

6.033 - Naming

Lecture 2

Katrina LaCurts, lacurts@mit.edu

1. Previous Lecture/Intro

- Complexity limits what we can build
- Enforced modularity mitigates complexity
- Modularity requires names (how else will modules communicate?)

2. Naming in General

- Examples of names: mit.edu, schedule.shtml, 617-253-7341, etc.
- Systems manipulate/pass objects either by value or by name
- Benefits of using names
 - Retrieval
 - Sharing
 - User-friendliness
 - Addressing
 - Hiding (+access control)
 - Indirection
- We can get certain functionality in our systems by the way we pick our names
- Design of the naming scheme should reflect the properties we want in the system as a whole

3. Abstract view of naming schemes

- Three components: namespace, set of values, look-up algorithm to translate
- Lots of questions to ask:
 - What is the syntax of the names?
 - Is there any internal structure to the names?
 - Is the name space global (context-free) or local?
 - What are the values?
 - Does every name have a value (or can you have "dangling" names)?
 - What part of the system has the authority to bind a name to a value?
 - Can a single name have multiple values?
 - Does every value have a name (i.e., can you name everything)?
 - Can a single value have multiple names (i.e., are there synonyms)?
 - Can the value corresponding to a name change over time?
 - What context is used to resolve the names? What part of the system specifies it?
 - Where does resolution happen ("who does it")?
- Designing a naming system means balancing engineering tradeoffs
 - Example tradeoffs: distribution, scalability, delegation

5. Naming on the Internet: DNS

- Maps hostnames to IP addresses. Necessarily because routers can only operate on IP addresses, not hostnames
- Provides:

- user-friendliness
- load balancing (single name -> multiple values)
- single value -> multiple names
- mappings can change over time
- Look-up algorithm
 - Bad design 1: hosts.txt. One file on every machine. Hard to keep files updated.
 - Bad design 2: one powerful machine with one big table. That machine is hit with a ton of requests.
 - DNS' algorithm: divide names into hierarchy, each zone contains its own mappings. send request to root, which delegates down the tree.
 - Zones have authority over their own names
- Enhancements
 - 1 zone can = more than 1 physical name server (MIT has 3, root server has hundreds distributed globally)
 - Initial DNS request can go to any server, not just root
 - Many name servers support recursive queries
 - Name servers and clients can cache
 - Those three combined result in the performance improvement
- Design questions (will be discussion in recitation)
 - DNS has a hierarchical design. Why? What's good about that?
 - Is it simple?
 - Does it scale? In what senses?
 - Is it fault tolerant?
 - Are there any lingering policy issues that are unsatisfying?
 - Are there aspects of the design that *don't* perform well?
 - Is it secure?
 - Was it a successful design?