Team Meeting

January 24, 2017

Meeting Agenda

- Milestone Update
- Brief bit on project timelines
- Abbreviated intro to computer networks
- Data modeling revisit?

I didn’t get through anywhere near as much management/design material as I’d hoped last time… Since the work this week is primarily with python-like technology, I figured I’d start with

Phase o Deliverables

- Demonstration contractor can use OSNAP environment
- Demonstration contractor can migrate OSNAP legacy data
- Demonstration contractor can make a web application
- Demonstration contractor can meet OSNAP documentation requirements

We’ve completed the first two milestones. It was a lot of uncomfortable work. Well done. The next milestone is a demonstration that we can generate a simple web application.

A few choice status entries...

- Dear lord is it ever dull, writing a whole bunch of SQL statements. Even with loops, there's still a whole bunch of statements that had to be written.
- Finalized all scripts for the data migration and pushed them to github. Which circle of hell do people who use inconsistent date formats go to? Seems like fraud, so I'm going to go with the eighth.
- Figuring out how to relate all the tables... this is hard.
- My other classes have assignments due mid-week so it's tempting to postpone but being far enough into the project to participate in the design discussion would have been really useful.

I really appreciate the status information. I wish I'd had a little more to go on before the weekend regarding what the anticipated or encountered difficulty was for the project. I was hearing a lot of confident status about cruising through the assignment over the weekend and only a couple status messages about the difficulty of working with the data. This messaging kept me from rethinking the delivery date and producing more examples. By Thursday lecture, we're pretty well locked into what is available for the assignment over the weekend… which for some of us sounded really rough.

Abbr. Intro to Networks

- High-level Idea
- Layered Model and Internet Protocol
- Early Web and HTTP
- Web Applications and cookies

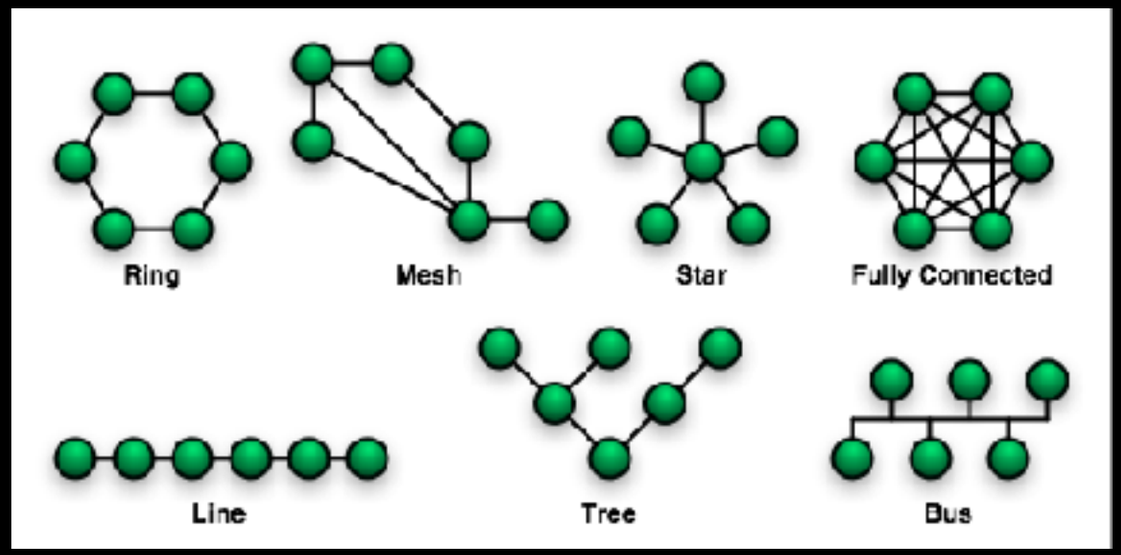
Feel free to stop me and ask for depth. I will give you what information I can… I did a little bit of network research back in 2004 with Reza but that was a long time ago now.

