SIENA*college*
Computer Science

Computer Science 180
Web Design
Siena College
Fall 2011

Topic Notes: Networks

We will take a little break from learning new web authoring techniques to study some of the underlying network technology that enable the World Wide Web.

But first, some basics. What is a *network*? Simply put, it is a connection among computers that allow those computers to share information. The computers and other devices connected to a network are often called *nodes*. Nodes are connected within a network via a *router* or similar device.

We will think of the information we wish to send on a network as a *bitstream*, just a collection of 0's and 1's. The network is just a way to transmit collections of binary data. Just as the contents of a file are meaningless unless we have some rules to lay out the bits of the file in an organized way and to interpret them as the information they are supposed to represent, the information transmitted on a network is a meaningless stream of 0's and 1's unless we have rules for how to organize it and interpret it.

Siena has a network on campus. I have a network at home.

There is no direct connection between my home network and Siena's network – that connection comes via the *Internet* (an "inter network" or "network of networks").

Network Basics

We can study networks in each of three layers:

- Low-level infrastructure
  - Ethernet
  - TCP/IP
  - packets
- Higher-level architecture (topology)
  - nodes, routers, wired/wireless links
- Services delivered across the network
  - file downloads (ftp)
  - world wide web (http)
  - email (smtp)

- ...

Low-Level Infrastructure

There are three low-level technologies required for a network to function:

1. Rules for "talking" on the communication medium (who can be sending a signal on the wire). *Ethernet* is a very common example.
2. An *addressing* scheme to allow information sent from one computer to another. *IP* is the Internet addressing scheme.
3. Some structure in this communication (a *protocol*) to manage a "conversation" between two computers. *TCP* is a very common protocol for reliable communication.

Ethernet is built into most modern computers and other networked devices.

- An Ethernet port looks kind of like a phone jack, and an RJ45 connector clicks into the port.
- The Ethernet protocol manages collisions on the communication medium (the wire). If more than one device attempts to "talk" at the same time, the "back off" protocol allows the devices to try again at different times. Ethernet will allow each device to have a turn to talk, since each device is required to let other devices have a chance to talk.

IP (the *Internet Protocol*) network addressing scheme gives each computer on the network a unique address.

This address is used to *route* the data from a sender to a receiver.

These addresses take the form of a 32-bit number, written as a sequence of 4 8-bit numbers, separated by .'s. The names we know (like `siena.edu`, `google.com`, `courses.teresco.org`) are translated to these numbers. We can look these up using the `nslookup` tool.

Every computer on the Internet has an *IP address*. When your computer sends something onto the network, it must provide the appropriate IP address of the destination.

TCP (the *Transmission Control Protocol*) manages the IP "conversations":

- making the initial connection (get the other computer's attention)
- agreeing on how to connect
- how to deal with errors, if part of the "conversation" is lost, how does a retransmission get requested and sent?

Packet Switching

Suppose we would like to have a "conversation" involving 8 MB of data (*e.g.*, a file download).

We could send the data as one large "chunk" of 8 MB of data (think: a big freight train that can carry all of our data, or sending a single package containing a 1000 page document through the mail).

Or, we could break down the data into smaller, independent pieces (called *packets*), and send them out (think: a fleet of smaller data trucks, or sending that 1000 page document in 1000 separate envelopes, with pages numbered so we can put them in order when they arrive).

Which of these approaches will be more reliable and robust when there are traffic disruptions (as there will be on a network)?

Well, if the big chunk of data does not arrive successfully (which would be the case if any part of it was lost along the way), the entire chunk would need to be retransmitted. And since it's so large and will take a long time to transmit, there is a very good chance that a problem will arise.

On the other hand, if we sent out a large collection of smaller packets, there is a very good chance that some of those packets will not arrive successfully. But..there is also a very good chance that the majority of them will arrive successfully. In this case, we need only retransmit the ones that failed.

All bitstreams on the Internet are broken down into packets.

Each packet has the sender's IP address and the destination IP address.

One of the reasons that the Internet is so reliable is that each packet can take a different route to its destination. If a link becomes unavailable, unreliable, or simply too busy, traffic can be routed along other paths.

Internet Services

The world wide web is just one example of a *service* that can be transmitted via the Internet.

There are thousands of services, many of which we can see at `http://www.iana.org/assignments/port-numbers`, the list of *ports* assigned to services by the *Internet Assigned Numbers Authority (IANA)*.

If we looked through this list, we would see only two deal with the web: `http` on port 80, and `https` on port 443.

