Content-Based Routing

Routing Schemes and Space Complexity

Antonio Carzaniga

Faculty of Informatics
University of Lugano
Switzerland

December 7, 2009

Applications

- System monitoring and management
  - e.g., management of a large data center
- Stream processing
  - e.g., analysis of financial data
- Service discovery
  - e.g., in a "cloud" computing infrastructure
  - e.g., in service-oriented computing
- Distributed simulation
  - e.g., multi-player games

Applications: Not Only Data Centers

- RSS News feeds
- Facebook notifications
- Twitter
- Google Alerts
- eBay Alerts
- . . .
- Facebook "Beacon"
  - Art.com, Blockbuster, Bluefly, CBS Interactive, eBay, ExpoTV, Fandango, Gamefly.com, Kiva, Kongregate, LiveNation, Mercantila, NY Times, Overstock.com, Redlight Mgmt, Seamless Web, Six Apart, STA Travel, TheKnot, Travelocity, Viagogo
- Many other "aggregators"

They want to be the content-based network!

Content-Based Communication

a.k.a. Publish/Subscribe Communication

- Receivers decide what they want to receive—they *subscribe*
  - "what they want" is predicated upon *the content of messages*
- Senders simply send information out—they *publish*
  - no need to address specific addresses or groups
- The system ("broker" or "dispatcher") does the rest
  - it delivers published messages to all interested subscribers

Messages: Example 1

- *A set of attributes* (plus possibly some opaque content)

channel = BBC News | News Front Page | World Edition
link = http://news.bbc.co.uk/go/rss/-/2/hi/default.stm
language = en-gb
title = Berliners celebrate fall of Wall
description = World leaders past and present join thousands of Berliners marking the 20th anniversary of the fall of the Berlin Wall.
link = http://news.bbc.co.uk/2/hi/europe/8349742.stm
pubDate = Mon, 09 Nov 2009 15:52:04 GMT
category = Europe

...

Messages: Example 2

- *A set of "tags"* (plus possibly some opaque content)

art, **design**, **culture**, **gallery**, **museum**, **new york**

Welcome to the MoMA Online Press Office. This site is designed for use by the working press. Here you will find the latest press releases on MoMA's exhibitions, programs, and building complex [...] High-resolution images for publication are available through our password-protected Press Image Center.
If you have any questions, press are welcome to contact the Department of Communications at (212) 708-9431 or pressoffice@moma.org
The Museum of Modern Art, 11 West 53 Street, New York, NY 10019, (212) 708-9400

Predicates: Example 1

- *An expression of attribute constraints*

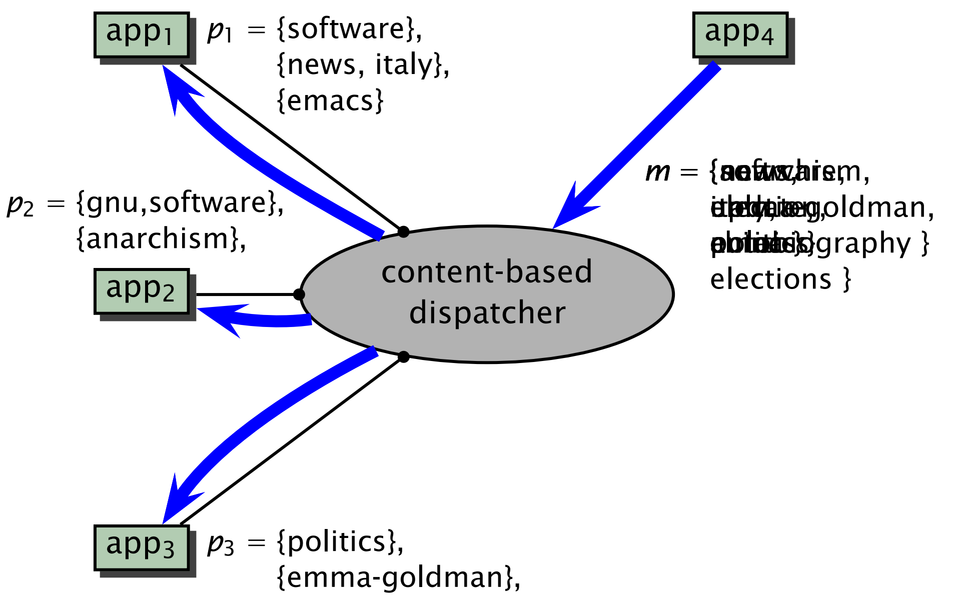
p = **author** = "Glenn Greenwald"
  $\vee$ **channel** = "BBC News"
    $\wedge$ **category** = "Europe"
    $\wedge$ **title** $=^{\text{regex}}$ ".$*$Italy.$*$"
  $\vee$ **management-event** = "disk-failure"
    $\wedge$ **host** = "research.inf.usi.ch"
  $\vee$ **management-event** = "disk-usage"
    $\wedge$ **host** = "research.inf.usi.ch"
    $\wedge$ **space-available** $<$ 1Gb

Predicates: Example 2

- *Sets of "tags"*

p = { **development**, **free-software**, **gnu/linux** },
  { **programming**, **c++** },
  { **tools**, **gnu/linux** },
  { **emacs** },
  { **philosophy**, **anarchism** },
  { **politics**, **emma-goldman** },
  { **chomsky**, **interview** },
  { **lugano**, **grotto** },
  { **lugano**, **pizzeria** }
  { **lugano**, **music**, **live** }

Example: Centralized Architecture



Content-Based Networking

- Content-based communication (a.k.a., publish/subscribe) *designed and implemented as a network service*
  - architecture
  - routing
  - forwarding
  - ...
- Host interface
  - *send-message(m)*
  - *set-predicate(p)*
- Type of service
  - datagram service (i.e., “best effort”)

Outline

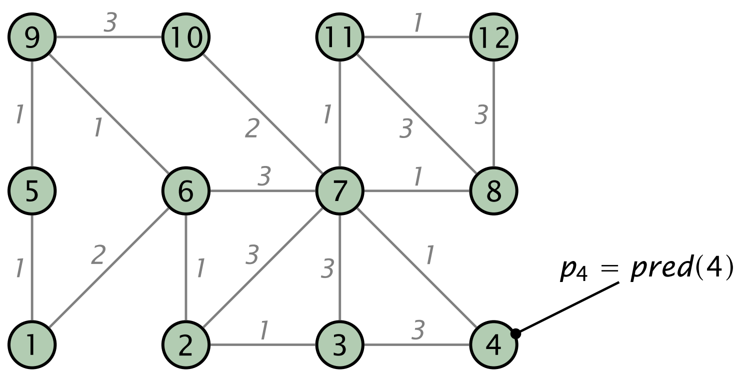
PART I: Routing Schemes

- Routing on a tree cover
- Routing on the whole graph
  - routing with per-receiver unicast information
  - routing with per-sender trees

