MIT 18.996: Topics in TCS: Internet Research Problems **Spring 2002**

Lecture 4 — February 27

Lecturers: T. Leighton, D. Shaw, R. Sudaran *Scribes: K. Battocchi and M. Monteleone*

4.1 Introduction

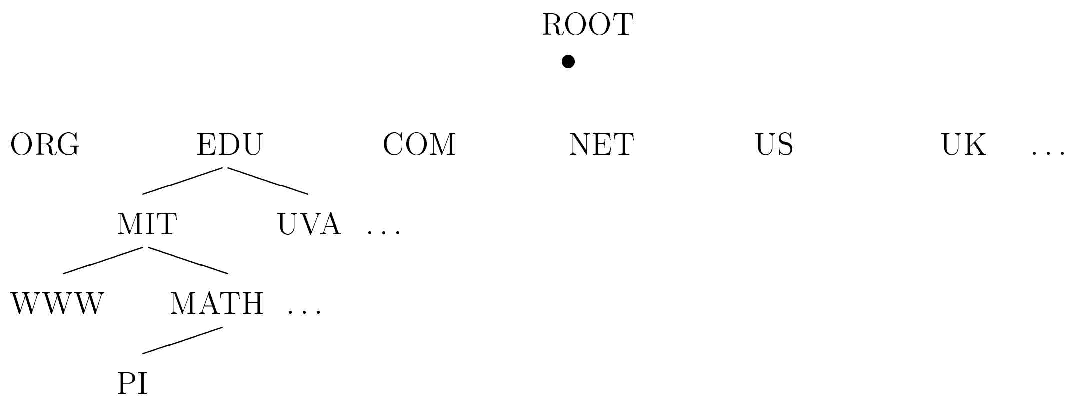
One could say, simply, that DNS is used to find something on the Internet. The traditional (old) design of DNS functions essentially like a name and number directory in which a user could type in a name and have a number returned (or vice versa) from a text file. As you may guess, this system was not very efficient and so it did not work very well in practice. Rather than serving as a true naming system, it was essentially more of a card catalog.

The current design of DNS does not function quite like a card catalog; rather, it can be thought of as a large, distributed directory. There are four important terms that describe the basic layout of DNS:

1. Name – includes ".edu", ".com", ".org", etc.
2. Type – refers to the data type being stored; data can be of any type, not just addresses.
3. Class – with the exception of a few internal classes, class merely refers to the Internet.
4. Data – there are several types of data. Examples include: A (IP Address), CNAME (Alias), H-Info (Host Info) (it is interesting to note that this is not as widespread as in the past for security reasons, since users would be able to identify information about a particular computer and its location).

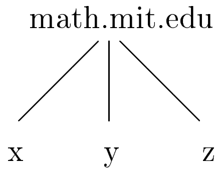
4.2 Layout of DNS

There are 13 root servers, each one with a zone that defines top-level names. Consider the following diagram, which illustrates the hierarchical structure of DNS:



One should note that heavy caching is done at all steps of the tree. In addition, the information contained at each node has a time-to-live (TTL) associated with it. In many cases, this is usually about one day, but it could be shorter or longer. For obvious efficiency reasons, it is useful for nameservers to build up a cache. Likewise, Internet Service Providers (ISPs) seek to build up caches in order to save on bandwith and, ultimately, cost.

The owner of a particular zone keeps a record of all of the new subzones. For example, consider the following diagram:



Math.mit.edu was created on MIT's network; furthermore, there are three nameservers – x, y, and z – that belong to the math department. The concept of a glue record describes this type of delegation. If a zone is delegated to a nameserver whose hostname is a descendant of that particular zone, then a glue record for that hostname must be included in the delegation. For instance, the MIT Administrator keeps the information about math's nameservers (i.e. the information is kept at mit.edu as opposed to math.mit.edu).

