

---

# TierStore

## Delay Tolerant Networking

Michael Demmer

Joint work with Bowei Du, Eric Brewer, Kevin Fall,  
Sushant Jain, Melissa Ho, Rabin Patra

# Potential Applications

---

- Many information-based applications may be beneficial to developing regions
  - Educational content distribution
  - Sensor data collection (water quality, disease)
  - Weather forecast distribution
  - Commodity price dissemination
  - Social and community network building
  - Logistical coordination (e.g. disaster recovery)
  - Internet applications such as e-mail and web

# Constraints

---

- Lack of infrastructure
  - Limited network connectivity
  - Poor quality electric power
  - Shortage of networking experts
- Economics
  - Limits application of “first world” technologies
  - Both equipment and maintenance costs

# Overall Goal

---

- Storage and data distribution technologies for information based applications in developing regions
- Requirements:
  - Support for many different applications
  - Low (amortized) cost
  - Amenable to limited infrastructure deployments
  - Capable of unattended operation

# Case for Delay Tolerant Networking

---

- 'Always on' networking can be hard and/or costly
  - High installation and operational costs
  - Poor connectivity reflected in poor application performance
- Even if 'always on' networking is deployed, many deployments are *intermittent*
  - Inconsistent power, weather disruptions, user errors
- Deployments may benefit from a blend of network connectivity technologies

# DTN Goals

---

- Functional networking in the presence of high loss / delay / error / disconnection
  - Decent performance for low loss/delay/errors
- Interoperable with 'radically heterogeneous' networking technologies
  - Ranging from high-bandwidth/delay deep space radio to small burst SMS transmissions
- Novel quality of service and reliability mechanisms
  - Traditional Internet solutions often inappropriate

# DTN Architecture

---

- Store-and-forward, message delivery service
  - Variable length “bundles” (not fixed length packets)
  - Intermediate nodes buffer across disruptions
- “Convergence layer” abstraction to adapt to different transport layers
  - e.g. TCP, Bluetooth, USB key “sneakernet”, Data Mule, ...
- “Postal service” service classes and options
  - Bulk vs. normal vs. expedited
  - Options for return receipt, hop by hop acks
  - Custody transfer based reliability

