Architectural Issues for Identifiers and Locators

Dave Thaler
dthaler@microsoft.com
Outline

• Terminology
• Core Problems
  – Routing scalability
  – Mobility (host and site)
  – Multihoming (host and site)
  – Location Privacy
• What can be changed?
• Design space questions
Starting from basics

• Users deal with *names*, not addresses (esp. in IPv6)
  – Humans need “friendly” identifiers that can be remembered and typed
  – Name = who (informally) you are

• Routing deals with *locators* (e.g., IP addresses)
  – Locator = where you are

• Security deals with *identities* that can be used as principals
  – Identity = who you are (really!)
  – May or may not be tightly bound to a name or locator (e.g., CGA)
Routing Scalability Basics

• Today we use hierarchical aggregation, which is broken by:
  
  – **Provider Independent (PI) addressing**: Sites want to be able to change providers without renumbering, to have a sense of "ownership" of their address space, to ease site multihoming.
  
  – **Site Multihoming**: Even if PI addressing is not used, multihoming injects more-specific routes from one provider to another which the entire global routing table must then carry.
  
  – **Traffic Engineering**: Providers inject more-specific routes to influence the behavior of the routing system, in order to control various traffic patterns

• All of these challenges are due to local operational state propagated globally
Mobility Basics

• If routing had no scaling and convergence time limitations, mobility could be handled by routing
  – Just use dynamically updated host routes

• If name resolution had no scaling and convergence time limitations, mobility could be handled by name resolution
  – Just use dynamically updated name records
Host Mobility 1: Accept new connections immediately after a move

Q: So what’s the problem?
A: Mainly design limitations of current solutions:
   – Inability of name resolution (DNS) to deal with rapid changes
     • Some DNS servers don’t respect small TTLs
   – Addresses are cached by applications and services
     • Applications don’t respect TTLs either
Host Mobility 2: Preserve established connections

• locators change over time
• There can also be periods of complete disconnectivity
  – Travel between work and home (long)
  – Ride in an elevator (medium)
  – Just walk past a cement pillar (short)
• To deal with disconnectivity, some layer must do a reconnect transparent to the user
• There is benefit to applications handling disconnectivity themselves
• Even if application does reconnects, reconnect time is still long enough that dealing with mobility below the application is still necessary for real-time interactive apps
Site Mobility: Ease Renumbering

• Renumbering pains depend on how many places addresses are configured:
  • Routers
  • Hosts
  • DNS servers
  • DHCP servers
  • Firewall
  • Remote monitoring systems
  • Intrusion detection systems
  • Load balancers
  • Management tools/databases
  • Etc.

• Whether renumbering is any easier or not depends how many of above have to change
Multihoming: Support redundancy, load sharing, etc (RFC3582)

- Named entities exist on machines with a set of locators.

- Efficient load sharing & redundancy needs a locator set to be communicated somehow
  - One end chooses which locators are communicated
  - Other end chooses among locators learned

- Problems:
  - Various applications and protocols (TCP, SIP, etc.) today only communicate one address
  - They also don’t re-bind during connections
Location Privacy

• Ability to hide topology details from outsiders

• Locator is visible to remote endpoint unless:
  – A translator is in between, or
  – End wanting privacy tunnels to/from something in between

  – Both separate identifier seen by remote endpoint from locator used by local routing
What can we change? (1/2)

- “Managed” systems are easier
  - “managed” = frequently/automatically upgraded software/patches

- “Unmanaged” systems are hard
  - “unmanaged” = someone rarely (if ever) looks at it and patches aren’t automatic

<table>
<thead>
<tr>
<th>Common Cases:</th>
<th>Managed</th>
<th>Unmanaged</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hosts</td>
<td>Homes</td>
<td>Corporate, Embedded, Legacy</td>
</tr>
<tr>
<td>Routers</td>
<td>Corporate</td>
<td>Homes</td>
</tr>
</tbody>
</table>
What can we change? (2/2)

• Applications:
  – Can’t change all of them
  – But can affect new applications
    • Note that many applications are moving up to higher-layer APIs anyway, so a host change may be sufficient for them

• Management & Security systems:
  – These are often the last/hardest to change
  – Most of them assume upper-layer identifier == locator
    • Separation makes it harder for intermediate system to peek in and look at the identifier
  – Unlike apps, you have to work with all of them before you can deploy in a corporation
  – Implies either blocked on changing them, or else must have identifier == locator within a corporate network
Incentive Structure

