# Internet Indirection Infrastructure

Ion Stoica, Daniel Adkins, Shelley Zhuang, Scott Shenker*, and Sonesh Surana*

Presented By

Xiang Gao

Oct 15, 2014

# Overview

- Motivation
- Design Overview
- Application
- Additional Design and Performance Issue
- Evaluation
- Comments

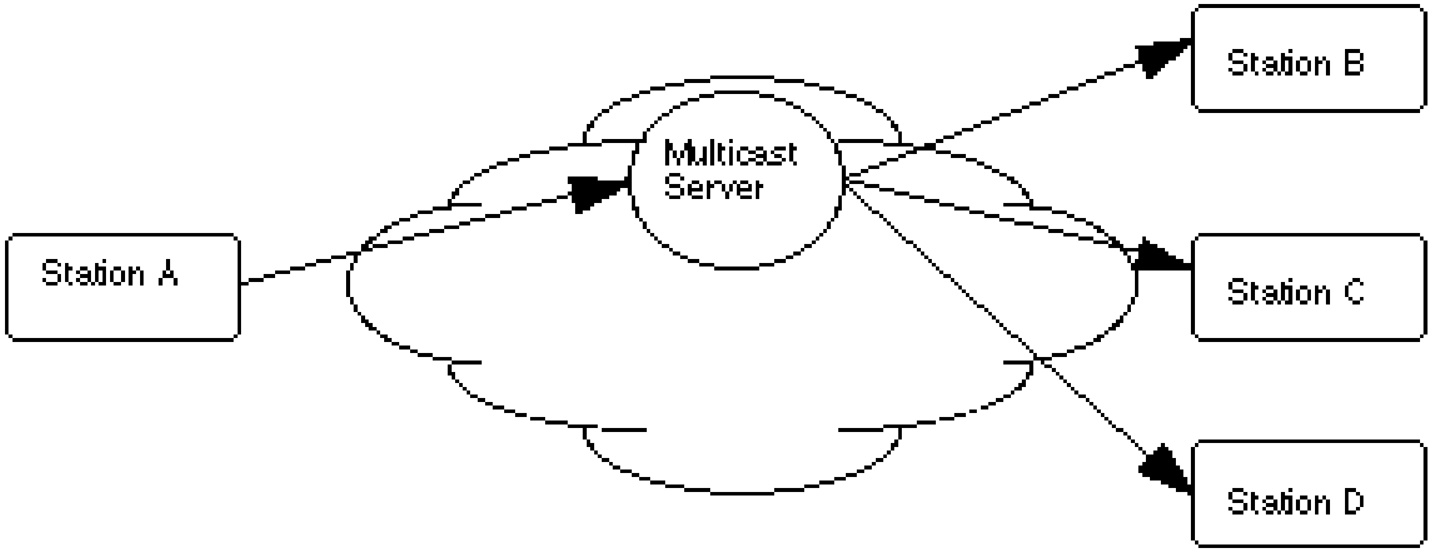
## Overview

- Motivation
- Design Overview
- Application
- Additional Design and Performance Issue
- Evaluation
- Comments

## Motivation

- Original Internet
  - Designed for Unicast, point-to-point
- More general communication abstractions are needed by modern application
  - Multicast, Anycast, host mobility
- Solutions so far
  - IP Layer:
    - Scalability
    - Deployment
  - Application Layer
    - Disjointed, redundant
    - Deployment

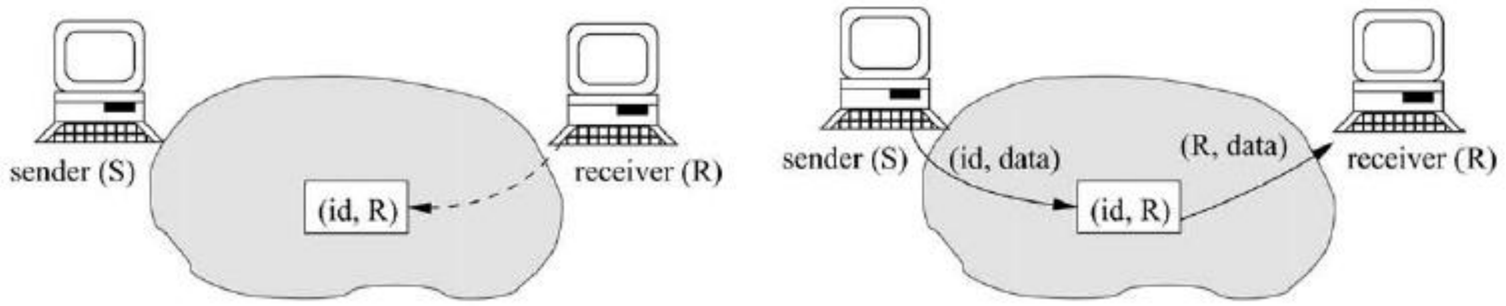
# A typical multicast scheme



# Overview

- Motivation
- Design Overview
- Application
- Additional Design and Performance Issue
- Evaluation
- Comments

