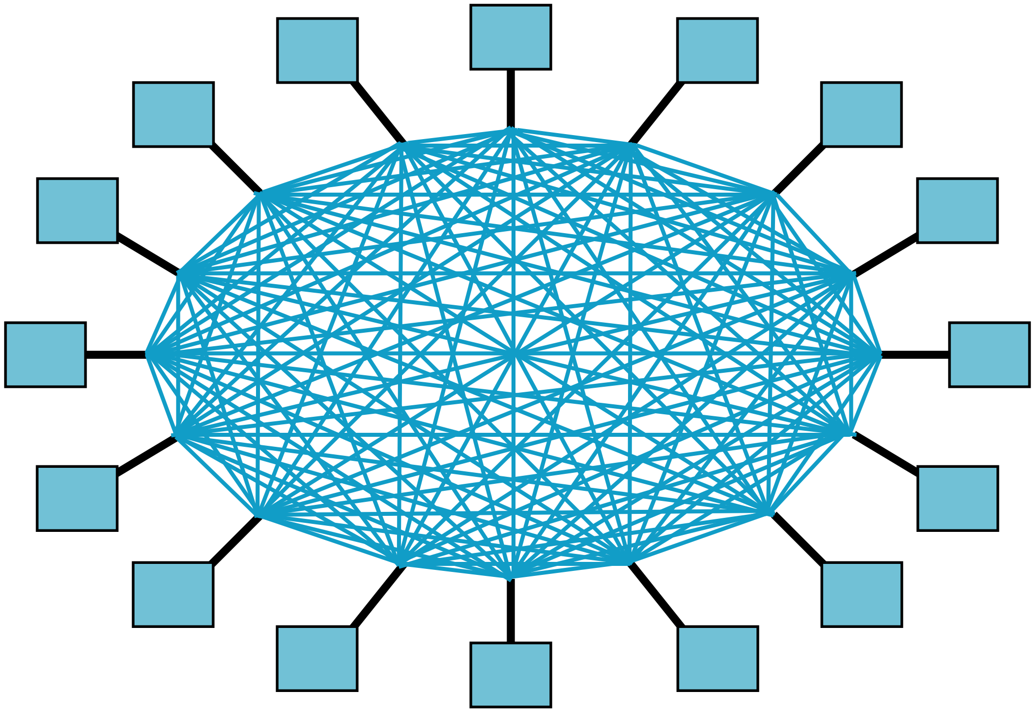
Back to the Future: Postmodern Routing and Forwarding Architecture

Ken Calvert
University of Kentucky
Laboratory for Advanced Networking

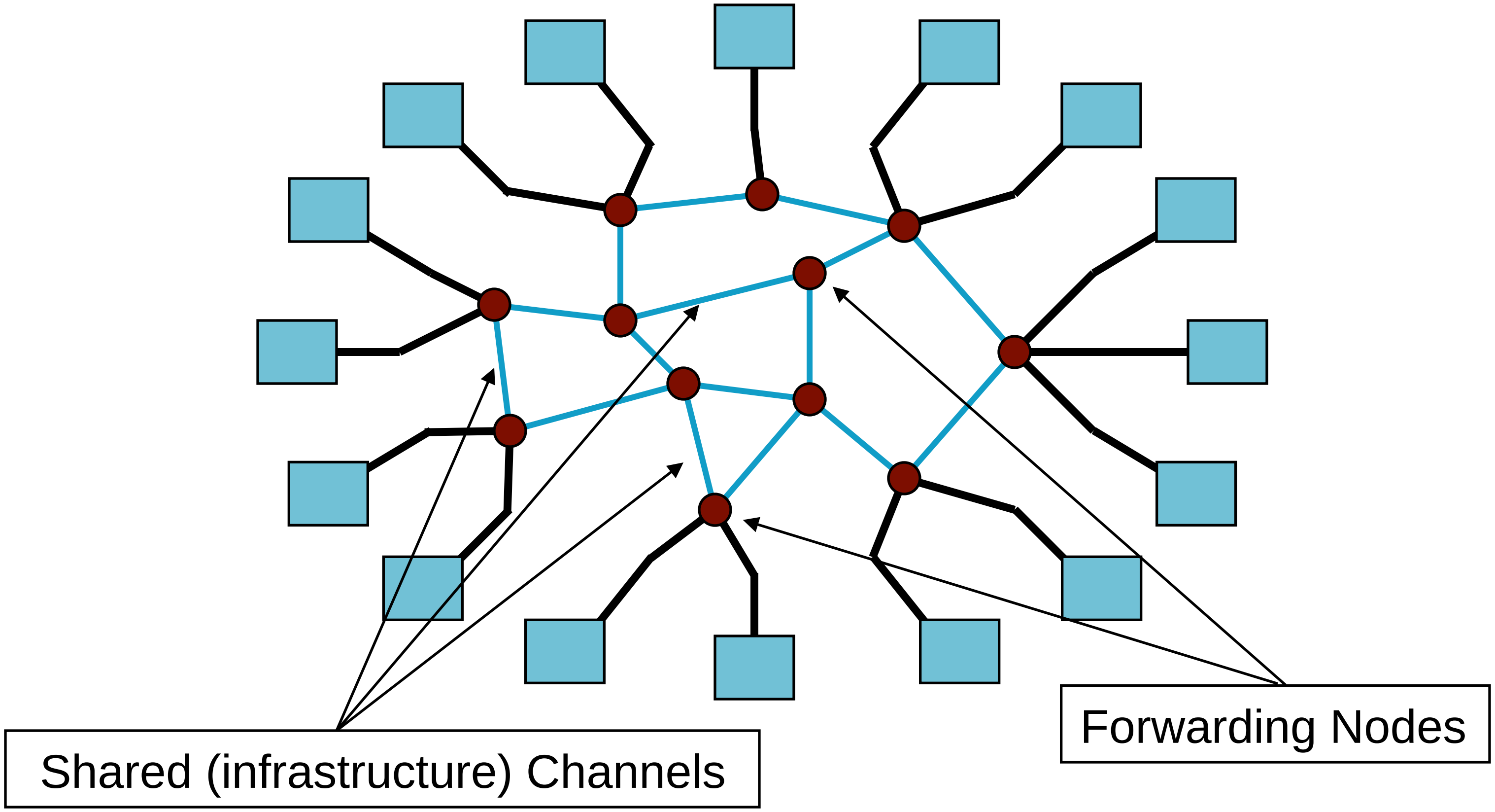
FIND: Postmodern Internet Architecture Project
Collaborators: J. Griffioen, O. Ascigil, S. Yuan

Why do we build networks?



