A widespread internet outage caused several major websites to shut down Thursday afternoon, including Amazon, Delta, Capital One and Costco.

Akamai, a content distribution network that helps with the spread of data around the internet, posted on Twitter that a software configuration update caused a bug in its DNS system.

A DNS, or domain name service, helps match a website’s name to its IP address. If the DNS fails, it becomes impossible to search and connect to a website by name.
Lecture #2: Naming
plus a case-study on DNS
last time: enforced modularity via client/server

today: naming, which allows modules to interact
why use names?
they let us achieve modularity by providing communication and organization, as well as a number of other properties

client

Class Browser
(on machine 1)

def main():
    html = browser_load_url(URL)
...

def browser_load_url(url):
    msg = url # could reformat
    send request
    wait for reply
    html = reply # could reformat
    return html

server

Class Server
(on machine 2)

def server_load_url():
    ...
    return html

def handle_server_load_url(url):
    wait for request
    url = request
    html = server_load_url(URL)
    reply = html
    send reply

retrieval
the client can retrieve the dog-food page because it can name it

sharing
the server can share the dog-food page with multiple clients (i.e., multiple clients can view this page)

user-friendly IDs
kaws.com is easier to remember than (say) 18.25.4.171; the variable name “html” is easier to remember than a particular location in memory

addressing
some names also specify location information
**why use names?**

They let us achieve modularity by providing communication and organization, as well as a number of other properties.

### hiding

Code on the server can access `dogfood_data.txt` without having to worry about how the file is laid out in memory.

### indirection

The server can change the memory layout of `dogfood_data.txt` without notifying the user.
why use names? they let us achieve modularity by providing communication and organization, as well as a number of other properties

the design of a system’s **naming scheme(s)** helps it achieve these properties

a naming scheme includes

1. the set of all possible **names**
2. the set of all possible **values**
3. a **look-up algorithm** to translate a name into a value (or a set of values, or “none”)

Katrina LaCurts | lacurts@mit.edu | 6.033 2022
naming case study: the **domain name system (DNS)**, which maps hostnames (eecs.mit.edu) to **IP addresses** (18.25.0.23)

the **look-up algorithm** has to scale to the size of the Internet, while dealing with constant updates and issues of delegation

- a partial view of the DNS hierarchy. Each box represents a **zone**. Name servers within a zone keep track of that zone’s mappings

```
<table>
<thead>
<tr>
<th>198.41.0.4</th>
<th>eduardo. 192.14.171.191 NS</th>
</tr>
</thead>
<tbody>
<tr>
<td>root</td>
<td>com. 192.14.171.192 NS</td>
</tr>
<tr>
<td>nameserver</td>
<td>net. 192.14.171.193 NS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>192.14.171.191</th>
<th>eduardo. 18.72.0.3 NS</th>
</tr>
</thead>
<tbody>
<tr>
<td>mit</td>
<td>berkeley.edu. 128.32.136.14 NS</td>
</tr>
<tr>
<td>nameserver</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>18.72.0.3</th>
<th>eduardo. 18.25.0.23 A</th>
</tr>
</thead>
<tbody>
<tr>
<td>mit</td>
<td>web.mit.edu. 18.9.2.69 A</td>
</tr>
<tr>
<td>nameserver</td>
<td><a href="http://www.mit.edu">www.mit.edu</a>. 18.9.22.169 A</td>
</tr>
</tbody>
</table>

| DNS client      | query sent to:                |
|-----------------| response:                     |
| e.g., your laptop |                               |
```
naming case study: the domain name system (DNS), which maps hostnames (eecs.mit.edu) to IP addresses (18.25.0.23)

The look-up algorithm has to scale to the size of the Internet, while dealing with constant updates and issues of delegation.

A partial view of the DNS hierarchy. Each box represents a zone. Name servers within a zone keep track of that zone’s mappings.
naming case study: the **domain name system (DNS)**, which maps **hostnames** (eecs.mit.edu) to **IP addresses** (18.25.0.23)

the **look-up algorithm** has to scale to the size of the Internet, while dealing with constant updates and issues of delegation

a partial view of the DNS hierarchy. each box represents a **zone**. name servers within a zone keep track of that zone’s mappings

DNS client

e.g., your laptop

query sent to: 192.14.171.191
response: try 18.72.0.3

<table>
<thead>
<tr>
<th>198.41.0.4</th>
<th>192.14.171.191</th>
<th>18.72.0.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>root nameserver</td>
<td>edu. 192.14.171.191 NS</td>
<td>mit.edu. 18.72.0.3 NS</td>
</tr>
<tr>
<td>root nameserver</td>
<td>com. 192.14.171.192 NS</td>
<td>berkeley.edu. 128.32.136.14 NS</td>
</tr>
<tr>
<td>root nameserver</td>
<td>net. 192.14.171.193 NS</td>
<td></td>
</tr>
<tr>
<td>mit nameserver</td>
<td>eecs.mit.edu. 18.25.0.23 A</td>
<td>web.mit.edu. 18.9.2.69 A</td>
</tr>
<tr>
<td>mit nameserver</td>
<td></td>
<td><a href="http://www.mit.edu">www.mit.edu</a>. 18.9.22.169 A</td>
</tr>
<tr>
<td>mit nameserver</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
naming case study: the domain name system (DNS), which maps hostnames (eecs.mit.edu) to IP addresses (18.25.0.23)

the look-up algorithm has to scale to the size of the Internet, while dealing with constant updates and issues of delegation

a partial view of the DNS hierarchy. each box represents a zone. name servers within a zone keep track of that zone’s mappings
naming case study: the domain name system (DNS), which maps hostnames (eecs.mit.edu) to IP addresses (18.25.0.23)

the look-up algorithm has to scale to the size of the Internet, while dealing with constant updates and issues of delegation

the domain name system (DNS), which maps hostnames (eecs.mit.edu) to IP addresses (18.25.0.23)

a partial view of the DNS hierarchy. each box represents a zone. name servers within a zone keep track of that zone’s mappings

performance issue: this is a lot of queries, especially to the root server

reliability issue: what happens when a nameserver fails or (security issue) is attacked?

control issue: who should manage the root server?
**modularity** and **abstraction** mitigate complexity. A **client/server model** allows us to enforce modularity by putting modules on physically separate machines.

**naming** is what allows modules to interact, and can help us achieve other goals through properties such as indirection, user-friendliness, etc.

The **domain name system** is a great case-study in naming, and also illustrates principles such as **hierarchy**, **scalability**, **delegation**, and **decentralization**.

The example you saw in lecture was a fairly basic one; you will talk more about DNS’s performance enhancements in recitation tomorrow, which change how some (many) DNS queries are resolved and client/server models, and (tomorrow) caching, and (in May) security…