## Design Overview



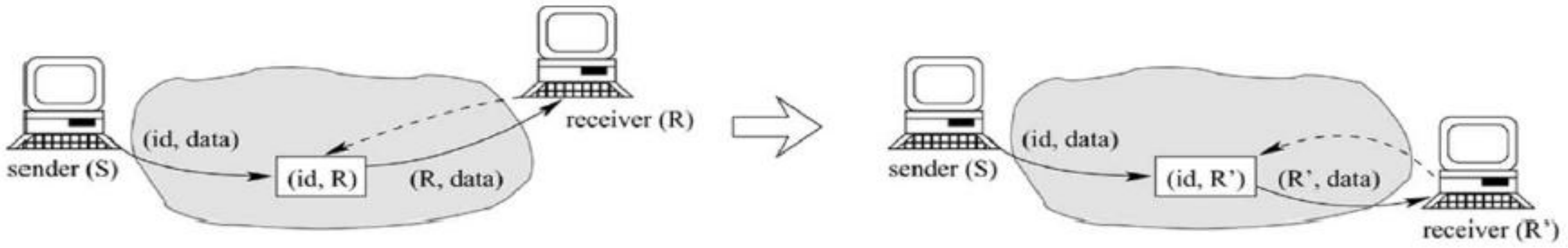
- Basic model
  - Id: A logical identifier.
    - *S*ources send packets to an id*.*
    - *R*eceivers express interest in packets sent to an id.
  - Trigger: Inserted by receiver
    - Allows receivers to control the routing
      - End-hosts can create application level services
      - End-hosts have the responsibility for efficient tree construction

# Design Overview

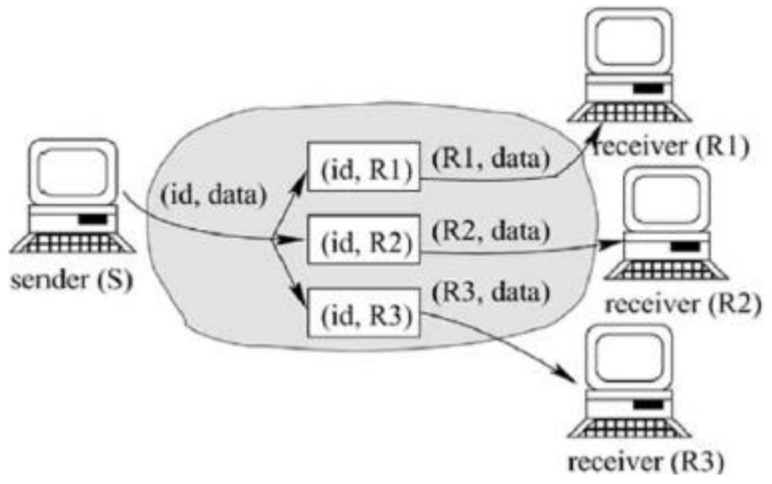
- Match packets (id, data) with triggers (id, addr)
  - Id: A logical rendezvous between the sender's packets and the receiver's trigger
    - Bits length *m,* exact-match threshold *k, k* < *m*
  - Trigger's $id_t$ matches packet's id iff
    - A prefix match of at least k bits
    - No other trigger has a longer prefix match
- Efficiently match
  - Mapping each identifier to a unique server
  - Provide best-effort service on top of IP

# Design Overview

- Primitive communication provided
  - Mobility: Receiver updates its trigger



  - Multicast: seamlessly switch



## Design Overview

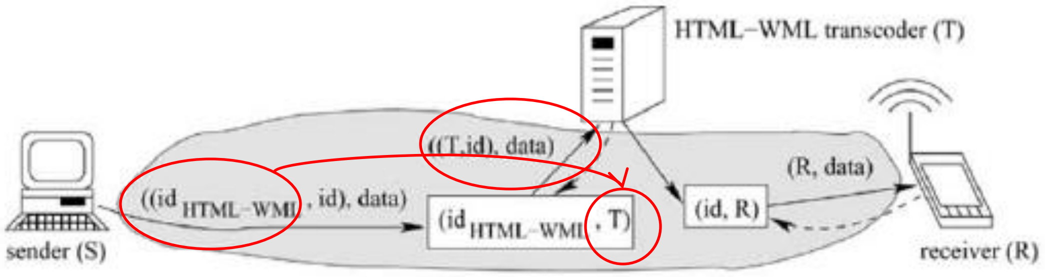
- Stack of ids
  - Packet P = ($id_1$, $id_2$,…, data)
  - Trigger T = (id, $id_{stack}$)
  - Operations
    - T match P:
      - Replace the match id with T’s id stack
    - Else:
      - Pop id stack until find a match
      - Drop the packet if the id stack is empty