Because direct connection is too expensive – need to share channels.

Why do we build networks?



Forwarding Nodes

Shared (infrastructure) Channels

Project Goals & Approach

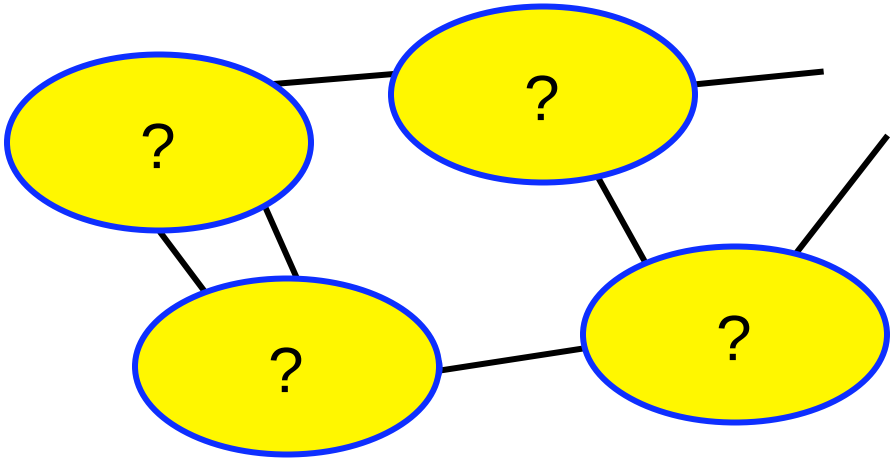
- Answer this question: How might the network layer be designed “from scratch” today?
  - Network layer: Share channels
    - “Get packets from A to B via C”
- “Stand on the shoulders of giants”
  - Steal good ideas from last 20 years
  - Design and implement, then see how to map onto current Internet
- Approach:
  - Omit what’s not crucial
  - Separate mechanism and policy
    - “Design for tussle” – ask “What mechanism could make this better?”
  - Don’t be overly constrained by today’s technology; have faith in engineering

What we leave out

- Node Identifiers
  - The network exists to share channels
  - Name channels, not nodes!
  - Why
    - Don’t have to change names when changing levels of abstraction
    - Clean inductive approach

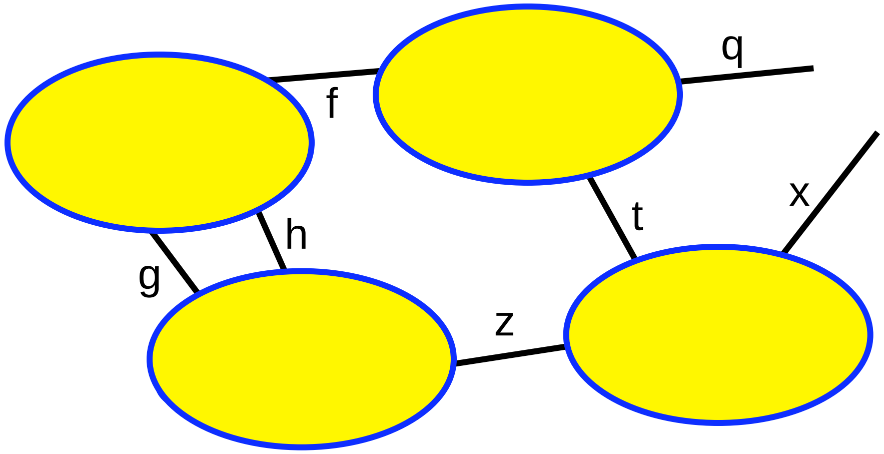
What to name:
Nodes vs. Channels

- Naming nodes



What to name:
Nodes vs. Channels

- Naming Channels



What we leave out

- Node Identifiers
  - Network is about sharing channels
  - Name channels, not nodes
  - Why:
    - Don’t have to change names when changing levels of abstraction
    - Clean inductive structure
- Topology-based addressing
  - Why: no address-assignment authority needed
  - Channel IDs can be self-configuring
    - Each has associated (self-certifying) public key [see CGA]

What we leave out

- Higher-level demultiplexing
  - Why: hide clues about packet's purpose
    - Make sure providers have other mechanisms to protect their interests
  - Instead: demux protocol is outside network layer
    - Dynamic or by prior agreement
- Hop-by-hop path determination
  - Why:
    - Enable a greater range of path selection policies
    - Allow appropriate party to specify its part of the path
  - Instead: Loose source routing
    - Packets carry sequence of channel IDs; nodes forward to next channel (possibly after pushing another sequence)