Some others that are important and perhaps familiar:

- 21: file transfer protocol (ftp)
- 22: secure shell (ssh)
- 23: telnet

- 25: simple mail transfer protocol (smtp; email transmission)
- 143: internet message access protocol (imap; email fetch from server)

These port numbers corresponding to services allow incoming network data that arrives at a computer to be directed to the appropriate program running on that computer. Port 80 traffic goes to the web server, port 22 traffic goes to the interactive remote login server, port 25 goes to the email server, *etc.*.

Low-Level Internet Summary

Key ideas about the low level network:

- Rules for traffic control (Ethernet)
  - avoid and resolve collisions
- Common addressing scheme (IP)
  - host name (`siena.edu`) translated to IP address
- A conversation protocol (TCP)
  - conversation sent in packets
- Services offered over the connection
  - http, ftp, smtp, IM, etc,

Higher-Level Network Architecture

How does your network device get information to "the Internet"?

- The network device needs a connection to an *Internet Service Provider (ISP)*
- This connection is often called "The Last Mile"
- Typical connection types:
  - "dialup" over a traditional telephone line (24-52 Kbps)
  - DSL - a digital subscriber line, provided by phone companies (400-800 Kbps)
  - Cable (1+ Mbps)
  - Wireless/cell phone network

This gets you to the ISP. As an individual, you are making a connection to the ISP's *Local Area Network (LAN)*.

The ISP then needs to connect to a *Network Access Point (NAP)*.

- Typically done with cable connections such as T1 (1.5 Mbps) or T3 (42 Mbps) lines.
- NAPs are owned by large telecommunication companies.
- NAPs are interconnected into a *Wide Area Network (WAN)* using cable connections such as OC3 (155 Mbps), OC48 (2.5 Gbps), OC192 (9.6 Gbps). These make up the Internet *Backbone*.

See a visualization of what the Internet "looks like" at `http://en.wikipedia.org/wiki/File:Internet_map_1024.jpg`

A Network of Routers

Recall that all Internet traffic is broken down into a set of packets. It is these packets that need to be routed from one computer to another on the Internet.

This is achieved by sending the packets through a network of routers.

Each connection point on the Internet has a *router computer*.

A router keeps track of:

- which other routers it is connected to
- how much traffic is going to the other routers

Each packet (of a bitstream) is examined by each router it goes through. The router decides the best direction (*i.e.*, route) to send the packet to next.

At the destination, the original bitstream is assembled from its many packets, some of which may have been lost and retransmitted, and which may have arrived in any order.

Each router needs only understand where to send a packet next to get it closer to its destination. No one router has knowledge of the entire path the packet has taken and/or will be taking.

An interesting tool to experiment with: `traceroute`. On many networks, this tool will allow you to see the series of routers that are involved in a connection between your computer and any host on the Internet.

Domain Name Service (DNS)

How do all those Internet names (*hostnames* or *domains*) that we know so well get translated into IP addresses, and who decides which names go to which addresses?

The answer to both is that there is a central authority in charge of names and numbers:

The Internet Corporation for Assigned Names and Numbers (ICANN), `http://www.icann.org/`

Any name to be recognized on the Internet must be registered with ICANN. Any IP addresses associated with those names must be assigned by ICANN.

Individuals or organizations do not contact ICANN directly to register names and obtain numbers. Other companies/organizations called *domain registrars* do these registrations.

You've almost certainly seen ads for some of these like GoDaddy or Network Solutions. I use one in France called Gandi.

The individual or organization that desires a name finds one that's not already in use and pays one of these domain registrars to add that name to the appropriate *top-level domain* (`.com`, `.org`, `.net`, *etc.*) and optionally allocates one or more IP addresses for that domain's use.

With the domain registered (*e.g.*, `siena.edu`, `teresco.org`), a domain name server can then be set up to map names within the domain (*e.g.*, `www.siena.edu`, `blackboard.siena.edu`, `courses.teresco.org`) to IP addresses.

Each domain has an *authoritative domain name server* that is the one place that manages name to number mapping for that domain.

But how does every computer in the world know how to find the authoritative domain name server for a particular domain like `siena.edu` or `teresco.org`?

ICANN maintains a set of name servers called *root servers* that know how to look up the authoritative domain server for each *top-level domain* (*e.g.*, `.com`, `.org`, `.net`). So the `.org` server knows how to find my name server for `teresco.org`.

But does that mean every time anyone on the planet needs anything in my domain, my server needs to do that translation? No, because the name servers throughout the Internet will maintain a *cache* of the names they have looked up in the recent past. If the same name is requested again, the number in the cache is used and the authoritative server is not queried again.

This system is hierarchical in the sense that a local name server looks up names for one local network (like Siena's).

