Towards a Common API for Structured Peer-to-Peer Overlays

Frank Dabek, Ben Zhao, Peter Druschel,
John Kubiatowicz, Ion Stoica

Presented for Cs294-4 by
Benjamin Poon

Outline

- Background
- Motivation
- API Specification
- Example Usage
- Example Implementation

Outline

- Background
- Motivation
- API Specification
- Example Usage
- Example Implementation

Background (1/2)

- Many applications for peer-to-peer
  - Information storage and retrieval
  - Group communication
- Applications built on decentralized (scalable, self-organizing) overlay systems
  - Tapestry, Pastry, Chord, CAN

Background (2/2)

- Distributed Hash Tables (DHT)
  - Provides traditional hashtable functionality
    - Stores key→value mapping
- Group anycast/multicast (CAST)
  - Scalable group communication
- Decentralized Object Location and Routing (DOLR)
  - Decentralized directory service for objects
    1) Objects (endpoints) placed anywhere within system
    2) Applications announce presence of endpoint
    3) Messages are routed to nearest available endpoint
  - Why isn't DOLR implemented on DHT?

Aside: DOLR on DHT

- *Towards a Common API for Structured . . .* (this paper):
  - "This is not possible because DOLR routes messages to the nearest available endpoint—providing a locality property not supported by DHTs"
- *Structured Peer-to-peer Overlays Need Application-Driven Benchmarks* - Rhea, Roscoe, Kubi (IPTPS '03):
  - "While at a functional level, a DOLR may be implemented by a DHT and vice-versa, we show in the results section of this paper that there are performance consequences of doing so."

Outline

- Background
- Motivation
- API Specification
- Example Usage
- Example Implementation

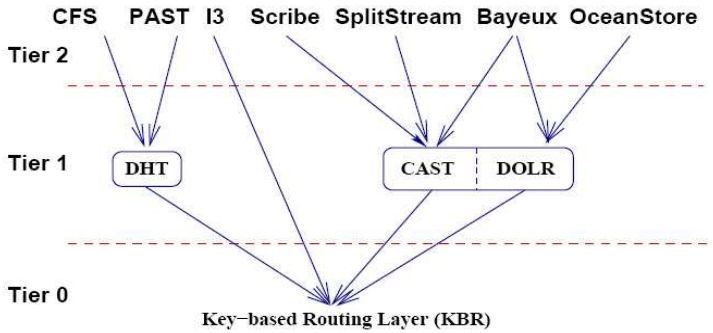
Motivation

- Overlay systems have commonalities
  - If commonalities are found:
    a) Independent innovation will be made easier
    b) Experimental comparisons easier to make
    c) Adoption of overlay systems will accelerate
- Describe commonalities in API
  - API described by paper expressive enough to implement all known applications built so far on CAN, Chord, Pastry, Tapestry

Outline

- Background
- Motivation
- API Specification
- Example Usage
- Example Implementation

API Specification



Goal: standardize an API for Tier 0

API Specification – Tier 1 Interfaces

DHT	DOLR	CAST
put(key, data)	*publish(objectId)*	*join(groupId)*
remove(key)	*unpublish(objectId)*	*leave(groupId)*
value = get(key)	*sendToObj(msg, objectId, [n])*	*multicast(msg, gId)* *anycast(msg, gId)*

API Specification – Vocabulary

- Node: instance of a participant in overlay
- ID space: n-bit integers (here, n = 160)
- Node ID: integer from the ID space assigned to a node uniformly at random
- Node handle: [transport address, nodeId]
- Key: integer from the ID space assigned to application-specific objects
- Root: a live node to which the overlay deterministically maps a given key
- Routing table: list of [node ID, IP address] tuples representing overlay links maintained locally by a node
- Msg: application data of arbitrary length
- R-root of key K: the live node that would be the root of K given R - 1 previous root failures

API Specification – Overview

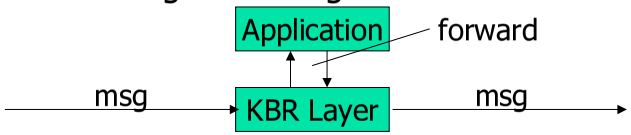
- Three types of calls:
  1. Main routing call
     - route()
  2. Supplementary upcalls
     - forward(), deliver()
     - Allow applications to customize routing behavior
  3. Routing state access calls
     - local_lookup(), neighborSet(), replicaSet(), range()
     - All local: no communication with other nodes
     - Applications access routing state to obtain nodes to be used in main routing call or supplementary upcalls

