Scalable Routing for Tag-Based Information-Centric Networking

ICN 2014

Michele Papalini (University of Lugano)
Antonio Carzaniga (University of Lugano)
Koorosh Khazaei (University of Lugano)
Alexander L. Wolf (Imperial College London)
Our Vision
Our Vision

Addressing Information
Addressing Information

descriptors:

{ICN14, conference, Paris}
Our Vision

Addressing Information

descriptors:

{ICN14, conference, Paris}

1. expressive (> names)
Our Vision

Addressing Information

descriptors:

\{ICN14, conference, Paris\}

1. expressive (\(\geq\) names)

2. user-defined
Our Vision

Addressing Information

descriptors:

{ICN14, conference, Paris}

1. expressive (> names)

2. user-defined

Rich Service Model

network

FIBs

(register: predicate)
Our Vision

Addressing Information

descriptors:

\{ICN14, conference, Paris\}

1. expressive (> names)

2. user-defined

Rich Service Model

network

\langle register: predicate \rangle

\langle notification: descriptor data \rangle

PUSH

FIBs
Our Vision

Addressing Information

descriptors:

{ICN14, conference, Paris}

1. expressive (> names)
2. user-defined

Rich Service Model

network

FIBs

PUSH

〈register: predicate〉

〈notification: descriptor data〉

PULL

〈request: descriptor〉

〈reply: descriptor data〉
Content Descriptor
Architectural Principles: Addressing Scheme

D0: \{ICN14, conference\}

set of tags
Architectural Principles: Addressing Scheme

Content Descriptor

D0: \{ICN14, conference\}

set of tags

matching: subset relation (⊆)
Architectural Principles: Addressing Scheme

Set of tags

D0: \{ICN14, conference\}

Matching: subset relation (\subseteq)

D1: \{ICN14, conference, banquet\}

D2: \{ICN14, paper, routing\}
 Architectural Principles: Addressing Scheme

Content Descriptor

D0: \{ICN14, conference\}

matching: subset relation (\(\subseteq\))

D1: \{ICN14, conference, banquet\}

✓

D2: \{ICN14, paper, routing\}
Architectural Principles: Addressing Scheme

Content Descriptor

D0: \{ICN14, conference\}

matching: \textbf{subset} relation ($\subseteq$)

D1: \{ICN14, conference, banquet\} \quad \checkmark

D2: \{ICN14, paper, routing\} \quad \times
Architectural Principles: Addressing Scheme

Function: Describe content, meta-data, user interests, ...

Characteristics: True content-based, User-defined, Location independent

Issues: Not a unique identifier, Addressing objects/blocks, High-throughput forwarding?
Function: Describe content, meta-data, user interests, ...
Architectural Principles: Addressing Scheme

**Function**: Describe content, meta-data, user interests, …

**Characteristics**: True content-based
User-defined
Location independent
Architectural Principles: Addressing Scheme

**Function:** Describe content, meta-data, user interests, . . .

**Characteristics:** True content-based
User-defined
Location independent

**Issues:** Not a unique identifier
Addressing objects/blocks?
High-throughput forwarding?
Architectural Principles: Addressing Scheme

Packet Header

Content Descriptor

Content Identifier

Host Locator

...
Architectural Principles: Addressing Scheme

Packet Header
- Content Descriptor
- Content Identifier
- Host Locator
- ...

Function: Identify a specific object/block
Characteristics: Location independent, Globally unique
**Architectural Principles: Addressing Scheme**

**Packet Header**

- Content Descriptor
- Content Identifier
- Host Locator
- ...

**Function:** Identify a specific object/block
Architectural Principles: Addressing Scheme

Function: Identify a specific object/block

Characteristics: Location independent
                Globally unique
**Architectural Principles: Addressing Scheme**

- **Function**: Locate and route to a specific host
- **Characteristics**: Network-defined

Diagram:

- Packet Header
  - Content Descriptor
  - Content Identifier
  - Host Locator
  - ...
Architectural Principles: Addressing Scheme

Function: Locate and route to a specific host
Architectural Principles: Addressing Scheme

**Function**: Locate and route to a specific host

**Characteristics**: Network-defined
Architectural Principles: Addressing Scheme

Packet Header

- Content Descriptor
- Content Identifier
- Host Locator
- ...