# Overview

- Motivation
- Design Overview
- Application
- Additional Design and Performance Issue
- Evaluation
- Comments

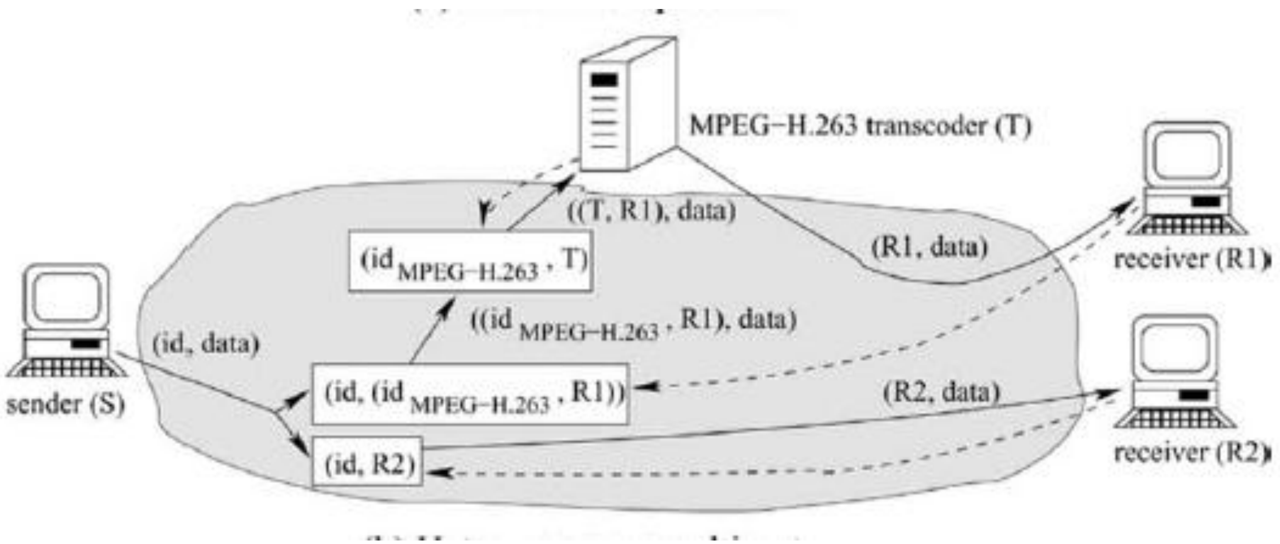
# Application

- Service composition



# Application

- Heterogeneous Multicast

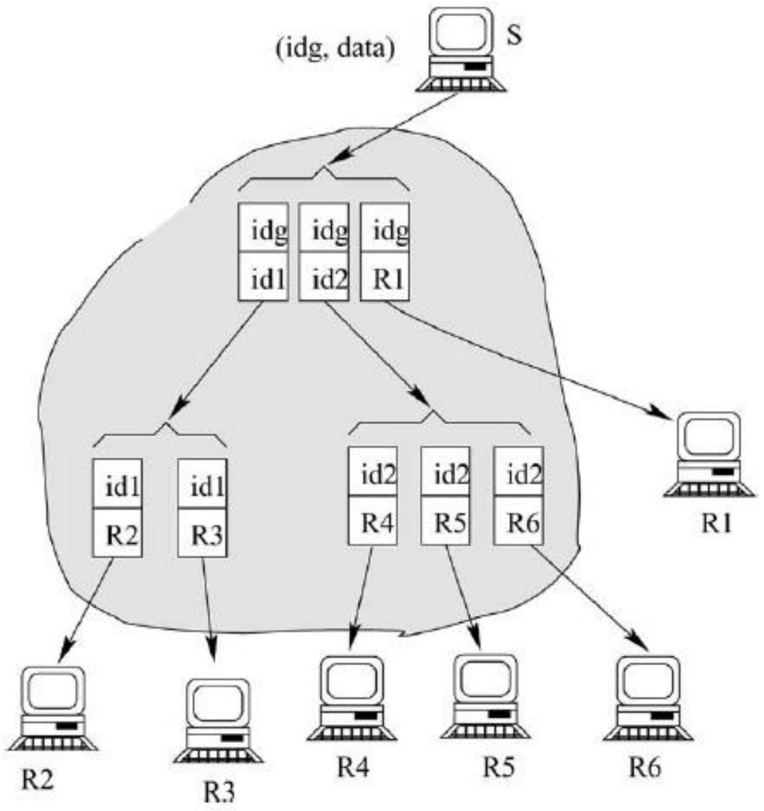


# Application

- Server Selection
  - Load balance:
    - set the *m-k* least significant bits to random values
  - Select closest server
    - Server encode its location into the last *m-k* least significant bits (e.g., zipcode)
    - Sender encode its location into the last m-k least significant bits

# Application

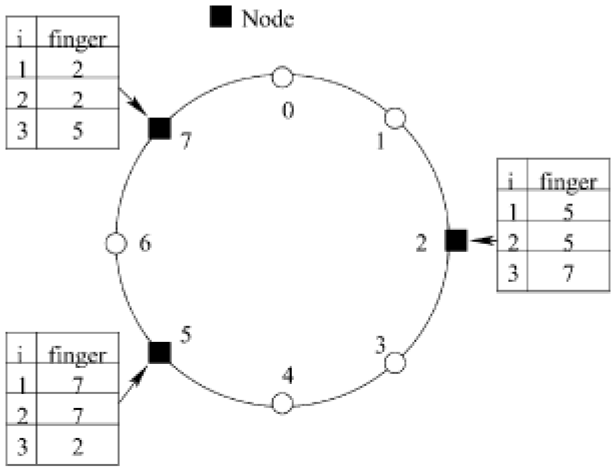
- Large Scale Multicast



## Overview

- Motivation
- Design Overview
- Application
- Additional Design and Performance Issue
- Evaluation
- Comments

# Additional Design



- Properties of the Overlay
  - Chord lookup protocol
    - *m* bits, id space [0, $2^m$-1]
    - Routing table, *i*th entry of server *n* contains the first server that follows n+$2^{i-1}$
    - How to route (O(logN) hops):
      - Incoming Id lies between itself and its successor:
        - Forward to its successor