Let us also define a few additional relevant terms. A PTR record, or pointer record, is also known as a reverse record. A PTR record associates an IP address with a canonical name (the real name of a host). PTR records should point to a name that can be resolved back to the IP address. The name of the pointer record is not the IP address itself, but is the IP address' four IP octets in reverse order followed by IN-ADDR.ARPA. For example, 192.168.0.1 becomes 1.0.168.192.IN-ADDR.ARPA.

Before moving on with an in depth discussioin of how DNS is used in practice, we should briefly discuss lame delegation. Lame delegation occurs when a nameserver record points to an incorrect host. This can occur if a zone has been delegated to a server that has not been correctly configured to be authoritative for the zone. Likewise, lame delegation can also occur when a server that is authoritative for the zone has a nameserver record that points to another server that is not authoritative for the zone. This causes servers to either not respond, or not respond authoritatively, resulting in network traffic or unnecessary server workload.

4.3 Using DNS in practice

We will explain how DNS is used in practice for downloading a website. We will also detail the process of a DNS lookup as it occurs in this context. It is helpful to consider two ways of using DNS to download a website: the traditional way and the Akamai way. We will describe each method in detail below.

4.3.1 The Traditional Way

Let us assume that the user wants to visit web.mit.edu. After entering [1]http://web.mit.edu, the browser seeks to resolve the IP address for the site. DNS plays a role here, since DNS maps the address web.mit.edu to the IP address 18.7.21.77. DNS returns the IP address and the browser now contacts the server at that address. The server then returns HTML (including embedded links). For each embedded object, the process is repeated. We will now describe the DNS lookup process in greater detail. We have mentioned before that caching plays an important role in DNS lookups. Thus, when web.mit.edu is entered into the browser, the browser consults its own cache first to see if the particular name has been previously resolved. If it is not found in the browser's cache, the operating system's (OS) cache is consulted. If it is not found here, the browser connects to its local name server. Again, the local nameserver will return the address if it has previously resolved it; otherwise, it will query higher authorities. At worst, the queries will eventually reach a root DNS server. After the address is resolved, the local DNS server contacts the mit.edu DNS server in order to obtain the resolution for web.mit.edu. An IP address is received (and cached) and is passed back to the users browser. Then, the request for HTML begins, as described in the previous paragraph.

4.3.2 The Akamai Way

A website that employs Akamais technology is downloaded in a more optimal way. The basic principle of resolving an IP address using DNS still applies, but the address that is returned from the lookup is one of an optimal Akamai server. The browser then contacts the Akamai server to request HTML. The Akamai server is responsible for assembling the website and delivering the content to the users browser. Optimal Akamai servers would also be responsible for delivering any embedded objects to the browser as well. We will now describe the DNS lookup process as it applies to the Akamai way. Since we are modifying the DNS lookup procedure to return the address of an optimal Akamai server, we must develop a way to ensure that this optimal server is found (as opposed to the traditional way where the IP address would be returned for the requested website). An alias is thus used to ensure that the address is resolved to the appropriate optimal server. The lookup process thus begins with the local nameserver being directed to the DNS server for the requested site (this would be mit.edu from the previous example; we will assume for the sake of consistency that the MIT website is Akamaized so that we can continue to use it as the example for the Akamai case). When the mit.edu DNS server is contacted, however, an alias called a CNAME is given to the local nameserver. This is an intermediate DNS name (not an IP address) which will will eventually resolve to the IP address for web.mit.edu. For now, the local nameserver must resolve this DNS address. Let us assume that the CNAME is a123.b.akamai.net. The local nameserver would then query akamai.net and would receive an IP address for a high-level Akamai DNS server. This server would then resolve b.akamai.net. Now, we will start realizing the benefits of Akamais way. The high-level DNS server now determines which IP address should be resolved from b.akamai.net by taking into account geographical considerations. The IP address that it settles on is the address of a low-level Akamai DNS server. The low-level DNS server runs algorithms to optimize performance in real-time, taking into account factors such as net congestion, server loads, and network conditions. It then determines the optimal webserver for the user. Thus, the local nameserver ultimately receives the IP address for an optimal Akamai server that hosts the requested website content. As one can see, Akamai has optimized the way that content is delivered via the web by developing ingenious methods for optimal website downloads and DNS lookups.

