Content-Based Communication: The *Network* Underneath Event Processing

Antonio Carzaniga

Faculty of Informatics
University of Lugano

April 2008

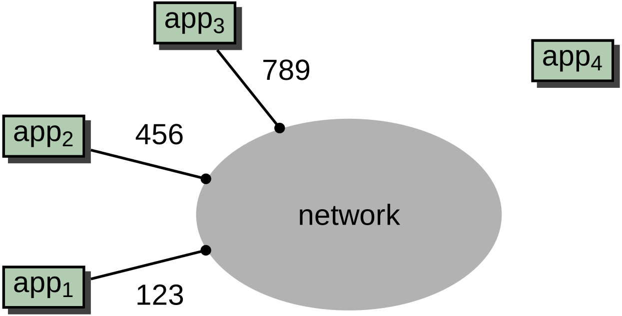
http://www.inf.unisi.ch/carzaniga/

Traditional Network Service

- Traditional networking is centered around two service models
  - “telephone service”
  - “postal service”

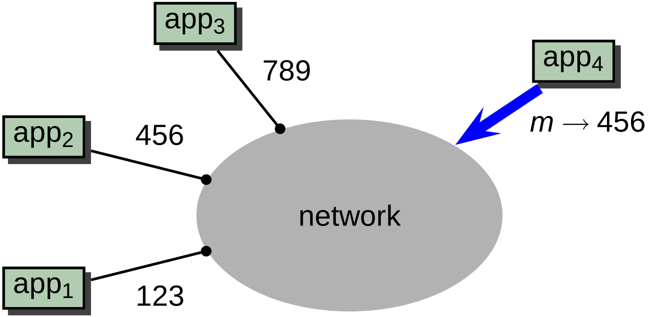
Traditional Network Service

- Traditional networking is centered around two service models
  - “telephone service”
  - “postal service”



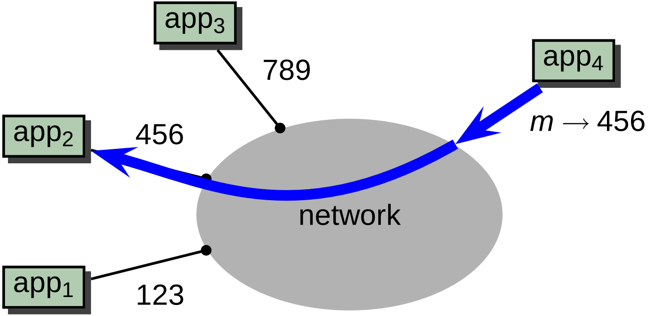
Traditional Network Service

- Traditional networking is centered around two service models
  - "telephone service"
  - "postal service"



Traditional Network Service

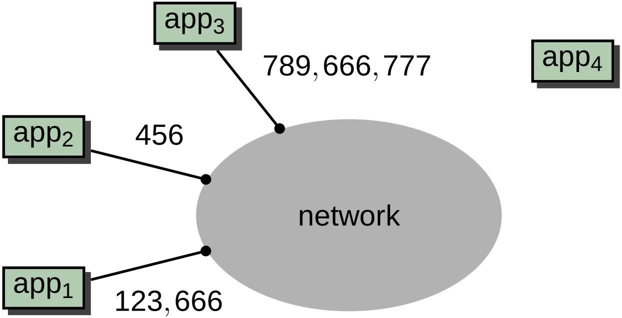
- Traditional networking is centered around two service models
  - "telephone service"
  - "postal service"



Multicast Service

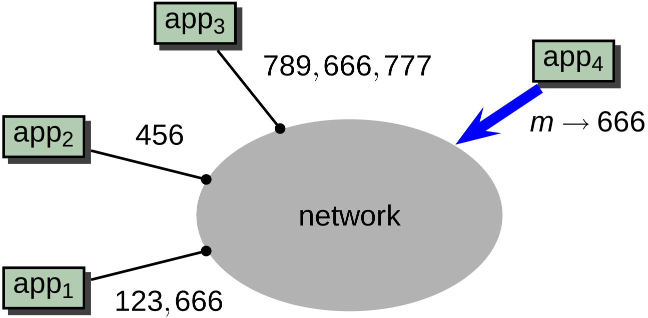
Multicast Service

- An address can represent a *group* of hosts
  - multiple host may *join* the same group
  - a host may *join* multiple groups



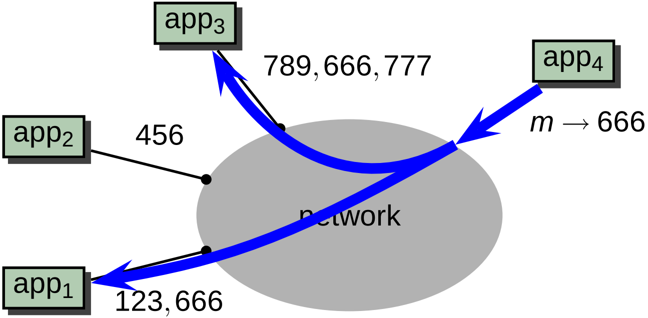
Multicast Service

- An address can represent a *group* of hosts
  - multiple host may *join* the same group
  - a host may *join* multiple groups



Multicast Service

- An address can represent a *group* of hosts
  - multiple host may *join* the same group
  - a host may *join* multiple groups



Publish/Subscribe Communication

Publish/Subscribe Communication

- Receivers decide what they want to receive—they *subscribe*

Publish/Subscribe Communication

- Receivers decide what they want to receive—they *subscribe*
- Senders simply send information to the network—they *publish*

Publish/Subscribe Communication

- Receivers decide what they want to receive—they *subscribe*
- Senders simply send information to the network—they *publish*
- The system ("broker" or "dispatcher") does the rest
  - it delivers every publication to all interested subscribers

Publish/Subscribe Communication

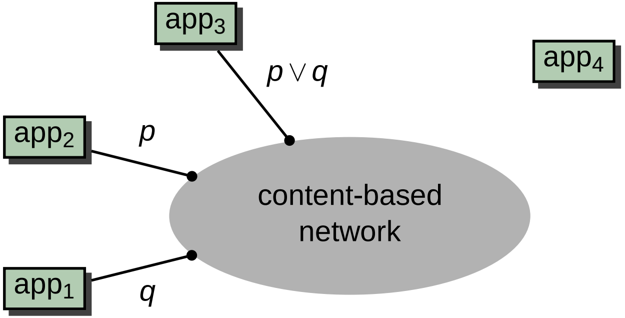
- Receivers decide what they want to receive—they *subscribe*
- Senders simply send information to the network—they *publish*
- The system (“broker” or “dispatcher”) does the rest
  - it delivers every publication to all interested subscribers
- Publish/subscribe looks like IP multicast
  - subscribing $\leftrightarrow$ joining a group
  - publishing $\leftrightarrow$ sending

Publish/Subscribe Communication

- Receivers decide what they want to receive—they *subscribe*
- Senders simply send information to the network—they *publish*
- The system ("broker" or "dispatcher") does the rest
  - it delivers every publication to all interested subscribers
- Publish/subscribe looks like IP multicast
  - subscribing $\leftrightarrow$ joining a group
  - publishing $\leftrightarrow$ sending

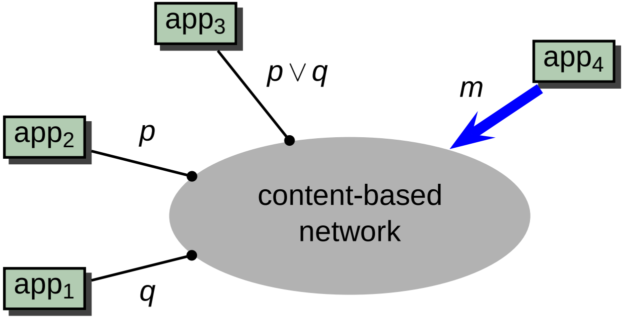
Publish/subscribe differs from IP multicast
only insofar as it is content-based