What we leave out

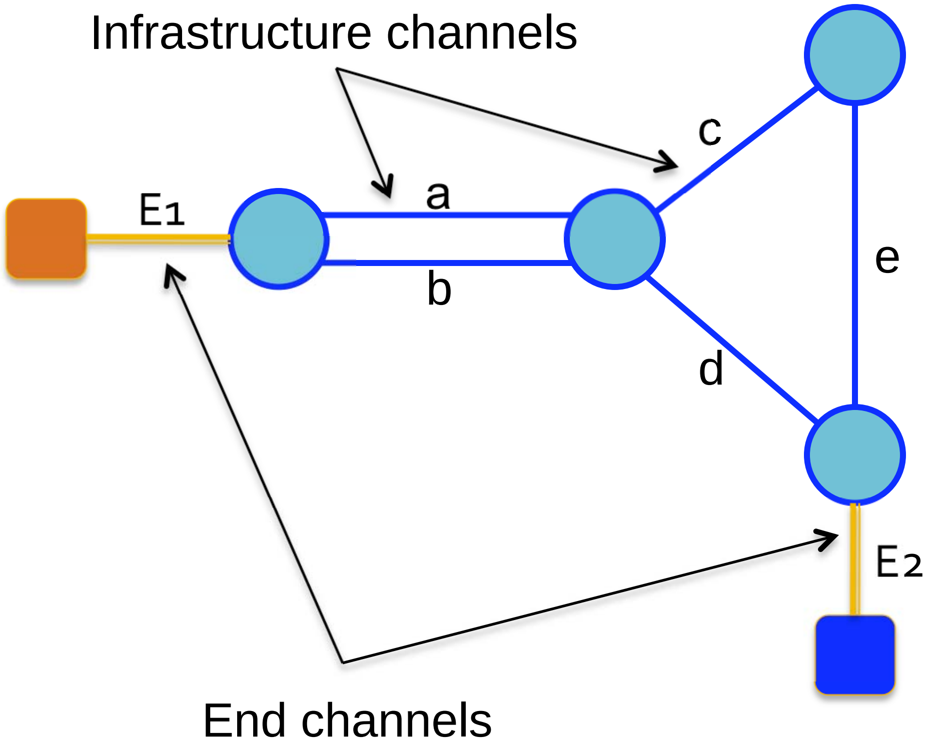
- Universal Destinationhood
  - Why: Give endpoints control over their own reachability
  - Instead: Endpoints register with EID-to-Locator mapping service to be “findable”

Architecture Features

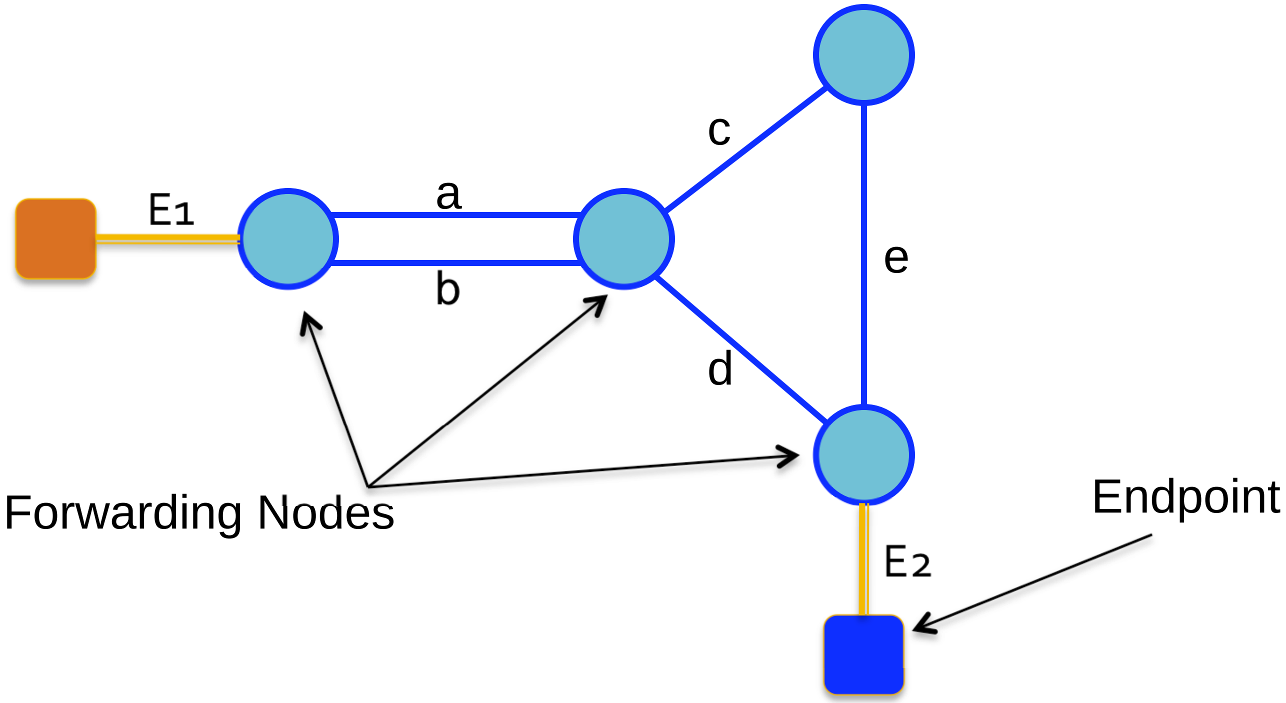
Simple inductive model

- Base case:
  - Link-state discovery of infrastructure channel topology
    - Topology service collects information about infrastructure channels and transit between them (pricing,
    - Nodes advertise willingness to transit packets between infrastructure channels
  - Routing service computes routes to destination
    - May be done on demand
  - Destinations register with EID-to-Locator (E2L) service