PART II: Space Complexity

- Analysis for three routing schemes
- Reducing the space complexity
  - encoding (compressing) predicates
  - grouping sources and destinations, folding predicate tables

Content-Based Network Model



- $CBN = (V, E, weight, \mathcal{M}, \mathcal{P}, pred)$
  - $v \in V$ is a processor (host or router)
  - $e \in E$ is a reliable bidirectional communication link
  - $weight : E \to \mathbb{R}$ is a link-weight function
  - $\mathcal{M}$ is a set of *messages*
  - $\mathcal{P}$ is a set of *predicates*; $p \in \mathcal{P}$ is a function $p : \mathcal{M} \to \{0, 1\}$
  - $pred : V \to \mathcal{P}$ associates a processor $v \in V$ to a predicate $p \in \mathcal{P}$

Content-Based Routing *Scheme*

Extension of a standard model by Peleg and Upfal [JACM'89]

- Messages travel in *packets*

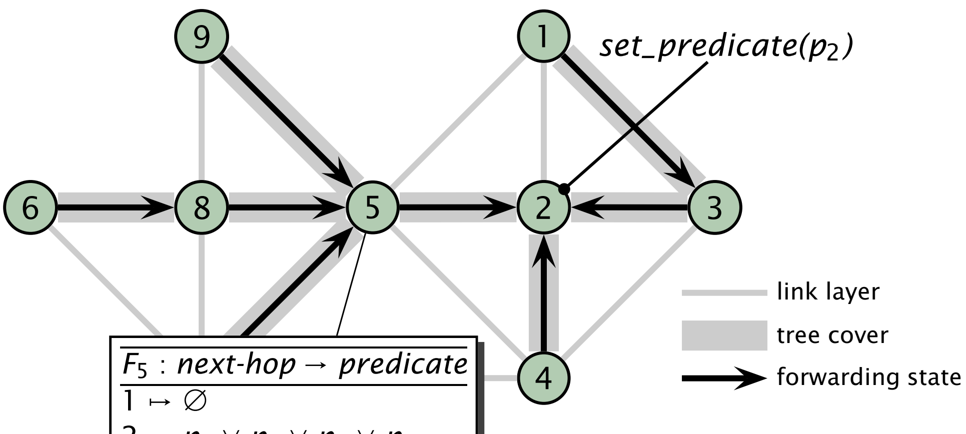
$$c = \langle m, h \rangle$$

  - $m = msg(c)$ is a message; $m \in \mathcal{M}$
  - $h = hdr(c)$ is a *header*; $h \in \mathcal{H}$
  - a scheme defines $\mathcal{H}$, the set of allowable message headers

- Packets are forwarded hop-by-hop from source to destinations

- A routing scheme is a *distributed algorithm* consisting of *per-processor, processor-local routing functions*
  - (re)writing packet headers
  - deciding where to forward a packet

Routing on a Tree Cover



- Routing protocol propagates predicates along the tree cover

- Forwarding state *attracts* messages towards matching predicates

Tree-Cover Scheme

- Headers are used to store the last hop of a packet

$$\mathcal{H} = V$$

$$\textbf{Init}_v(\cdot) = v$$

$$\textbf{Hdr}_u(v) = v$$

- Processor u forwards $c = \langle u, m \rangle$ using F_u

ProcessPacket$_u(c = \langle v, m \rangle)$
1 $\triangleright$ we are at processor u
2 **for** $w \in \{w \neq v | m \in F_u(w)\}$
3 **do** forward $\langle u, m \rangle$ to w

Per-Processor Routing Functions

For each processor v

- *Initial header function*

$$\textbf{Init}_v : \mathcal{M} \to \mathcal{H}$$

given a message m originating at v, $\textbf{Init}_v(m)$ is m's initial header, so v starts by forwarding a packet $c = \langle \textbf{Init}_v(m), m \rangle$

- *Header (rewriting) function*

$$\textbf{Hdr}_v : \mathcal{H} \to \mathcal{H}$$

given a packet $c = \langle h, m \rangle$, v forwards $c' = \langle \textbf{Hdr}_v(h), m \rangle$

- *Forwarding function*

$$\textbf{Fwd}_v : \mathcal{H} \times \mathcal{M} \to \mathbb{P}(\mathit{neighbors}(v))$$

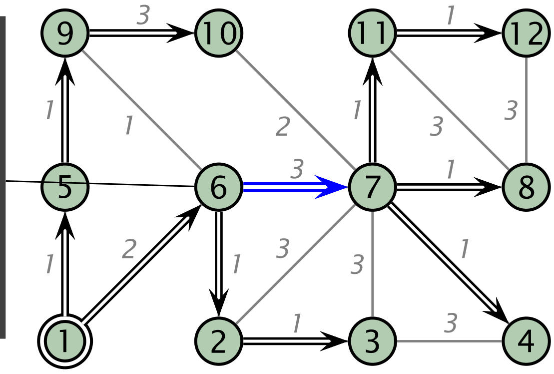
v forwards $\langle h, m \rangle$ to the subset of its neighbors $\textbf{Fwd}_v(h, m)$

Basic Per-Source Forwarding (PSF)

- Idea
  - a per-source spanning tree T_v for each source v
  - annotate edges $e = (u, w)$ in T_v with the disjunction of the predicates of processor w and all its descendents in T_v
  - processor-local functions F store edge annotations

F_6: annotations for processor 6

source,next-hop $\to$ *predicate*
...
$1, 1 \mapsto \varnothing$
$1, 2 \mapsto p_2 \vee p_3$
$\boxed{1, 7 \mapsto p_4 \vee p_7 \vee p_8 \vee p_{11} \vee p_{12}}$
$1, 9 \mapsto \varnothing$
...