- All computers at Siena will first query a local name server.
- If the server already knows the number for a given name (and it will for the most common and popular lookups), it returns it and the work is done.
- If not, it knows who to ask: the name server at Siena's network provider.
- If the network provider's servers know the number for the given name, it returns it to Siena's server (which remembers it for a while, just in case it's asked again) and provides the number to the computer that issued the request.

- Only when the request gets all the way up the chain to the root servers without being answered does the authoritative domain name server need to answer the query.

Some tools to try out:

- nslookup: look up IP addresses for host names.
- whois: look up information about domain registrations. Fun with whois: find out where your junk email is coming from.

Client-Server Model for Network Services

We've talked about services on the Internet, but how are these services provided?

The basic method here is the *client-server model*. A central *service provider* or *server* on the network awaits requests from other machines, the *clients*, for whatever information that server can provide.

The server has a known address (its IP address, which we obtain from a name using DNS), so clients know "who" to ask.

When a request arrives at the server, it sets up communication with the client to find out what information is being requested. If the information is available, it is sent to the client.

This model is used by a wide variety of services on the Internet:

- DNS
- Web
- time servers
- some file sharing (like our SOSAD files in lab)
- email

Let's think about what happens when you wish to view a web page.

- You enter a URI (aside: this is a *uniform resource identifier*, also commonly referred to as a URL, or uniform resource locator), such as `http://xkcd.com` into your web browser (the client).
- The browser looks up the name by contacting a DNS server (and possibly having that request forwarded to other servers as we discussed earlier).
- The name server responds with the IP address 72.26.203.99.

- The browser then sends a request for `http` service from the web server at IP address 72.26.203.99.
- The server receives the request and sends back a response, in this case the contents of a web page.
- The browser receives this information and displays the page.

Web Servers

A *web server* is a computer connected to the Internet that has a program listening for http requests and responds to those requests.

The most popular web server program is called *Apache*. Apache is just a program like any other, except that its job is to respond to web server requests. Microsoft's Internet Information Server (IIS) is also a popular web server running on Windows.

The server also needs to have the pages and images that the web server program will send out. These are in files on the web server just like other files.

The web server program interprets the request and provides the appropriate file.

The URI includes information used by the server to find the appropriate file. For example, the URI

`http://courses.teresco.org/cs180_f11/lectures/lect7/index.html`

consists of three main parts:

1. `http` indicates Hyper Text Transfer Protocol (the standard web service) is to be used, which uses port 80.
2. `courses.teresco.org` is the host name of the web server. We have already seen how DNS converts this to the appropriate IP address.
3. `cs180_f11/lectures/lect7/index.html` provides the name of the file on the web server that we wish to display.

In this case, the web server running at `courses.teresco.org` will look on its hard disk in the location that contains web server files for the site `courses.teresco.org`. The server configuration defines this as a *directory* (folder) named `/home/www/courses`. (We will look at server configuration soon.)

Within that directory, the server looks for the file specified in the rest of the URI: `cs180_f11/lectures/lect`

This means it looks in `/home/www/courses` for a directory named `cs180_f11`, that in turn contains a directory `lectures`, which in turn contains a directory `lect7`, which in turn contains

the file `index.html`. And that is the file the web server sends back across the network to the client (browser).

Example Apache Server Configuration

We will look in a bit more detail at the server configuration on my server.

Note: you are not expected to know the details of the configuration file or the configuration of this server specifically, but you should have an idea of what is possible to configure with a web server.

The server is on a computer named `blizzard.teresco.org`. It serves many purposes for the `teresco.org` domain – our concern here is web service. The server runs a variant of the Unix operating system called FreeBSD. A more popular alternative today is Linux.

We do not need to know too much about Unix for our purposes, but a few things that will be helpful:

- A Unix system has a unified file system – no matter how many disk drives and partitions it contains, they are all available in file paths that start with a `/`. These paths consist of a series of directory (folder) names, followed by the name of a file. The idea is familiar from other systems, but the specification may be a bit different.

  The path to the Apache web configuration file on the server is

  ```
  /usr/local/etc/apache22/httpd.conf
  ```

- Unix systems have a collection of programs that start up when the computer boots up and which run "in the background" for the entire time the computer is up. These programs are often called *daemons*. The Apache web server is one of these, and it is named `httpd` (the HTTP Daemon).

If we look at the web configuration file, we can see that there are many, many configurable options. We will look at a few.

- `Listen 80`

  – This line tells `httpd` which port it should accept connections on. Remember that port 80 is the default port for web servers, but we can use this line to set any port we wish (like the Siena Jira server which listens on port 8080).