• Best if *only* requires changes by entities actually feeling pain, e.g.
  – Service Provider (Routers): routing scalability
  – End-user (Hosts): mobility, host multihoming

• Often *only one entity* experiences the pain, and so is incented to change
  – Best if provides actual benefits when only that entity is changed
Design Questions

• What properties should an *identifier* have?
  – Take as given (per BOF description):
    • Works with legacy applications
    • Works with legacy destinations
    • Supports referrals
  – Open questions:
    1. How is mapping secured?
    2. How do you map?
    3. Is identifier routable or not?
    4. Explicit in data packet or not?
Support referrals

• One application/user/service wants to refer/redirect you to another one
  – Would like the new identifier to be authenticatable
    • I.e., want chain of trust from identifier to connection
  – Why not just use a name? (example: HTTP redirect URL contains hostname)
  – Inefficiency of subsequent name-to-locator mapping step
    • But refer/redirect could provide a locator hint
  – Further complicated by current design/deployment limitations:
    • Many protocols are defined to refer/redirect to IP address
    • Some apps might only cache addresses
    • Not all applications/users/services have a name
1. How is mapping secured? (1/3)

Currently defined (examples):

Simple id/loc separation:

Multiple levels:
Security Basics

• Need a chain of trust from a user-friendly name to a connection
  – DNSsec alone is not sufficient if the locator can be spoofed
  – Self-signed CGAs alone are not sufficient if the name-to-locator mapping can be spoofed
  – If names are authenticated directly (e.g., TLS/DTLS) then any spoofing attacks are reduced to DoS

• Need a chain of trust from whatever an application starts from, to a connection
  – Not all applications act on behalf of humans (e.g., server apps)
  – Either application always needs to start from a name, or also need chains of trust from whatever other type of identifier is in use
How is mapping secured? (3/3)

- Identifier algorithmically derived from identity?
  - Chain of trust goes from identifier to connection

- Identifier/locator mapping is signed by some trusted identity?
  - Chain of trust goes from identifier to locator, and relies on something else for locator-to-connection

- Return routability check only?
2. Mapping name to identifier

• If name == identifier, this is a no-op
• Otherwise:
  – Need to work with existing name resolution mechanisms (DNS, SIP, etc.)
  – Need to deal with security
  – Can there be multiple identifiers?
    • Some names map to multiple hosts (www.example.com)
Mapping identifier to locators (1/3)

• What?
  – Need to deal with dynamically changing locators
  – Need to deal with multiple locators
  – Need to deal with security
Mapping identifier to locators (2/3)

Where?

• **Vertical locus:**
  – Application
  – Session Layer
  – Transport Layer
    • Do the above 3 work for *all* applications?
  – Network Layer
  – Below IP
    • Does it work for *all* link types?

• **Horizontal locus:**
  – Within host
  – Out in the network somewhere
    • Doesn’t work if the host attaches to different networks
    • Farther out centralizes burden
Mapping identifier to locator (3/3)

• When?
  A priori:
    • How much data has to be learned a priori?

On demand:
  – Name resolution time
    • But not all apps resolve names (server apps, referrals, etc.)
  – At time of first packet
    • Forced to buffer/drop packets
      • How avoid circular dependency between routing and lookup?

  – When do you get changes to the mapping?
Implications on Mapping Function (nod Ross Callon)

• If in a router:
  – On-demand forwarding plane update implies demands on the router control plane
    • May be ok if rare enough, but not if too frequent
    • Imagine multiple 100G interfaces sending packets “every so often” to a general purpose CPU
  – Will some routers have mapping tables that are just as or more large and dynamic as today’s FIBs?
3. Is identifier routable or not?

- Non-routable
  - May need both “ends” to change (deployability impact)
  - May have to drop/queue pending lookup (app impact)

- “Routable” (in some topology)
  - May increase routing state (“router” impact)
4. Explicit in data packet or not?

• Is id->locator mapping explicit in every data packet, or implicit (only communicated in signaling plane)?
• Do we provide the ability for intermediate systems to see the identifier or not?

• Explicit (e.g., tunneling):
  – Identifier can be seen in packet by intermediate systems that change to look for it
  – Causes increase in packet size, more fragmentation

• Implicit (e.g., index or translation):
  – Identifier not findable in data packets
  – Asymmetric paths mean intermediate systems may not have mapping state
Summary

• Need to align ease of deployment and incentives
• There are plenty of interesting problems, but incentives are in different places
• There might not be one solution, but rather complementary ones
  – Different problem -> incentive -> location of change
  – E.g., routing scalability in provider-managed routers vs host mobility in hosts