PSF Scheme

- Headers are used to store the source of a message

$$\mathcal{H} = V$$

$$\mathbf{Init}_v(\cdot) = v$$

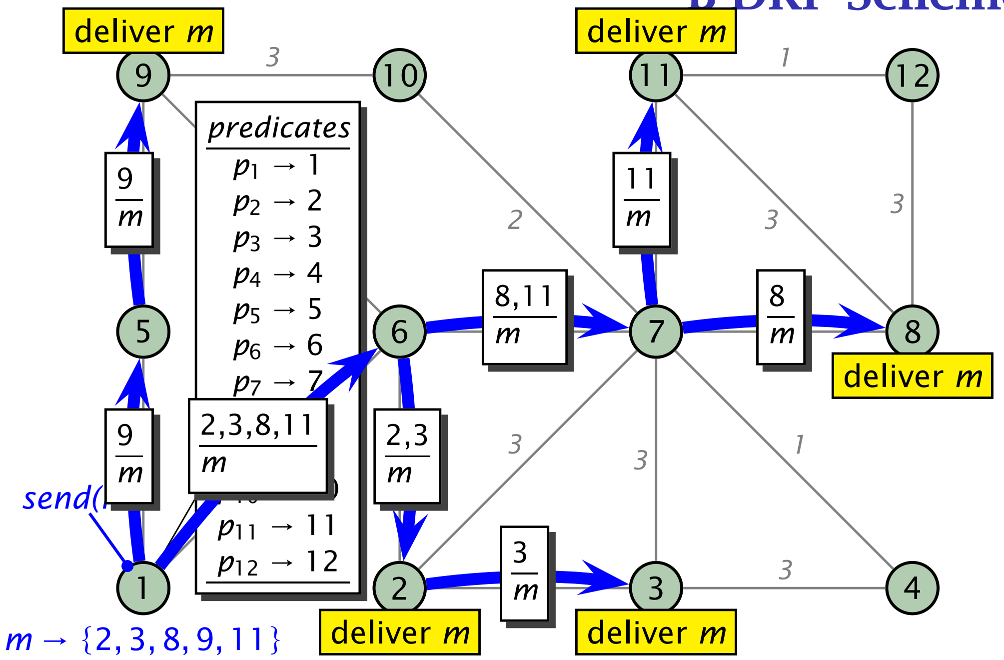
$$\mathbf{Hdr}_u(v) = v$$

- Processor u forwards $c = \langle v, m \rangle$ using F_u

$$\mathbf{Fwd}_u(v, m) = \{w | m \in F_u(v, w)\}$$

notation: if p is a predicate, $m \in p$ means $p(m) = 1$

B-DRP Scheme



(router-level topology)

Basic Steps

1. Perform a content-based matching of a message *once* at the source
   - compile a complete list of receivers
2. Use *unicast* routing information to forward to all receivers
   - using *dynamic receiver partitioning* (DRP)

B-DRP: Basic Ingredients

- Unicast routing information

unicast : *router-id* → *neighbor-link*

- Predicates from all routers in the network

predicates : *router-id* → *predicate*

Memory Requirements of a Scheme

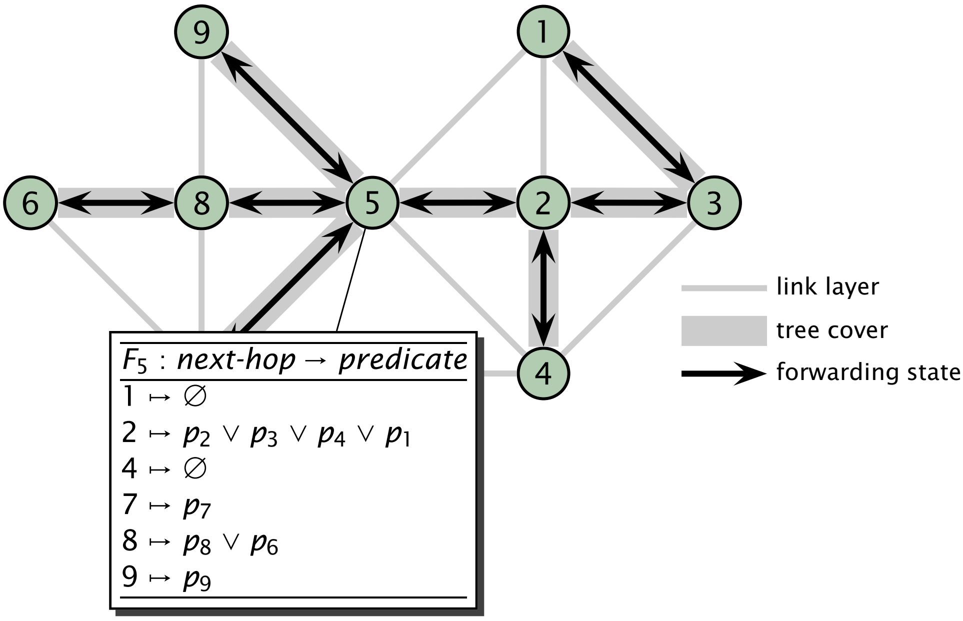
- How much memory do we need to implement a scheme?
  - over the entire network
  - average, max for each processor/router

- *Notation*

  Let $M(\cdot)$ to denote the memory requirements for a generic component or function

  - literally, the number of *bits*

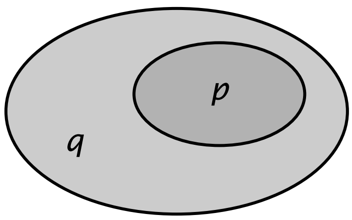
Tree-Cover Complexity



Covering Relation

- Covering relation $p \prec q$: *q covers p* when every message matching p also matches q

$$p \prec q \stackrel{\text{def}}{=} \forall m : p(m) \Rightarrow q(m)$$



- Represents the *content-based* *subnet* *address* relation

Reduced Table Sizes

F_5 : *next-hop* $\to$ *predicate*
$1 \mapsto \emptyset$
$2 \mapsto p_2 \vee p_3 \vee p_4 \vee p_1$
$4 \mapsto \emptyset$
$7 \mapsto p_7$
$8 \mapsto p_8 \vee p_6$
$9 \mapsto p_9$

$\wedge$

$$p_1 \prec p_2, p_4 \prec p_2$$

$\Longrightarrow$

F_5 : *next-hop* $\to$ *predicate*
$1 \mapsto \emptyset$
$2 \mapsto p_2 \vee p_3$
$4 \mapsto \emptyset$
$7 \mapsto p_7$
$8 \mapsto p_8 \vee p_6$
$9 \mapsto p_9$

Disjunction Advantage

- Given a set of predicates $P = \{p_1, p_2, \ldots, p_n\}$, we define the *disjunction advantage*

$$\alpha(P) = \frac{M(p_1 \vee p_2 \vee \ldots \vee p_n)}{M(p_1) + M(p_2) \cdots + M(p_n)}$$

- In the case $M(p_1) \approx M(p_2) \approx \ldots \approx M(p_n) \approx M_p$, we define

$$\alpha(k) = \frac{M(p_1 \vee p_2 \vee \ldots \vee p_k)}{kM_p}$$

- How does α affect the space complexity of a given scheme?
- Can we quantify α?

