Announcements

- My office hours this week: not today, but Th 10-11
- HW #4 will be out shortly....

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

Challenges

- Interface
- Security (securing interface)
- Resource allocation
- Beyond rendezvous
  - ReDiR
  - Range queries

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?

Basic Interface

- put(k,v,t): write (k,v) for TTL t
- Why TTL? No garbage collection...
Security Worries

- Modifying: changing data stored by someone else
- Squatting: getting key first and not allowing others to use it
- Drowning: storing many values at certain key, so that client can’t get data they want without sifting through huge pile

Put/Get Interface w/Authentication

- put-unauth(k,v,t): append-only (no remove), no auth
  - no modifying, no squatting, but drowning
  - for easy use
- put-immutable(k,v,t): k=H(v)
  - no modifying, squatting or drowning, but no flexibility in key choice
- put-auth(k,v,t;n,K,s): removable, authenticated
  - n is sequence number
  - Public key K
  - s=H(k,v,n) signed with private key
  - get-auth(k,H(K)) retrieves only entries with that public key
  - no modifying, squatting, or drowning, and flexibility of key choice

Resource Allocation

- Consider a put of size B and TTL T
- The resource consumed by that put is BT
- Resource allocation strategy:
  - At any one time, allocate resources to keep instantaneous rate of resource allocation even
  - Leave enough room for future puts

Storage Constraint Equation

- Baseline rate: \( r_{\text{min}} \)
- 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 \( b \) can be accepted iff
  \[
  S(t)+b + t·r_{\text{min}} \leq C \quad \text{for } 0 \leq t \leq T
  \]
  different notation than Sean’s

Does it Fit?

- Sum must be < max capacity
- Reserved for future puts. Slope = \( r_{\text{min}} \)
- Candidate put

Does it Fit?

- Violation!
**Key point:** slopes of all lines the same at all times!

**Beyond Rendezvous**
- More complicated queries
- Application-specific processing
  *without changing interface*

**Range Queries**
- Useful for many applications like databases and publish-subscribe systems
  - but not directly supported by a DHT
- Existing approaches require changes to DHT implementation
  - Skip Graphs [Shah et al, SODA 2003]
  - Load Balancing [Karger et al, SPAA 2004]
- How can range queries be supported on top of a generic put/get interface?

**Prefix Hash Tree**
- Trie data structure
- Data items are stored at leaf nodes with matching prefixes
- Logical structure mapped to DHT nodes by hashing prefix labels
- \( \text{HASH}(110^*) \Rightarrow \text{key} \)

**Insertion / Deletion**
- Leaf nodes have a capacity constraint \( B \)
- Insertion could result in the splitting of a leaf node
  - For example, \( \text{Insert}(110011) \) (\( B = 4 \))
- Conversely, deletion could result in the merging of two sibling leaf nodes

**Range Queries**
- Parallel traversal of smallest sub-trie completely covering range query
- Root of this sub-trie can be directly accessed instead of top-down traversal
  - For example, \( \text{Query}(9,11) \)
  - \( \text{HASH}(0010^*) \)
### PHT Properties

- Efficient: Operations are doubly logarithmic in domain size due to direct access
- Load Balanced: Top-down traversal is not required, reducing load in upper levels of the trie
- Fault-tolerant: Node failure does not affect the ability to access data available at other nodes

### Application-Specific Functionality

- How can you apply application-specific functionality to OpenDHT applications?
- Approach: use OpenDHT for routing, use external 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

### ReDiR

- Each set of app-specific nodes is assigned a namespace
- Each node has a key in that namespace
- ReDiR supports:
  - `join(namespace, key, ip)`
  - `lookup(namespace, key)`

### ReDiR “Homes”

<table>
<thead>
<tr>
<th>Level</th>
<th>Keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>L0</td>
<td><img src="image" alt="L0 Diagram" /></td>
</tr>
<tr>
<td>L1</td>
<td><img src="image" alt="L1 Diagram" /></td>
</tr>
<tr>
<td>L2</td>
<td><img src="image" alt="L2 Diagram" /></td>
</tr>
</tbody>
</table>

### ReDiR Join Rule

- Store your (key, IP) at the namespace key in the lowest partition and continue to higher levels, stopping only after you’ve stored at a level where you aren’t the lowest key
  - There’s a more sophisticated method using both highest or lower
- When you are the highest or lowest, “kick out” the previous lowest
  - Not necessary with soft state, but for the sake of the presentation
**ReDiR Lookup Rule**

- Keep going up levels until you find a successor
- You are guaranteed that that’s your successor (aside from the lowest level)
  - Why?

