COMPSCI 630 Systems **Fall 2016**

Lecture 13: Fault Tolerance

Lecturer: Emery Berger *Scribe(s): Emily Herbert*

13.1 Admission's Control

Admission's control is a term used to describe managing load in a system.

13.1.1 Load Shedding

The example given was of a bar:

Imagine a bar. People enter, they are served, and then they exit. The load of the bar is determined by a combination of (1) how quickly people arrive and (2) how quickly they are served.

This brings us to **Little's Law**:

$$L = \lambda W \tag{13.1}$$
$$\text{average number of customers} = \text{arrival rate} \cdot \text{service time} \tag{13.2}$$
$$E[L] = E[\lambda] \cdot E[W] \tag{13.3}$$

This means that the bar is always in one of two states:

1. If the service time is lower than the arrival rate, customers are served as quickly as possible and are happy :)
2. If the service time is higher than the arrival rate, this builds a queue. This increases wait times and makes everyone sad :(

This concept introduces **load shedding**. The idea that turning away customers can (1) improve Quality-of-Service (QoS) and (2) mitigate against Denial-of-Service (DoS) attacks.

13.1.2 Thread Pools

Servers use thread pools for admission's control. Have a set number of threads "at the ready", so that when one is available it can be used. This is instead of creating threads on demand.

13.2 Cryptography

Security used to be done through "security through obscurity". This relies on the internals of the system being hidden from the users, and it really isn't secure at all. Modern security uses a different threat model.

It is assumed that the resources that you want secure (e.g. a secret message) *will* be intercepted, but security is applied to ensure that the details or content are unreachable. This is where cryptography comes into play.

Types of cryptography:

- Stegonography. A key is used to determine the location of the secret things. Codewords.
- Caesar Cipher. Symmetric crypto scheme where the key is a mapping of letters of the alphabet to other letters. The secret message is encrypted with forward mapping and decrypted with the backward mapping. This mapping can take two forms:

  1. All letters are "shifted" by some number.
     A B C D E
     J K L M N
     This is bad because it only takes 26 brute-force attempts to decrypt.
  2. All letters are mapped to a random letter.
     A B C D E
     Z K T Q N
     This is better because it takes many brute-force attempts to decrypt. But...

  Those "many brute-force attempts to decrypt" can be reduced by **frequency analysis**. Frequency analysis uses common word patterns to make guesses at decryption (e.g. a single encrypted letter is likely "I" or "a" when decrypted). This brings the random letter Caesar Cipher down to a manageable range.
- One time pad. Best thing ever. Theoretically unable to be decrypted. The message is turned into an array of random numbers, where the max of those numbers is the length of the message, and the key is the corresponding mapping.
- Rotating keys + codebooks. If you have a codebook with a bunch of keys, all you have to do is choose a new key each day. This is hard to decrypt. Apparently Germany used this and called it Enigma.

Known plaintext attacks are when you a priori known what part of the message is going to say, and you use this information to decrease the huge space of possible decryption patterns.

13.3 Encryption

The goal of encryption is to make it as if the encrypted message is just random noise. When something is encrypted to be "random noise" the best that a bad actor can do is brute-force the decryption. So, if we use large encryption keys, this becomes difficult. A system such as this offers a probabilistic security gaurentees:

$$E[\text{time to decrypt with brute force}] = \frac{1}{2} \cdot \text{key space} \tag{13.4}$$

Of course, a bad actor can attain your key. In this case, they will be unable to do both encryption and decryption. This is a property of asymmetric keys.

The best case is if we can encrypt without being able to decrypt. Thus, we want some function F that is easy to perform, that has some F^{-1} that is difficult to perform. ... prime factoring!

All this leads us to the modern day scheme: **public keys and private keys**. And also our cast of characters:

- Alice and Bob are good actors, and they are attempting to send messages back and forth.
- Mallory is malicious.
- Eve is an evesdropper.
- Sybil is the representation of reputation based systems (?)

A public key consists of multiple components: (e, d, n), where e is computed from d, and d is relatively prime to $(p-1, q-1)$. Using a combination of public and private keys, Alice and Bob can achieve:

$$\begin{aligned} \text{Signature} &= D_{bob}(message) - > secret \quad (13.5)\\ \text{Encrypt} &= E_{alice}(secret) - > secret_{alice} \quad (13.6)\\ \text{Decrypt} &= D_{alice}(secret_{alice}) - > secret_{*} \quad (13.7)\\ \text{now} &= message = E_{bob}(secret_{*}) \quad (13.8) \end{aligned}$$

Encrypted messages must be of a certain length, so they are filled up to a certain size with random numbers.

Who keeps they keys? Some public key infrastructure (PKI). It is a known condition, so to speak, that there must ultimately be one key origin, from which all trusted exchanges originate. If this is compromised, the whole system is compromised.

13.4 Fault Tolerance

$$\begin{aligned} \text{availablity} &= \frac{MTBF}{MTBF + MTTR} \quad (13.9)\\ MTBF &= \text{mean time between failures} \quad (13.10)\\ MTTR &= \text{mean time to recover} \quad (13.11)\\ & \quad (13.12) \end{aligned}$$

So to increase fault tolerance, you can either increase the MTBF or reduce the MTTR. There is a system, ROC, that seeks to reduce MTTR.

Fault tolerance recover can either be fail-safe, fail-stop, or fail-over. Fail-safe systems rollback or undo the error, fail-stop systems blowup on error, and fail-over systems rely on replicas to vote on errors.