CS 194: Lecture 28

Review, Part II

Location of exam: A1 HEARST ANNEX

1

Special Topics

- DHTs (mostly covered in project)
- Sensornets
- Robust protocols
- Resource allocation
- Incentives

2

Three Classes of DHT Applications

- Rendezvous: uses DHT to store current "location"
- Storage: uses DHT to store data objects
  - Case 1: only uses put/get interface
  - Case 2: data manipulated in an application-specific manner
- Routing: uses DHT to contact all DHT nodes along the path to the appropriate DHT "root".

3

Rendezvous Applications

- Each client (telephony, chat, etc.) periodically stores the IP address (and other metadata) describing where they can be contacted
- When A wants to contact B, it first does a get on B's key, and then contacts B directly
- This can handle:
  - IP mobility
  - Chat
  - Internet telephony
  - DNS
  - The Web!

4

Storage Applications

- File Systems
- Backup
- Archiving
- Electronic Mail
- Content Distribution Networks
- .....

5

CFS Uses Self-authentication

Immutable block: (Content-Hash Block)

key = CryptographicHash(value)

encourages data sharing!

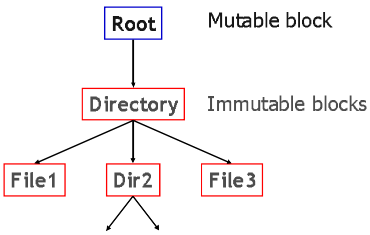
Mutable block: (Public-key Block)

key = K_{pub}

value = data + Sign[data]$_{K_{priv}}$

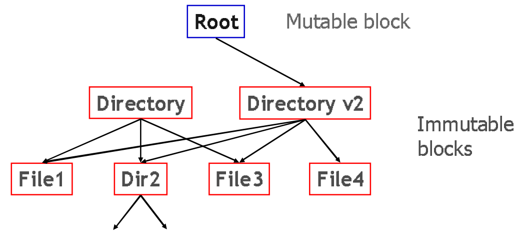
6

CFS



- This is a single-writer mutable data structure

Adding a File to a Directory



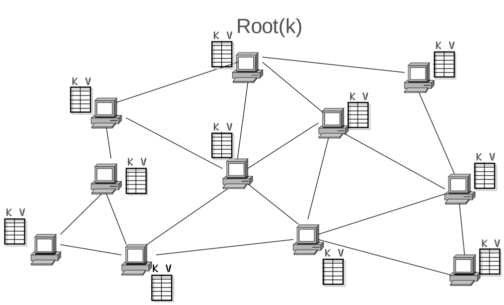
Usenet over a DHT

- Standard usenet: broadcast all articles to all N sites
- UsenetDHT: broadcast headers to all sites
  - store full articles in DHT (only once, not N times)
  - only transmit to site when requested
- Bandwidth/storage performance advantage function of:
  - header (H) to full (F) article ratio
  - fraction (R) of articles ever read at each site
  - total volume V of messages
- Bandwidth/Storage: roughly VNF vs V[NH + F + (N-1)RF]

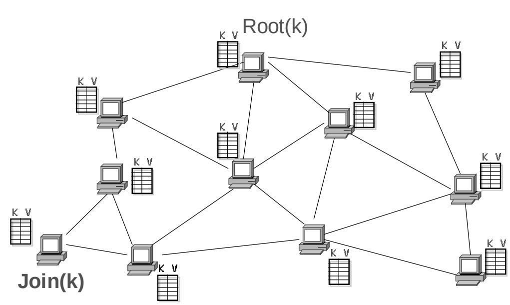
"Routing" Applications

- Examples:
  - Application-layer multicast
  - Video streaming
  - Event notification systems
  - ...
- Multicast:
  - Group is associated with key
  - "root" of group is node that owns key
  - Any node that wants to join sends message to root, leaving forwarding state along path
  - Message stops when it hits existing state for group
  - Data sent from root reaches all nodes