4.4 BIND Algorithms – An Overview

The general objective of the BIND algorithms is to settle on the best server for quick, reliable (accurate) answers. We will start with the Bind 4 algorithm. It is helpful to illustrate the BIND 4 algorithm with an example. Say that we have two name servers for math.mit.edu: 1.2.3.4 and 1.2.3.5. We start by querying one, say 1.2.3.4, and by penalizing it every time it is queried. The penalty is proportional to the time that it takes for the query. Thus, 1.2.3.5 will eventually be queried and penalized in the same fashion. The process will continue in such a way that the server with the least total penalty is the one that is queried.

The BIND 8 algorithm incorporates similar principles, but introduces three factors: α, β, and γ. The α factor reflects query time. It is based 70% on the previous query time and 30% on the current query time. Thus, α helps determine which server will give the fastest response. The β factor deals with accuracy. It penalizes servers that do not answer correctly, including servers that are lame. Finally, the γ factor is introduced to ensure that all machines get tried. It lowers the query time of any machine that has not been queried. Thus, it helps to ensure that potential fast and/or accurate servers are not neglected by the algorithm. These three factors should hopefully enable the BIND 8 algorithm to converge on the server that answers quickly and accurately. We will now discuss the BIND algorithms and their performance in more detail.

4.5 BIND 4 Algorithm

The BIND 4 algorithm attempts to pick the fastest server, but not to use one server exclusively, so that if traffic patterns change, the optimal server can be found. It tries to do this by keeping a running total, which includes the total time of all replies thus far. To be sure that the same server is not always picked, the total also includes a penalty for being the server picked on a given try.

We will now introduce some notation to make these ideas precise. Let N be the number of resources we can query. We let $R_i(t)$ be the running total for resource i before step t, and define $S_i(t)$ to be the time it would take resource i to complete request t. If we define $i^*(t)$ to be the resource our algorithm chooses to query on step t, we would like to minimize the average performance,

$$T = \lim_{t\to\infty} \frac{1}{t} \sum_{i=1}^{t} S_{i^*(t)}(t).$$

The BIND 4 algorithm works as follows on each step:

1. $i^*(t)$ is picked to be the i such that $R_i(t)$ is a minimum (arbitrarily break ties).
2. $R_{i^*(t)}(t+1) \leftarrow R_{i^*(t)}(t) + S_{i^*(t)}(t)$
3. All other R_i are unchanged.

Assuming that the $S_i(t)$ are constant with respect to time, it is easy to see that the best possible performance is to always pick the resource with the smallest S. We now prove that BIND 4 is N-competitive.

Theorem 4.1. *Assuming the S_i are constant, the BIND 4 algorithm described above performs with average running time*

$$T = \frac{N}{\sum_i \frac{1}{S_i}}.$$

Proof: Let $n_i(t)$ be the number of times resource i is picked before step t. Then, since the S_i are constant, $R_i(t) = S_i n_i(t)$. Consider two resources, i and j. After t steps, resource i will be chosen if $R_i(t)$ is less than $R_j(t)$, which implies $S_i n_i(t) < S_j n_j(t)$. If the reverse inequality holds, j will be chosen. Thus the ratio of the number of times i is chosen to the number of times j is chosen tends to $n_i(t)/n_j(t) = S_j/S_i$, and resource i is chosen a number of times inversely proportional to S_i.

The probability that a given resource will be chosen is c/S_i for some c. The total probability is 1, so $\sum_{i=1}^N c/S_i = 1$, and $c = 1/\sum_i S_i$. To calculate T, note that resource i will be picked with probability c/S_i and take S_i time to run. The expected running time is therefore

$$T = \sum_{i=1}^N S_i \frac{c}{S_i} = \sum_{i=1}^N c = Nc = \frac{N}{\sum_i \frac{1}{S_i}}.$$

□

The factor of N in the average running time is worrying since the ideal algorithm would pick the fastest resource every time, regardless of how many resources there are. It is conceivable that the sum in the denominator could mitigate this problem. We show that this is not the case.