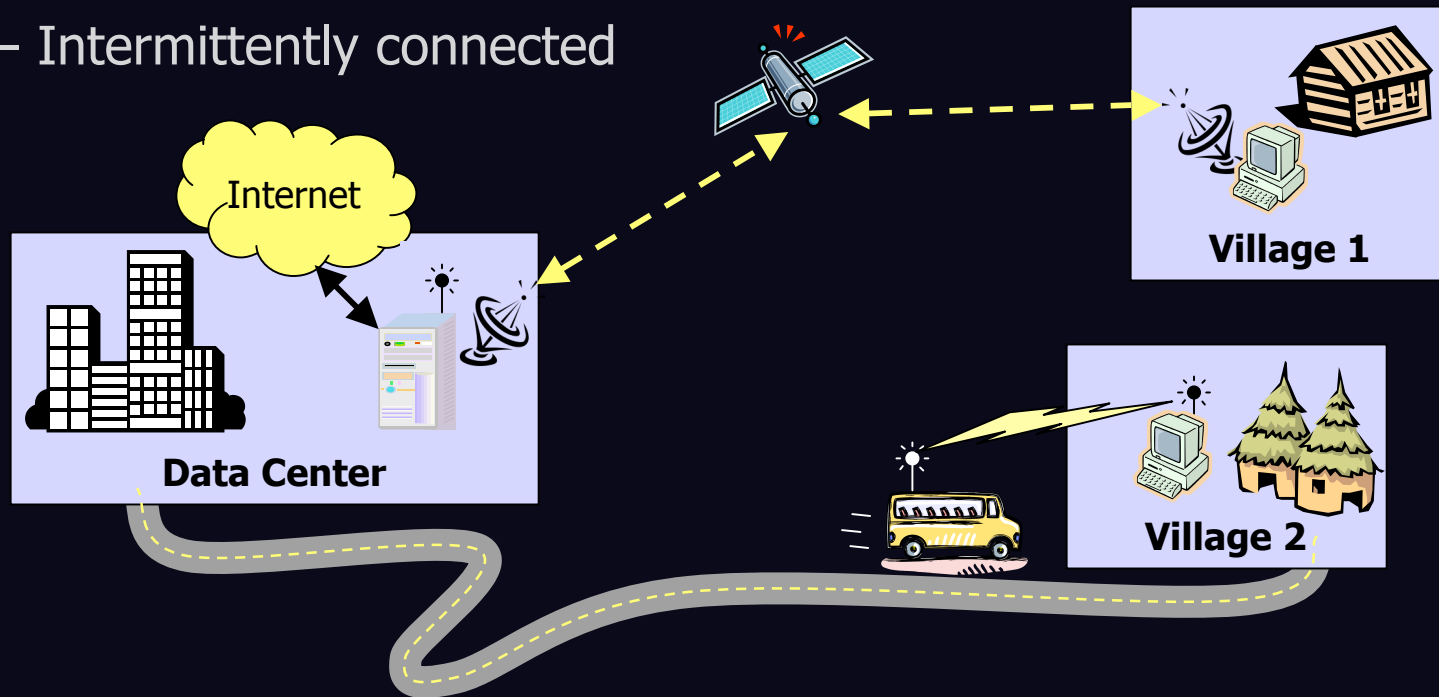
# TierStore Motivations

---

- Applications need long-term storage
  - DTN by itself is not enough
- Similar data flow for many applications
  - Basic distribution and/or collection maybe sufficient
- A common platform can ease application development and improve robustness
  - Avoids reinventing the wheel
  - Framework can be optimized

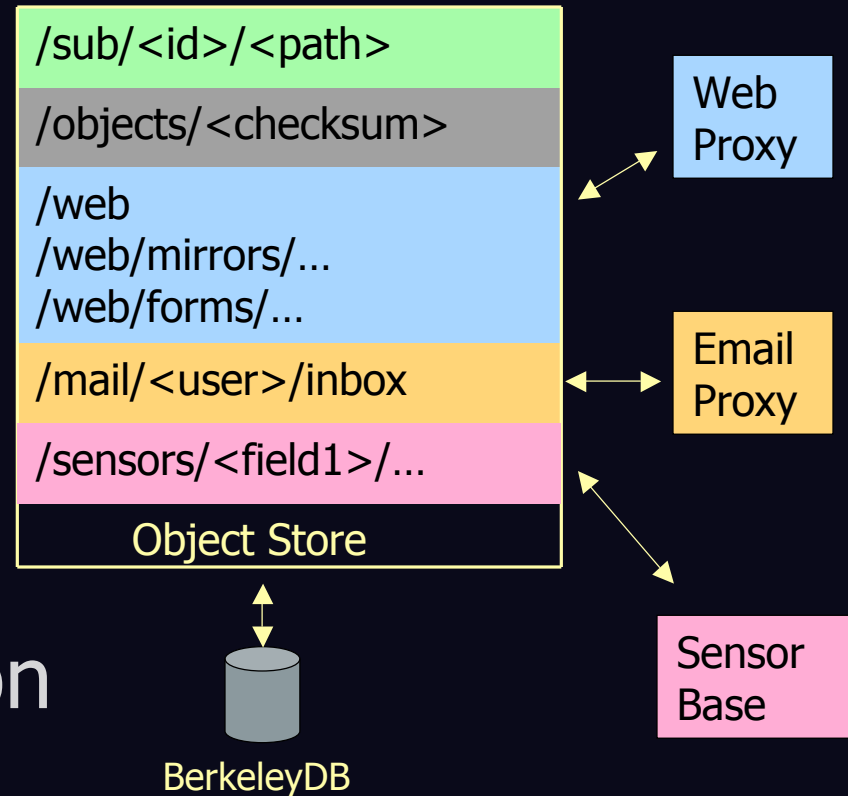
# “Tier”ed Deployment Architecture

- Data Center in major cities
  - Well connected, well powered, amortized cost
- Proxies / devices in villages
  - Commodity class PCs or PDAs (can be flaky)
  - Intermittently connected