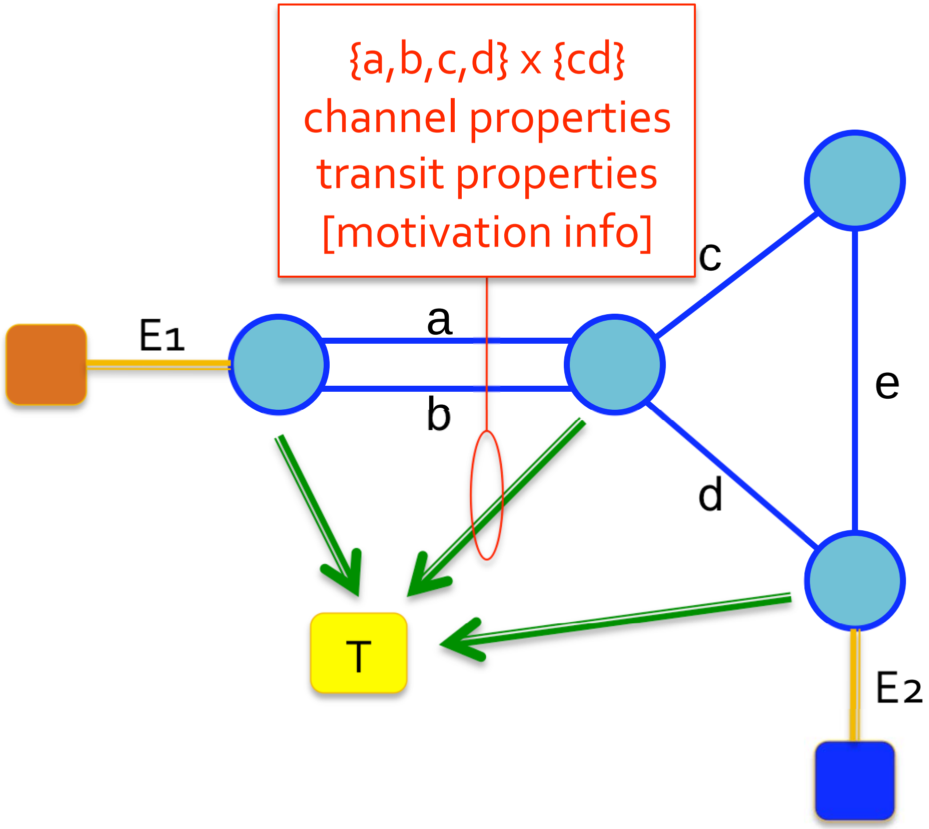
Base case



Base case



Base case

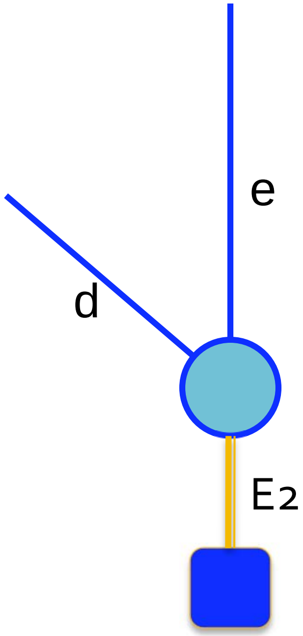

{a,b,c,d} x {cd}
channel properties
transit properties
[motivation info]

Nodes advertise transit capabilities to topology/routing service.

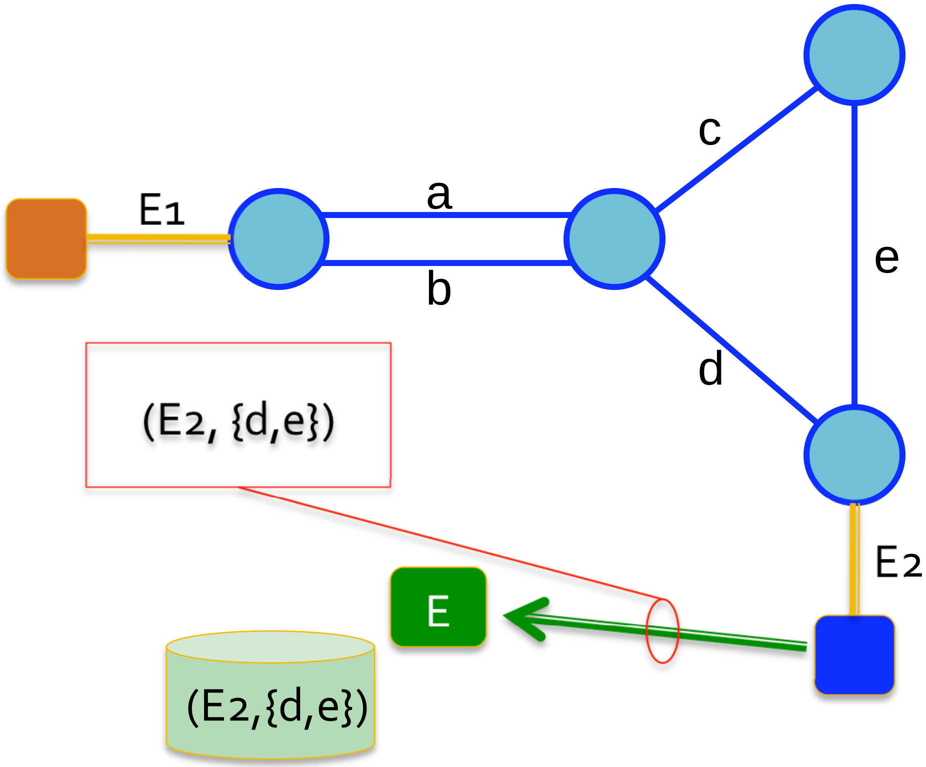
Locators

- Problem: scalability of including end-channels in topology
- Solution: Topology service only knows about infrastructure channels
- Locator = (EID, {path_1, ..., path_k})
- Resolve destination EID to locator
- Consequences:
  - Destinations control whether they are “findable”
  - Multhihoming is handled naturally

Locator: (E2, {d, e})