Content-Based Communication



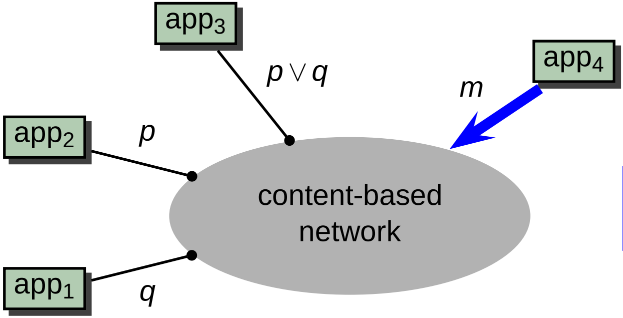
- Receivers declare a *predicate*
  - Boolean function $P : \mathit{Message} \rightarrow \{0,1\}$
  - $p(m) = 1$ means the the receiver is interested in receiving m

Content-Based Communication



- Receivers declare a *predicate*
  - Boolean function $P : Message \rightarrow \{0,1\}$
  - $p(m) = 1$ means the the receiver is interested in receiving m
- Senders send *messages*
  (without specifying a destination)

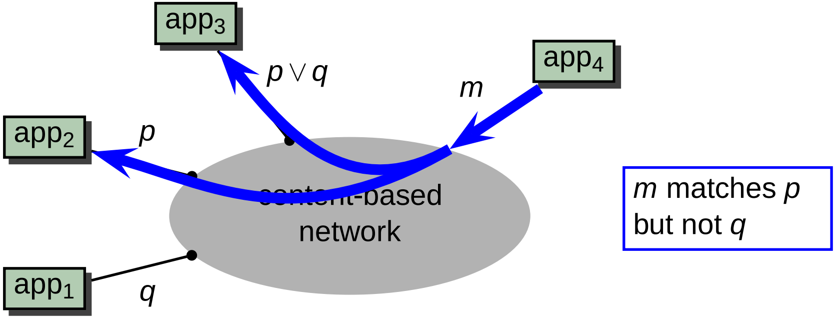
Content-Based Communication



m matches p
but not q

- Receivers declare a *predicate*
  - Boolean function $P : \textit{Message} \to \{0,1\}$
  - $p(m) = 1$ means the the receiver is interested in receiving m
- Senders send *messages*
  (without specifying a destination)

Content-Based Communication



- Receivers declare a *predicate*
  - Boolean function $P : Message \rightarrow \{0,1\}$
  - $p(m) = 1$ means the the receiver is interested in receiving m
- Senders send *messages* (without specifying a destination)
- Message m goes to all interested receivers

Application Domains

- *Publish/subscribe communication*

- News distribution
- System/network monitoring and management
- Intrusion detection
- Service discovery and brokering
- Peer-to-peer data sharing
- Distributed electronic auctions
- Multi-player games
- Caching systems
- . . .

Messages

- Example: *a set of attributes*

```
alert-system = “IT-ANAS”
alert = “conjestion”
cause = “accident”
date = [20/Aug/2006:06:14:40 +0200]
location-road = “A1”
location-km = 231
location-dir = “North”
delay-min = 35
detour-info = “sms:3141592653/5897”
report-to = “sms:2718281828/4590”
```

Predicates

- Example: *an expression of attribute constraints*

$$
\begin{array}{ll}
 & \text{alert} = \text{“conjestion”} \\
 & \wedge\ \text{location-road} = \text{“A1”} \\
 & \wedge\ \text{location-dir} = \text{“South”} \\
\vee & \text{alert} = \textit{any} \\
 & \wedge\ \text{cause} = \text{“accident”} \\
\vee & \text{alert} = \text{“weather”} \\
 & \wedge\ \text{severity} > 4 \\
\vee & \text{news-topic} =^{\text{regex}} \text{“sport/soccer/.}{*}\text{”} \\
 & \wedge\ \text{team} = \text{“Milan”}
\end{array}
$$

Content-Based Networking

- Content-based communication (a.k.a., “pub/sub”) *designed and implemented as a network service*
  - architecture
  - routing
  - forwarding
  - . . .

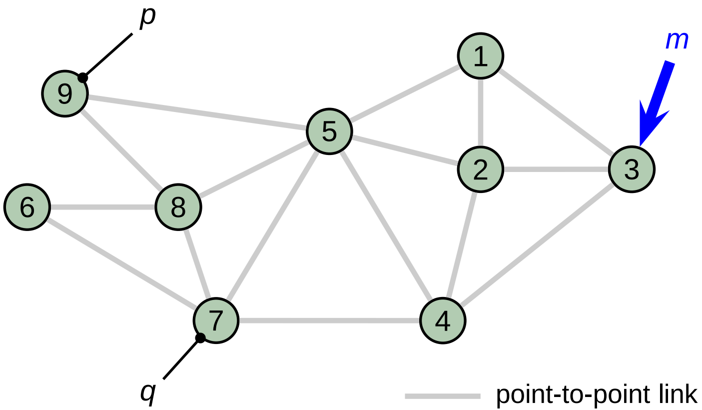
Content-Based Networking

- Content-based communication (a.k.a., “pub/sub”) *designed and implemented as a network service*
  - architecture
  - routing
  - forwarding
  - . . .

- Host interface
  - *send(m)*
  - *set_predicate(p)*

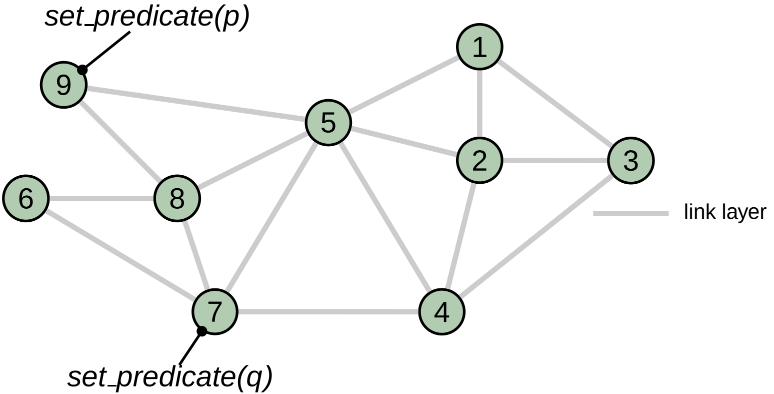
- Type of service
  - *datagram service (i.e., “best effort”)*

Content-Based Routing

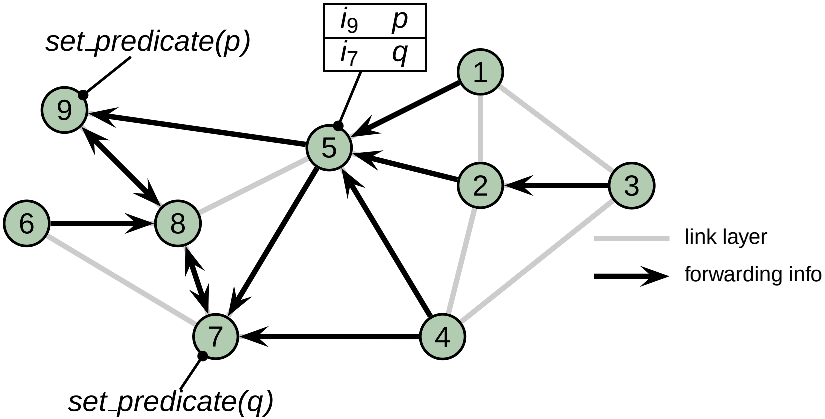


- Where and how to forward *m*?
- Based on which kind of routing information?