**ReDiR Join**

- Join cost:
  - Worst case: $O(\log n)$ puts and gets
  - Average case: $O(1)$ puts and gets

---

**ReDiR Lookup**

- Lookup cost:
  - Worst case: $O(\log n)$ gets
  - Average case: $O(1)$ gets
All Three Words Count

- Wireless: no lines (network or power)
- Sensors: tied to real world
- Networks: not just a single hop

Networked Sensing Enabler

- Small (coin, matchbox sized) nodes with
  - Processor
    - 8-bit processors to x86 class processors
  - Memory
    - Kbytes – Mbytes range
  - Radio
    - 20-100 Kbps initially
- Battery powered
- Built-in sensors!

Application Areas

- Seismic Structure response
- Marine Microorganisms
- Structural Condition Assessment
- Ecosystems, Biocomplexity
- Contaminant Transport

Seismic Structure Response

- Interaction between ground motions and structure/foundation response not well understood.
- Current seismic networks not spatially dense enough
  - to monitor structure deformation in response to ground motion.
  - to sample wavefield without spatial aliasing.
Condition Assessment

- Longer-term
- Challenges:
  - Detection of damage (cracks) in structures
  - Analysis of stress histories for damage prediction
  - Applicable not just to buildings
  - Bridges, aircraft

Contaminant Transport

- Industrial pollutant disposal can be enormously damaging to the environment
- Marine contaminants
  - Groundwater contaminants
- Study of contaminant transport involves
  - Understanding the physical (e.g., structural), chemical, interaction with sediments, and biological (e.g., plants and marine life) aspects of contamination
  - Modeling their transports
  - Munro field?
  - Fine-grain sensing can help

Lab-Scale Experiments

- Use surrogates (e.g., heat transfer) to study contaminant transport
- Testbed
  - Tank with heat source and embedded thermistors
  - Measure and model heat flow

Field-Level Experiments

- Nitrates in groundwater
- Application
  - Wastewater used for irrigating alfalfa
  - Wastewater has nitrates, nutrients for alfalfa
  - Over-irrigation can lead to nitrates in ground-water
  - Need monitoring system, wells can be expensive
  - Pilot study of sensor network to monitoring nitrate levels
**Marine Micro-organism Monitoring**

- Algal Blooms (red, brown, green tides) impact
- Human life
- Industries (fisheries and tourism)
- Causes poorly understood, mostly because
  - Measurement of these phenomena can be complex and time consuming
- Sensor networks can help
  - Measure, predict, mitigate

**Lab-Scale Experimentation**

- Build a tank testbed in which to study the factors that affect micro-organism growth
- Actuation is a central part of this
  - Can’t expect to deploy at density we need
  - Mobile sensors can help sample at high frequency
- Initial study:
  - thermocline detection

**Ecosystem Monitoring**

- Remote sensing can enable global assessment of ecosystem
- But, ecosystem evolution is often decided by local variations
  - Development of canopy, nesting patterns often decided by small local variations in temperature
- In-situ networking can help us understand some of these processes

**James Reserve**

- Clustered architecture
- Weather-resistant housing design
- Sensors: Light, temperature, pressure, humidity

**Great Duck Island**

- Study nesting behavior of Leach’s storm petrels
- Clustered architecture:
  - 802.11 backbone
  - multihop sensor cluster

**Challenges**
**Energy**

- Nodes are untethered, must rely on batteries
- Network lifetime now becomes a performance metric

**Communication is Expensive**

- The Communication/Computation Tradeoff
  - Received power drops off as the fourth power of distance
  - $10 \text{ m}: 5000 \text{ ops/transmitted bit}$
  - $100 \text{ m}: 50,000,000 \text{ ops/transmitted bit}$
- Implications
  - Avoid communication over long distances
  - Cannot assume global knowledge, or centralized solutions
  - Can leverage data processing/aggregation inside the network

**Can’t Ignore Physical World**

- Can’t hide in the machine room!
- Conditions variable and sometimes challenging

**No Configuration**

- System must be self-organizing

**Generality vs Specificity**

- Internet: single infrastructure capable of supporting many apps
- Sensornets: each deployment will have limited number of users and limited number of apps
- But basic technology should be general

**Components of Infrastructure**

- Collaborative Event Processing
  - Querying, Triggering
  - Data-centric Routing, Aggregation and Compression, Data-centric Storage
  - Collaborative Signal Processing
  - Localization, Time Synchronization, Medium Access, Calibration
  - Operating Systems
  - Processor Platforms, Radio, Sensors
  - Security
Components of Infrastructure

- Processor Platforms
- Radios
- Sensors

How to Access Data

- Sensors are useless if one can’t access the required data
- Can’t send it all to external storage:
  - limited bandwidth
  - limited energy
- How can you get only the data you need?

Name the Data!

- Don’t know which nodes have data
- Don’t think in terms of point-to-point protocols (as in Internet)
- Think in terms of data

Ask for Data!