High-level Ideas

- Computers are awesome
- Sharing is awesome
- It would be really great if our computers could help us share

- Can we connect physically connect computers together?
- Can we write code that allows the connection to be useful?

Modeled As Graphs



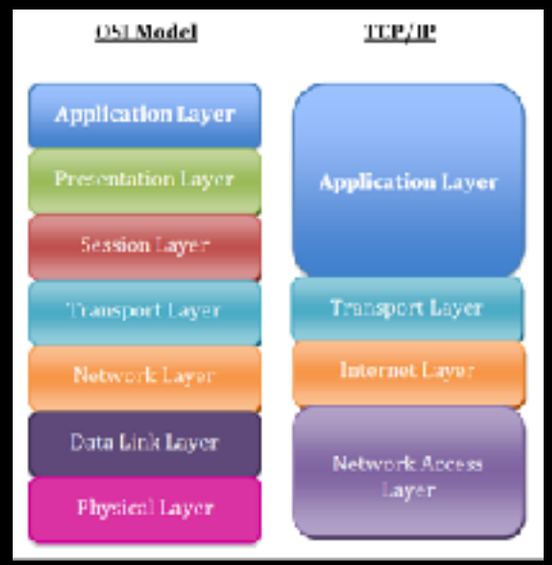
https://commons.wikimedia.org/wiki/File:NetworkTopologies.png

Typically computer networks are modeled as graphs. The graph nodes represent computers and the edges represent wires/connections. Some low-level network protocols require particular topology shapes.

Internet protocol (IP) is pretty flexible and does not require a specific network topology. Using IP, each node in the network is assumed to have a unique numeric address that can be used to send messages to that node. Since multiple programs can run simultaneously on a single node, a port is used to identify the application that should receive a given message/packet.

Layered Model

- Data needs to move from the source application to signals in a medium to data in the target application
- This generally happens in steps
- The ISO network model generalizes and orders the steps



https://commons.wikimedia.org/wiki/File:Application_Layer.png

A bunch of work goes on to move data from one machine to another. At the top of the network stack is the application processing the conceptual data. At the bottom of the stack is the actual physical representation of the data while in transit.

Since doing nothing is easier than doing something, TCP was a good choice for implementing the early web.

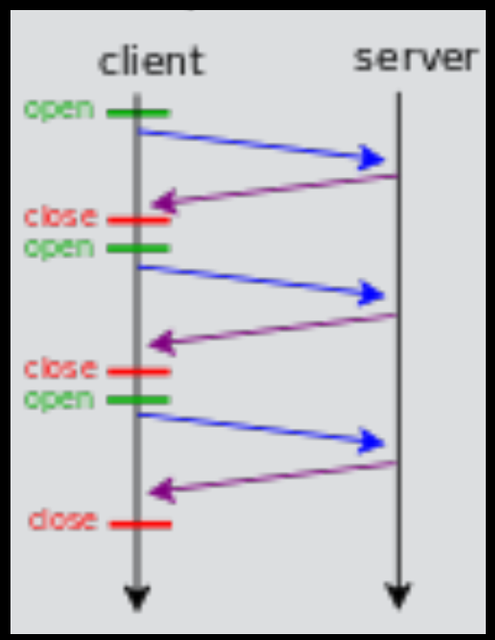
Internet Protocol is a network layer protocol. IP is generally supported by IEEE 802.3 (Ethernet) or 802.11 (wireless) in current networks.

The two primary transport layer protocols on top of IP are TCP (Transmission Control Protocol) and UDP (User Datagram Protocol). UDP is stateless and all the data for a UDP message must fit in a single UDP packet. TCP sets up a stream and guarantees in order delivery of data written to the stream.