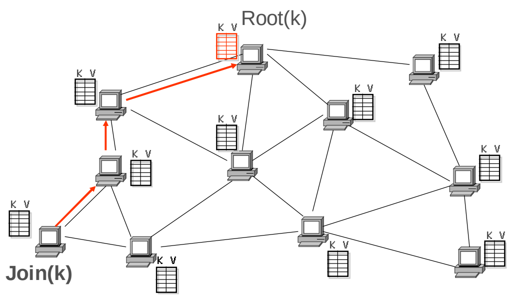
Multicast



Multicast Join

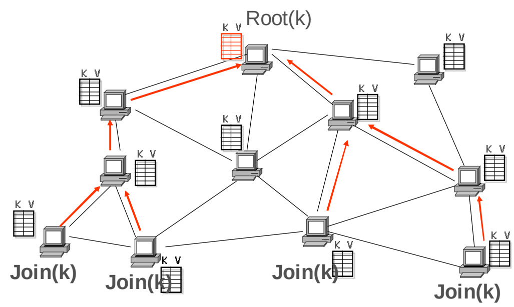


Multicast Join



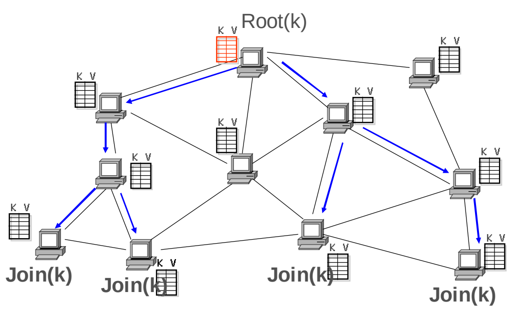
13

Multicast Join



14

Multicast Send



15

Four DHT Scenarios

- Donated distributed infrastructure
- Scalable enterprise applications
- Super-resilient applications
- “Tiny” applications

16

Library or Service

- Library: DHT code bundled into application
  - Runs on each node running application
  - Each application requires own routing infrastructure
  - Allows customization of interface
  - Very flexible, but much duplication
- Service: single DHT shared by applications
  - Requires common infrastructure
  - But eliminates duplicate routing systems
  - Harder to get, and much less flexible, but easier on each individual app

17

Why OpenDHT?

Consider FreeDB (the CD metadata database)

- Two options to implement large-scale FreeDB

1. Implement your own DHT:
   - Find 500 nodes you can use
   - Run DHT 24/7
   - Debug DHT problems when they occur
2. Use OpenDHT:
   - 58 lines of Perl

18

Two Challenges

- Resource allocation
- Beyond rendezvous

Resource Allocation

- Two resource allocation issues:
  - Leave enough room for future puts
  - Allocate evenly between users

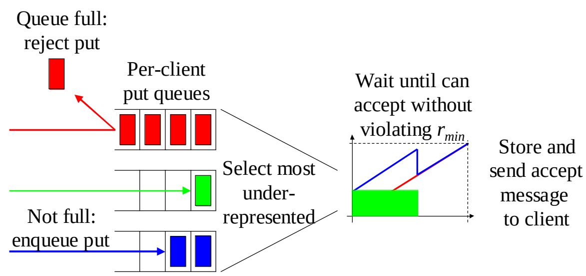
Storage Constraint Equation

- Rate reserved for future puts): r_{min}
- Disk capacity C
- Let S(t) be total number of current bytes that will still be on disk at time t
- A put of size b and TTL T can be accepted iff

$$S(t) + b + t \cdot r_{min} \leq C \text{ for } 0 \leq t \leq T$$

different notation than Sean's

Allocating Storage Fairly



The Big Decision: Definition of "most under-represented"

Using FQ-like Allocation Rule

- Equalize *rate* of commitments granted
  - Service granted to one client depends only on others putting "at same time"
- Mechanism inspired by Start-time Fair Queuing
  - Have virtual time, v(t)
  - Each put gets a start time S and finish time F
    - S = max[v(arrival), F(previous)]
    - v(t) = maximum start time of all accepted puts
    - F = S + B × T
  - Take client with smallest S at head of queue

Generality of Interface

Three classes of DHT interfaces:

- Routing: app-specific code at each hop
- Lookup: app-specific code at endpoint
- Storage: put/get

For a shared infrastructure that can't incorporate app-specific code, the only choice is put/get

- Limited flexibility
- Does convenience outweigh constraints?

