Nathan Tung

CS111 Sections of Chapter 7, 9, Article Outline #17

Chapter 7, section 7.1, pages 7-1 through 7-20. Fallacies of Distributed Computing Explained. Chapter 9, section 9.6.3.

- Network as System Component
  - Involves layering as a modulation technique
  - Other useful network techniques: *framing, multiplexing, exponential backoff, best-effort contracts, latency masking, error control,* and *the end-to-end argument*
- Interesting Network Properties
  - Fundamental physical properties
    - Speed of light is finite: there is a propagation delay between long-distance signal transmits
    - Communication environments are hostile: threats like noise bursts can wipe out bits or operators might sever cables that need to be repaired
    - Communication media have limited bandwidth: data rate is increased (more so than signal rate) because of limiting physical properties
  - Mechanics of sharing
    - Any-to-any connection: communication system must be shared (might involve lots of entities)
    - Sharing of communication costs: while processors, memory, disk get cheaper from cheap silicon, laying wires or launching satellites are still expensive; they also involve regulatory/legislative costs
  - Remarkably wide range of parameter values: propagation times, data rates, number of communicating computers, or amount of load (size of files being transmitted)
- Isochronous and Asynchronous Multiplexing
  - Communication link is multiplexed, or used for many different communications at the same time
    - Ex: Phone network with switch in LA and Boston plus electrical circuit comm link (can become busy)
  - Network busy phenomenon plagues **isochronous** multiplexing technique
    - Hard-edged in limit; the first 703 phone calls might go through, but not the 704th one
    - Time-division multiplexing (TDM) even divides data into frames and transmits them in intervals
  - Usually "messages" are sent in bursts rather than bit stream; this is **asynchronous** multiplexing
    - Untimed, less even, but gets message through faster
    - Frames are any length and carried at any time when link is not used by other frames
    - Frame and guidance bits holding delivery info form a "packet"
      - Guidance info might be a destination address; framing finds the start/end of the frame
      - Connectionless transmissions that don't use state-maintaining switches can be used
    - When message exceeds max frame size, break them into segments and reassemble them later
  - Packet Forwarding; Delay (asynchronous)
    - In packet forwarding network, packet switches are interconnected with asynchronous links and allow packets to route through the network
    - Following an outgoing link leads to forwarding, which takes some transit time due to:
      - Propagation delay: depends on the speed of light
      - Transmission delay: depends on length of frame and the link's data rate
        - "Cut-through" is a clever trick to send packet before completely receiving it
      - Processing delay: packet switch examines guidance info to choose correct outgoing link
        - Checksum on block of data detects errors by breaking block into k-bit chunks and XORs them (k=1 is a parity check); this catches only one-bit errors and sometimes two-bit ones, but doesn't show when block chunks are interchanged
        - Checksum that detects integrity of all changes is a "witness" or "fingerprint"
        - Uses more time and increases processing delay; usually use simple checksum
      - Queuing delay: waiting for other packets to finish transmitting
        - Queuing theory is not actually useful in predicting network behavior
- Buffer Overflow and Discarded Packets
  - Buffer space is an issue in queuing transmission-waiting packets; how large should buffer be?
    - Plan for worst case
    - Plan for usual case and fight back (if buffer fills up, send messages back/tell them not to stop)
    - Plan for usual case and discard overflow (size that usually works, throw away missed packets)
  - Best-Effort Network