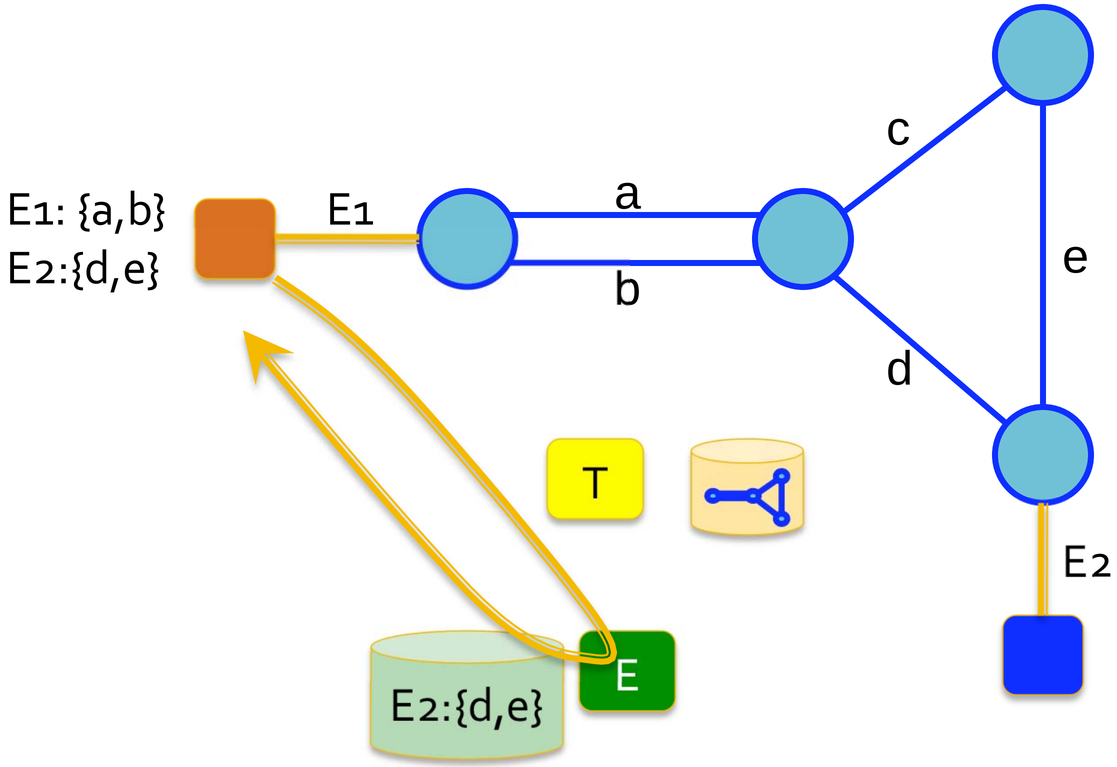


Base case, cont.



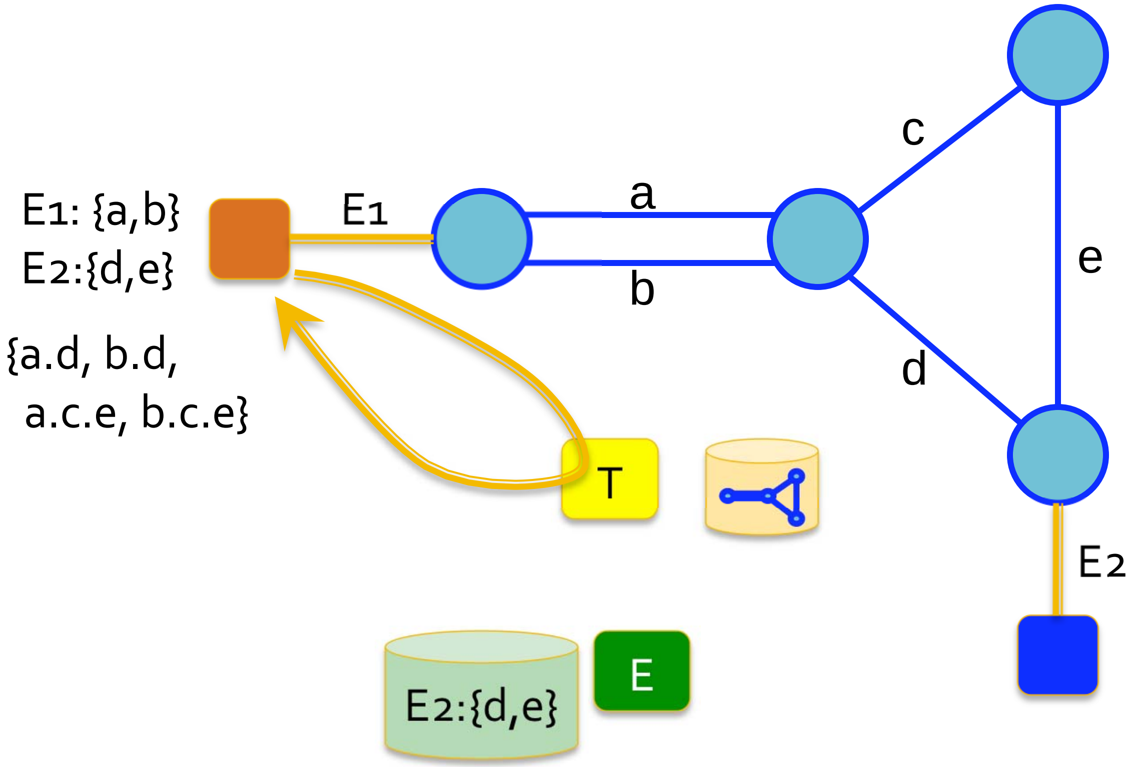
Destination endpoints register their mappings with EID-to-Locator service.

Base case



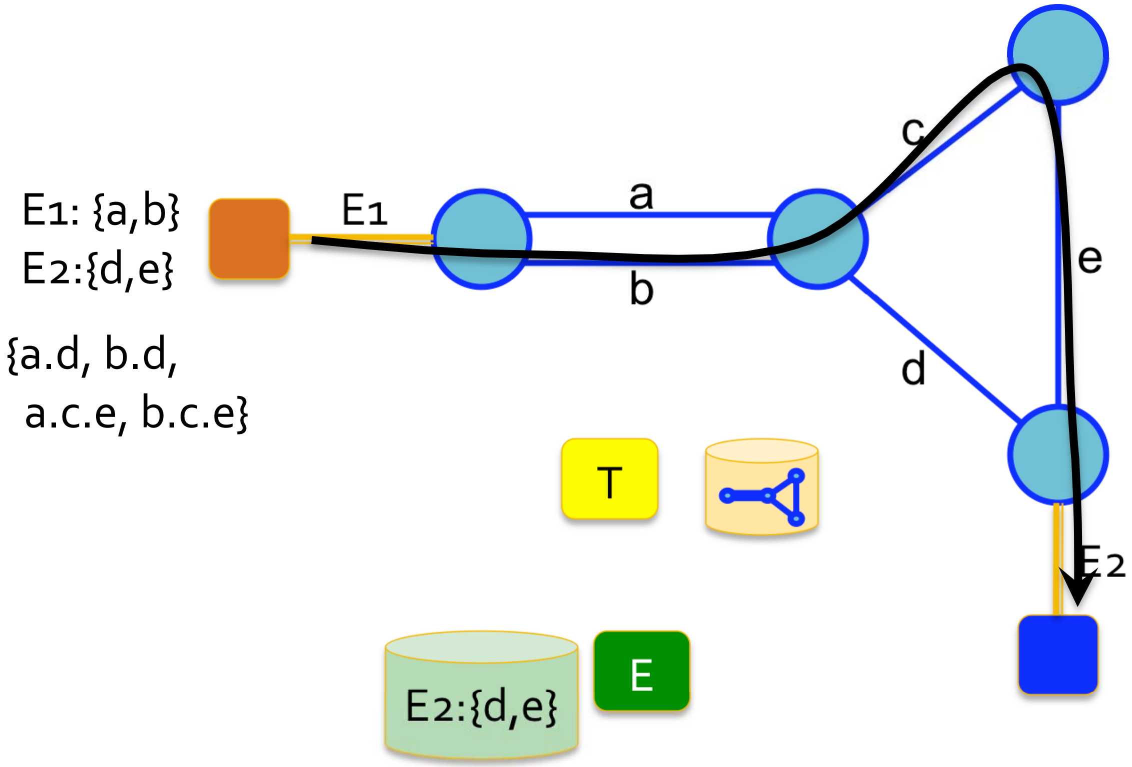
To send to E_2:
1. Get locator from E2L.