Content-Based Routing Example

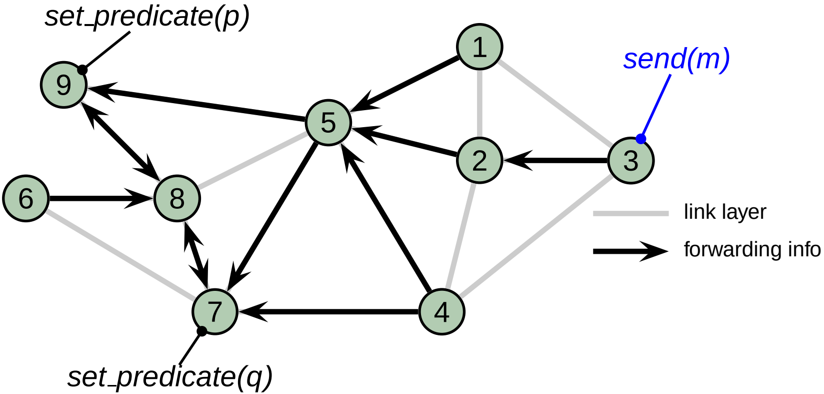


Content-Based Routing Example

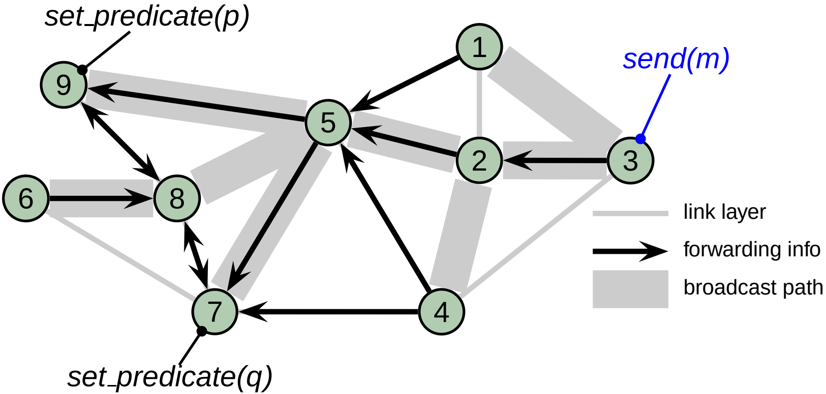


- Routing protocol propagates predicates
- Forwarding state “attracts” messages towards matching predicates

Content-Based Routing Example

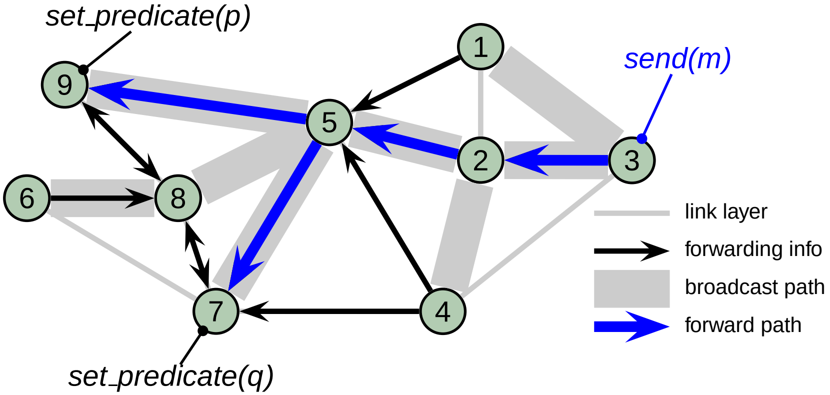


Content-Based Routing Example



- A message *m* is treated as a broadcast packet

Content-Based Routing Example



- A message *m* is treated as a broadcast packet
- But only forwarded along matching paths

Some Research Questions

Some Research Questions

- How do senders and receivers *connect?*
  - you say “soccer,” I say “football”
  - in a content-based network, this is a problem

Some Research Questions

- How do senders and receivers *connect?*
  - you say “soccer,” I say “football”
  - in a content-based network, this is a problem

- How do we design a routing protocols that allows autonomous systems to control input, output, and transit traffic?
  - a “content-based” firewall?
  - routing policies

Some Research Questions

- How do senders and receivers *connect?*
  - you say “soccer,” I say “football”
  - in a content-based network, this is a problem
- How do we design a routing protocols that allows autonomous systems to control input, output, and transit traffic?
  - a “content-based” firewall?
  - routing policies
- What is the *complexity* of content-based routing?
  - theoretical basis for content-based networking
  - correctness and complexity (memory requirements)
  - lower bounds

Some Research Questions (2)

- Routing: any good idea?
  - peer-to-peer network models and protocols
  - new old ideas: broadcast (as in link-state routing) and almost-random walks

Some Research Questions (2)

- Routing: any good idea?
  - peer-to-peer network models and protocols
  - new old ideas: broadcast (as in link-state routing) and almost-random walks

- How do we *evaluate* our protocols and systems?
  - we need good models of networks and applications

Some Research Questions (2)

- Routing: any good idea?
  - peer-to-peer network models and protocols
  - new old ideas: broadcast (as in link-state routing) and almost-random walks
- How do we *evaluate* our protocols and systems?
  - we need good models of networks and applications
- How do we preserve the *privacy* of receivers?
  - we want predicates to remain confidential, and at the same time allow the (untrusted) network to do its job

Some Research Questions (2)

- Routing: any good idea?
  - peer-to-peer network models and protocols
  - new old ideas: broadcast (as in link-state routing) and almost-random walks
- How do we *evaluate* our protocols and systems?
  - we need good models of networks and applications
- How do we preserve the *privacy* of receivers?
  - we want predicates to remain confidential, and at the same time allow the (untrusted) network to do its job
- Is there a *content-based middleware?*
  - traditional subscriptions, content directories, etc.
  - synthesis of predicates, integration with applications, etc.

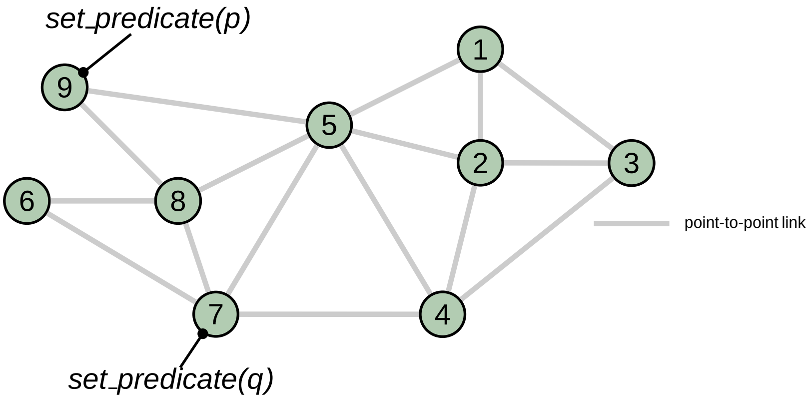
Menu

- A concrete routing protocol
- A concrete forwarding algorithm
- Theory of content-based routing
- Security in content-based networking
- Conclusions