Network  Transport  Application
Architectural Principles: Addressing Scheme

Network
Content-based forwarding

Transport
—

Application
Describe content

Packet Header

- Content Descriptor
- Content Identifier
- Host Locator
- ...

Architectural Principles: Addressing Scheme

Packet Header
- Content Descriptor
- Content Identifier
- Host Locator
-...

<table>
<thead>
<tr>
<th>Network</th>
<th>Transport</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content-based forwarding</td>
<td>Stream ID, object ID, blocks</td>
<td>Describe content</td>
</tr>
<tr>
<td>Caching</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>
Architectural Principles: Addressing Scheme

**Packet Header**
- Content Descriptor
- Content Identifier
- Host Locator
- ...

<table>
<thead>
<tr>
<th>Network</th>
<th>Transport</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content-based forwarding</td>
<td>Stream ID, object ID, blocks</td>
<td>Describe content</td>
</tr>
<tr>
<td>Caching</td>
<td>End-point address</td>
<td>—</td>
</tr>
<tr>
<td>Locator-based forwarding</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>
Our Vision

Address Information

Packet Header
Content Descriptor
Content Identifier
Host Locator
...

Rich Service Model

network

⟨register: predicate⟩
⟨notification: descriptor data⟩
⟨request: descriptor⟩
⟨reply: descriptor data⟩

PUSH

PULL
Routing protocol based on trees

- Supports rich “push/pull” primitives
- Supports both descriptors and very efficient locators
Routing protocol based on trees

- Supports rich “push/pull” primitives
- Supports both descriptors and very efficient locators
- Heuristics to build trees
Routing protocol based on trees

- Supports rich “push/pull” primitives
- Supports both descriptors and very efficient locators
- Heuristics to build trees
- Hierarchical (inter and intra AS) routing
Routing protocol based on trees
- Supports rich “push/pull” primitives
- Supports both descriptors and very efficient locators
- Heuristics to build trees
- Hierarchical (inter and intra AS) routing

Data structure for RIBs
- Efficient compression
- Maintenance algorithm
Routing protocol based on trees
  ▶ Supports rich “push/pull” primitives
  ▶ Supports both descriptors and very efficient locators
  ▶ Heuristics to build trees
  ▶ Hierarchical (inter and intra AS) routing

Data structure for RIBs
  ▶ Efficient compression
  ▶ Maintenance algorithm

Scalability analysis
Routing on One Tree
T, next-hop

P, w

T, green, c

p, g ∨ p, h

T, green, f

p, f ∨ p, j ∨ p, k

T, green, e

p, e ∨ p, a ∨ p, d ∨ p, i
Descriptor-Based Forwarding

<table>
<thead>
<tr>
<th>Tree $T$, next-hop $w$</th>
<th>Predicate $P_{T,w}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{\text{green}}, c$</td>
<td>$p_c \lor p_g \lor p_h$</td>
</tr>
<tr>
<td>$T_{\text{green}}, f$</td>
<td>$p_f \lor p_j \lor p_k$</td>
</tr>
<tr>
<td>$T_{\text{green}}, e$</td>
<td>$p_e \lor p_a \lor p_d \lor p_i$</td>
</tr>
</tbody>
</table>
Descriptor-Based Forwarding

<table>
<thead>
<tr>
<th>Tree $T$, next-hop $w$</th>
<th>Predicate $P_{T,w}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{\text{green}, c}$</td>
<td>$p_c \lor p_g \lor p_h$</td>
</tr>
<tr>
<td>$T_{\text{green}, f}$</td>
<td>$p_f \lor p_j \lor p_k$</td>
</tr>
<tr>
<td>$T_{\text{green}, e}$</td>
<td>$p_e \lor p_a \lor p_d \lor p_i$</td>
</tr>
</tbody>
</table>