      - else:
        - Lookup the routing table, send it to the preceding server

## Additional Design

- Public and Private Triggers
  - Public trigger is known by all end-hosts
  - Employed for efficiency and security
- Robustness
  - End-hosts use periodic refreshing to maintain their triggers
  - If a trigger is lost:
    - Eventually be reinserted
  - Avoid the failure time:
    - Receiver maintains an extra backup trigger, and sender sends packet with the extra id
    - I3 replicate triggers and manage the replicas

# Additional Design

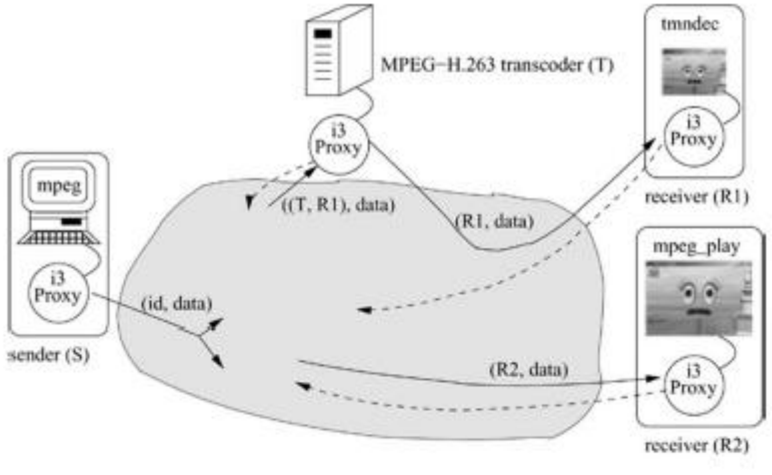
- Self-Organizing
  - Bootstrapping mechanism for nodes to join the i3
  - End-hosts only need to know one single server
- Routing Efficiency
  - Reduce the hops:
    - Sender caches the i3 server's IP address
  - Triangle routing problem:
    - Receivers choose their private triggers such that they are located on nearby servers
    - End-host can sample the identifier space to find ranges of identifiers that are stored at nearby servers. Use RTT to probe the range (insert (id, A), then send (id, dummy) to itself).