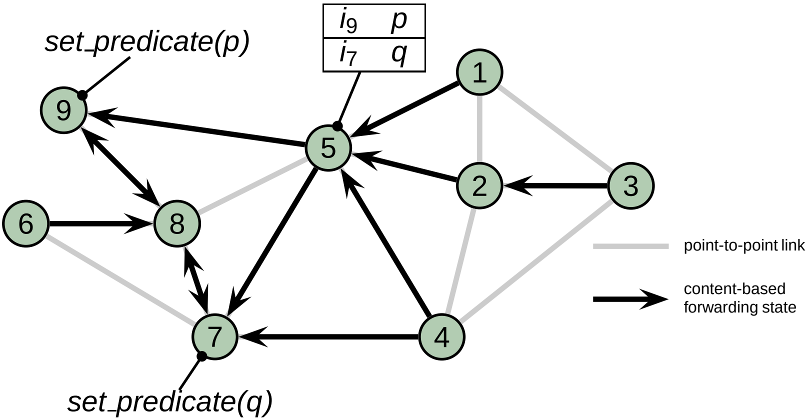
Part I

A Concrete Routing Protocol

Content-Based Routing

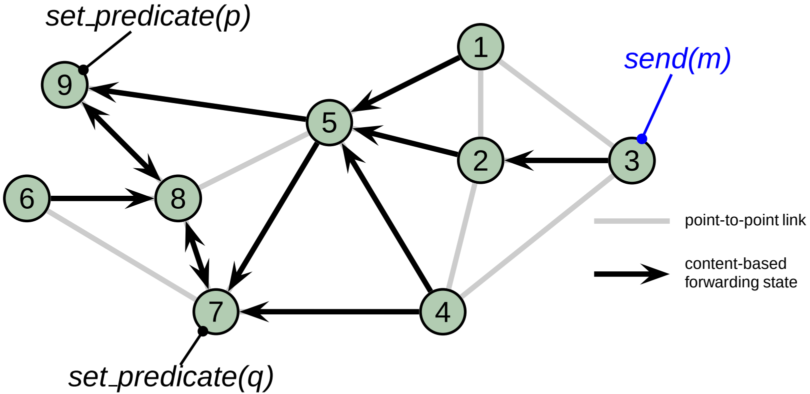


Content-Based Routing

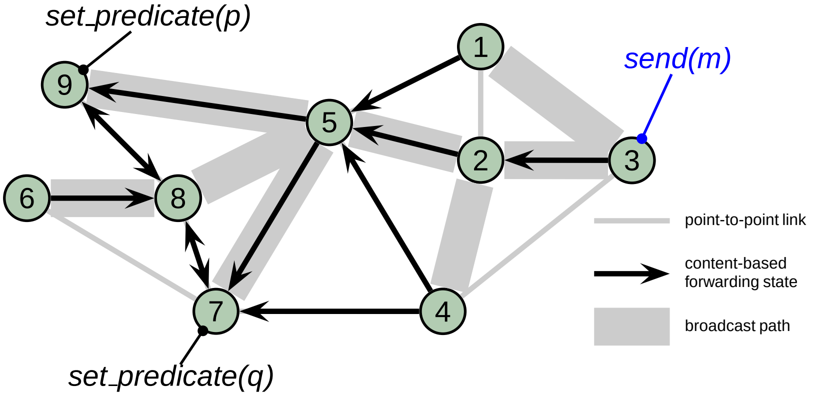


- Routing protocol propagates predicates
- Forwarding state “attracts” messages towards matching predicates

Content-Based Forwarding

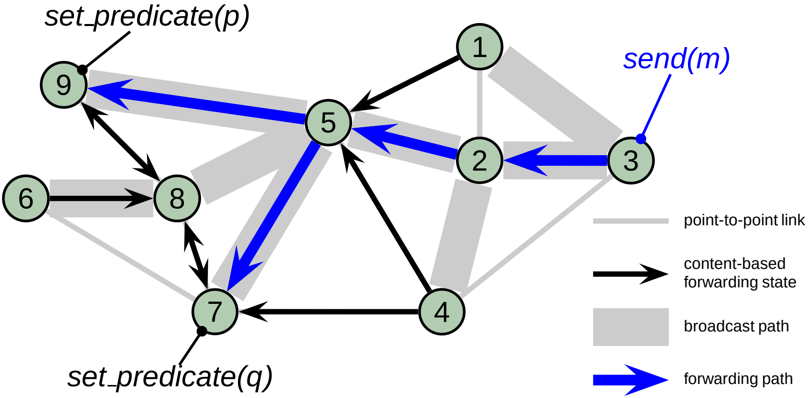


Content-Based Forwarding



- Every message *m* is treated as a broadcast packet

Content-Based Forwarding



- Every message *m* is treated as a broadcast packet
- But only forwarded along “matching paths”

CBCB Routing Scheme

Combined Broadcast and Content-Based Routing

- A broadcast layer takes care of avoiding loops
- A content-based layer forms forwarding state out of predicates

CBCB Routing Scheme

Combined Broadcast and Content-Based Routing

- A broadcast layer takes care of avoiding loops
- A content-based layer forms forwarding state out of predicates

Broadcast Layer

- Well-known techniques
- A few additional requirements

Content-Based Routing

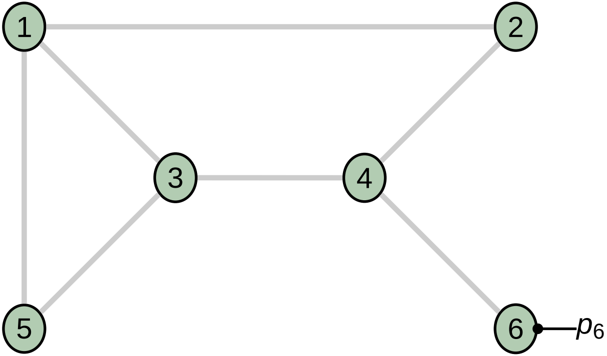
- "Push" propagation of predicates (RA protocol)
- "Pull" propagation of predicates (SR/UR protocol)

Receiver Advertisements (RA)

- *Receiver advertisements (RAs)* push predicates from receivers out to all potential senders

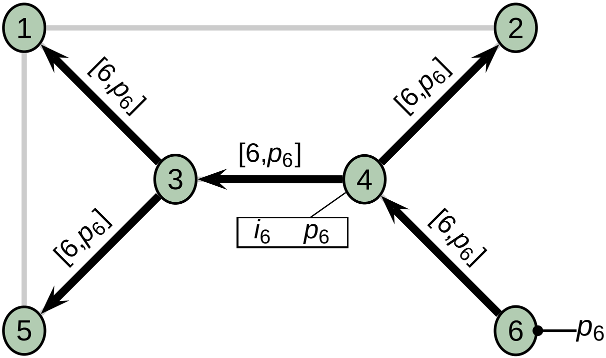
Receiver Advertisements (RA)

- *Receiver advertisements (RAs)* push predicates from receivers out to all potential senders



Receiver Advertisements (RA)

- *Receiver advertisements (RAs)* push predicates from receivers out to all potential senders

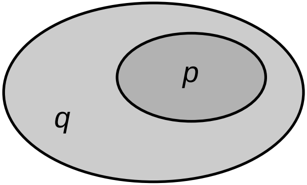


- *Forwarding/routing table* associates a predicate with each interface

Covering Relation

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

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



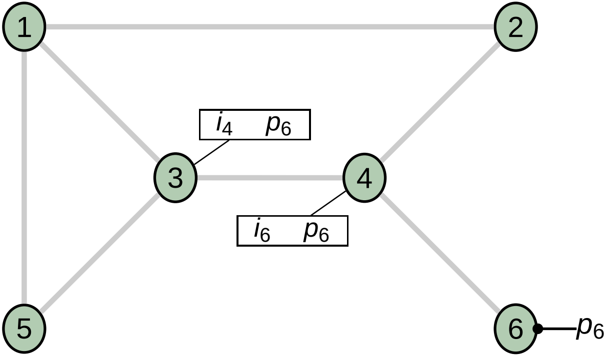
- Represents the *content-based* *<u>subnet</u>* *address* relation

Content-Based RA Ingress Filtering

- RA propagation stops when a new predicate is *covered* by an old one ($p_{RA} \prec p$)

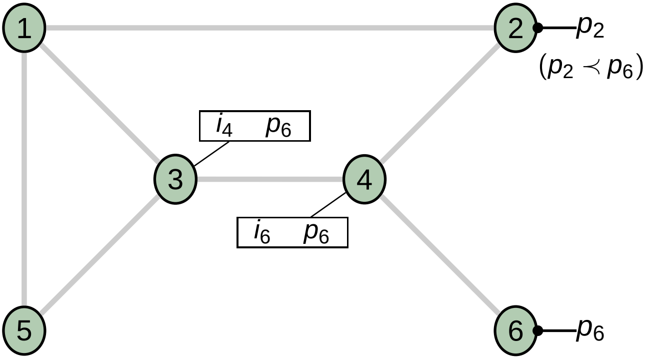
Content-Based RA Ingress Filtering

- RA propagation stops when a new predicate is *covered* by an old one ($p_{RA} \prec p$)



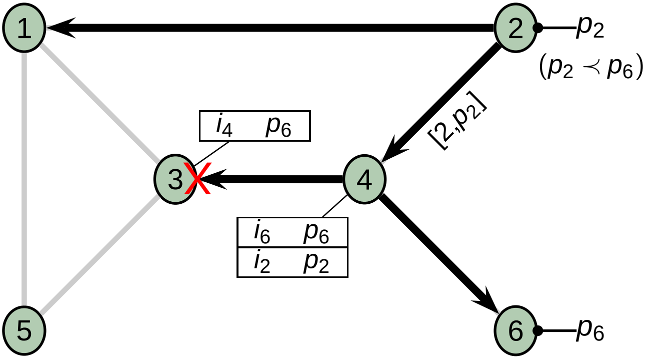
Content-Based RA Ingress Filtering

- RA propagation stops when a new predicate is *covered* by an old one ($p_{RA} \prec p$)