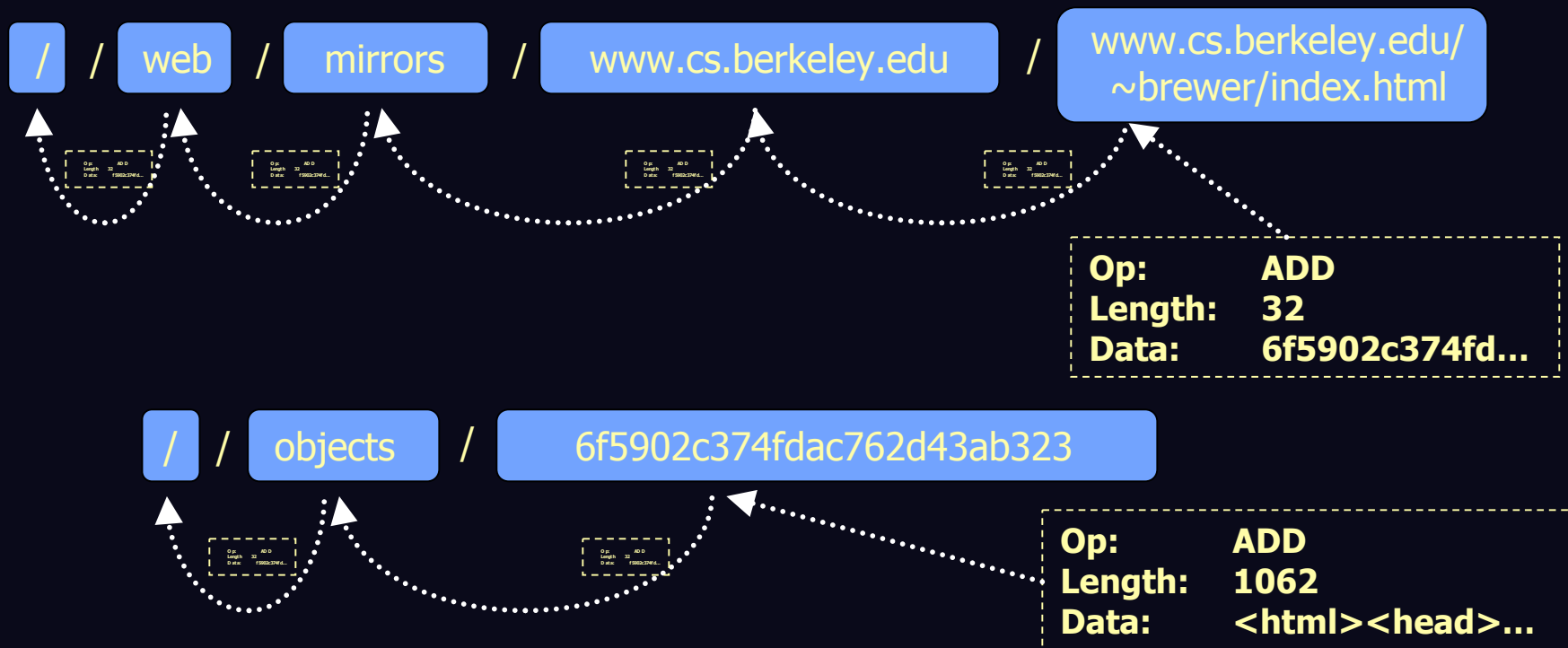
# TierStore Storage Model

- Partitioned Namespace
  - Applications have control over data organization
- Hierarchical layout
- Typed elements
- Fine-grained replication
  - Any path in the tree



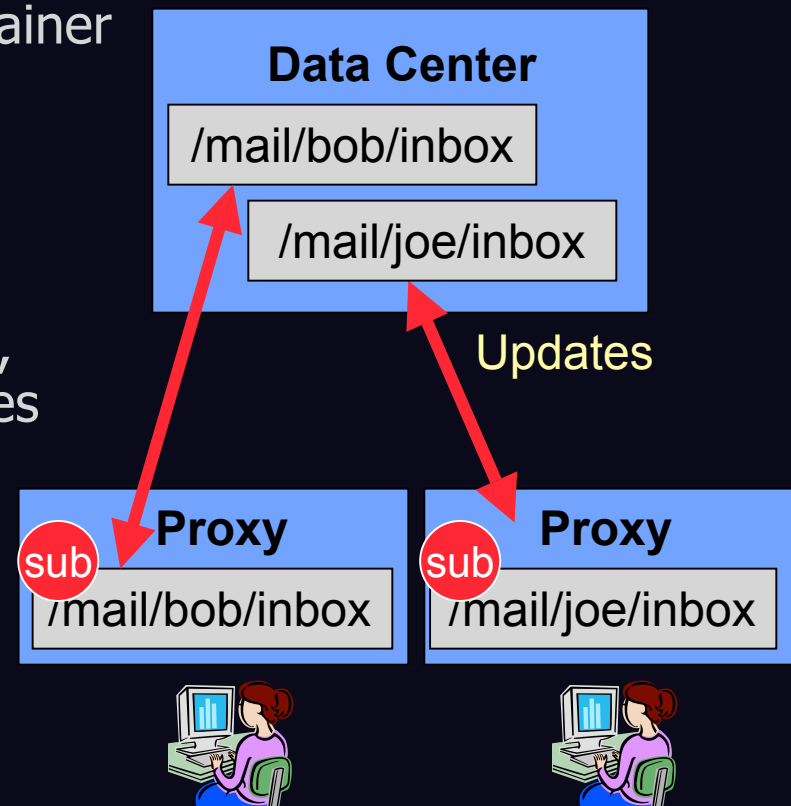
# Updates

- Update: encapsulation of a modification
  - Notification passed up from the modified object
  - Hooks can be attached at every container