α in a Generic Predicate Model

- A predicate $p \in \mathcal{P}$ is a subset of a finite universe of messages $\mathcal{M}$, therefore $M(p) = p \log |\mathcal{M}|$
- Assuming a uniform distribution of predicates p of size $|p| = h$

$$\mathrm{E}(\alpha) = \frac{\mathrm{E}(|P|)}{nh}$$

$\mathrm{E}(|P|)$ is the expected size of the union of n random sets of size h

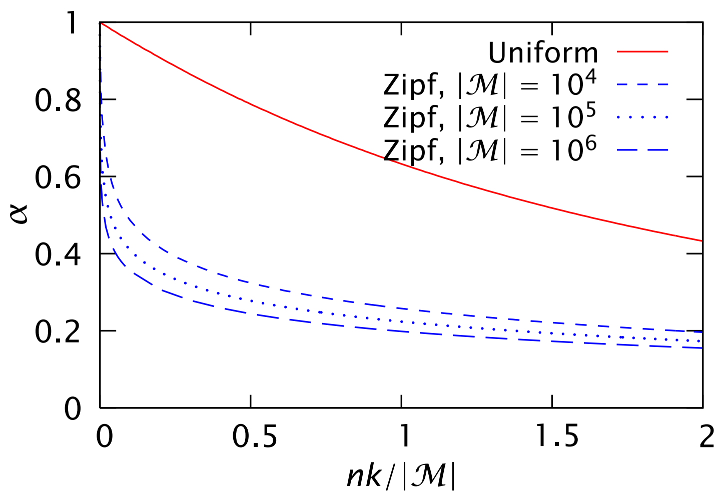
$$\Pr[m \in P] = 1 - \left(1 - \frac{h}{|\mathcal{M}|}\right)^n \approx 1 - e^{-\frac{nh}{|\mathcal{M}|}}$$

expected size of P and then the expected disjunction advantage

$$\mathrm{E}(\alpha) = \frac{|\mathcal{M}|}{nh}\left(1 - \left(1 - \frac{h}{|\mathcal{M}|}\right)^n\right) \approx \frac{|\mathcal{M}|}{nh}\left(1 - e^{-\frac{nh}{|\mathcal{M}|}}\right)$$

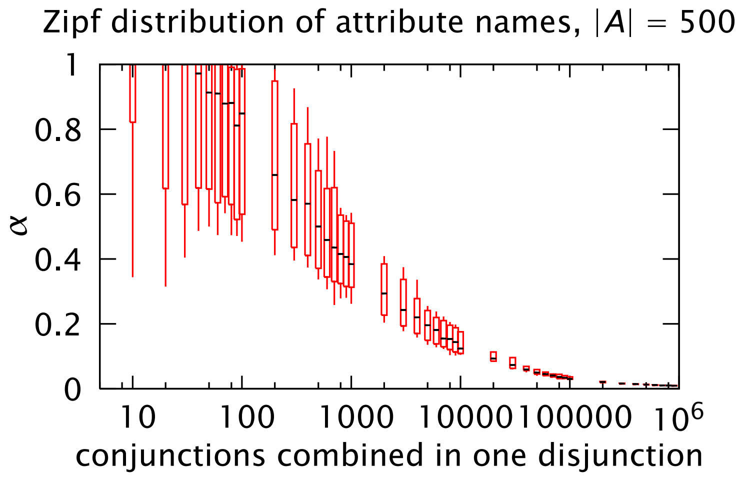
α in a Generic Predicate Model (2)

- Monte Carlo simulation
- Uniform vs. Zipf distribution for messages



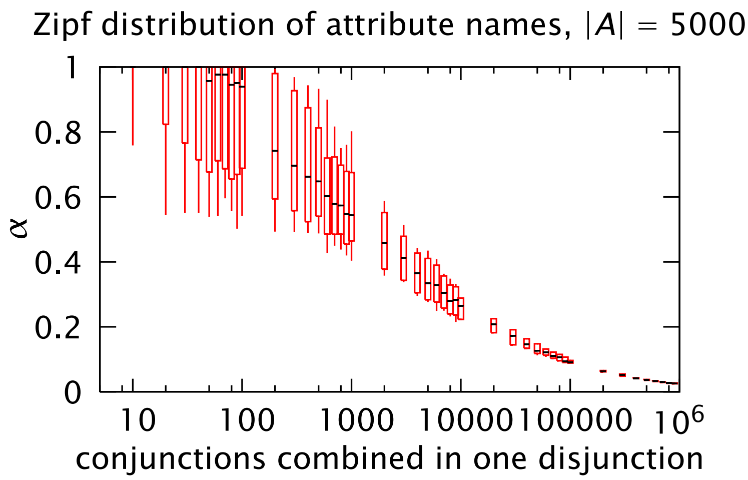
α in a Specific Predicate Model (1)

- Monte Carlo simulation
- Disjunctive normal form of attribute constraints



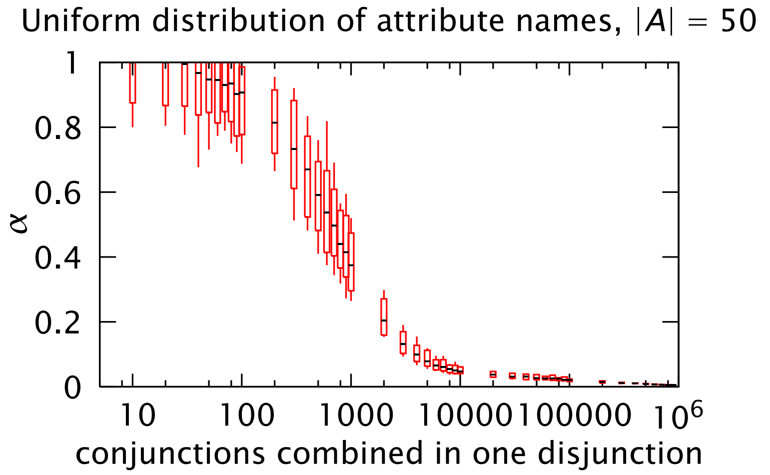
α in a Specific Predicate Model (2)