Theorem 4.2. *For any ϵ greater than zero and any N, there is a set of resources such that the expected running time of the BIND 4 algorithm is at least $N(1-\epsilon)$ times as bad as the optimal algorithm.*

Proof: Set $S_1 = 1$, and $S_i = (N-1)/\epsilon$ for $i \in \{2, \ldots, N\}$. Then the average performance given by Theorem 4.1 is

$$T = \frac{N}{\sum_{i=1}^N \frac{1}{S_i}} = \frac{N}{1 + (N-1)\frac{\epsilon}{N-1}} = \frac{N}{1+\epsilon}.$$

From the identity $(1-\epsilon)(1+\epsilon) = 1 - \epsilon^2$ we deduce that $(1-\epsilon)(1+\epsilon) \leq 1$, and that for ϵ in the range $(0,1)$, $1/(1+\epsilon) \geq 1-\epsilon$. Thus $T \geq N(1-\epsilon)$, while the optimal algorithm would always choose the resource with running time 1. □

Not only is it possible for BIND 4's average time to go up as the number of resources does, but the example used in the proof of Theorem 4.2 is not unlikely in practice. If there is one resource nearby and several on the opposite coast, performance times will look much like those used in the proof.

At this point, we consider a subtle change in the BIND 4 algorithm. Instead of setting $R_{i^*(t)}(t+1) = R_{i^*(t)}(t) + S_{i^*(t)}(t)$ as in the original algorithm, what happens if we set $R_{i^*(t)}(t+1) = R_{i^*(t)}(t) + f(S_{i^*(t)}(t))$ for a function f other than the identity? It seems like we might be able to achieve a competive algorithm if we use, say, $f(x) = x^2$, or certainly with $f(x) = 2^x$. To address these issues we first prove the following theorem.

Theorem 4.3. *With R defined as above, the average operation time T of the modified BIND 4 algorithm is*

$$\frac{\sum_{i=1}^{N} S_i/f(S_i)}{\sum_{i=1}^{N} 1/f(S_i)}.$$

Proof: This proof is virtually identical to the proof of Theorem 4.1. The probability that resource will be picked is now proportional to $1/S_i$. The constant of proportionality must be $1/\sum_i f(S_i)$ to make the probabilities sum to one. Since resource i takes S_i time to run, the expected performance of the algorithm is

$$T = \sum_{i=1}^{N} S_i \frac{1/\sum_i 1/f(S_i)}{f(S_i)} = \frac{\sum_i S_i/f(S_i)}{\sum_i 1/f(S_i)}.$$

□

How much of a difference is there between the new and old BIND 4 algorithms? We will first prove that any function asymptotically worse than that used in the original BIND 4 is *very* bad.

Theorem 4.4. *If $f\colon \mathbb{R}^+ \to \mathbb{R}$ is a function such that* $\liminf(f(x)/x) = 0$*, then the performance of the new algorithm can be arbitrarily bad.*

Proof: It is clear that if $f(x)$ is less than or equal to zero for some x greater than zero, the performance of the algorithm can be worse than optimal by an arbitrarily large factor. Therefore, assume for the rest of the proof that $f(x)$ is positive for all positive x.

Assume that f has the property that $\liminf f(x) < \infty$. Then to show that performance can be worse than any fixed number A, let N be 2. Take $S_1 = 1$. There are arbitrarily large x such that $f(x) < L$, so take S_2 such that $S_2 > \frac{L(A-1)}{f(1)} + A$ with $f(S_2) < L$. Then

$$S_2 > f(S_2)\frac{A - S_1}{f(S_1)} + A.$$

After some routine algebra, we obtain

$$\frac{S_2 - A}{f(S_2)} > \frac{A - S_1}{f(S_1)}.$$

Collecting terms in A, we get

$$\frac{S_1}{f(S_1)} + \frac{S_2}{f(S_2)} > A\left[\frac{1}{f(S_1)} + \frac{1}{f(S_2)}\right].$$

From this it immediately follows that

$$T = \frac{S_1/f(S_1) + S_2/f(S_2)}{1/f(S_1) + 1/f(S_2)} > A,$$

so it is indeed possible to get arbitrarily bad performance regardless of the number of sources.