- Send out requests for data by name
- If nodes have the relevant data, they respond

Three Communication Patterns

- Data-centric routing
- Tree-based aggregation/collection
- Data-centric storage
Diffusion messages

- Messages are sets of attribute-value pairs
- Message types
  - Interest (from sinks)
  - Data (from sources)
  - Control (reinforcement)

Diffusion Routing: Two phase pull

- Flood interest
- Flood data in response
- Sink reinforces
- Forward data along the reinforced paths

Components of Infrastructure

- Collaborative Event Processing
- Querying, Triggering
- Data-centric Routing
- Aggregation and Compression
- Data-centric Storage
- Collaborative Signal Processing
- Localization
- Time Synchronization
- Medium Access
- Calibration
- Operating Systems
- Processor Platforms
- Radius
- Sensors

TinyDB/TAG

- Set up spanning tree from source
  - not as easy as it sounds!
- Flood query down tree
- Data sent back along reverse path
- Apply various aggregation operators

So Far....

- Data access methods are flood-response
- Good for long-lived queries
- What about one-shot queries?
Sensor Networks
Peer-to-peer Systems
Database Systems

Distributed Data Structures

Efficient sensor network querying and triggering

Data-Centric Storage

• Simple primitives
• Efficient information retrieval in sensor networks
• Challenges
  • Geographic routing
  • Robustness to failure

An Instance of Data-Centric Storage

• Geographic Hash Tables (GHTs)
• *Hash* the name of the data to a geographic location
• Store data at the node closest to that locations
  • Use a geographic routing protocol (e.g., GPSR)
• Can retrieve data the same way

GPSR Internals

• Nodes are named by their geographic locations
• Greedy routing as far as possible
• Perimeter routing when greedy fails
  • Fundamentals: Right-hand rule
  • Planarization removes crossing links
• Recover to greedy whenever possible
• Drop a packet when it is going to enter a perimeter along the same route again!

GHT = GPSR + DHT

• Answer queries for exact matched data, just like any other hash tables.

More Sophisticated Queries

• Spatio-temporal aggregates
• Multi-dimensional range queries
• Approach
  • Use hashing and spatial decomposition
• Data-centric storage not yet deployed

Current UCB Project

• Defining a sensornet architecture (SNA)
Today's SensorNet Landscape

- **App**: Hood EnviroTrack Regions TinyDB
- **Transport**: TTDQ SPIN
- **Routing**: CCSR AQD VSR DSBV DBR Resynch
- **Scheduling**: CGS R ARA GSR GR S P A GPR
- **Topology**: PC SPAN GAF FPS
- **Link**: Pico WiseMAC WisemAC
- **PHY**: RadioMetrix RFM CC1000 Bluetooth 802.15.4 nordic

Not Just a Messy Picture

- Many components developed in isolation
  - Differing assumptions about overall structure...
- Some vertically integrated systems
  - Not much interoperability between them

Our Conjecture

- The biggest impediment to progress is *not* any single technical challenge
- It is the lack of an overall architecture that would increase composability and interoperability

The "Internet Architecture"

- **Goal 1**: universal connectivity
  - Problem: diversity of network technologies
  - Solution: universal and opaque IP protocol
- **Goal 2**: application flexibility
  - Problem: application-aware networks limit flexibility (*because network is static*)
  - Solution: end-to-end principle
  - *Put app. functionality in hosts, not network*
  - *Hosts under our control, and can be changed*

Internet Architecture

- **Goal 1**: universal connectivity
  - Problem: diversity of network technologies
  - Solution: universal and opaque IP protocol
- **Goal 2**: application flexibility
  - Problem: application-aware networks limit flexibility (*because network is static*)
  - Solution: end-to-end principle
  - *Put app. functionality in hosts, not network*
  - *Hosts under our control, and can be changed*

The Internet Architecture

- Shields applications from hardware diversity
- Shields network from application diversity
- Speeds development and deployment of both
### How Do Sensornets Differ?

- Apps: data-centric, not host-centric
  - Endpoints not explicitly addressed by apps
  
  ⇒ Can’t organize around end-to-end abstractions

- Goal: code portability and reuse
  - Not universal connectivity
  - Not application flexibility for static network
  
  ⇒ End-to-end principle not (automatically) applicable

  *In-network processing is often much more efficient*

### Where is the Narrow Waist?

- Internet: best-effort end-to-end packet delivery (IP)

- Sensornets: best-effort single-hop broadcast (SP)?

- Expressive abstraction of a universal link layer
  - Single abstraction for all lower layer technologies

- Abstraction should allow higher-layers to optimize without knowing the underlying technology

### How Do Sensornets Differ?

- Constraints: scarce resources (energy)

- Internet: opaque layers as easy abstraction
  - Willing to tolerate significant efficiency loss

- Sensornets: need *translucent* layers
  - Hide details of hardware underneath
  - But expose abstractions for control

- Goal: trade (small) efficiency loss for (much) less reprogramming

### The Sensornet "Hourglass"

![Diagram of Sensornet "Hourglass"](image-url)