FIB router $b$

$D \supseteq p_g$
Locator-Based Forwarding

TZ-labels

Thorup M. and Zwick U., Compact Routing Schemes, SPAA 2001
Locator-Based Forwarding

TZ-labels
Locator-Based Forwarding

TZ-labels

[Thorup M. and Zwick U., Compact Routing Schemes, SPAA 2001]
Locator-Based Forwarding

TZ-labels

[Thorup M. and Zwick U., Compact Routing Schemes, SPAA 2001]
Locator-Based Forwarding

**TZ-labels**


- Distributed labeling algorithm (DFS)
Locator-Based Forwarding

TZ-labels

- Distributed labeling algorithm (DFS)
- Compact labels
  - Label size: 46 bits
  - FIB size: 1 label
Locator-Based Forwarding

TZ-labels


- Distributed labeling algorithm (DFS)
- Compact labels
  - Label size: 46 bits
  - FIB size: 1 label
- Efficient forwarding
  - Matching: 10 CPU cycles
  - Throughput: 250M pkt/sec
    (on this laptop!)
Communication Flow

Content-based matching, using descriptor D

Locator-based forwarding, using TZ-label

Caching, using Obj-ID
Communication Flow

request

\[ D \supseteq p_c \]

src: \( TZ_i \)
Content-based matching, using descriptor $D$
Communication Flow

- **FIB router**
- **T**
- **w**
- **predicate**

```
P(T, w) = \begin{cases} 
  \text{green}, & p_e \\
  \text{green}, & p_i \\
  \text{green}, & p_f \\
  \text{green}, & p_j \\
  \text{green}, & p_k \\
  \end{cases}
```

- Request
- **Obj-ID**
- **dst:** TZ
- **src:** TZ

- **Locator-based forwarding**, using TZ-label

- **Caching**, using Obj-ID

- **Reply**
  - **Obj-ID**
  - **dst:** TZ
  - **src:** TZ

- **Data**
Locator-based forwarding, using TZ-label $TZ_i$
Locator-based forwarding, using TZ-label $TZ_c$
Caching, using Obj-ID
Issues with a Single Tree

Higher latency

Lower throughput
Issues with a Single Tree

Higher latency
Issues with a Single Tree

Higher latency

Lower throughput
Multiple Trees

Low latency and high throughput (in expectation)

How do we build the trees?
Multiple Trees

Low latency and high throughput (in expectation)

How do we build the trees?
Multiple Trees

Low latency and high throughput (in expectation)

\[
d_G(i, j) = 1 \\
d_{T_{\text{green}}}(i, j) = 4
\]
Multiple Trees

Low latency and high throughput (in expectation)

How do we build the trees?
Multiple Trees

Low latency and high throughput (in expectation)

\[ d_G(i, j) = 1 \]
\[ d_{T_{\text{green}}}(i, j) = 4 \]
\[ d_{T_{\text{red}}}(i, j) = 1 \]
\[ E[d_T(i, j)] = 2.5 \]
Multiple Trees

Low latency and high throughput (in expectation)

How do we build the trees?

\[
\begin{align*}
    d_G(i, j) &= 1 \\
    d_{T_{\text{green}}}(i, j) &= 4 \\
    d_{T_{\text{red}}}(i, j) &= 1 \\
    E[d_T(i, j)] &= 2.5
\end{align*}
\]
Building Trees: Latency Only

Diagram of a network with nodes labeled a, b, c, d, e, f, g, h, i, j, k, and connections between them.
Building Trees: Latency Only

Diagram of a network with nodes labeled a, b, c, d, e, f, g, h, i, j, and k.
Building Trees: Latency Only

Diagram of a network with nodes labeled a, b, c, d, e, f, g, h, i, j, and k.
Building Trees: Latency and Congestion
Building Trees: Latency and Congestion
Building Trees: Latency and Congestion