- Monte Carlo simulation
- Disjunctive normal form of attribute constraints



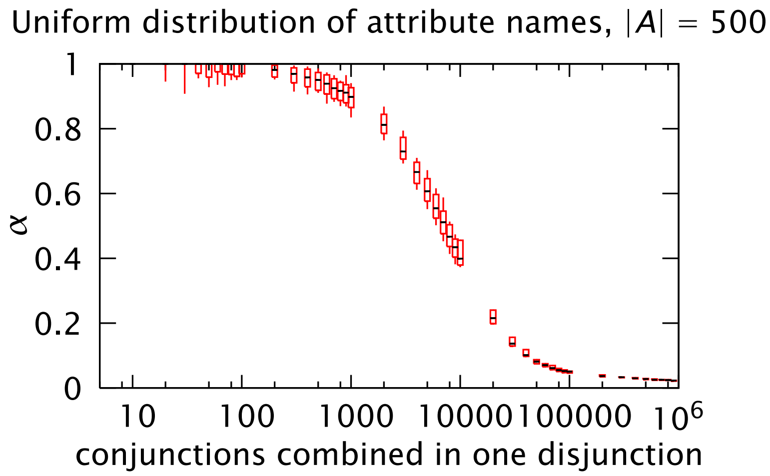
α in a Specific Predicate Model (3)

- Monte Carlo simulation
- Disjunctive normal form of attribute constraints



α in a Specific Predicate Model (4)

- Monte Carlo simulation
- Disjunctive normal form of attribute constraints



Memory Requirements of PSF

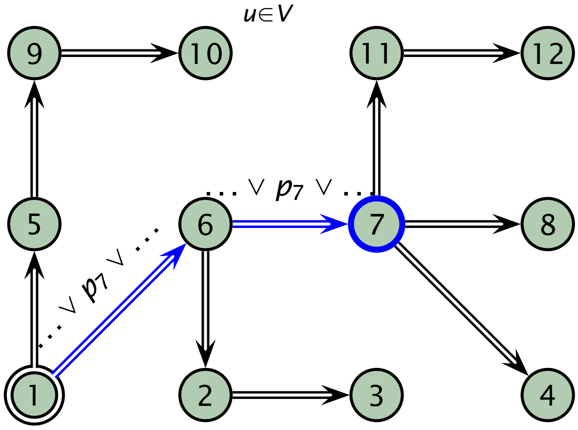
- Let $M(PSF)$ represent the memory requirements of PSF (over the entire network)
  - let V be the set of *routers*
  - let $R \subseteq V$ be the set of routers that have active *receivers*
  - let $S \subseteq V$ be the set of routers that have active *senders*
- Therefore,

$$M(PSF) = \sum_{u \in V} M(F_u) = \sum_{v \in S} M(T_v)$$

Memory Requirements of PSF (2)

- Memory requirement of a source-rooted tree T_v

$$M(T_v) = \sum_{u \in V} M(F_u(v, \cdot))$$



$$M(T_v) = \sum_{x \in R} M(p_x) distance(v, x)$$

Memory Requirements of PSF (3)

- Total memory requirement for PSF

$$M(PSF) = \sum_{v \in S} \sum_{x \in R} M(p_x) distance(v, x)$$

- A couple of uniformity assumptions
  - $\forall u \in R : M(pred(u)) = M_p$
  - senders and receivers are uniformly distributed
  - Let d be the average distance between two processors

$$M(PSF) = |S||R|M_p d$$

- Obviously $d = O(n)$
  - but in power-law random graphs $d = O(\log \log n)$ is very likely

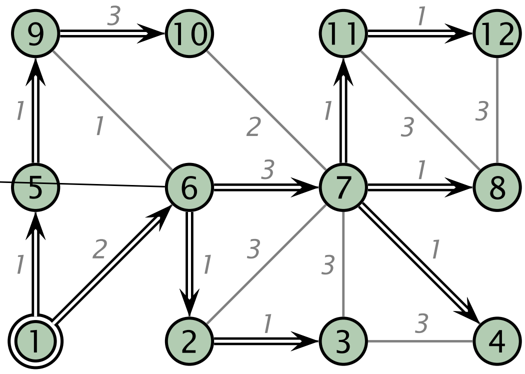
$$M(PSF) = O(n^2 \log \log n)$$

PSF Distinguishes Every Receiver

F_6: annotations for proc. 6

source,next-hop → *predicate*

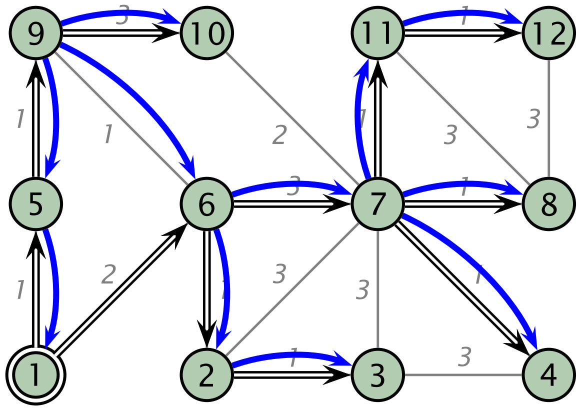
...
$1, 1 \mapsto \emptyset$
$1, 2 \mapsto p_2 \vee p_3$
$1, 7 \mapsto p_4 \vee p_7 \vee p_8 \vee p_{11} \vee p_{12}$
$1, 9 \mapsto \emptyset$
...



- The previous analysis assumes that, e.g., $M(p_2 \vee p_3) = M(p_2) + M(p_3)$

- In general, $M(p_2 \vee p_3) \leq M(p_2) + M(p_3)$
  - e.g., $p_2 =$ (port > 1000) $\wedge$ (port < 3000) and $p_3 =$ (port > 2000) $\wedge$ (port < 4000) can be combined in the disjunction $p_2 \vee p_3 =$ (port > 1000) $\wedge$ (port < 4000)

PSF Distinguishes Every Sender



- The previous analysis assumes that, e.g., processor 6 sees $T_1 \neq T_9$
- In general, from the viewpoint of u, $F_u(v_1, \cdot)$ may be identical to $F_u(v_2, \cdot)$ for some v_1 and v_2
  - e.g., $F_6(1, \cdot) = F_6(5, \cdot) = F_6(9, \cdot)$