Application-Specific Functionality

- How can you apply application-specific functionality to OpenDHT applications?
- Approach: use OpenDHT for routing, use external nodes nodes for application-specific processing
  - Application owner doesn't need to understand DHTs
  - Can write application assuming a lookup(key) operation just works
- Accomplished through a client library called ReDiR
  - takes application lookup(key) calls and returns with proper IP address (of external node) using put/get interface on OpenDHT
- Works for storage apps, and many routing apps can be converted to storage....

25

Sensornet Characteristics

- Energy constraints: communication more expensive tham computation
- Harsh conditions: radio connectivity variable
- Self-organizing: no manual configuration
- Data-centric (not node-centric): users want data, but don't know which node has the data....

26

Isn't the Internet Robust?

- Robustness was one of the Internet's original design goals
- Adopted failure-oriented design style:
  - Hosts responsible for error recovery
  - Critical state refreshed periodically
  - Failure assumed to be the common case
- Proof from experience: Internet has withstood some major outages with minimal service interruption
  - 9/11
  - Baltimore tunnel fire
  - etc.

27

Counterexamples

- Arpanet routing problem
- Frequent routing failures due to misconfiguration
- Vulnerability to congestion control misbehavior
- Vulnerability to SYN floods
- .....

28

General Lesson

- Most Internet protocols are design with (at most) two failure models in mind:
  - Participating nodes: fail-stop
  - Other nodes: malicious
    - Denial-of-service, spoofing, etc.
- They are usually vulnerable to participating nodes misbehaving:
  - Subverted nodes
  - Misconfigured nodes
  - Bug in software

29

Semantic vs Syntactic Failures

- Syntactic failures:
  - Node doesn't respond, message ill-formed, etc.
- Semantic failure:
  - Node responds with well-formed message, that is semantically incorrect
- Internet designed for syntactic failures, not semantic ones

30

Why Didn't Traditional Tools Work?

- Formal verification:
  - Verifies that <u>correct</u> protocol operation leads to the desired result
- Cryptographic authentication:
  - Verifies <u>who</u> is talking, but not <u>what</u> they say
- Fault-tolerance via consensus: (Byzantine techniques)
  - Requires that several nodes have enough information to do the required computation
  - In network routing, for instance, only the nodes at the end of a link know about its existence

31

Some Guidelines

1. Value conceptual simplicity
2. Minimize your dependencies
3. Verify when possible
4. Protect your resources
5. Limit scope of vulnerability
6. Expose Errors

- #3 and #4 pose the most difficult technical challenges

32

Lightweight Verification

- Checking for semantic correctness
- Not trying to guarantee correct behavior, merely trying to detect lying (even if after-the-fact) with some probability
- Mechanism must be lightweight enough to scale

33

Explicit Congestion Notification (ECN)

- Bit in IP header flipped when routers experience congestion
  - Replaces packet drops with explicit signaling of congestion
- Receiver returns this bit back to sender in TCP header
  - Keeps sending bit until sender returns CWR
  - CWR = congestion window reduced
- ECN advantages:
  - Doesn't require drops
  - No confusion between corruption losses and congestion losses

34

Problem

- ECN requires receiver to give information back to sender
- If receiver lies (doesn't return bit), then sender keeps increasing window
- Lying receiver gets more bandwidth than truthful ones or non-ECN-enabled ones

35

Robust Congestion Signaling (Ideal)

- Use bits in IP header to send two separate signals:
  - Congestion-bit: on or off
  - Nonce: large random number
- When congestion bit is set, nonce is erased
- Receiver must send back cumulative sum of nonces in ACK
- When congestion is signaled, receiver can't see nonce, so must guess about it
  - If many nonce bits, this is very unlikely

36

Robust Congestion Signaling (Real)

- Use ECN bits in IP header to send two separate signals:
  - Congestion-bit: on or off
  - Nonce: randomly 0 or 1
- When congestion bit is set, nonce is erased
- Receiver must send back cumulative sum of nonces in ACK
- When congestion is signaled, receiver can't see nonce, so must guess about it
  - Improbable it can continue to guess right

Protecting Resources

- Different contexts:
  - web servers: SYN cookies
  - routers: fair queueing
  - ....
- Will describe fair queueing
  - Can be used to support QoS
  - But was initially proposed to protect flows from each other