Content-Based RA Ingress Filtering

- RA propagation stops when a new predicate is *covered* by an old one ($p_{RA} \prec p$)



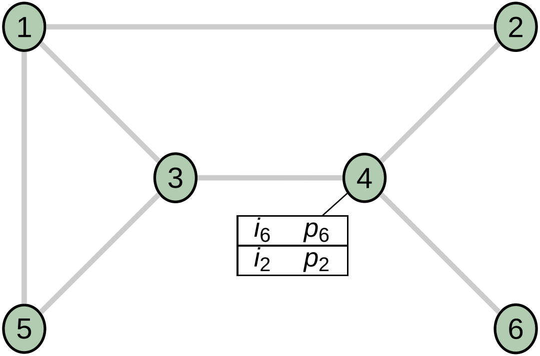
- Table update in RA protocol: $p \leftarrow p \vee p_{RA}$
  notice that $(p \vee p_{RA} = p) \Leftrightarrow (p_{RA} \prec p)$

Content-Based Address Inflation

- Content-based RA ingress filtering generates an “inflation” of content-based addresses

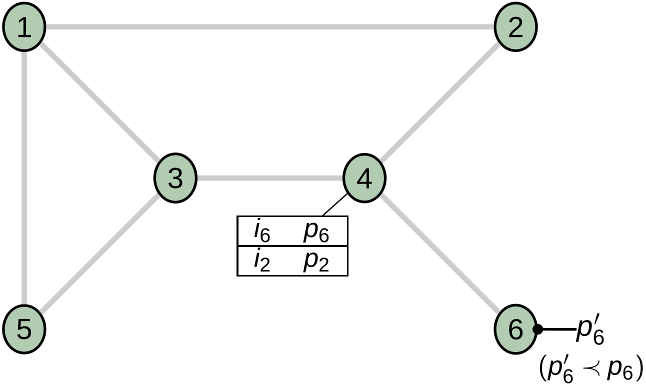
Content-Based Address Inflation

- Content-based RA ingress filtering generates an "inflation" of content-based addresses



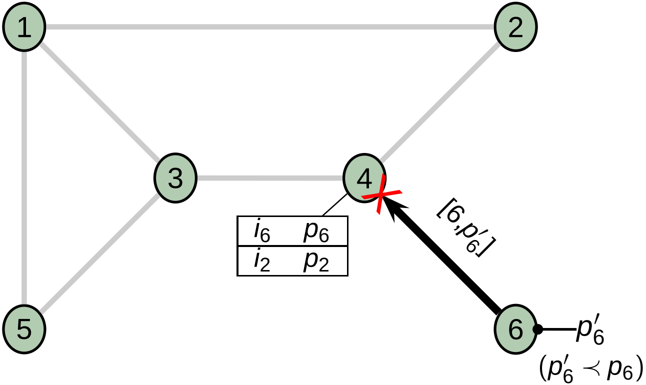
Content-Based Address Inflation

- Content-based RA ingress filtering generates an “inflation” of content-based addresses



Content-Based Address Inflation

- Content-based RA ingress filtering generates an “inflation” of content-based addresses



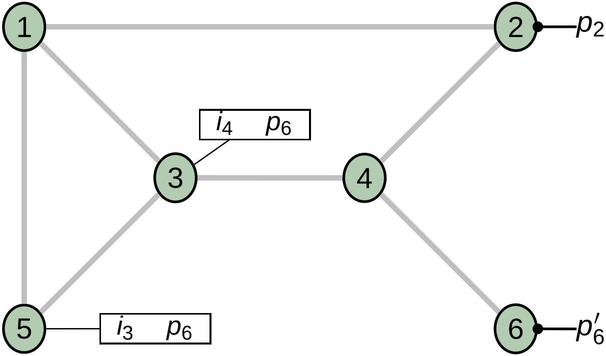
- Node 6 could now receive (from node 4) unwanted messages, i.e., messages that match p_6, but not p'_6

Sender Request/Update Reply

- Sender requests and update replies (SR/UR) “pull” routing information from receivers to senders

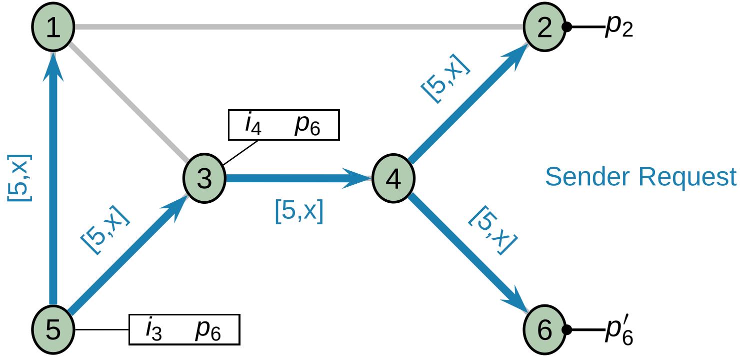
Sender Request/Update Reply

- Sender requests and update replies (SR/UR) “pull” routing information from receivers to senders



Sender Request/Update Reply

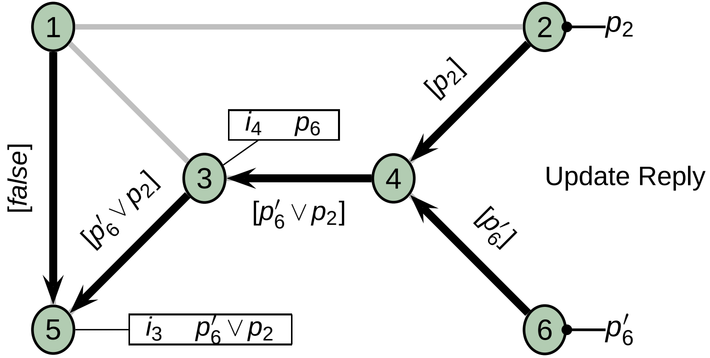
- Sender requests and update replies (SR/UR) “pull” routing information from receivers to senders



- Node 5 uses the URs to update its table
- Node 3 does not update its table…

Sender Request/Update Reply

- Sender requests and update replies (SR/UR) “pull” routing information from receivers to senders



- Node 5 uses the URs to update its table
- Node 3 does not update its table...

Options and Optimizations

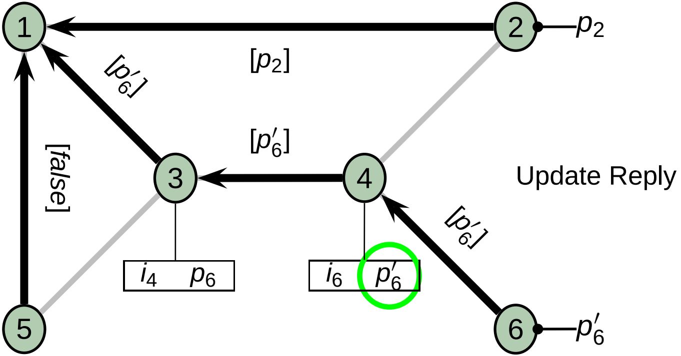
- The SR/UR protocol can be expensive
  - URs can be cached and reused, depending on the network topology
  - the SR/UR protocol can be triggered by the amount of *false positives*

Options and Optimizations

- The SR/UR protocol can be expensive
  - URs can be cached and reused, depending on the network topology
  - the SR/UR protocol can be triggered by the amount of *false positives*

- Both RA and SR/UR manage complex predicates
  - updates in RAs ($p \leftarrow p \vee p_{RA}$) and URs ($p_{UR} \leftarrow p_1 \vee p_2 \vee \ldots \vee p_k$) can be "simplified"

Caching and Reusing URs



- Node 4 may reuse the update reply received from node 6
  - because the 4–6 link is a *bridge*