# Subscriptions

- Replication mechanism for arbitrary paths
  - Downstream nodes register interest in a path
  - Upstream node attaches to a container and forwards updates over DTN
- Common abstraction layer
  - Handles lost or reordered updates, timeouts, inter-object dependencies
  - Network hierarchy can be exploited for multicast benefits



# Subscription Management

---

- Must inform the Data Center of changes to downstream subscription interests
- Trick: use the object store and subscription system for this as well
  - Provides durability and coordinated delivery
  - Objects in /sub/<node>/<path> for downstream interests
- Default subscription for bootstrap problem
  - All nodes boot with a “meta-subscription” on /sub to replicate changes to immediate upstream parent
  - Interior nodes coalesce subscriptions for multicast

# Conflicts

---

- All distributed storage systems must worry about conflicting operations
  - Intermittency makes this even harder
- How to deal with conflicts?
  - Subscription granularity
  - Conflict-free naming
  - Application specific handlers



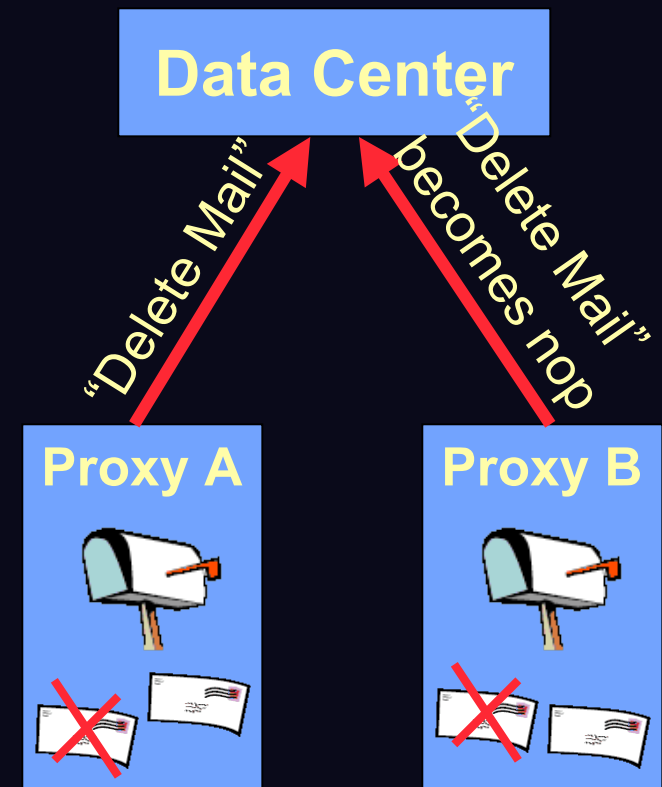
# Conflict Avoidance

---

- Subscription Granularity
  - Since a subscription can be placed on any container, can limit false conflicts
  - E.g. no dependencies between updates to web cache and updates to an email inbox
- Conflict free naming
  - Use unique node identifier and timestamp to ensure that operations won't conflict
  - e.g. /web/forms/node-100/1105019697.401902

# Conflict Management

- Not all operations can be made conflict-free
- Example:
  - Alice modifies her inbox in two different locations.
  - How are these two versions of the inbox reconciled?
- Typed objects enable app-specific hooks for conflict resolution
  - Example: A delete of a nonexistent message can always be ignored