## Additional Design

- Avoiding Hot Spots
  - Spreading triggers
  - Where to ?
    - Push the trigger to the server most likely to route the packets matching that trigger
    - Try to minimize the state it needs to maintain(e.g., the predecessor server)
- Scalability
  - Suppose *n* triggers and *N* servers, each server will store *n/N* triggers on the average
  - Scale up by adding more servers

# Additional Design

- Incremental Deployment
  - Transparent to other nodes
- Legacy Applications
  - Proxy on end-hosts:
    - Translates between the applications' UDP packets and i3 packets
    - Inserts triggers on behalf of the applications.



## Additional Design

- Security
  - Attacks to (id, R):
    - Using triggers pointing to end-hosts:
      - Eavesdrop*:   insert (id, X)*
      - Reflection*: insert(id’, R)*
      - Impersonation*: cause R to drop its public trigger*
    - Form arbitrary topologies with triggers:
      - Form a loop by inserting triggers in a cycle
      - construct a confluence
      - construct a chain of triggers that leads to a dead end

# Additional Design

- Security
  - Solutions
    - Constrained Triggers:
      - Define a constraint for a trigger *(x,y),* such that choosing *x* constrains the choice of *y* or vice versa.
    - Pushback:
      - Removing the previous trigger in the chain cascading when reach a dead end
    - Trigger Challenges:
      - i3 challenges trigger insertion

## Overview

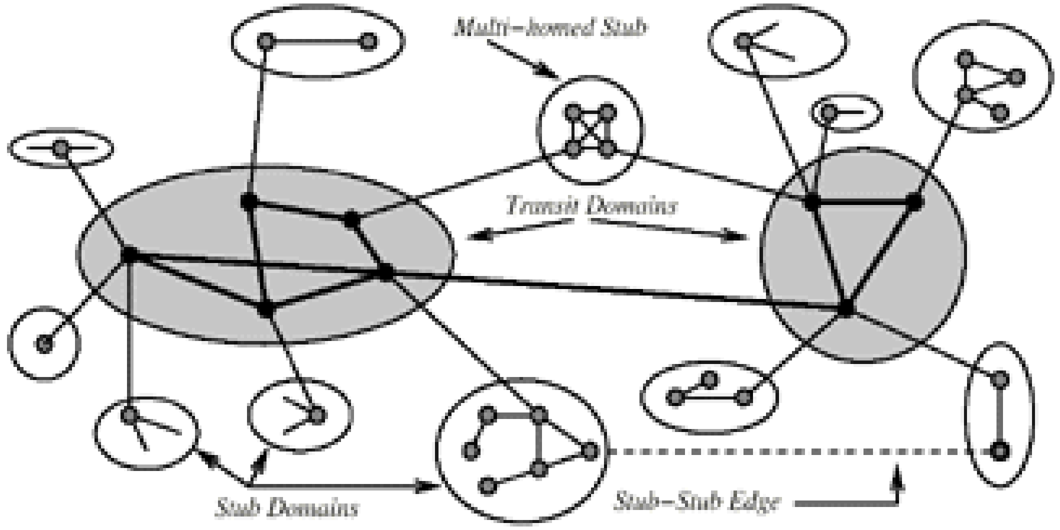
- Motivation
- Design Overview
- Application
- Additional Design and Performance Issue
- Evaluation
- Comments

# Evaluation

- Metric
  - Latency stretch: the ratio of the inter-node latency on the network to the inter-node latency on the underlying IP network.
- Network topologies
  - power-law random graph topology
    - 5000 Nodes, servers randomly assigned, delay uniformly distributed in [5, 100)
  - transit-stub topology
    - 5000 Nodes, 100ms for intra-transit, 10ms for transit-stub, 1ms for intra-stub domain links.