![Graph](image)
Building Trees: Latency and Congestion
Building Trees: Latency and Congestion
Building Trees: Latency and Congestion
Routing Over Multiple Trees: Evaluation
Expected Stretch, AS-level Topology

N: 42112  E: 118040

min, 1%, 50%, 99%, max

L

LC

50% of the nodes experience no congestion

Expected Congestion, AS-level Topology

LC generates less congestion than L
Expected Stretch, AS-level Topology

The expected stretch and congestion for the AS-level topology are shown in the graph. The graph compares the stretch for different numbers of trees using LC and L approaches. The x-axis represents the number of trees (2, 4, 8, 16, 32), and the y-axis represents the stretch. The line with orange dots represents the LC approach, and the line with white dots represents the L approach. The graph indicates that LC generates less congestion, and 50% of the nodes experience no congestion. The dataset size is N:42112, E:118040.
Expected Stretch, AS-level Topology

N: 42112 E: 118040

min, 1%, 50%, 99%, max

L

LC

1% of the nodes experience no congestion.
### Expected Stretch, AS-level Topology

<table>
<thead>
<tr>
<th>Number of Trees</th>
<th>Stretch</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>16</td>
<td>7</td>
</tr>
<tr>
<td>32</td>
<td>9</td>
</tr>
</tbody>
</table>

- **min, 1%, 50%, 99%, max**

- N: 42112  E: 118040

LC generates less congestion than L

50% of the nodes experience no congestion

---

**Diagram:**

- Y-axis: Stretch
- X-axis: Number of Trees
- Comparison between L and LC
- Legend: L (light gray), LC (dark gray)
- Data range: 1 to 11
- Number of trees: 2, 4, 8, 16, 32

---

**Notes:**

- **Stretch Distribution:**
  - Min: 1
  - 1%: 3
  - 50%: 7
  - 99%: 9
  - Max: 11

- **Network Information:**
  - N: 42112
  - E: 118040

---

**Conclusion:**

- LC reduces congestion compared to L.
- Approximately 50% of nodes experience no congestion with LC.
**Expected Stretch, AS-level Topology**

- **LC** generates less congestion than **L**
- **50%** of the nodes experience no congestion

---

**Expected Congestion, AS-level Topology**

- **LC** generates less congestion than **L**
- **50%** of the nodes experience no congestion

- **N**: 42,112
- **E**: 118,040
Hierarchical Multi-Tree Routing
The diagram illustrates the connectivity between two autonomous systems (AS1 and AS2) through gateways (G1 and G2). The gateways have TZ-labels, which are used for content-based and locator-based forwarding. The paths between the gateways and the ASes are color-coded to indicate different types of connectivity: global/local trees, AS connectivity, and TZ-labels of gateways.
local router
- local trees
- TZ-labels of gateways

D ⊇ p B

content-based forwarding
locator-based forwarding

T AS1, TZ A

T green, TZ AS1

T AS2, TZ G1

T green, TZ AS2

data

Obj-ID

dst stack

src stack

T AS1

T AS2

G1

G2
content-based forwarding
locator-based forwarding
request

\[ D \supseteq p_B \]

dst stack

src stack

\[ T_{AS1}, TZ_A \]

content-based forwarding
locator-based forwarding

\[ D \supseteq p_B \]
request

\[ D \supseteq p_B \]

dst stack

src stack

\[ T_{green}, TZ_{AS1} \]

\[ T_{AS1}, TZ_A \]

content-based forwarding

locator-based forwarding
content-based forwarding
locator-based forwarding
content-based forwarding
locator-based forwarding
content-based forwarding
locator-based forwarding
content-based forwarding
locator-based forwarding
gateway
global/local trees
AS connectivity
TZ-labels of gateways
local router
local trees
TZ-labels of gateways

\[ D \supseteq p \]