Base case



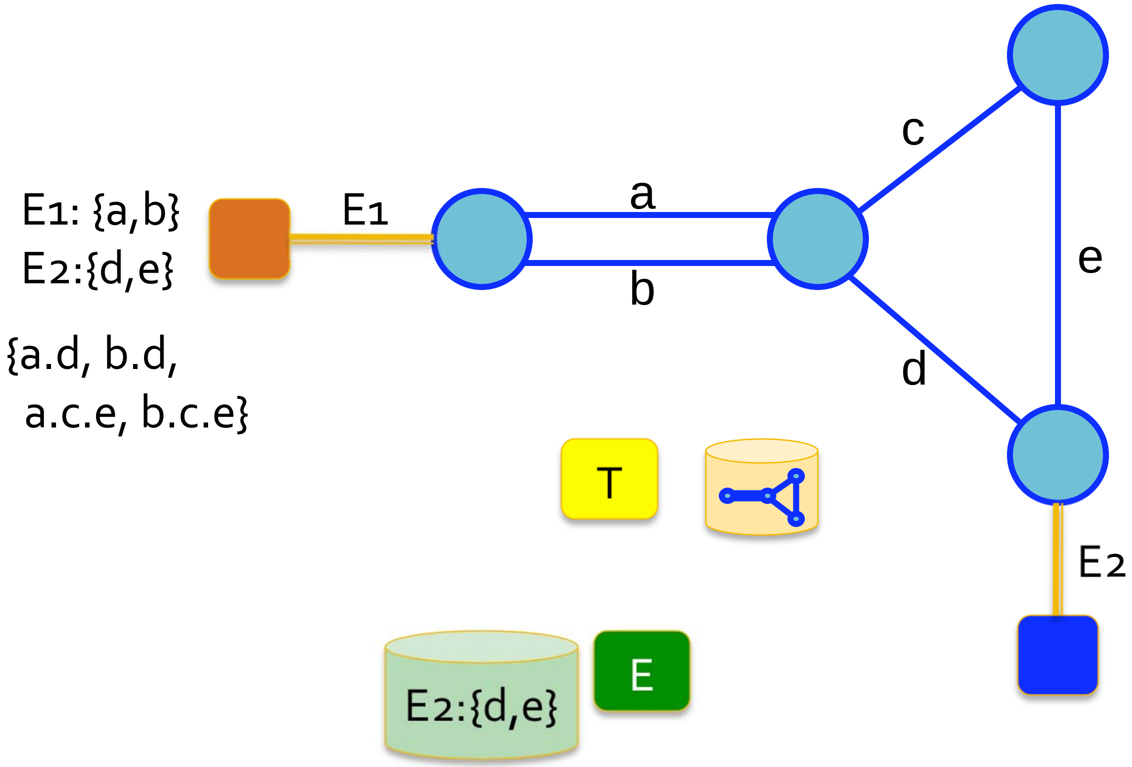
To send to E_2:
1. Get locator from E2L.
2. Ask topology service for paths connecting source and destination locators.

Base case



To send to E_2:
1. Get locator from E2L.
2. Ask topology service for paths connecting source and destination locators.
3. Choose path, transmit.

Base case



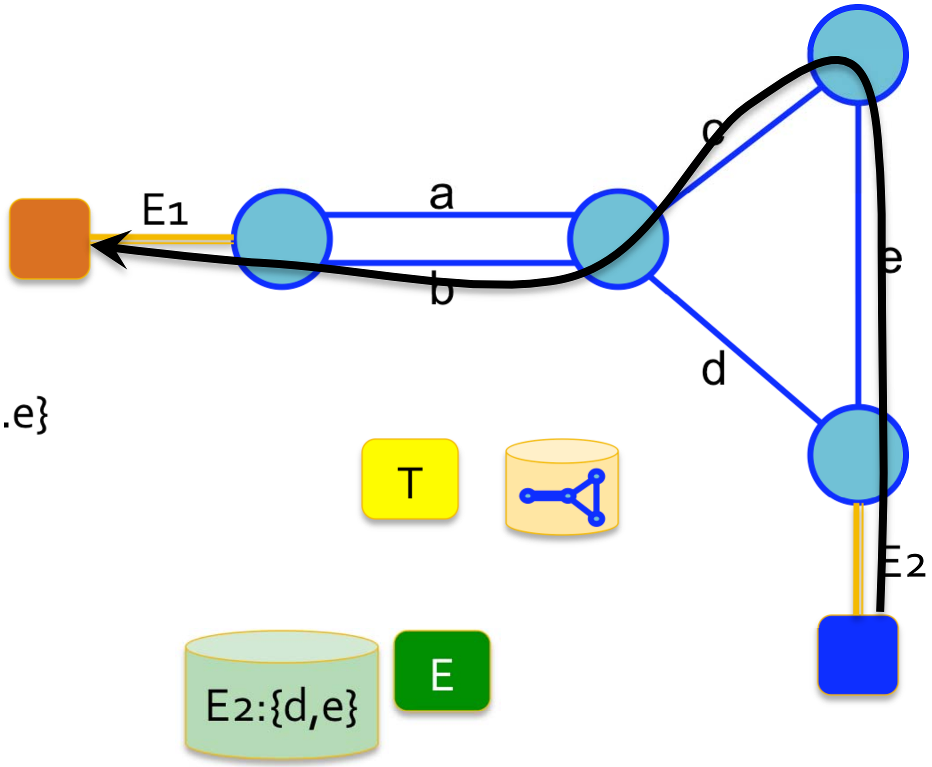
To send to E_2:

1. Get locator from E2L.
2. Ask topology service for paths connecting source and destination locators.
3. Choose path, transmit.
4. Cache paths for later use.