# Evaluation

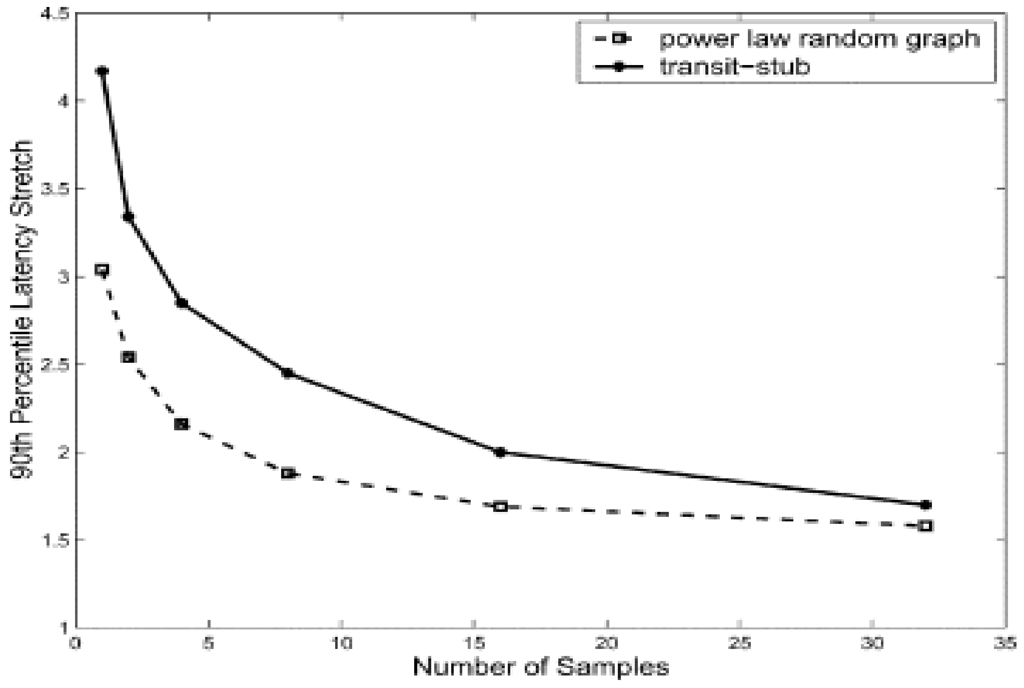
- A transit-stub topology



http://www.cs.columbia.edu/~hgs/teaching/ais/1998/projects/Panagiotis_Sebos/report.html

## Evaluation

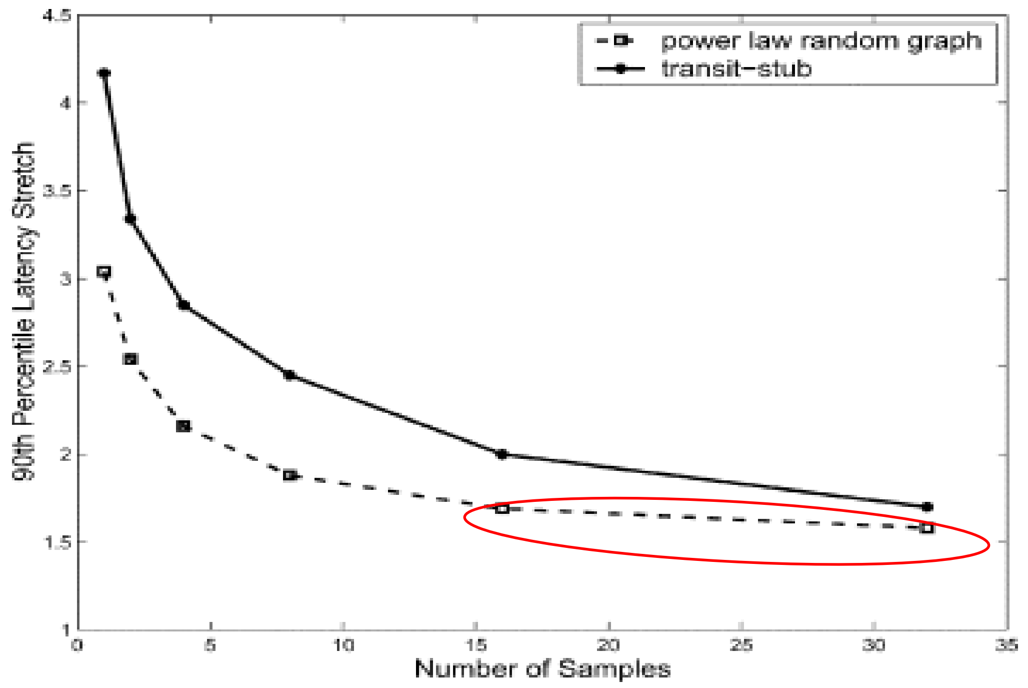
- Measure the sample effectiveness



The 90th percentile latency stretch versus number of samples for PLRG and transit-stub with 5000 nodes.