Assume then that $\liminf f(x) = \infty$. Then for large enough x, $f(x) > 1$. For any A, again take $N = 2$ and set $S_1 = 1$. Since $\liminf f(x)/x = 0$, if follows that for *any* n_0 and c greater than zero, there is an x greater than n_0 such that $f(x) < cx$. Letting $c = 1/C$, this is equivalent to saying that for any C, there is an $x > n_0$ such that $x/f(x) > C$. Since $\liminf f(x) = \infty$, there is some n_0 such that for all x greater than n_0, $f(x) > 1$. Then take S_2 to be a number greater than n_0 for which $S_2/f(S_2) > (A-1)/f(1) + A$. Then, since $f(S_2) > 1$, it follows that

$$\frac{S_2 - A}{f(S_2)} = \frac{S_2}{f(S_2)} - \frac{A}{f(S_2)} > \frac{A-1}{f(1)} + A - \frac{A}{f(S_2)}.$$

But since $f(S_2) > 1$, $A/f(S_2) < A$, and

$$\frac{A-1}{f(1)} + A - \frac{A}{f(S_2)} > \frac{A-1}{f(1)} + A - A = \frac{A-1}{f(1)}.$$

So

$$\frac{S_2 - A}{f(S_2)} > \frac{A-1}{f(1)}.$$

Clearing denominators, we get $S_2 f(1) - Af(1) > Af(S_2) - f(S_2)$. Collecting terms in A, $S_2 f(1) + f(S_2) > A\left[f(S_2) + f(1)\right]$. Dividing through, we get

$$\frac{S_2 f(1) + f(S_2)}{f(S_2) + f(1)} > A.$$

Dividing the numerator and denominator by $f(1)f(S_2)$, we obtain

$$T = \frac{S_1/f(S_1) + S_2/f(S_2)}{1/f(S_1) + 1/f(S_2)} = \frac{1/f(1) + S_2/f(S_2)}{1/f(1) + 1/f(S_2)} > A,$$

so that now we have proved our result in its full generality. □

In a sense this shows that the algorithm actually used by BIND 4 is really not *too* bad, since at least the worst case performance is bounded by N. Choosing a penalty asymptotically smaller than S_i can lead to extremely bad performance. What happens if we choose a larger penalty? Can we ever get the average performance to go to the optimal result? The answer is that we can get very close to optimal performance, but we can't ever get there.

To motivate this discussion, consider what happens if $f(x) = x^2$. Similarly to what we did to prove Theorem 4.2, take $S_1 = 1$ and $S_2 = S_3 = \ldots = S_N = \sqrt{N-1}$. Then

$$T = \frac{1 + (N-1)\frac{\sqrt{N-1}}{(\sqrt{N-1})^2}}{1 + (N-1)\frac{1}{(\sqrt{N-1})^2}} = \frac{1 + \sqrt{N-1}}{2}.$$

So the performance is still not independent of N, despite increasing the penalty. We can do a similar analysis for any function with a suitably defined inverse.

Theorem 4.5. *Given a bijective function $f\colon \mathbb{R}^+ \to \mathbb{R}^+$, there is a set of S_i such that the new BIND 4 algorithm runs in average time $\Omega(f^{-1}(N-1))$ worse than competitive.*

Proof: Take $S_1 = 1$ and $S_i = f^{-1}(N-1)$ for $i \in 2 \ldots N$. Then

$$T = \frac{\frac{1}{f(1)} + (N-1)\frac{f^{-1}(N-1)}{f(f^{-1}(N-1))}}{\frac{1}{f(1)} + (N-1)\frac{1}{f(f^{-1}(N-1))}} = \frac{1/f(1) + f^{-1}(N-1)}{1/f(1) + 1} = \Omega(f^{-1}(N-1)).$$

□

So while we can get performance that grows arbitrarily slowly with N, we cannot get performance that is independent of N.