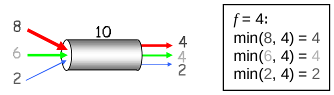
Fair Queueing (FQ)

- Implements max-min fairness: each flow receives $min(r_i, f)$ where
  - r_i – flow arrival rate
  - f – link fair rate (see next slide)
- Weighted Fair Queueing (WFQ) – associate a weight with each flow
  - We will ignore this for now

Fair Rate Computation: Example 1

- If link congested, compute f such that

$$\sum_i \min(r_i, f) = C$$



Another Example

- Rates: 1 3 5 7 8 9 15
- Capacity: 40
- Testing values of f:
  - f=5 has rates 1 3 5 5 5 5 5 so total is 29
  - f=6 has rates 1 3 5 6 6 6 6 so total is 33
  - f=7 has rates 1 3 5 7 7 7 7 so total is 37
  - f=8 has rates 1 3 5 7 8 8 8 so total is 40
  - f=9 has rates 1 3 5 7 8 9 9 so total is 42
  - .....
- Fair share is 8, so fair allocations are 1 3 5 7 8 8 8

Implementing Fair Queueing

- Fluid model: have queue for each flow, and serve each flow in a bit-by-bit round robin
- Packet model: send packets in the order that they would have finished in the fluid model

Implementing Fair Queueing

- Fluid model: have queue for each flow, and serve each flow in a bit-by-bit round robin
  - Let V(t) denote the "virtual time",which is the round number in progress at time t
  - $\partial V/\partial t = C/n(t)$ where n(t) is the number of flows with nonempty queues
- Packet model: send packets in the order that they would have finished in the fluid model
  - If you calculate finishing times in terms of V(t) rather than t, you don't have to change when new packets arrive

43

Finish Time Calculation

- Define
  F_i^k = virtual finishing time of packet *k* of flow *i*
  a_i^k = arrival time of packet *k* of flow *i*
  L_i^k = length of packet *k* of flow *i*
- The finishing time of packet *k+1* of flow *i* is

$$F_i^{k+1} = \max(\ V(a_i^{k+1}),\ F_i^k) + L_i^{k+1}$$

- Hard part: calculating virtual time V(t)

44

FQ Variations

- Use start or finish time of packet in service as the value of virtual time when new packet arrives
- Use start time rather than finish time for ordering packets

45

Living in the Brave New World....

- Most modern Internet-scale distributed systems involve independent users
  - Web browsing, DNS, etc.
- Users typically care only about their own service
- Users may not want to use mandated algorithms, but instead will act to maximize their own welfare
- What happens when users behave selfishly, and what can you do about it?

46

Selfish Behavior in Congestion Control

- Users adjust rate r_i to maximize utility U_i which is function of both bandwidth and delay d_i
- Delay d_i is function of all sending rates r_j
  - This is how users affect each other
  - Function for delays depends on packet scheduling algorithm
- A Nash equilibrium is a vector of r's such that no user can increase their U_i by unilaterally changing r_i
  - First order condition: $\partial U_i/\partial r_i = 0$

47

Simple Poisson Model with FIFO Queue

- In Poisson model with FIFO queues (and link speed 1):

$$d_i = 1/(1-r_{tot})$$

- Assume $U_i = r_i/d_i$
- Then $U_i = r_i(1 - r_{tot})$

48

Three Questions

- What is the result of selfish behavior? (Nash equilibrium)
  - $r_i = 1/(n+1)$ where n is number of users
  - $U_{tot} = (n+1)^{-2}$
- What is the socially optimal level of usage?
  - $r_i = 1/2n$
  - $U_{tot} = 1/4$
- What packet scheduling algorithm would have a Nash equilibrium with the socially optimal usage?
  - Fair queueing

49

Designing for Selfishness

- Assume every user (provider) cares only about their own performance (profit)
- Give each user a set of actions
- Design a "mechanism" that maps action vectors into a system-wide outcome
  - Mechanism design
- Choose a mechanism so that user selfishness leads to socially desirable outcome
  - Nash equilibrium, or other equilibrium concepts

50

Strategyproof Mechanism

- Clients have no incentive to lie
- They are asked to reveal their utility
- And no matter what other clients do, they are at least as well off telling the truth
- Examples:
  - second-price auction
  - VCG mechanisms

51