that may not actually exist.
    - Example: Querying for b.google.com would return an NSEC response stating no domain exists between a.google.com and b.google.com

Potential Improvements

- Confidentiality
  - No confidentiality options specified.
  - Allows for network analysis attacks at DNS level.
  - Could provide option for encryption of Queries and Responses

Potential Improvements

- Better Education and Discussion with Varied Members of the Community
  - Many assumptions made about knowledge
  - Not many specifications on best practices provided
  - Example: Could have improved adoption of changes by specifying types and classes not be hardcoded at compile time.
  - Did not seem to consider many ways the Internet was already being used.
  - Many times it is mentioned there was no consensus found – could have spent more time trying to get a majority agreement on some features.
  - Example: TLD and root servers.

Potential Improvements

- Better experimentation and testing
  - Not much mention of how testing and initial trials were performed.
  - Could potentially have tested longer and more thoroughly to address problems before DNS was put into production.
  - Distribute out to candidates first, receive feedback?

Takeaways

- DNS was a huge effort to replace an outdated system – HOSTS.TXT.
- Openness and variability worked great for the Distributed nature of the Internet.
- Lack of direction, guidelines, and discussion with the community and admins led to large issues such as problems with TTL times leading to poor caching practices.
- Hard to account for all of the potential use cases.
- Availability was a much bigger concern than authentication or confidentiality.
- Overall, worked well. Most issues were caused by poor practices of end users/administrators and unforeseen issues such as the amount of negative responses that could have been cached to improve performance.

References

- Mockapetris, P., & Dunlap, K. J. (1988, August). Development of the domain name system. In *Symposium proceedings on Communications architectures and protocols* (pp. 123-133).
- R. Arends, R. Austein, M. Larson, D. Massey, & S. Rose. DNS Security Introduction and Requirements. RFC 4033. https://tools.ietf.org/html/rfc4033
- S. Weiler (2006, April). Minimally Covering NSEC Records and DNSSEC On-line Signing. RFC 4470. https://tools.ietf.org/html/rfc4470

THANK YOU!