content-based forwarding
locator-based forwarding

\( T_{green}, TZ_{AS1} \)
\( T_{AS1}, TZ_A \)
\( T_{green}, TZ_{AS2} \)
\( T_{AS2}, TZ_B \)

reply
Obj-ID
dst stack
src stack
data

AS1
AS2
G1
G2
A
B

Obj-ID
dst stack
src stack
data

content-based forwarding
locator-based forwarding
content-based forwarding
locator-based forwarding
RIB Representation and Maintenance
Neighbors tree T neighbors

e, f, c
tz-labels

tz green

tz red

Rib tree T, next-hop w predicate P

T green, c p c ∨ p g ∨ p h

T green, f p f ∨ p j ∨ p k

T red, c p c ∨ p g ∨ p h

T red, e p e ∨ p a ∨ p d ∨ p i

Compress and update the RIB
Neighbors tree $T$ neighbors $T$ green $e, f, c$ $T$ red $e, c$

TZ-labels $TZ$ green $b$ $TZ$ red $b$

RIB tree $T$, next-hop $w$ predicate $P$ $T$, $w$ green $c$ $p_c \lor p_g \lor p_h$ $T$ green $f$ $p_f \lor p_j \lor p_k$ $T$ red $c$ $p_c \lor p_g \lor p_h$ $T$ red $e$ $p_e \lor p_d \lor p_i \lor p_j \lor p_k$

Compress and update the RIB
Neighbors

<table>
<thead>
<tr>
<th>tree $T$</th>
<th>neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{\text{green}}$</td>
<td>e, f, c</td>
</tr>
<tr>
<td>$T_{\text{red}}$</td>
<td>e, c</td>
</tr>
</tbody>
</table>
Neighbors

<table>
<thead>
<tr>
<th>tree $T$</th>
<th>neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{\text{green}}$</td>
<td>e, f, c</td>
</tr>
<tr>
<td>$T_{\text{red}}$</td>
<td>e, c</td>
</tr>
</tbody>
</table>

TZ-labels

$T_{Z_b}^{\text{green}}$

$T_{Z_b}^{\text{red}}$

Compress and update the RIB
Neighbors

<table>
<thead>
<tr>
<th>tree $T$</th>
<th>neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{green}$</td>
<td>e, f, c</td>
</tr>
<tr>
<td>$T_{red}$</td>
<td>e, c</td>
</tr>
</tbody>
</table>

TZ-labels

| $T_{green}$, c |
| $T_{green}, f$ |
| $T_{green}, e$ |

| $T_{red}, c$ |
| $T_{red}, e$ |

RIB

<table>
<thead>
<tr>
<th>tree $T$, next-hop $w$</th>
<th>predicate $P_{T,w}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{green}, c$</td>
<td>$p_c \lor p_g \lor p_h$</td>
</tr>
<tr>
<td>$T_{green}, f$</td>
<td>$p_f \lor p_j \lor p_k$</td>
</tr>
<tr>
<td>$T_{green}, e$</td>
<td>$p_e \lor p_a \lor p_d \lor p_i$</td>
</tr>
<tr>
<td>$T_{red}, c$</td>
<td>$p_c \lor p_g \lor p_h$</td>
</tr>
<tr>
<td>$T_{red}, e$</td>
<td>$p_a \lor p_d \lor p_e \lor p_f \lor p_i \lor p_j \lor p_k$</td>
</tr>
</tbody>
</table>
Neighbors

<table>
<thead>
<tr>
<th>tree $T$</th>
<th>neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{green}$</td>
<td>e, f, c</td>
</tr>
<tr>
<td>$T_{red}$</td>
<td>e, c</td>
</tr>
</tbody>
</table>

TZ-labels

<table>
<thead>
<tr>
<th>$TZ_b^{green}$</th>
<th>$TZ_b^{red}$</th>
</tr>
</thead>
</table>