# Evaluation

- End-to-End Latency:
  - Measure the sample effectiveness

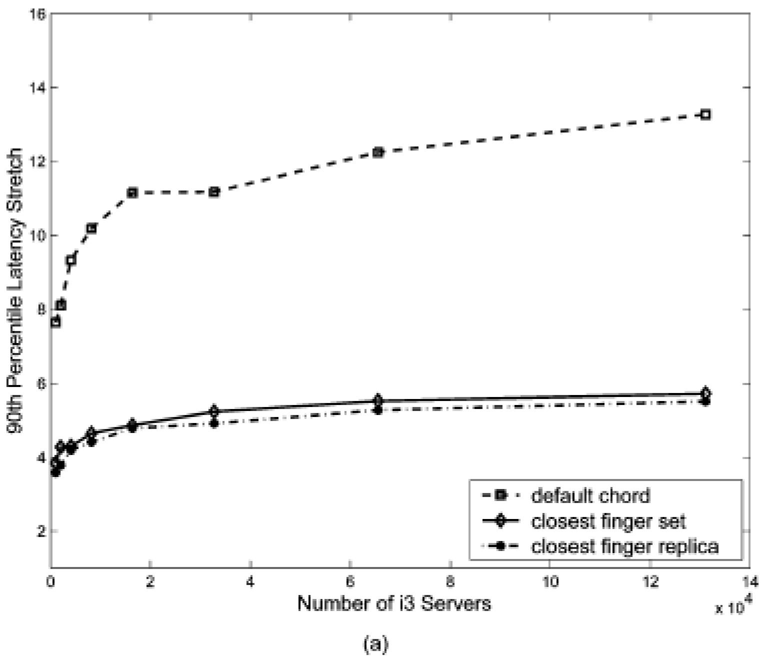


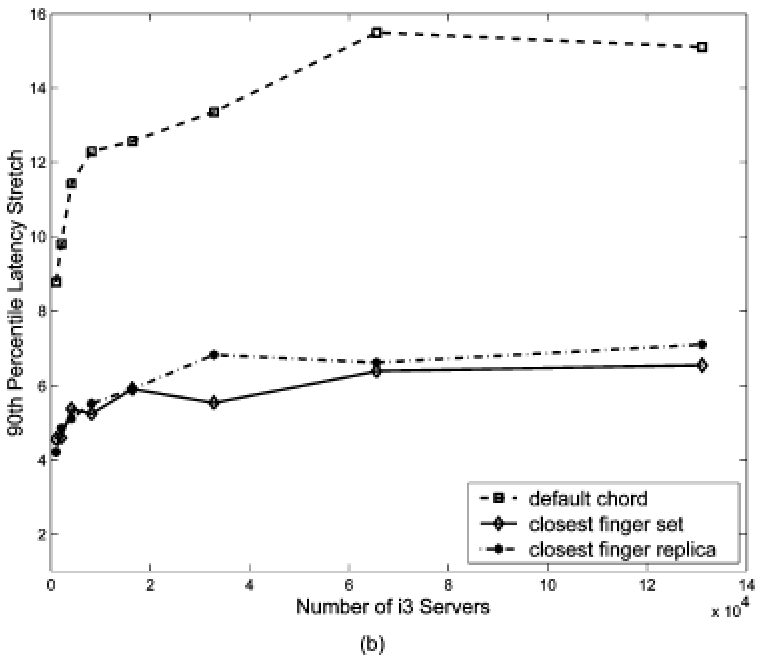
The 90th percentile latency stretch versus number of samples for PLRG and transit-stub with 5000 nodes.

# Evaluation

- Proximity Routing
  - Closest finger replica: In addition to each finger, a server maintains *r-1* immediate successors of that finger.
  - Closest finger set: To route a packet, server considers only the closest $log_2N$ fingers in terms of network distances among all its $log_bN$ fingers ($b < 2$).