API Specification – Main Routing Call

- route(key *key*, msg *msg*, [nodeHandle *hint*])
  - Main routing call that routes *msg* to node currently responsible for *key*
  - Optional argument *hint* specifies a node that should be used as first hop
  - Provides best-effort service

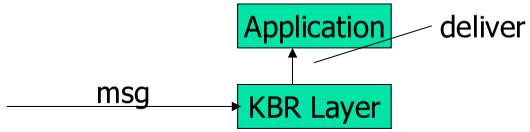
API Specification – Supplementary Upcalls (1/2)

- forward(&key *key*, &msg *msg*, &nodehandle *nextHopNode*)
  - Informs application that *msg* is about to be forwarded to *nextHopNode*
  - Invoked at all nodes that forward the message including source and root nodes



API Specification – Supplementary Upcalls (2/2)

- deliver(key *key*, msg *msg*)
  - Delivers *msg* to application at *key*'s root



API Specification – Routing State Access Calls (1/3)

- nodehandle[] = local_lookup(key *key*, int *num*, boolean *safe*)
  - Produces an unordered list of *num* suitable next hop nodes given a route towards *key*
  - Safe feature
    - If *safe*, fraction of faulty nodes in list <= fraction of faulty nodes in overlay
    - If !*safe*, list may be chosen to optimize performance (at expense of higher fraction of faulty nodes)
    - Allows support for overlays with Byzantine failures
      - Fail-stop behavior overlays may ignore this feature

API Specification – Routing State Access Calls (2/3)

- nodehandle[] = neighborSet(int *num*)
  - Produces an ordered list of *num* neighbors
- nodehandle[] = replicaSet(key *key,* int *max_rank*)
  - Produces an ordered list of 1..*max_rank*-root nodes on which replicas of the object associated with *key* can be stored
- update(nodehandle *node*, rank *rank*, key *lkey*, key *rkey*)
  - Informs the application that *node* has either joined or left the local node's neighborSet

API Specification – Routing State Access Calls (3/3)

- range(nodehandle *node*, int *rank*, key *lkey*, key *rkey*)
  - Provides information about keys in range [*lkey, rkey*] for which *node* is a *rank*-root
  - If overlay has disjoint ranges of keys, the range in which *lkey* resides is the range examined

Outline

- Background
- Motivation
- API Specification
- Example Usage
- Example Implementation

Example Usage – Distributed Hash Table (DHT)

- Interface Review
  - put(key, value)
  - value = get(key)
- Usage
  - Put
    - Source of request, S, sends route(key, [PUT, value, S])
    - Root receives deliver upcall and stores (key, value) pair in local storage
  - Get
    - Client of request, C, sends route(key, [GET, C])
    - Root R receives deliver upcall and sends route(NULL, [value, R], C)
- How to use API for caching DHT?

Example Usage – Caching DHT

- Put
  - Source of request, S, sends route(key, [PUT, value, S])
  - When a node receives a forward upcall, that node also stores (key, value) pair in local storage
  - Root receives deliver upcall and stores (key, value) pair in local storage
- Get
  - Client of request, C, sends route(key, [GET, C])
  - When a node S receives a forward upcall, if (key, value) is cached, send route(NULL, [value, S], C) and do not forward message to nextHop
  - If root R receives deliver upcall, it sends route(NULL, [value, R], C)
- Other usages similar (CAST, DOLR, i3, replication, data maintenance) similar
  - Refer to paper

Outline

- Background
- Motivation
- API Specification
- Example Usage
- Example Implementation

Example Implementation – Chord

- route()
  - Local node invokes RPC in next node in lookup path
  - RPC invokes appropriate call (route() or deliver()) returns next hop node
- local_lookup()
  - Returns closest *num* successors in node's successor list
- replicaSet()
  - Return node's successor list
- neighborSet()
  - Return node's successor list + predecessor
- range()
  - Examine range [successor(n), successor(n+1)]
- Other implementations similar (CAN, Pastry, Tapestry)
  - Refer to paper

Future Work

- Implement API as an RPC program
- Better articulation of tier 1 services (DHT, DOLR, CAST)