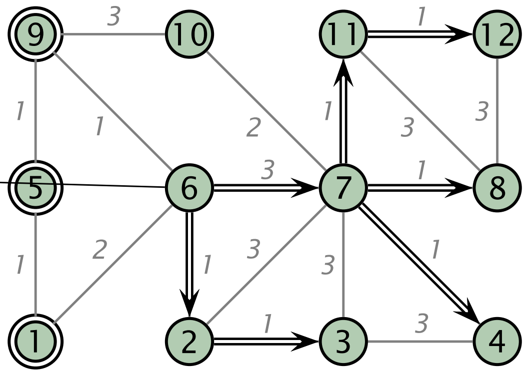
Improved Per-Source Forwarding

- *Destination grouping*
  - old idea: compression of predicates (used also on a tree-cover)
- *Source indistinguishability*
  - one forwarding entry per *equivalence class of indistinguishable sources*
- *Table folding*

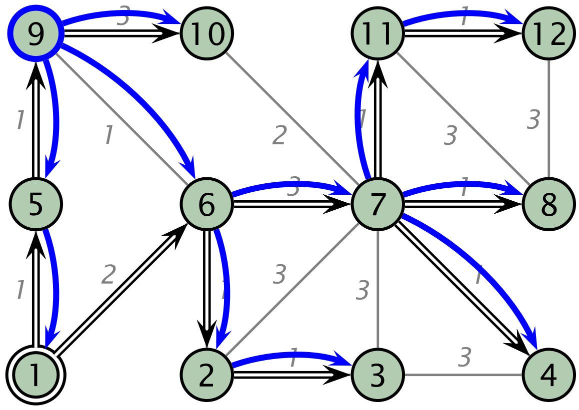
F_6: annotations for proc. 6

source,next-hop → *predicate*

. . .
$\{1,5,9\}, 1 \mapsto \emptyset$
$\{1,5,9\}, 2 \mapsto p_2 \vee p_3$
$\{1,5,9\}, 7 \mapsto p_4 \vee p_7 \vee p_8 \vee \ldots$
$\{1,5,9\}, 9 \mapsto \emptyset$
. . .



Rewriting Sources



- if a and b are indistinguishable by u, then they are also indistinguishable by all the descendants of u on T_a and T_b
  - those two sets of descendants are identical
- So, we can further compress forwarding state by rewriting the source with the most recent *indistinguishable* representative

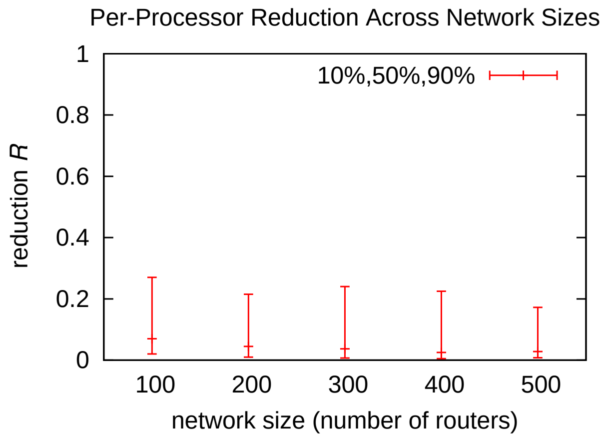
Evaluation

- Analysis of the compression of predicates
- Analysis of the indistinguishability relation
- Actual memory usage in practice
  - simple method to treat sources as content
  - concrete implementation of a forwarding table

Indistinguishability of Sources

- 25 networks for each size; BRITE; Waxman model of autonomous systems; for each processor u we compute

$$R_u = \frac{\text{number of equivalence classes}}{\text{total number of processors}}$$

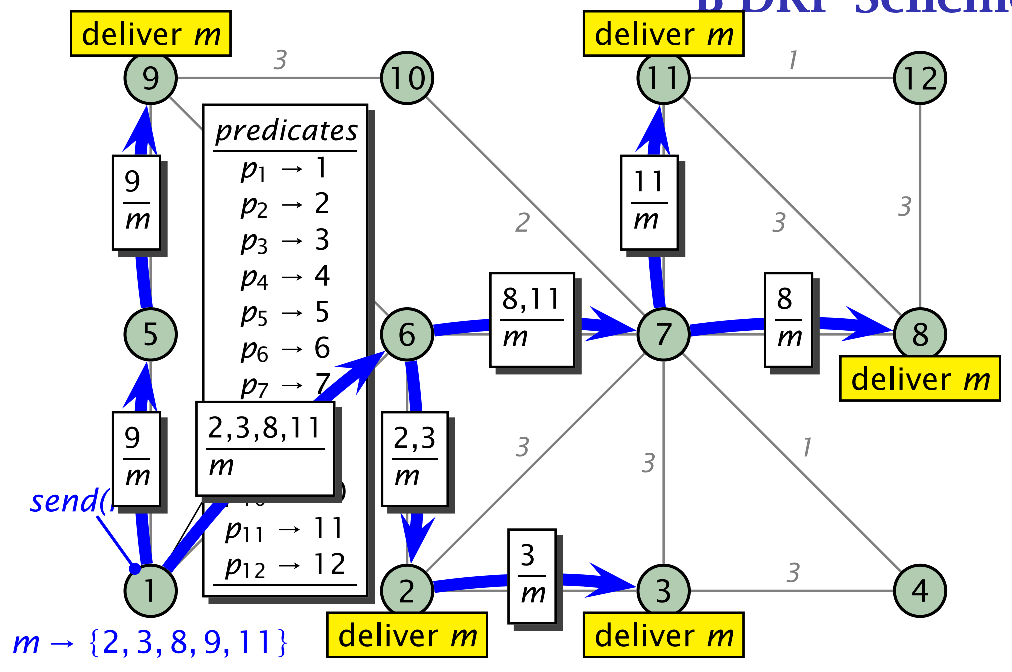


Folding Sources into Predicates

source	*next-hop*	→ *predicate*
1	1	$\mapsto \emptyset$
1	2	$\mapsto p_2 \vee p_3$
1	7	$\mapsto p_4 \vee p_7 \vee p_8 \vee p_{11} \vee p_{12}$
1	9	$\mapsto \emptyset$
1	10	$\mapsto \emptyset$
10	1	$\mapsto p_1$
10	2	$\mapsto p_2 \vee p_3$
10	7	$\mapsto \emptyset$
10	9	$\mapsto \emptyset$
10	10	$\mapsto \emptyset$
...		