If your application is going to move large amounts of data, like images or audio files, UDP packets are too small. A developer could still use UDP to transmit the data but would need to write their own code for fragmenting and reassembling the file. Or, a developer could use TCP and not need to solve the fragmentation and reassembly problem.

HTTP

- Hyper Text Transfer Protocol
- Client/Server
- Single request/response per connection
- Stateless interaction with the client
- Public data



HTTP was intended to use a stateless client server model. For every connection, a client would make a single request for a file and get a single file back. Each request would be treated as completely unrelated by the server. The web server publishes data, so restricting access while data is in transit or checking who was requesting the data weren't design concerns.

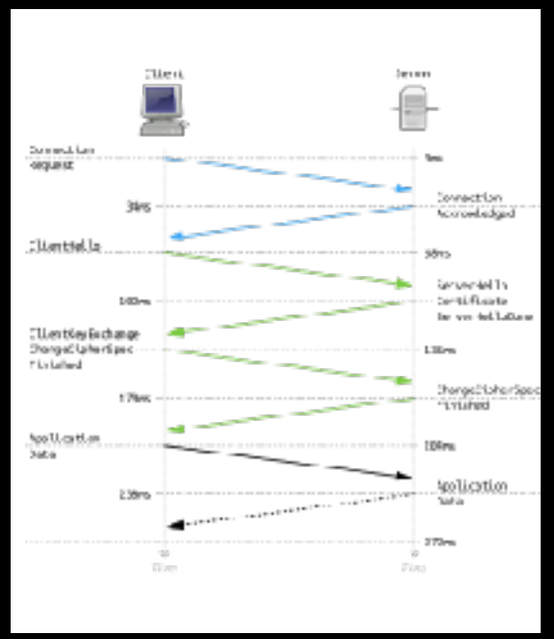
HTML (hyper text markup language) files are only one kind of data that HTTP can transmit. HTML provides a language to annotate parts of text files with tags; it is up to the web browser to determine how to render the text and tags (separating the concerns of content and display). Some people want to control how things are displayed, which led to a dark age where embedded tables were used to do layout and there was no go way to tease apart content from layout. CSS has been, to some extent, and attempt to return to a separation between content and display.

Early Websites

- A URL uniquely identifies a single document
- Documents are static or change infrequently
- Websites form an overlay graph on the Internet

Web Applications

- Anti-thesis of several original design goals
- Interactive widgets can break single request/response (hack: AJAX)
- Requires state between requests (hack: cookies/ nonces)
- Requires access control (hack: SSL+cookies/nonces)



Really, the only things modern applications use about the old web are HTTP for file transfer and HTML to describe the page to render to the browser. Pretty much everything else is hacked around by intermediate technology.

Constructing Timelines

- What do I want?
- When do I want it by?
- What needs to happen to get from here to there?
- Is it possible? How can I adjust my expectations?
- What could go wrong? How can I mitigate?

https://www.flickr.com/photos/johnbullas/7788230372

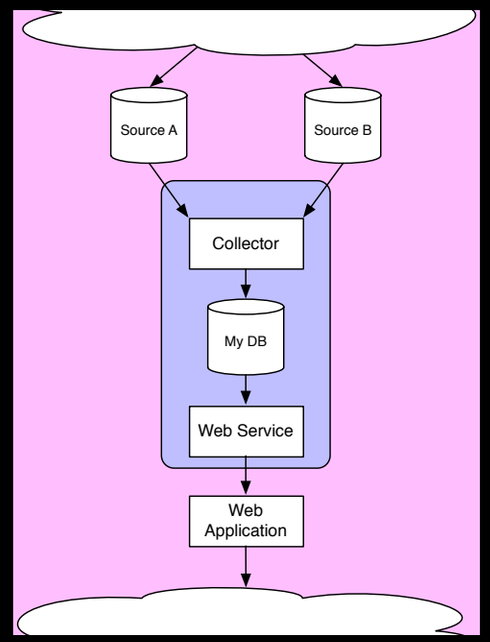
For timelines and estimates to make sense there has to be some objective to work toward and notion of progress over time towards that goal. We can work forwards, by asking first what needs to be done and then estimating the time for each part. Which gets us to an estimated delivery date. Or, we can work backwards from when the deliverable is needed.