RIB

<table>
<thead>
<tr>
<th>tree $T$, next-hop $w$</th>
<th>predicate $P_{T,w}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{green}$, c</td>
<td>$p_c \lor p_g \lor p_h$</td>
</tr>
<tr>
<td>$T_{green}$, f</td>
<td>$p_f \lor p_j \lor p_k$</td>
</tr>
<tr>
<td>$T_{green}$, e</td>
<td>$p_e \lor p_a \lor p_d \lor p_i$</td>
</tr>
<tr>
<td>$T_{red}$, c</td>
<td>$p_c \lor p_g \lor p_h$</td>
</tr>
<tr>
<td>$T_{red}$, e</td>
<td>$p_a \lor p_d \lor p_e \lor p_f \lor p_i \lor p_j \lor p_k$</td>
</tr>
</tbody>
</table>

Compress and update the RIB
Compression
(1) Subset Aggregation
(1) Subset Aggregation

\{ICN14, Paris, routing, paper\}

\{hotel, Paris, ICN14\}

\{Paris, social, event, ICN14\}

\{ICN14, Paris\}
(1) Subset Aggregation

\{ICN14, Paris, routing, paper\}
\{hotel, Paris, ICN14\}
\{Paris, social, event, ICN14\}
\{ICN14, Paris\}
(2) Bloom Filters
(2) Bloom Filters

\{ICN14, Paris\}
(2) Bloom Filters

**our setting**: $k=7$ and $m=192$ (24B)
Compression

(3) Index by Descriptors
## Compression

### (3) Index by Descriptors

<table>
<thead>
<tr>
<th>tree $T$, next-hop $w$</th>
<th>predicate $P_{T,w}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{green}$, c</td>
<td>00100101</td>
</tr>
<tr>
<td></td>
<td>01010000</td>
</tr>
<tr>
<td></td>
<td>01000001</td>
</tr>
<tr>
<td>$T_{green}$, f</td>
<td>00100100</td>
</tr>
<tr>
<td></td>
<td>01010000</td>
</tr>
<tr>
<td></td>
<td>01100100</td>
</tr>
<tr>
<td>$T_{green}$, e</td>
<td>00010000</td>
</tr>
<tr>
<td></td>
<td>10000101</td>
</tr>
<tr>
<td>$T_{red}$, c</td>
<td>00100101</td>
</tr>
<tr>
<td></td>
<td>01000001</td>
</tr>
<tr>
<td>$T_{red}$, e</td>
<td>00010000</td>
</tr>
<tr>
<td></td>
<td>10000101</td>
</tr>
<tr>
<td></td>
<td>00100100</td>
</tr>
</tbody>
</table>
## Compression

### (3) Index by Descriptors

<table>
<thead>
<tr>
<th>RIB</th>
<th>compressed RIB</th>
</tr>
</thead>
<tbody>
<tr>
<td>tree $T$, next-hop $w$</td>
<td>descriptor $D$</td>
</tr>
<tr>
<td>$T_{\text{green}}, c$</td>
<td>$(T_{\text{green}}, c)$</td>
</tr>
<tr>
<td>$T_{\text{green}}, f$</td>
<td>$(T_{\text{green}}, f)$</td>
</tr>
<tr>
<td>$T_{\text{green}}, e$</td>
<td>$(T_{\text{green}}, e)$</td>
</tr>
<tr>
<td>$T_{\text{red}}, c$</td>
<td>$(T_{\text{red}}, c)$</td>
</tr>
<tr>
<td>$T_{\text{red}}, e$</td>
<td>$(T_{\text{red}}, e)$</td>
</tr>
</tbody>
</table>
(3) Index by Descriptors