# Conflict Management Alternative

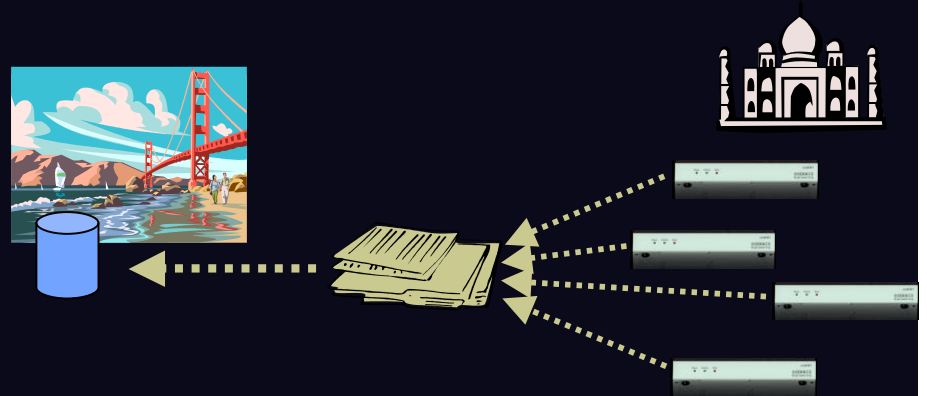
---

- App-specific hooks can be cumbersome
  - Basically an ad-hoc approach – hard to reason about and ensure that all cases are covered
- Instead, handle conflicts at the Data Center
  - DC is a natural serialization point, always up-to-date
  - Proxies maintain log of intended operations to be either accepted or rejected by the data center
  - All subscriptions are by definition one-way and ordered
  - Operations would include dependencies by name
  - Apps could use both committed and pending operations for user interface (ala Bayou)

# App 1: Data collection

---

- Collect log files from wireless routers in India
  - Simple one-way replication from router nodes to a “data center” in Berkeley
  - Deployed last week



- Future work: generic data collection service
  - Data center would publish requests by inserting XML specification objects
  - Downstream proxies notice requests, gather samples that are in turn replicated back to the data center

## App 2: Web Cache

---

- Disconnection tolerant web proxy cache
  - “Interesting” sites periodically crawled at the Data Center
  - Downstream proxies register interest in some sites
  - Real world apps: e-government, educational content
- Forms handled asynchronously
  - Filled out by user, propagated to DC via subscription
  - DC submits the form, results returned via subscription
  - User can then later return and use a token to retrieve the form results

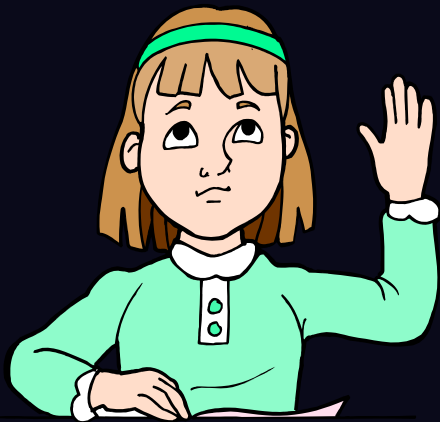
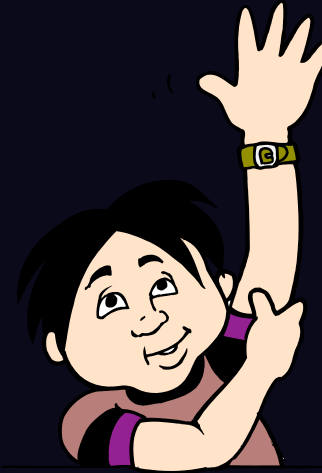
## App 3: E-mail

---

- Reimplementation of E-Mail4B (CS262A class project) using TierStore
  - Complexities found when implementing the first project served as the motivation for this platform
  - Several mail-specific protocols (message delivery, cache consistency, registrations) handled via subscriptions
- Use application type system for conflicts
  - Mailbox containers avoid conflicts since they are inherently unordered
  - As in Bayou, distinction between local objects and committed objects are exposed to user via UI

# Questions

---



# Optimization: Indirect Objects

---

- Observation: Many apps have duplicate data
  - Examples: Email messages sent to many people, different URLs for the same web page or image, etc.
  - Inefficient in storage and communication costs
- Solution: Indirect Objects
  - Shared read-only data cache under /objects/<checksum>
  - “Smart pointer” implementation to hide complexities, similar to symbolic links in a file system
  - Replication automatically bundles referenced objects along with other updates

# Typed Data Objects

---

- Applications define types for data
  - Unlike traditional binary blobs
  - E.g. mail message has fields for sender, recipient, flags, subject and message body
- Use a C++ class representation
  - Serialization layer to marshal / unmarshal object bits and reflect to the backing store
- Type-specific hooks for optimizations
  - Applications can override default behavior
  - Used for conflict resolution

# Future Optimizations

---

- Type specific diff / compression
  - Improve over the wire efficiency
- Integrate with LLADD storage backend
  - ARIES-inspired transactional backing store
  - Tight integration to eliminate unnecessary in-memory copy of persistent data