next-hop	→ *predicate*
1	$\mapsto$ *source* $= 10 \wedge p_1$ $\vee \ldots$
2	$\mapsto$ *source* $= 10 \wedge (p_2 \vee p_3) \vee$ *source* $= 1 \wedge (p_2 \vee p_3)$ $\vee \ldots$
7	$\mapsto$ *source* $= 1 \wedge (p_4 \vee p_7 \vee p_8 \vee p_{11} \vee p_{12})$ $\vee \ldots$
9	$\mapsto \ldots$
10	$\mapsto \ldots$

B-DRP Scheme



Encoding Idea

$$\begin{tikzcd}[column sep=huge]
m \arrow[r, leftrightarrow, "p(m)"] \arrow[d, "X_S"'] & p \arrow[d, "X_P"] \\
S_m \arrow[r, leftrightarrow, "S_m \supseteq S_p^i \text{ for some } i"] \arrow[d, "B"'] & S_p^1, S_p^2, \ldots \arrow[d, "B"] \\
B_m \arrow[r, leftrightarrow, "B_m \supseteq B_p^i \text{ for some } i"] & B_p^1, B_p^2, \ldots
\end{tikzcd}$$

1. Map m and p into sets of tags S_m and $S_p^1, S_p^2, \ldots$, such that

$$p(m) \Rightarrow \left(\exists S_p^i \in \{S_p^1, S_p^2, \ldots\} : S_m \supseteq S_p^i\right)$$

2. Represent S_m and $S_p^1, S_p^2, \ldots$ with Bloom filters B_m and $B_p^1, B_p^2, \ldots$

Bloom Filters

- $U = \{x_1, x_2, \ldots\}$ is the universe of values we intend to represent
- A Bloom set over U is defined by
  - a bit vector B of size m
  - k distinct hash functions $h_1, h_2, \ldots, h_k$ with signature $H : U \to \{0, 1, \ldots, m-1\}$
- $B(x)$ is computed as follows

Input: x, an element of the universe U
Output: $B(x)$, Bloom filter representing the singleton $\{x\}$

$B \leftarrow \emptyset$ // all zeros
foreach $i \in \{1, \ldots, k\}$
  $B[h_i(x)] \leftarrow 1$
return B

Bloom Filters (2)

- Given a set of n elements $S = \{x_1, x_2, \ldots x_n\}$

$B(S) \leftarrow B(x_1) \cup B(x_2) \cup \cdots \cup B(x_n)$

Input: S, a set elements from the universe U
Output: $B(S)$, Bloom filter representing S

$B \leftarrow \emptyset$
foreach $x \in S$
  foreach $i \in \{1, \ldots, k\}$
    $B[h_i(x)] \leftarrow 1$
return B

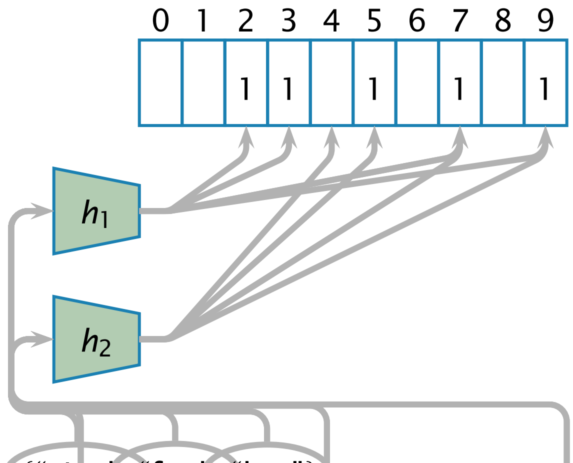
- Testing $x \in S$ amounts to testing $B(x) \subseteq B(S)$

i.e., (assuming B is implemented as an integer)

$x \in S \Leftrightarrow$ (Bx & BS) == Bx

Bloom Filters: Example

U is the universe of character strings; $k = 2$; $m = 10$



$S = \{$"ciao", "foo", "bar"$\}$

Test: "foo": *yes*, "abc": *no*, "xyz": *yes* (false positive)

Encoding with Tags

$$
\begin{tikzcd}
m = \{\text{software, update, emacs}\} \arrow[rr, leftrightarrow, "p(m)"] \arrow[d, "X_S"'] & & p = \{\text{gnu,software}\}, \{\text{emacs}\}, \{\text{anarchism}\} \arrow[d, "X_P"] \\
S_m = \{\text{software, update, emacs}\} \arrow[rr, leftrightarrow, "S_m \supseteq S_p^i \text{ for some } i"] \arrow[d, "B"'] & & S_p = \{\text{gnu,software}\}, \{\text{emacs}\}, \{\text{anarchism}\} \arrow[d, "B"] \\
B_m = [0110010101] \arrow[rr, leftrightarrow, "B_m \supseteq B_p^i \text{ for some } i"] & & B_p = [0101010010]\ [0010000100]\ [0001000001]
\end{tikzcd}
$$

From Constraints to Tags

1. Constraint encoding

encoding rule			*type*
c	$\rightarrow$	$s(c)$	
name $=$ *value*	$\rightarrow$	"***name***=*value*"	equality
name $\langle$*other-operator*$\rangle$ *value*	$\rightarrow$	"$\exists$***name***"	existence

Example:

- **disk-space** $< 1Gb$ $\longrightarrow$ "∃disk-space"
- **disk-space** $= 2Gb$ $\longrightarrow$ "disk-space=2"

2. A *conjunction* $f = c_1 \wedge c_2 \wedge \cdots \wedge c_k$ is encoded with the union of the encodings of its constraints $S_f = \{s(c_1), s(c_2), \ldots, s(c_k)\}$

3. A predicate $P = f_1 \vee f_2 \vee \ldots \vee f_F$ P is encoded with a set of sets $S_P = \{S_1, S_2, \ldots, S_F\}$

Message Encoding

1. Every attribute **name** $=$ *value* is encoded with two strings

encoding rule		
a	$\rightarrow$	$s(a) = \{s^{=}(a), s^{\exists}(a)\}$
name $=$ *value*	$\rightarrow$	{"***name***=*value*", "$\exists$***name***"}

disk-space $= 2Gb \rightarrow$ {"disk-space=2Gb", "∃disk-space"}

2. A message $m = \{a_1, a_2, \ldots, a_n\}$ is therefore encoded with a set $S_m = s(a_1) \cup s(a_2) \cup \cdots \cup s(a_n)$