We can extend this result to functions satisfying less stringent conditions than those of Theorem 4.5.

Theorem 4.6. *Given a function $f\colon \mathbb{R}^+ \to \mathbb{R}^+$ there is a set of S_i such that the algorithm runs in arbitrarily bad time.*

Proof: If $\liminf f(x)/x = 0$ this is an immediate consequence of Theorem 4.4. So assume $\liminf f(x)/x$ is positive. Then there is some c and some n_0 such that for all $x > n_0$, $f(x) > cx$. Define $f^{-1}(x)$ to be the largest x such that $f(x) \le y$ and $f(x') > y$ for $x' > x$. Take $N > n_0 + 1$. Then, $f(f^{-1}(N-1)) \le N-1$ and $f(f^{-1}(N-1)) \ge cf^{-1}$. So if $S_1 = 1$ and $S_2 = S_3 = \ldots = S_N = f^{-1}(N-1)$,

$$T = \frac{1/f(1) + (N-1) \cdot f^{-1}(N-1)/f(f^{-1}(N-1))}{1/f(1) + 1/f(f^{-1}(N-1))}.$$

But $1/f(1) + (N-1) \cdot f^{-1}(N-1)/f(f^{-1}(N-1)) \ge 1/f(1) + f^{-1}(N-1)$, and $1/f(1) + 1/f(f^{-1}(N-1)) \le 1/f(1) + 1/cf^{-1}(N-1)$. Thus,

$$T \ge \frac{1/f(1) + f^{-1}(N-1)}{1/f(1) + 1/cf^{-1}(N-1)}.$$

But, if $f(x) > cx$ for $x > n_0$, then $f^{-1}(x)$ increases to infinity, and T grows to infinity with N, as was to be proved. □

4.6 BIND 8 Algorithm

As mentioned above, the BIND 8 algorithm works somewhat similarly to BIND 4. However, it uses three factors to determine performance. The equations governing the assignment of a penalty in round t are as follows:

1. If resource i is not selected to be queried, $R_i(t+1) = \gamma R_i(t)$.
2. If resource i is selected to be queried, and returns a correct response, $R_{i^*(t)}(t+1) = \alpha S_{i^*(t)}(t) + (1-\alpha)R_{i^*(t)}(t)$. Remember that the notation $i^*(t)$ indicates the resource chosen to be queried at step t.
3. If resource i is selected, but fails to return an acceptable answer, $R_{i^*(t)}(t+1) = \beta R_{i^*(t)}(t)$.

As previously mentioned, the BIND 8 code uses $\alpha = 0.3$, $\beta = 1.2$, and $\gamma = 0.98$. This algorithm intuitively seems like it should pick the fastest server most of the time, but we show that in fact its performance can be arbitrarily bad.

Theorem 4.7. *There is a set of two resources such that the BIND 8 algorithm's performance is worse than N-competitive.*

Proof: Assume $S_1 = 1$ for all t, and $S_2 = x$ for all t, where x is some fixed number. Then assume resource 2 is picked. R_2 will be x, γx, $\gamma^2 x$, etc., on each succesive step, as R_1 will be picked each time. Once $\gamma^n x$ is less than one, x will be picked again. But $\gamma^n x = 1$ is the same as $n = \log_{\gamma^{-1}}(x)$. So the average running time of the algorithm is approximately $1 + x/(\log_{\gamma^{-1}} x)$, which can clearly be made as large as we want by choosing an appropriate x. □

Bibliography

[1] Akamai Whitepaper: "Internet Bottlenecks: the Case for Edge Delivery Services." 2000, Cambridge, MA. Available URL: `http://www.akamai.com/en/html/services/white_paper_library.html`.

[2] Leighton, Tom; Shaw, David; Sudaran, Ravi. Presentation: "DNS." 2002, Cambridge, MA.