Base case

E1: {a,b}
E2:{d,e}

{a.d, b.d,
a.c.e, b.c.e}



All paths are symmetric. Destination uses reverse path to respond.

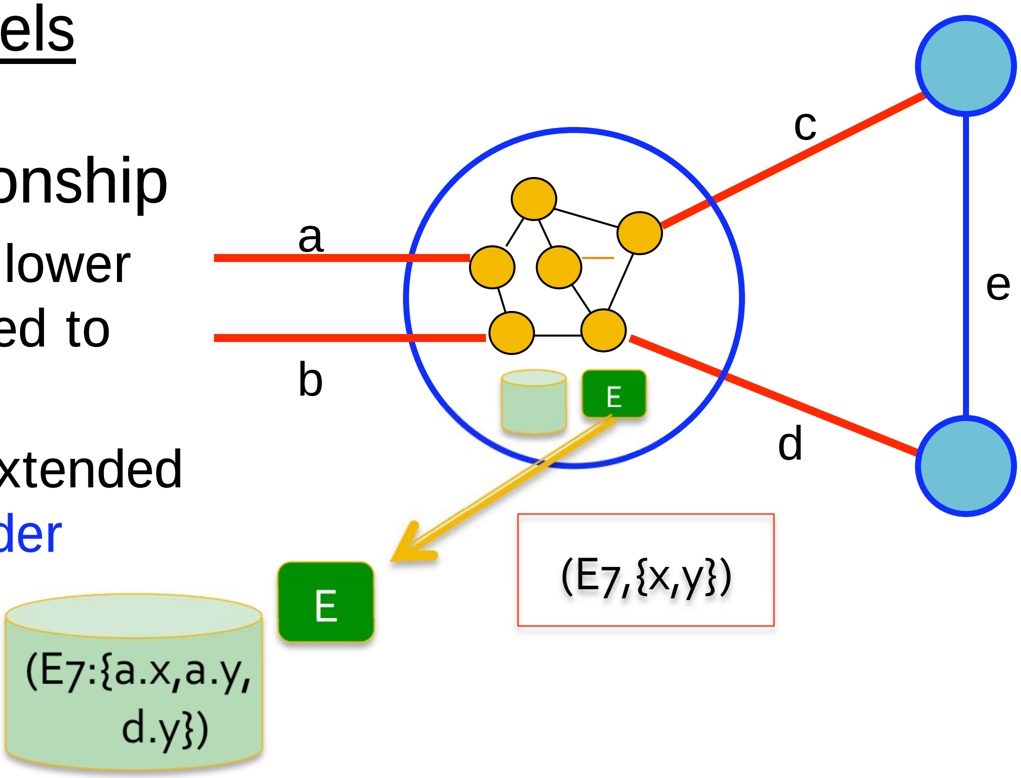
Architecture Features

Simple inductive model

- Inductive step
  - Realms = parts of the network that look like forwarding nodes
    - Border channels mark boundaries where abstraction happens
    - Internal topology info does not cross realm borders
    - Only transit service is advertised outside realm
  - Border channels are visible on both sides of realm boundary
  - Locators
    - E2L service is hierarchical
    - Extend paths associated with each EID as registration propagates up hierarchy

Inductive step

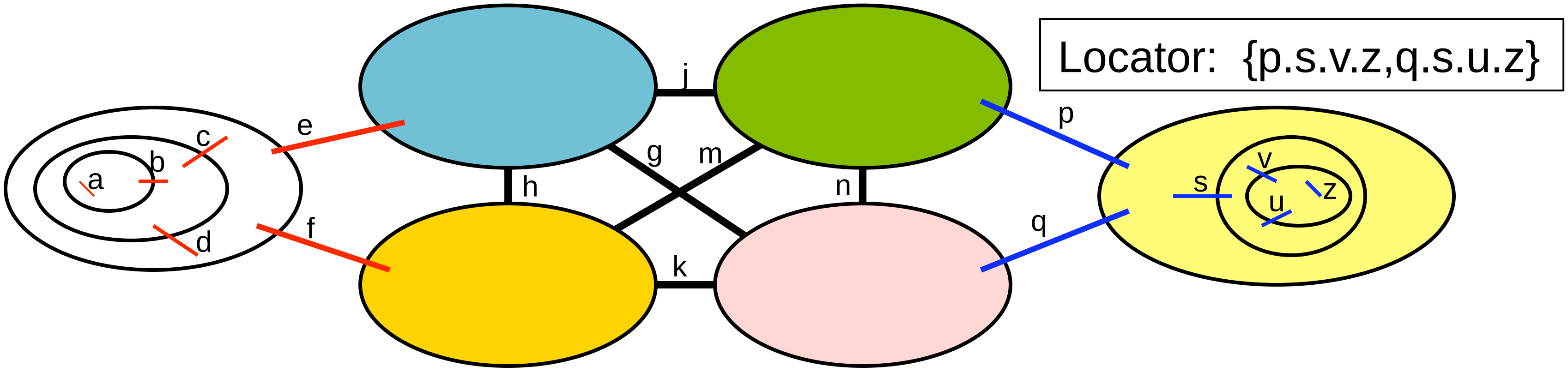
- Border channels are visible at <u>both levels</u>
- E2L services have hierarchical relationship
  - EIDs registered at lower level are propagated to higher level
  - Paths in locators extended according to provider policy



Architecture Features

Paths chosen according to stakeholders' policies

- Destination provider chooses ingress path(s) during locator construction (traffic engineering)
- Source provider chooses egress paths
- Source/Path broker/mediator chooses transit path



Locator: {p.s.v.z,q.s.u.z}

path: a.b.c.e.j.p.s.v.z

Architecture Features

In-band policy enforcement mechanism ("Motivation")

- Forwarding nodes use it to answer the question:
  "Why should I relay this packet?"
- Contains hard-to-forge evidence that
  - the source is a customer of the provider, or
  - the packet pertains to the operation of the network, or
  - a trusted upstream party vouches for the packet

  Possibly also: destination wants to receive it
- Why: enables market competition of transit providers
  [See Platypus, NIRA, MINT, ...]