        - Buffer memory is not costly, but worst case is hard to discover and might be overkill
        - Also, a very long buffer might mean long queuing times
        - Congestion occurs when traffic exceeds the average buffer size, causing it to run out of space
        - Push back in this case (tell sender to stop, AKA "quench" request)
            - But adjacent switches might also be congested...OR packet switches might deadlock!
            - We can send quench request to original source, but that's really hard
                - Sometimes there are many sources. That one source might not be sending anymore. Quench message might be delayed also.
        - In the worst case scenario, buffers will have to toss extra packets
            - Packet sender can tell that network is congested and so reduce send rate
                - Called "automatic rate adaptation"
            - Simple theoretical network behavior models might not apply when some requests are discarded
        - This is called the best-effort network (discard packet if we cannot dispatch it soon)
        - Example: US Postal Service for junk mail
    - Alternative design: guaranteed-delivery network (store-and-forward network)
        - Use complete messages and a disk for buffering to deliver In once piece
        - Switch tracks individual messages to prevent lost/damaged messages
        - Example: US Postal Service for first class mail
    - Difference: best-effort (contract) network has a probability of undetected loss
- Duplicate Packets and Duplicate Suppression
    - Packets that are discarded are not actually that big of a problem and pretty routine
        - Service might simple be available/crashed
    - Unfortunately, discarded packet means client gets a slower response and they might have to send duplicate requests/responses (important for banks, not so important for spell check)
        - Impact depends on whether the state changed
    - Nonce: unique ID that is never reused by A for sending to B
    - Idempodent: repeating same request has same effect as sending one request
- Damaged Packets and Broken Links
    - Need error detection mechanism in our data transmission system
    - Noise-damaged packets can be discarded
    - Packet network usually has multiple interconnection paths for flexible routing due to hostile environment
- Reordered Deliver
    - Some networks prevent packets from getting unordered
    - Force sequence of packets to use same path or delay delivery until previous packets arrive
    - Like a postcard, segments need an ID so that if they get split up, reader can read them in order

- Fallacies of Distributed Computing
    - The network is reliable.
        - Network is unreliable due to switch failure, power loss, software failure, security issues
    - Latency is zero.
        - Latency is good on LAN, but not on WAN or internet; also it's more problematic than bandwidth
    - Bandwidth is infinite.
        - Not as bad, since bandwidth is growing; however, so does the amount of info we try to send, and there is still packet loss to account for
    - The network is secure.
        - Internet attacks are growing and many networks only have a perimeter as defense
    - Topology doesn't change.
        - Topology changes once network hits the field; clients also mean it's changing constantly
        - Don't depend on specific routes; try abstract physical network structure (ex: DNS instead of IP)
    - There is one administrator.
        - Usually multiple admins; they need to have monitoring tools and handle upgrades
        - Admins also constrain your options (like disk quota, privileges, etc.)
    - Transport cost is zero.

    - Application to transport level is not free; we have to marshal (serialize bit info) data into wires
    - Networks also cost money (routers, security, leasing bandwidth, operation/maintenance, etc.)
  - The network is homogeneous.
    - Usually they're based differently (Linux, Windows, etc.); not homogeneous at application level
    - Don't rely on standard technology, but use interoperable technologies

- Multiple-Site Atomicity: Distributed Two-Phase Commit
  - Transactions on sites separated by best-effort network have difficulty obtaining atomicity
    - Message used to coordinate transactions can be lost, delayed, duplicated
    - RPC assurances we learned before aren't sufficient to ensure atomicity here
  - Solution: use two-phase commit protocol with persistent sending, duplicate suppression, and single-site transactions to create a multiple-site transaction
    - Assume each single site can handle its own transactions to guarantee atomicity
    - Use a higher-layer transaction and distributed version of two-phase commit protocol
  - Example:
    - Superior creates higher-layer transaction so each worker performs a nested transaction
    - Worker checks for duplicates of form, then makes a (nested) transaction
    - Two-phase commit occurs when all workers respond and the superior sends them a new message
    - Worker then responds with a PREPARED (to commit) or an abort
    - If all workers are PREPARED, two-phase commit's phase-one is complete
    - If the superior commits, then phase-two starts
    - Workers are then COMMITTED, does some action, and exits
  - Two-phase commit of N worker sites uses 3N messages
  - Sometimes a fourth acknowledgement is used so coordinator can safely discard the outcome record
  - Presumed commit, a further refinement, can be used
    - Saves record storage space and message cost concerns
    - Assume commits exist (non-existence)
  - Two-phase commit doesn't solve all multiple-site atomicity problems
  - Dilemma of the two generals
    - If the superior sends PREPARE message but "crashes forever" before sending COMMIT or ABORT
    - The workers are left in PREPARED state
    - Workers will do their parts eventually
    - If the needed to do simultaneous work, that's the dilemma of the two generals
      - Back and forth message passing, but never sure when to start