- The `LoadModule` entries load additional capabilities into the `httpd` server. Most of these are standard with any web server and correspond to things like ways to authenticate users to the server, or ways to have individual users on the system have their own files available as web pages.

- The `ServerAdmin` directive is the email address that will be displayed when an error occurs.

- The `ServerName` directive gives the name by which the server wishes to be known, in this case, `www.teresco.org`. This name needs to be a valid name available by DNS, but does not need to be the same as the name of the computer. In this case, the computer is named `blizzard.teresco.org`.

- `DocumentRoot "/home/www"`

  – This line specifies where in the computer's file system it will look, by default, for pages to serve.

- There is then a set of configuration options inside of a `<Directory "/home/www">` section. Most of these are not of specific interest, but we will look at the group of `SetEnvIfNoCase` lines and the `<FilesMatch>` block.

  Without worrying about the details, this section is an attempt to deny *hotlinking* of images on the site. Hotlinking is when someone, often on a blog, puts an inline image on their page, but the image is still located on a remote server. All of the `SetEnvIfNoCase Referer` lines set up the sites that are allowed to link to images from the server, and then either allows or denies the request.

- The next few sections are `<DirectoryMatch>` blocks that add access restrictions on some of the server's pages.

  We'll look in most detail at the last – this one applies to all files served from the directory `/home/www/pics/kate` on the server. The block sets the requirement that anyone wishing to access files which are below that directory needs to provide a `Basic` authorization, using the passwords in the `AuthUserFile` specified, requiring the specified `user`.

  If we look in that password file, we see the usernames and passwords that are understood by the server. The passwords are *encrypted* – those are not the things you need to type. An *encryption algorithm* was used to convert the password into the *cipher text* in the file. Then when someone needs to authenticate and types a password, that same algorithm is used to encrypt the typed password. If the result matches the cipher text in the file, it means that was the same password.

- The next item of interest is in the `<IfModule mime_module>` block, where the lines

```
AddType text/html .shtml
AddOutputFilter INCLUDES .shtml
AddOutputFilter INCLUDES .html
```

  set up *server-side includes*. This means that HTML documents can include other HTML documents by including a line such as:

```
<!--#include virtual="./title.html" -->
```

  somewhere in the document. It looks like an HTML comment (and is, really) but will be interpreted by the *server* as a directive to replace this line with the contents of the file

`title.html` from the same directory before returning the contents of the file in response to a request from a client.

So by the time the browser sees the file, it looks just like any other document – with that line replaced with the contents of the included file.

What good is this? In the case of the pages at `http://j.teresco.org/`, which are served from `/home/www/homepage`, it means all of the pages there have the same header information without having to retype it (or paste it) into all of the documents. More importantly, if a change needs to be made to the header, it can be made in one place and it will take effect on all of the pages.

- From here, we jump down near the bottom of the file where there are a series of `Include` directives. Most of these are commented out (the ones after an initial #) but a few are not.

  The `Include etc/apache22/extra/httpd-userdir.conf` line includes configuration options to allow individual users to serve web pages from their own directories. My username there is `terescoj`, so any files I place under the `public_html` directory of my account will be served under the URI `http://www.teresco.org/~terescoj`.

  The `Include etc/apache22/extra/httpd-vhosts.conf` line sets up virtual hosts. This is where `courses.teresco.org` and some other sites come into play. If we look at the included file `/usr/local/etc/apache22/extra/httpd-vhosts.conf`, we see a series of `<VirtualHost>` directives, each of which sets up one of the "virtual servers" managed by this Apache instance. For each, the `ServerName` specifies one of the possible host names that could result in a request to this server, and the corresponding `DocumentRoot` specifying the path to the files to use for those requests.

To understand this a bit better, we will set up a new virtual host on the server by adding a new name to the `teresco.org` DNS tables then creating a new `<VirtualHost>` block corresponding to it in the Apache configuration.

Web Clients (Browsers)

Web clients, or *browsers* are, of course, the component of the World Wide Web with which most people are most familiar.

A brief history:

- Berners-Lee's first web client was called "WorldWideWeb" first made available in late 1990 and 1991.
- The "Mosaic" browser from NCSA (National Center for Supercomputing Applications at the University of Illinois) was the first widely popular browser, starting in 1993.
- Mosaic went on to become "Netscape Navigator" and later "Mozilla" and eventually today's popular "Firefox" browser.

- Microsoft's entry came in 1995 with "Internet Explorer", which became and remains the most frequently used browser.
- "Opera" is a commercial browser that has a small but loyal user base.
- Apple's "Safari" browser arrived in 2003, and is the most popular Mac-based browser today.
- Google entered the browser market in 2008, with "Chrome".