CBCB Evaluation

- We implemented and tested CBCB within a simulator

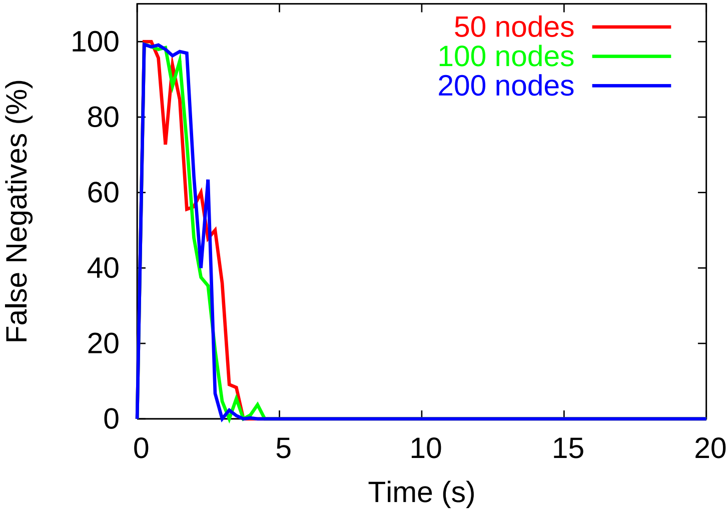
- Evaluation goals

  - *main functionality:* does the protocol deliver messages to nodes that are interested in them?

  - *traffic filtering:* does the protocol prevent unnecessary message traffic?

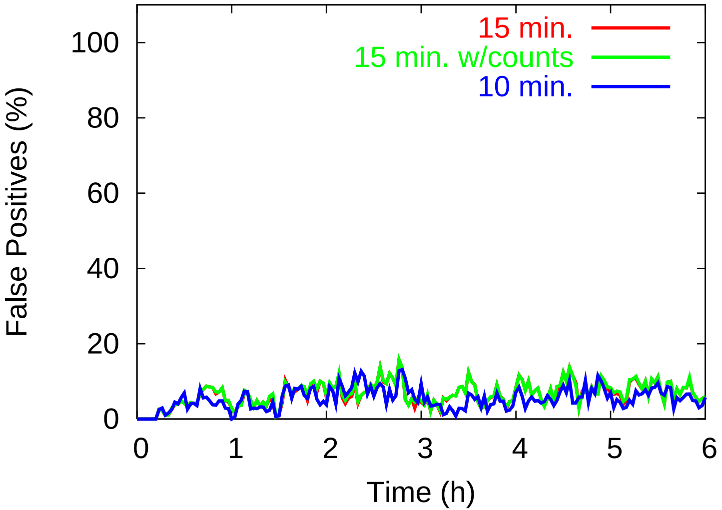
  - *protocol scalability:* does the protocol produce a reasonable and stable amount of control traffic?

Main Functionality



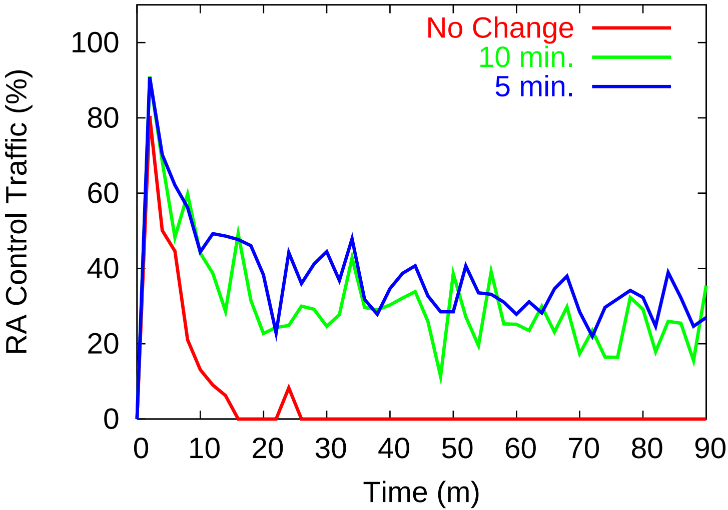
- RAs propagate predicates very quickly

Traffic Filtering



- The amount of false positives remains under 10%

Control Traffic Stability



total msg traffic 40 msg/min

- The amount of RA traffic is stable and contained

Summary of CBCB Routing

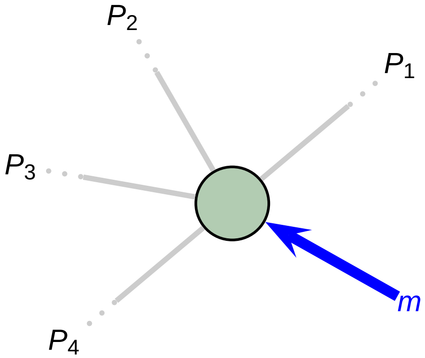
- Content-based routing protocol
- Generic networks (i.e., unrestricted topology)
- Idea 1: use a broadcast layer
- Idea 2: use a “push/pull” routing protocol
- Good behavior for both functionality and stability
- Software and documentation available at
  http://www.cs.colorado.edu/serl/cbn/

Part II

A Concrete Forwarding Algorithm

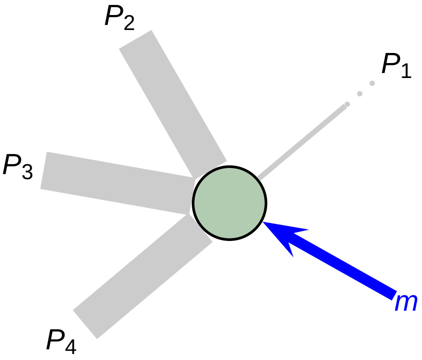
Content-Based Forwarding

- Forwarding table: *interface* ↔ *predicate*



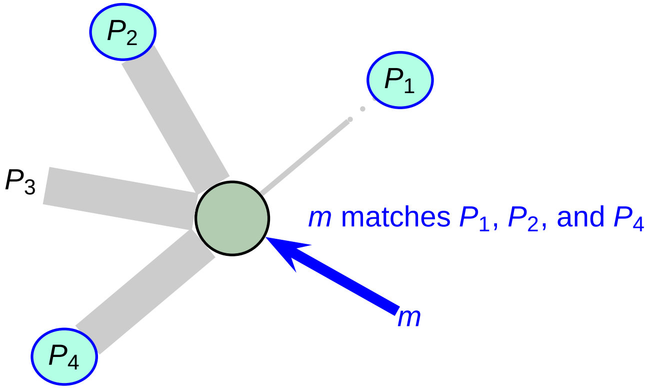
Content-Based Forwarding

- Forwarding table: *interface* $\leftrightarrow$ *predicate*
- Broadcast forwarding (or other constraints)



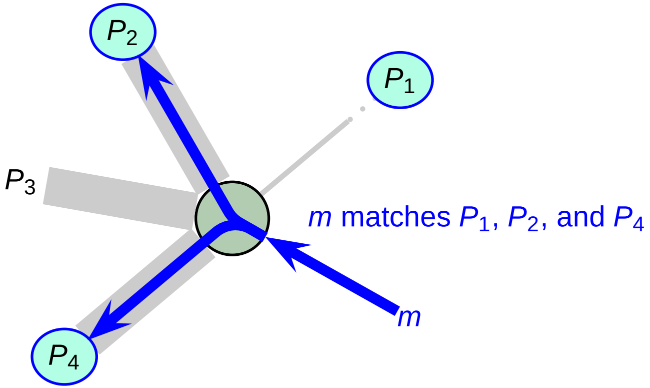
Content-Based Forwarding

- Forwarding table: *interface* $\leftrightarrow$ *predicate*
- Broadcast forwarding (or other constraints)



Content-Based Forwarding

- Forwarding table: *interface* $\leftrightarrow$ *predicate*
- Broadcast forwarding (or other constraints)
- Content-based forwarding