Architecture Features

In-band recovery from transient failures

- When path breaks, last hop sends notification back to originator (of that segment – origin or border node)
- Select alternative path
- Possibly also inform topology/routing service of outage
- Failure affects exactly those flows using the failed component

What we give up

- Paths are not transferable, only EIDs
- Small headers
  - Path segment: ~10^2 bytes, Motivation: ~10^2 bytes
... times hierarchy depth
  - Bigger MTU needed!
- Extra resolution step (EID -> Locator)

	Header Size	Access	Backbone
1981	10^2 bits	10^4 bits/sec	10^6 bits/sec
Today	10^5 bits	10^7-10^8 bits/sec	10^{10} bits/sec

UNIVERSITY OF KENTUCKY

Implementation Challenges

- Build a global EID-to-Locator system that runs on "bare hardware"
  - Present approach: hierarchical DHT [see Canon]
- Scalable, secure motivation system
  - Current approach: hierarchical delegation (time-bounded), single hash verification
  - Crypto hash sizes should not be wired into architecture

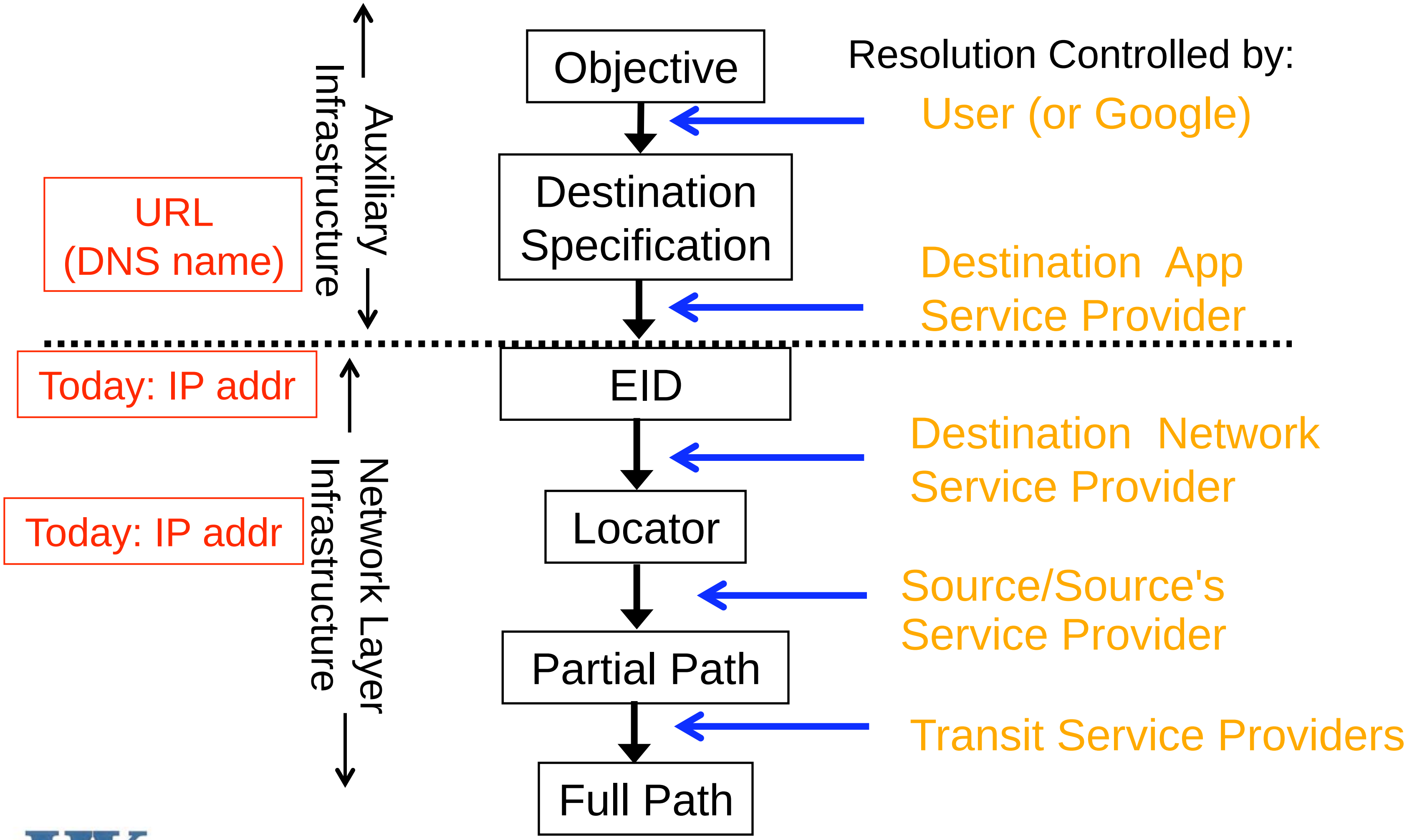
    … but will they grow without limit? Continue to exist?

Summary

- Pomo routing and forwarding: minimalist network layer
- Channel-oriented paradigm, simple inductive structure
  - Top and bottom levels use same mechanisms
- Self-configuring EIDs w/ EID-locator resolution
- Distributed, hierarchical, policy-compliant path selection
- Explicit data-plane policy enforcement
- Endpoint control over visibility/reachability
- Implementation status: prototype, Emulab evaluation

  (not ready for distribution)

Usage: Resolution Steps



Resolution Controlled by:

Objective

User (or Google)

Destination Specification

Destination App Service Provider

EID

Destination Network Service Provider

Locator

Source/Source's Service Provider

Partial Path

Transit Service Providers

Full Path

Auxiliary Infrastructure

URL (DNS name)

Network Layer Infrastructure

Today: IP addr

Today: IP addr