### RIB

<table>
<thead>
<tr>
<th>tree ( T ), next-hop ( w )</th>
<th>predicate ( P_{T,w} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T_{\text{green}}, c )</td>
<td>00100101 ( \bullet )</td>
</tr>
<tr>
<td></td>
<td>01010000 ( \bullet )</td>
</tr>
<tr>
<td></td>
<td>01000001 ( \bullet )</td>
</tr>
<tr>
<td>( T_{\text{green}}, f )</td>
<td>00100100 ( \bullet )</td>
</tr>
<tr>
<td></td>
<td>01010000 ( \bullet )</td>
</tr>
<tr>
<td></td>
<td>01100100 ( \bullet )</td>
</tr>
<tr>
<td>( T_{\text{green}}, e )</td>
<td>00010000 ( \bullet )</td>
</tr>
<tr>
<td></td>
<td>10000101 ( \bullet )</td>
</tr>
<tr>
<td>( T_{\text{red}}, c )</td>
<td>00100101 ( \bullet )</td>
</tr>
<tr>
<td></td>
<td>01000001 ( \bullet )</td>
</tr>
<tr>
<td>( T_{\text{red}}, e )</td>
<td>00010000 ( \bullet )</td>
</tr>
<tr>
<td></td>
<td>10000101 ( \bullet )</td>
</tr>
<tr>
<td></td>
<td>00100100 ( \bullet )</td>
</tr>
</tbody>
</table>

### Compressed RIB

<table>
<thead>
<tr>
<th>descriptor ( D )</th>
<th>tree ( T ), next-hop ( w )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{red} )</td>
<td>( (T_{\text{green}}, e), (T_{\text{red}}, e) )</td>
</tr>
<tr>
<td>( \text{green} )</td>
<td>( (T_{\text{green}}, c), (T_{\text{red}}, c) )</td>
</tr>
<tr>
<td>( \text{green} )</td>
<td>( (T_{\text{green}}, f), (T_{\text{green}}, e) )</td>
</tr>
<tr>
<td>( \text{green} )</td>
<td>( (T_{\text{red}}, c) )</td>
</tr>
<tr>
<td>( \text{green} )</td>
<td>( (T_{\text{green}}, f), (T_{\text{red}}, e) )</td>
</tr>
<tr>
<td>( \text{green} )</td>
<td>( (T_{\text{red}}, e) )</td>
</tr>
<tr>
<td>( \text{green} )</td>
<td>( (T_{\text{green}}, e) )</td>
</tr>
<tr>
<td>( \text{green} )</td>
<td>( (T_{\text{red}}, e) )</td>
</tr>
<tr>
<td>( \text{green} )</td>
<td>( (T_{\text{green}}, f) )</td>
</tr>
<tr>
<td>( \text{green} )</td>
<td>( (T_{\text{red}}, e) )</td>
</tr>
</tbody>
</table>
RIB as PATRICIA Trie and Updates

Diagram:

- Nodes represent Trie nodes.
- Edges indicate path through the Trie.
- Labels on edges represent bit values.
- Node labels show transitions from parent to child.
RIB as PATRICIA Trie and Updates

- Minimal size
RIB as PATRICIA Trie and Updates

- Minimal size
- Subset/superset check = walk
RIB as PATRICIA Trie and Updates

- Minimal size
- Subset/superset check = walk
- Shortcut by prefix
RIB as PATRICIA Trie and Updates

- Minimal size
- Subset/superset check = walk
- Shortcut by prefix

```
void apply_delta (map<int,delta> & result, 
delta update, int ifx, int tree) {
    for (filter f : update.remove())
        remove_filter(result, f, ifx, tree);
    for (filter f : update.add())
        add_filter(result, f, ifx, tree);
}

void add_filter (map<int,delta> & result, 
    filter f, int ifx, int tree) {
    if (!exists_subset_of(f, ifx, tree)) {
        add(f, ifx, tree);
        remove_supersets_of(f, ifx, tree);
        for (int i : interfaces[tree])
            if (i != ifx && no_subsets_on_other_ifx(f, i, tree))
                result[i].additions.add(f); }
}

void remove_filter (map<int,delta> & result, 
    filter f, int ifx, int tree) {
    if (exists_filter(f, ifx, tree)) {
        remove(f, ifx, tree);
        for (int i : interfaces[tree])
            if (i != ifx && no_subsets_on_other_ifx(f, i, tree))
                result[i].additions.add(supersets_of(f, tree)); }
}
```