B-DRP State

- *Local predicates*

  local-predicates : *host-id* → *predicate*

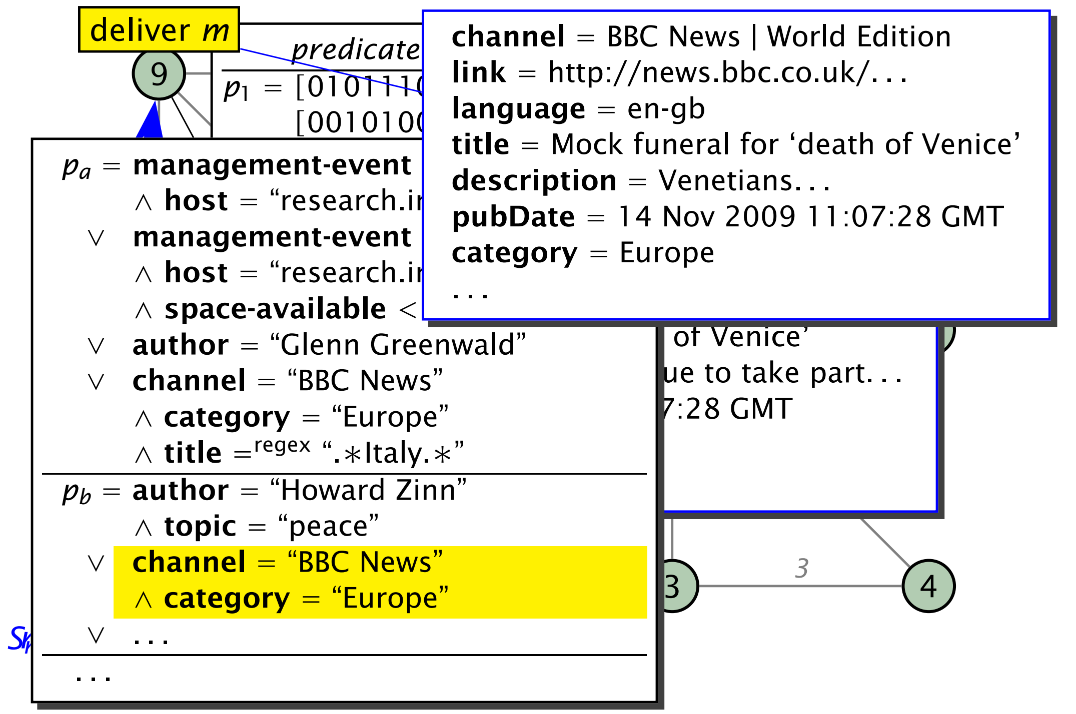
- *Predicates* from all other routers

  predicates : *router-id* → *encoded-predicate*

- *Unicast* routing information

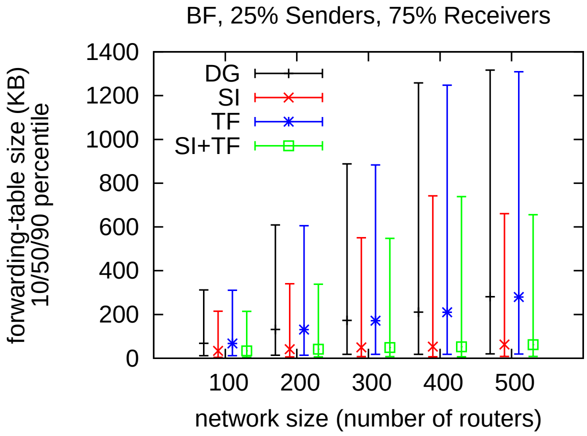
  unicast : *router-id* → *neighbor-link*

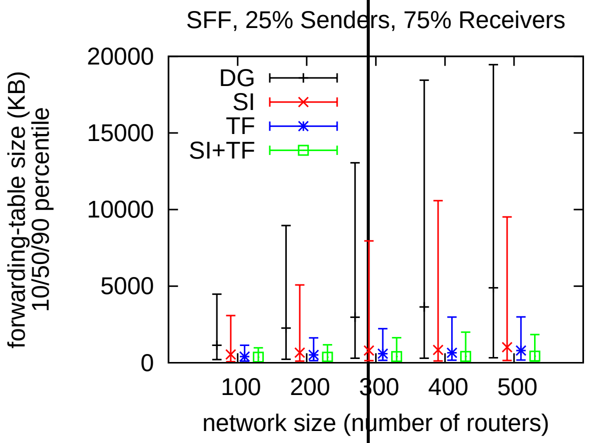
B-DRP: Two Matching Processes



Emulation with SFF

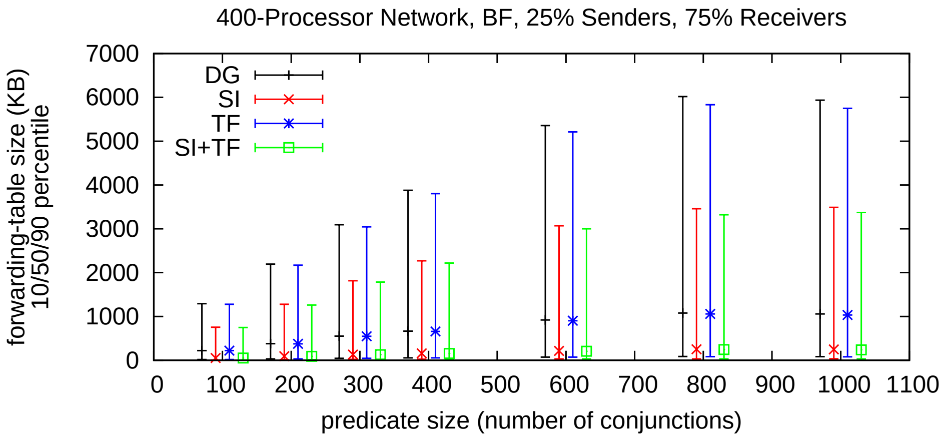
- Siena Fast Forwarding algorithm; for each processor we measure the *actual memory usage*

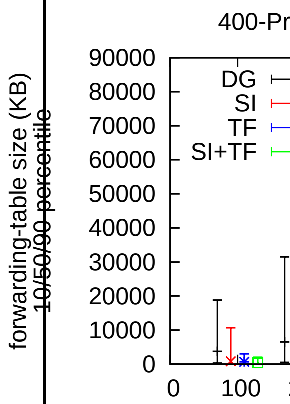




Emulation with SFF (2)

- Siena Fast Forwarding algorithm; for each processor we measure the *actual memory usage*





Emulation with SFF (3)

- Siena Fast Forwarding algorithm; for each processor we measure the *actual memory usage*