Lecture 1: introduction

1. Introduction of me
    a.
2. Course Overview
    a. Readings – 2/3 papers per week
    b. Projects – 1 implementation, 1 more open
    c. Discussion
    d. Some student groups lead discussions, read extra papers
3. Class intros
    a. Name, area, favorite technology
4. Class properties:
    a. Grade based on:
        i. class participation + reviews
            1. I will read all reviews, let you know if it was particularly good or needs improvement – otherwise satisfactory
            2. I would like everyone to ask questions in class, have things to say. I've been known to cold call on people
        ii. Midterm/final
            1. Still under discussion, will decide soon
        iii. Projects
            1. Each one worth the same
                a. First one: building a key/value store
                b. Second one: probably cloud computing project + paper & poster session last week of classes
    b. Readings: some days, we will all read the same paper. Other days, presenters will read additional material as background. Other days, parts of the class will read different papers.
        i. Reading types:
            1. Read: read thoroughly the whole paper
            2. Skim: read intro, first couple paragraphs of each section, a bit of evaluation
            3. Choose: pick one of N papers to read. If there is imbalance, I'll assign papers instead
        ii. Reviews:
            1. About one page (60 lines of 80-column text, 500 words)
5. Next lecture: Thursday
    a. Reading assignment up on the web:
        i. Background:
            1. Eight fallacies of distributed computing
            2. Introduction to distributed system design
        ii. Foreground:
            1. Grapevine – classic distributed system facing many of the problems

iii. No reviews yet

6. Why distributed systems?
    a. What is distribution for (ASK)
        i. Fault tolerance/availability
        ii. Scalability
        iii. Sharing
    b. Why are they interesting? (ASK)
        i. Independent failures
            1. File server goes down but client doesn't
        ii. Independent management
            1. Separate web sites on a network
        iii. Properties at scale
            1. Self-synchronization
            2. Congestion
            3. Dick Sites talk on Google
                a. Hard to understand whole system
                b. Small sources of latency/congestion can add up
        iv. Security
            1. Very hard to tell who is at the other end of a network
            2. Very hard to stop someone from sending packets to you
7. What will we cover
    a. Classic distributed system problems
        i. Communication: what are the right primitives
        ii. Scalability: how do you make a system that serves a population larger than a single machine can?
        iii. Reliability: how do you improve reliability with a distributed system rather than reduce it?
        iv. Consistency: how do you make sure your application gets the appropriate data/response to a question in the presence of multiple computers?
        v. Replication: how do you make copies of data/state available on multiple machines, and what is the impact?
        vi. Security: how do you identify who you are talking to and determine what they are allowed to do?
    b. Cloud computing: new take on distributed systems
        i. Heavily client-server
        ii. New programming models
        iii. New deployment models
        iv. Vast scalability
        v. Elastic consumption
    c. General tilt of course
        i. Most people here have a lot of practical systems experience, and can read systemsy papers and understand them. And if you are interested, you will

ii. Few people read the theoretical papers on distributed systems: the protocols, the proofs, etc.
iii. We will tilt a bit towards theory, to make up for this

8. What makes distributed computing hard?
    a. Two major environments:
        i. closed LANs
            1. Well connected,
            2. High bandwidth
            3. Low load
            4. Reliable
        ii. Internet
            1. Often unconnected
            2. Variable bandwidth
            3. Variable load
            4. Not reliable
        iii. How do you build services for both? Efficiently?
    b. Eight fallacies: Generally, LAN conditions don't always exist
        i. The network is reliable.
            1. What if network is 5 nines reliable – 99.999%. If you send a gigabit of data
            2. Network can fail for a variety of reasons: backhoes, operators, software failures
        ii. Latency is zero.
            1. "*But I think that it's really interesting to see that the end-to-end bandwidth increased by 1468 times within the last 11 years while the latency (the time a single ping takes) has only been improved tenfold. If this wouldn't be enough, there is even a natural cap on latency. The minimum round-trip time between two points of this earth is determined by the maximum speed of information transmission: the speed of light. At roughly 300,000 kilometers per second, it will always take at least 30 milliseconds to send a ping from Europe to the US and back, even if the processing would be done in real time.*"
        iii. Bandwidth is infinite.
            1. Getting better, definitely
            2. Problem comes not from a single client, so much, but from many clients acting simultaneously (e.g. refreshing every X minutes)
            3. Wide-area bandwidth limited by TCP/IP and losses
                a. At 40 msec RTT and 0.1% (1 in 1000 packet) loss, TCP/IP capped at 6.5 Mbps.
                b. To reach 500 Mbps, need $3x10^{-7}$ error rate
        iv. The network is secure.
            1. Example: FTP sends password, username in cleartext
            2. E.g. MS Windows RPC did not validate format – assume correct.

Malformed packet would crash server

3. Attacks:
    a. IP injection
    b. Snooping
    c. Denial of service
    d. Dictionary attacks
    e. Malware on client desktops (see Google in China), means firewalls aren't enough

v. Topology doesn't change.
    1. Machines move, to different networks, different routes
    2. Can't statically say how to route things, where servers are, etc.

vi. There is one administrator.
    1. Cannot change everything at once
    2. Cannot change everything at all – e.g. could change server settings but not all client settings

vii. Transport cost is zero.
    1. Network is not free – provided in this department, but for real systems someone must pay for it

viii. The network is homogeneous.
    1. Latencies, reliability, distances vary
    2. E.g.: DSL, dialup, LAN clients

9. Stories:
    a. First job at Microsoft: write a locator to find a domain controller for a client
        i. Turn on machine, find domain controller to log on to
        ii. I was told it would take about a month
        iii. Challenge:
            1. Could have multiple NICs on unrelated networks
            2. Could use multiple protocols (XNS, NetBEUI, IP)
            3. Unreliable network
            4. Set of servers could change dynamically
        iv. My solution:
            1. Cache old information
            2. Send datagram ping to server
            3. Timeout for 3 seconds
        v. Problems:
            1. What if you use the wrong network? Wait 3 seconds for every error can be slow
            2. After a year I gave up (the problem was taken away from me)
        vi. Final solution: (implemented by someone else in about 6 months)
            1. Move to IP only
            2. Use DNS: make server store location in DNS
            3. Make DHCP tell us where DNS servers are (push problem to someone else)
    b. Amazon backend:

i. Uses "The Information Bus" from TIBco, which uses transactions for reliability and re-transmit for errors
ii. Uses fixed timeout for errors
    1. Each client app written to use timeouts (not common code)
iii. What happens under overload of server?
    1. Clients start timing out, retransmitting
    2. Load on network and server goes up, causes more work on server and more timeouts
    3. Clients retransmit more
    4. Whole system jams and has to be rebooted