Evaluation: 500 descriptors/sec
RIB Compression (Aggregation): Evaluation
Gateway RIB Sizes
50M users, 8 trees

Required Memory (GB)

L
LC

8 trees
1 tree

Telstra  Sprint  Verio  Tiscali  Level3  ATT
AS1221  AS1239  AS2914  AS3257  AS3356  AS7018
Gateway RIB Sizes
50M users, 8 trees

aggregation across trees is effective
Gateway RIB Sizes
50M users, 8 trees

aggregation across trees is effective
8 trees < 2 × 1 tree (intra-AS)
Required Memory (GB)

Users (Millions)

subset aggregation is effective

2.5B users

∼ 8GB (projection)
Single AS
50–500M users, 1 tree

subset aggregation is effective
subset aggregation is effective
2.5B users $\sim 8$GB (projection)
Conclusion and Future Work
Scalable tree-based routing protocol
Conclusion and Future Work

- Supports rich “push/pull” communication

- Scalable tree-based routing protocol

- Approximates the Internet with a few trees
- Aggregates descriptors within and across trees
- Supports fast locator-based forwarding, content-based forwarding
- Policy-based heuristics for trees
- High throughput content-based forwarding

Comparison and interaction with other architectures

Use-case scenarios and applications
Supports rich “push/pull” communication

Approximates the Internet with a few trees

Scalable tree-based routing protocol
Conclusion and Future Work

- Supports rich “push/pull” communication
- Approximates the Internet with a few trees
- Aggregates descriptors within and across trees
- Scalable tree-based routing protocol
- Supports fast locator-based forwarding, content-based forwarding
- Policy-based heuristics for trees
- High throughput content-based forwarding
- Use-case scenarios and applications
- Comparison and interaction with other architectures

Scalable tree-based routing protocol
Conclusion and Future Work

Supports rich “push/pull” communication

Approximates the Internet with a few trees

Aggregates descriptors within and across trees

Supports fast locator-based forwarding, content-based forwarding

Scalable tree-based routing protocol

Comparison and interaction with other architectures

Use-case scenarios and applications

Policy-based heuristics for trees

High throughput content-based forwarding
Conclusion and Future Work

**Scalable tree-based routing protocol**

- Policy-based heuristics for trees
- Approximates the Internet with a few trees
- Supports fast locator-based forwarding, content-based forwarding
- Aggregates descriptors within and across trees
- High throughput content-based forwarding

Use-case scenarios and applications
Comparison and interaction with other architectures

Policy-based heuristics for trees
Approximates the Internet with a few trees
Supports fast locator-based forwarding, content-based forwarding
Aggregates descriptors within and across trees
High throughput content-based forwarding
Conclusion and Future Work

- Policy-based heuristics for trees
- Supports fast locator-based forwarding, content-based forwarding
- Scalable tree-based routing protocol
- Aggregates descriptors within and across trees
- High throughput content-based forwarding

Comparison and interaction with other architectures

Use-case scenarios and applications

Scalable tree-based routing protocol
Conclusion and Future Work

- Policy-based heuristics for trees
- High throughput content-based forwarding
- Use-case scenarios and applications
- Supports fast locator-based forwarding, content-based forwarding

Scalable tree-based routing protocol
Conclusion and Future Work

- Policy-based heuristics for trees
- Supports rich "push/pull" communication
- Approximates the Internet with a few trees
- Aggregates descriptors within and across trees
- Supports fast locator-based forwarding, content-based forwarding
- High throughput content-based forwarding
- Use-case scenarios and applications
- Comparison and interaction with other architectures

Scalable tree-based routing protocol
Scalable Routing for Tag-Based Information-Centric Networking

ICN 2014

code available at:
http://www.inf.usi.ch/carzaniga/maketrees.git