# Evaluation



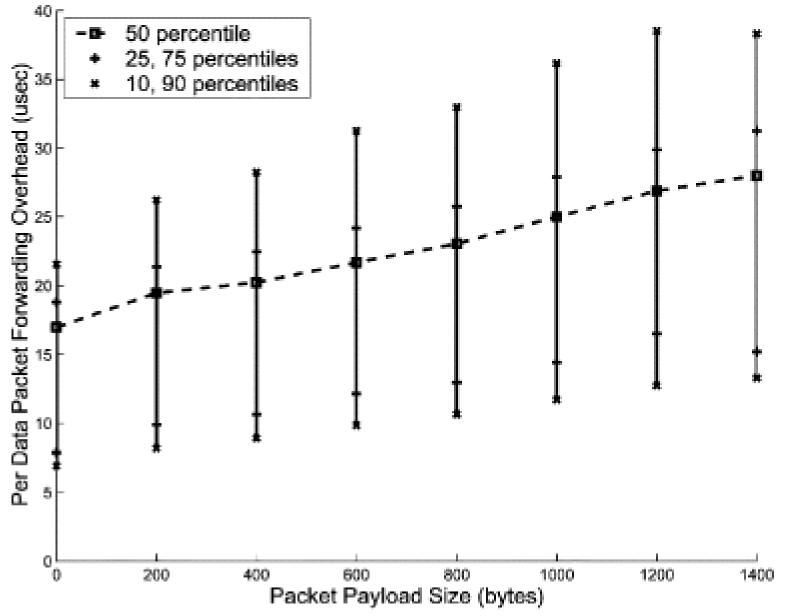


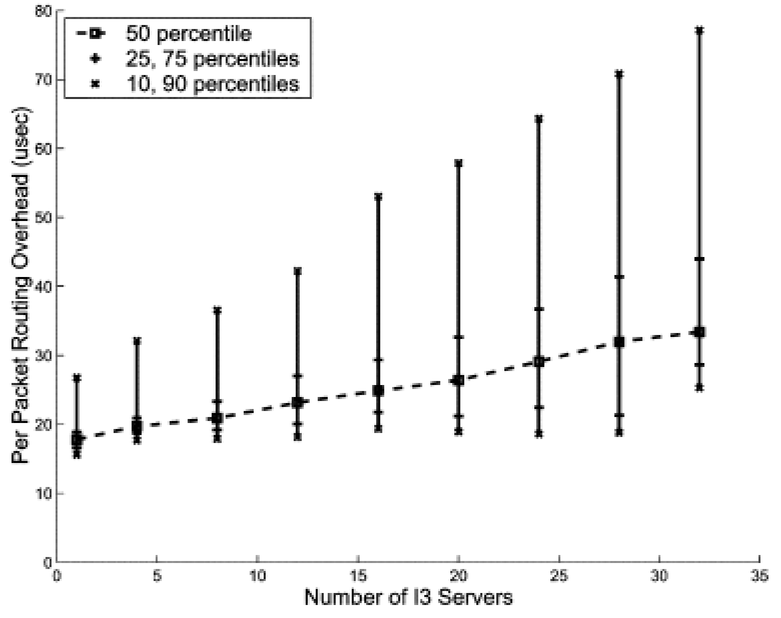
The 90th percentile latency stretch in the case of (a) a power-law random network topology with 5000 nodes, and (b) a transit-stub topology with 5000 nodes.
The i3 servers are randomly assigned to all nodes in case (a), and only to the stub nodes in case (b).

## Evaluation

- Performance
  - Trigger Insertion
    - Maintained in hashtable
    - The average and the standard deviation of the trigger insertion operation over 10,000 insertions are 12.5 and 7.12 µs, respectively. This is mostly the time it takes the operating system to process the packet and to hand it to the application.
  - Data Packet Forwarding
    - As packet size increases, memory copy operations and pushing the bits through the network dominate processing time.

# Evaluation



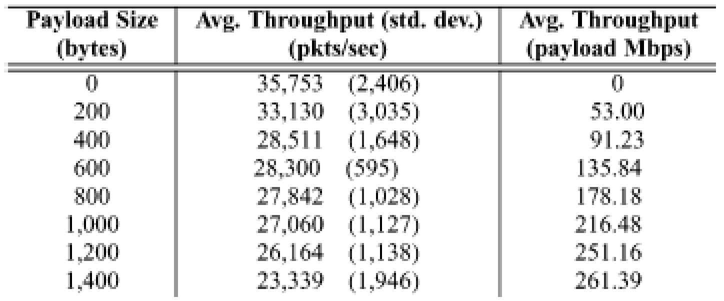


Per packet forwarding overhead as a function of payload packet size.
In this case, the i3 header size is 48 bytes.

Per packet routing overhead as a function of i3 nodes in the system.
The packet payload size is zero.

# Evaluation

- Performance
  - Throughput:
    - The user throughput in megabits per second increases as the packet payload increases because the overhead for headers and processing is roughly the same for both small and large payloads.



| Payload Size (bytes) | Avg. Throughput (std. dev.) (pkts/sec) | Avg. Throughput (payload Mbps) |
|---|---|---|
| 0 | 35,753 (2,406) | 0 |
| 200 | 33,130 (3,035) | 53.00 |
| 400 | 28,511 (1,648) | 91.23 |
| 600 | 28,300 (595) | 135.84 |
| 800 | 27,842 (1,028) | 178.18 |
| 1,000 | 27,060 (1,127) | 216.48 |
| 1,200 | 26,164 (1,138) | 251.16 |
| 1,400 | 23,339 (1,946) | 261.39 |

## Overview

- Motivation
- Design Overview
- Application
- Additional Design and Performance Issue
- Evaluation
- Comments

## Comments

- How to make id unique in global deployment?
- How to constrain long id stack in globally deployed applications ?
- Since when a lookup fails, the packet will be forwarded to next server until it is dropped. This seems to be resource consuming process, and also insecure.

Thank you!