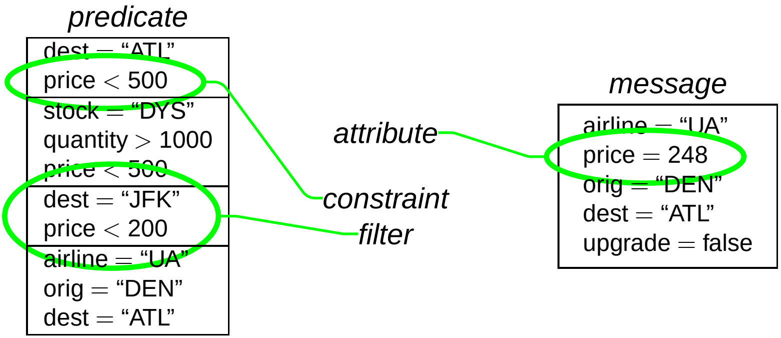


Predicates and Messages

predicate
dest $=$ "ATL" price $<$ 500
stock $=$ "DYS" quantity $>$ 1000 price $<$ 500
dest $=$ "JFK" price $<$ 200
airline $=$ "UA" orig $=$ "DEN" dest $=$ "ATL"

message
airline $=$ "UA" price $=$ 248 orig $=$ "DEN" dest $=$ "ATL" upgrade $=$ false

Predicates and Messages



- *Predicate:* a disjunction of *filters*
- *Filter:* a conjunction of *constraints*
- *Constraint:* a condition on the value of an *attribute*

Matching Problem

forwarding table

I_1	dest $=$ "ATL" price $<$ 500
	stock $=$ "DYS" quantity $>$ 1000 price $<$ 500
I_2	airline $=$ "UA" orig $=$ "DEN" dest $=$ "ATL"
	dest $=$ "JFK" price $<$ 200
	orig $=$ "DEN"
	airline $=$ "UA" upgrade $=$ true
I_3	stock $=$ "MSFT" price $<$ 200

message

airline $=$ "UA" fare $=$ "T" price $=$ 248 orig $=$ "DEN" dest $=$ "ATL" upgrade $=$ false

Matching Problem

forwarding table

I_1	dest = "ATL" price < 500
	stock = "DYS" quantity > 1000 price < 500
I_2	airline = "UA" orig = "DEN" dest = "ATL"
	dest = "JFK" price < 200
	orig = "DEN"
	airline = "UA" upgrade = true
I_3	stock = "MSFT" price < 200

message

airline = "UA" fare = "T" price = 248 orig = "DEN" dest = "ATL" upgrade = false

Matching Problem

forwarding table

I_1	dest $=$ "ATL" price $<$ 500
	stock $=$ "DYS" quantity $>$ 1000 price $<$ 500
I_2	airline $=$ "UA" orig $=$ "DEN" dest $=$ "ATL"
	dest $=$ "JFK" price $<$ 200
	orig $=$ "DEN"
	airline $=$ "UA" upgrade $=$ true
I_3	stock $=$ "MSFT" price $<$ 200

message

airline $=$ "UA" fare $=$ "T" price $=$ 248 orig $=$ "DEN" dest $=$ "ATL" upgrade $=$ false

Target: forwarding table containing millions of constraints

Matching Strategies

- Naïve
  - evaluate constraints one by one

Matching Strategies

- Naïve
  - evaluate constraints one by one
- Index-based
  - build an index structure for the forwarding table
  - define a look-up algorithm

Matching Strategies

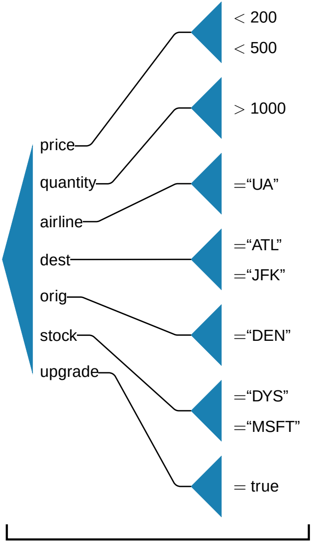
- Naïve
  - evaluate constraints one by one
- Index-based
  - build an index structure for the forwarding table
  - define a look-up algorithm
    - walk through the index as in a decision diagram [Gough+:ACSC95, Aguilera+:PODC99, Campailla+:ICSE01]
    - walk through the message [Yan+:TODS99, Fabret+:SIGMOD01, Carzaniga&Wolf:SIGCOMM03]

Predicate Index

I_1	$f_{1.1}$	dest = "ATL" price < 500
	$f_{1.2}$	stock = "DYS" quantity > 1000 price < 500
I_2	$f_{2.1}$	airline = "UA" orig = "DEN" dest = "ATL"
	$f_{2.2}$	dest = "JFK" price < 200
	$f_{2.3}$	orig = "DEN"
	$f_{2.4}$	airline = "UA" upgrade = true
I_3	$f_{3.1}$	stock = "MSFT" price < 200

Predicate Index

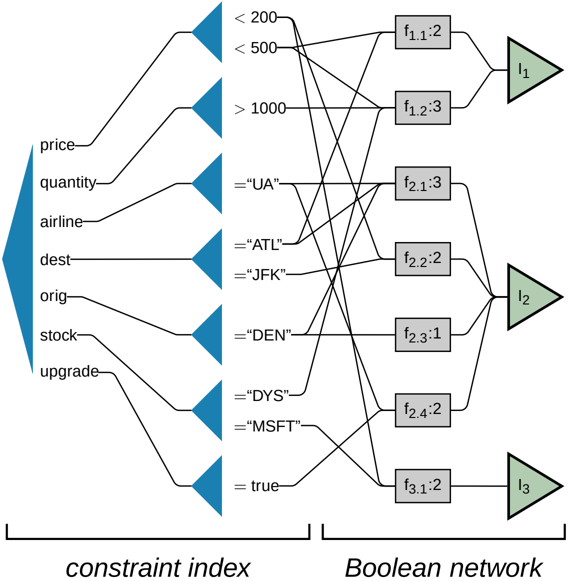
I_1	$f_{1.1}$	dest $=$ "ATL" price $<$ 500
	$f_{1.2}$	stock $=$ "DYS" quantity $>$ 1000 price $<$ 500
I_2	$f_{2.1}$	airline $=$ "UA" orig $=$ "DEN" dest $=$ "ATL"
	$f_{2.2}$	dest $=$ "JFK" price $<$ 200
	$f_{2.3}$	orig $=$ "DEN"
	$f_{2.4}$	airline $=$ "UA" upgrade $=$ true
I_3	$f_{3.1}$	stock $=$ "MSFT" price $<$ 200



constraint index

Predicate Index

I_1	$f_{1.1}$	dest = "ATL" price < 500
	$f_{1.2}$	stock = "DYS" quantity > 1000 price < 500
I_2	$f_{2.1}$	airline = "UA" orig = "DEN" dest = "ATL"
	$f_{2.2}$	dest = "JFK" price < 200
	$f_{2.3}$	orig = "DEN"
	$f_{2.4}$	airline = "UA" upgrade = true
I_3	$f_{3.1}$	stock = "MSFT" price < 200



Matching Algorithm

```
proc counting_CBF(Message msg) {
  map<Filter,int> counters ← ∅
  set<Interface> exclude ← broadcast_exclude(msg)
  foreach attribute in msg {
    set<Constraint> C ← find_matching_constraints(attribute)
    foreach constraint in C {
      foreach filter in constraint.filters {
        if filter.interface ∉ exclude {
          if filter ∉ counters {
            counters ← counters ∪ ⟨filter,0⟩ }
          counters[filter] ← counters[filter] + 1
          if counters[filter] = filter.size {
            output(msg, filter.interface)
            exclude ← exclude ∪ {filter.interface}
            if |exclude| = total_interface_count {
              return } } } } } } }
```