In either case, it is possible that there just simply isn't enough time. Not having time can be handled by changing the objective (reducing features or other changes that reduce the time needed for intermediate steps) or changing the expectation for when delivery will occur.

No plan survives contact with reality. Even with good estimates things can go unexpectedly good or bad. For your manager all surprises are bad; though good surprises are far and away better than bad surprises. Bad surprises often slowdown or halt a project, causing groups to idle or the project to fail. Good surprises cause you to idle while waiting for other teams or will disrupt the architecture removing the need for work already done. Good management will be prepared for both good and bad surprises.

Case Study

- Merge data from source A and source B
- Periodically look for data indicating problems
- Expose problems via a WSDL web service
- Deliver on Oracle, JAVA, Apache Tomcat



My first hard discussion with my mentor was when putting together the timeline for a project. The project is pretty simple, take in data from two existing databases that some other group develops/manages, look through the data for patterns that indicate trouble, report on the trouble when queried by another external system. I'd already completed a system almost exactly like this a few months earlier for the same customer.

I understood the problem. The easiest seeming approach was to tackle the problem in the same order as the data flow. Solve the intake problem, then the data correlation problem, then the query interface problem. I estimate the project at 6 weeks and deliver my timeline so that my mentor can get approval for the project from the customer. Getting the timeline approved also causes the upstream and downstream teams to start planning for the integration and feature delivery.

A couple days later, I get pulled from my office to have a meeting regarding the timeline. "We will work to your timeline, but I think you should change it." The conversation came down to how well risk had been taking into account. I'd put all the technologies I was good at in the beginning of the plan and all of the unknown technologies at the end; this left little in the timeline should the unknown technology be harder to work with than planned. After 45 minutes, we ended the meeting and I executed my original plan.

Everything was great until the Apache Tomcat part. Getting that software working with the WAR file implementing the WSDL web service was terrible. The project ended up weeks late, the team that needed our data was mad, and there was a member of the operations team that was a lot harder to work with after. Had I moved Apache Tomcat earlier in the timeline, there would've been time to address the issues over the course of the regular project timeline.

Application to LOST

- The major project has been broken down into sprints already. There may be value in thinking about upcoming sprints early.
- The steps may not be in the best order for you and some steps can be done concurrently.
- Uncover the challenges early so there is the most time to respond.

ZOOM Opportunity

- Zombie Observation and Outbreak Management (ZOOM)
- ZOOM has an application team but no database team
- Needs a vendor to design a database to hold the data needed for the UI
- Currently looking for high level proposals

We’re working on the project for OSNAP, but that work runs out in 8 weeks. We need to start hunting the next piece of work so that our income is not interrupted. There is a sales meeting upcoming with ZOOM where they would like us to outline a solution to their problem. It sounds like they need our database expertise. If we can show up at the meeting with a data model that we can talk through each of their reporting needs with, that will really help to close the sale.

Primary Use Cases

- Would like reports on the tagged zombies within a radius of specific population centers
- Would like to be able to track the migration history for tagged zombies
- Would like to run a report to allocate tagging bounties to agents

There are 6 reports ZOOM really would like to do that they can’t yet.

Primary Use Cases

- Would like to know which zombies have a particular tag type
- Would like to know where zombies with a tag running out of power are located so that they can be retagged
- Would like a report on which agents have become zombies and when they turned

More on Tagging

- There are different types of tags
- A single zombie may have multiple tags
- Some tags are passive and never run out of power
- Tags have a permanent identifier from the vendor
- ZOOM uses several tag vendors

With that information, what do we bring in for a diagram? What are the fields that are likely important for each entity?