Motivations

• Today’s Internet is built around a unicast point-to-point communication abstraction:
  – Send packet “p” from host “A” to host “B”
• This abstraction allows Internet to be highly scalable and efficient, but…
• … not appropriate for applications that require other communications primitives:
  – Multicast
  – Anycast
  – Mobility
  – …

Why?

• Point-to-point communication → implicitly assumes there is one sender and one receiver, and that they are placed at fixed and well-known locations
  – E.g., a host identified by the IP address 128.32.xxx.xxx is located in Berkeley

IP Solutions

• Extend IP to support new communication primitives, e.g.,
  – Mobile IP
  – IP multicast
  – IP anycast
• Disadvantages:
  – Difficult to implement while maintaining Internet’s scalability (e.g., multicast)
  – Require community wide consensus -- hard to achieve in practice

Application Level Solutions

• Implement the required functionality at the application level, e.g.,
  – Application level multicast (e.g., Narada, Overcast, Scattercast…)
  – Application level mobility
• Disadvantages:
  – Efficiency hard to achieve
  – Redundancy: each application implements the same functionality over and over again
  – No synergy: each application implements usually only one service; services hard to combine

Internet Indirection Infrastructure (i3)

• Each packet is associated an identifier id
• To receive a packet with identifier id, receiver R maintains a trigger (id, R) into the overlay network

Sender

Receiver (R)
Service Model

- API
  - `sendPacket(p);`
  - `insertTrigger(t);`
  - `removeTrigger(t)` // optional
- Best-effort service model (like IP)
- Triggers periodically refreshed by end-hosts
- ID length: 256 bits

Mobility

- Host just needs to update its trigger as it moves from one subnet to another

Multicast

- Receivers insert triggers with same identifier
- Can dynamically switch between multicast and unicast

Anycast

- Use longest prefix matching instead of exact matching
- Prefix `p`: anycast group identifier
- Suffix `s`: encode application semantics, e.g., location

Service Composition: Sender Initiated

- Use a stack of IDs to encode sequence of operations to be performed on data path
- Advantages
  - Don’t need to configure path
  - Load balancing and robustness easy to achieve

Service Composition: Receiver Initiated

- Receiver can also specify the operations to be performed on data
Quick Implementation Overview

- i3 is implemented on top of Chord
  - But can easily use CAN, Pastry, Tapestry, etc
- Each trigger $t = (id, R)$ is stored on the node responsible for $id$
- Use Chord recursive routing to find best matching trigger for packet $p = (id, data)$

Routing Table

- Chord uses an $m$ bit circular identifier space where 0 follows $(2^m) - 1$
- Each identifier $ID$ is mapped on the server with the closest identifier that follows $ID$ on the identifier circle. This server is called successor of $ID$
- Each server maintains a routing table of size $m$. The $i$th entry in the routing table of server $n$ contains the first server that follows $n + 2^{(i-1)}$
- A server sends the packet to the closest server (finger) in its routing table that precedes $ID$.
- It takes $O(\log N)$ hops to route a packet to the server storing the best matching trigger for the packet, where $N$ is the number of i3 servers.

Routing Example

- R inserts trigger $t = (37, R)$; S sends packet $p = (37, data)$
- An end-host needs to know only one i3 node to use i3
  - E.g., S knows node 3, R knows node 35

Optimization #1: Path Length

- Sender/receiver caches i3 node mapping a specific ID
- Subsequent packets are sent via one i3 node

Optimization #2: Triangular Routing

- Use well-known trigger for initial rendezvous
- Exchange a pair of (private) triggers well-located
- Use private triggers to send data traffic
Examples

– Heterogeneous multicast
– Scalable Multicast
– Load balancing
– Proximity

Example 1: Heterogeneous Multicast

• Sender not aware of transformations

Example 2: Scalable Multicast

• i3 doesn’t provide direct support for scalable multicast
  – Triggers with same identifier are mapped onto the same i3 node
• Solution: have end-hosts build an hierarchy of trigger of bounded degree

Example 3: Load Balancing

• Servers insert triggers with IDs that have random suffixes
• Clients send packets with IDs that have random suffixes

Example 4: Proximity

• Suffixes of trigger and packet IDs encode the server and client locations
Security: Some Attacks

Eavesdropping

Loop

Confluence

Dead-End

Possible Defense Measures

- End hosts can use the public triggers to choose a pair of private triggers, and then use these private triggers to exchange the actual data. To keep the private triggers secret, one can use public key cryptography to exchange the private triggers.

- Instead of inserting the trigger \((id, S)\), the server can insert two triggers, \((id, x)\) and \((x, S)\), where \(x\) is an identifier known only by \(S\).

- An \(i3\) server can easily verify the identity of a receiver \(S\) by sending a challenge to \(S\) the first time the trigger is inserted. The challenge consists of a random nonce that is expected to be returned by the receiver. If the receiver fails to answer the challenge the trigger is removed.

- Each server can put a bound on the number of triggers that can be inserted by a particular end-host.

- When a trigger that doesn’t point to an IP address is inserted, the server checks whether the new trigger doesn’t create a loop.

Traffic Isolation

- Drop triggers being flooded without affecting other triggers
  - Protect ongoing connections from new connection requests
  - Protect a service from an attack on another services

Traffic Isolation

- Drop triggers being flooded without affecting other triggers
  - Protect ongoing connections from new connection requests
  - Protect a service from an attack on another services

Traffic of transaction server protected from attack on web server.