Matching Algorithm

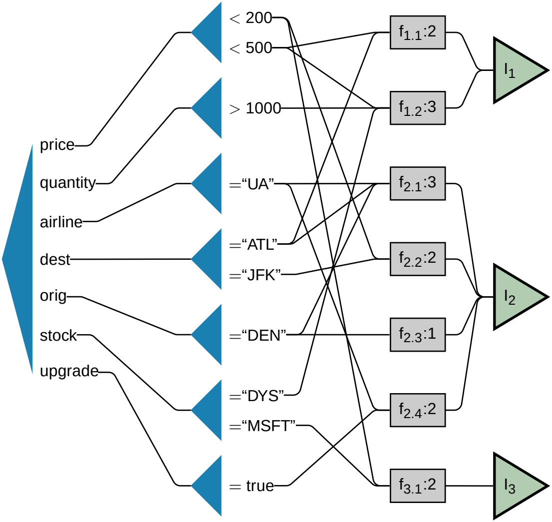
```
proc counting_CBF(Message msg) {
  map<Filter,int> counters ← ∅
  set<Interface> exclude ← broadcast_exclude(msg)
  foreach attribute in msg {
    set<Constraint> C ← find_matching_constraints(attribute)
    foreach constraint in C {
      foreach filter in constraint.filters {
        if filter.interface ∉ exclude {
          if filter ∉ counters {
            counters ← counters ∪ ⟨filter,0⟩ }
          counters[filter] ← counters[filter] + 1
          if counters[filter] = filter.size {
            output(msg, filter.interface)
            exclude ← exclude ∪ {filter.interface}
            if |exclude| = total_interface_count {
              return } } } } } } }
```

Counting Algorithm

message

airline = "UA"
fare = "T"
price = 248
orig = "DEN"
dest = "ATL"
upgrade = false

// local variables
excluded = $\{I_3\}$
output = $\emptyset$

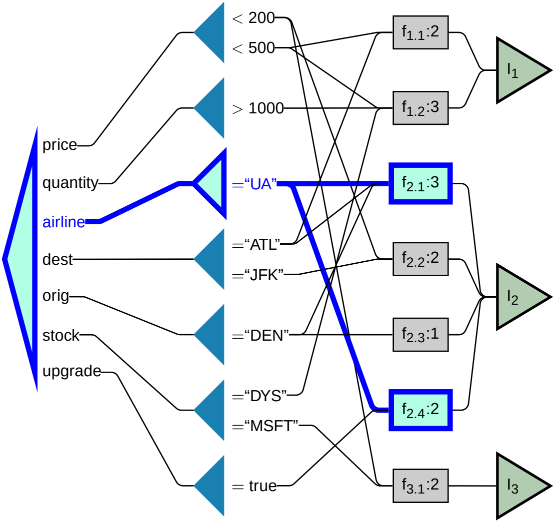


Counting Algorithm

message

airline $=$ “UA”
fare $=$ “T”
price $=$ 248
orig $=$ “DEN”
dest $=$ “ATL”
upgrade $=$ false

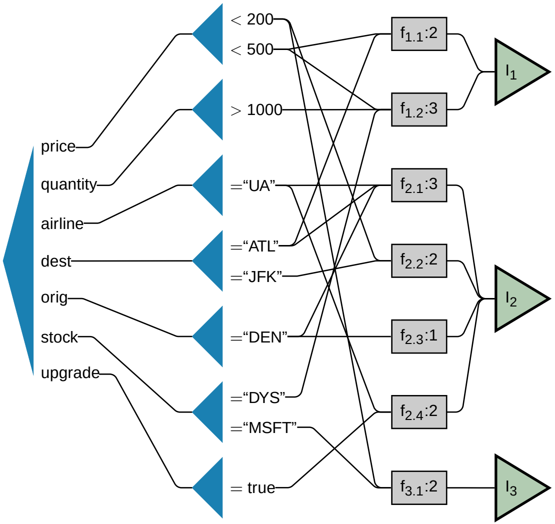
// local variables
excluded $= \{I_3\}$
output $= \emptyset$
counter[$f_{2.1}$] $= 1/3$
counter[$f_{2.4}$] $= 1/2$



Counting Algorithm

message

airline = "UA"
fare = "T"
price = 248
orig = "DEN"
dest = "ATL"
upgrade = false

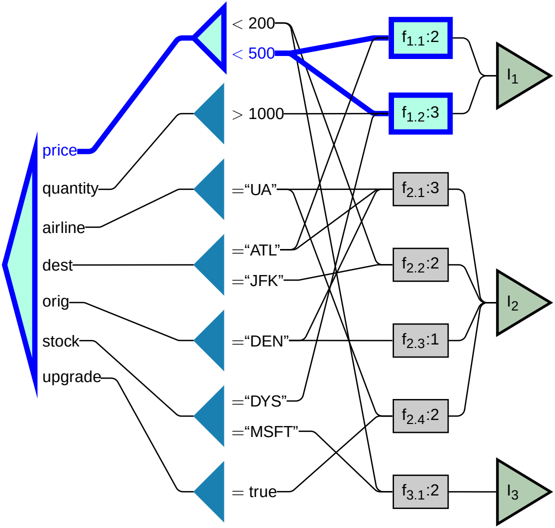


// local variables
excluded = $\{I_3\}$
output = $\emptyset$
counter[$f_{2.1}$] = 1/3
counter[$f_{2.4}$] = 1/2

Counting Algorithm

message

airline = "UA"
fare = "T"
price = 248
orig = "DEN"
dest = "ATL"
upgrade = false



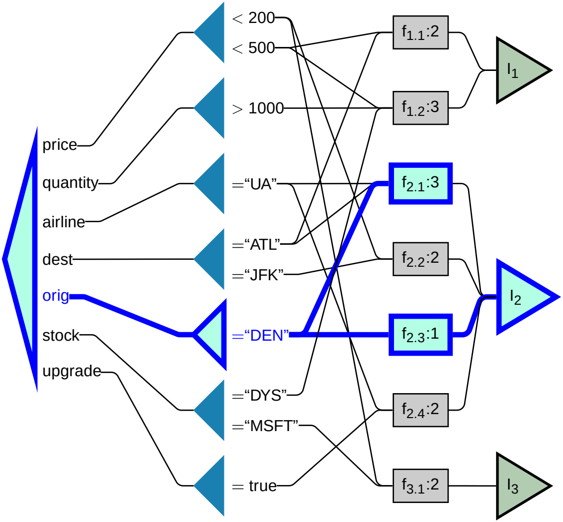
// local variables
excluded $= \{I_3\}$
output $= \emptyset$
counter[$f_{2.1}$] = 1/3
counter[$f_{2.4}$] = 1/2
counter[$f_{1.1}$] = 1/2
counter[$f_{1.2}$] = 1/3

Counting Algorithm

message

airline = "UA"
fare = "T"
price = 248
orig = "DEN"
dest = "ATL"
upgrade = false

// local variables
excluded = $\{I_3, I_2\}$
output = $\{I_2\}$
counter[$f_{2.1}$] = 2/3
counter[$f_{2.4}$] = 1/2
counter[$f_{1.1}$] = 1/2
counter[$f_{1.2}$] = 1/3
counter[$f_{2.3}$] = 1/1

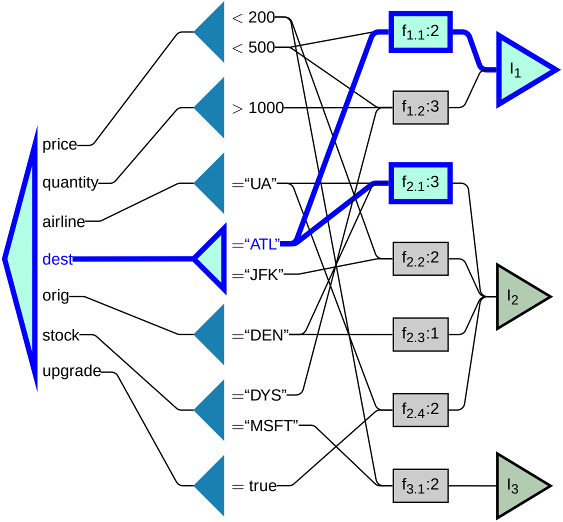


Counting Algorithm

message

airline = “UA”
fare = “T”
price = 248
orig = “DEN”
dest = “ATL”
upgrade = false

// local variables
excluded = $\{I_3, I_2, I_1\}$
output = $\{I_2, I_1\}$
counter[$f_{2.1}$] = 2/3
counter[$f_{2.4}$] = 1/2
counter[$f_{1.1}$] = 2/2
counter[$f_{1.2}$] = 1/3
counter[$f_{2.3}$] = 1/1

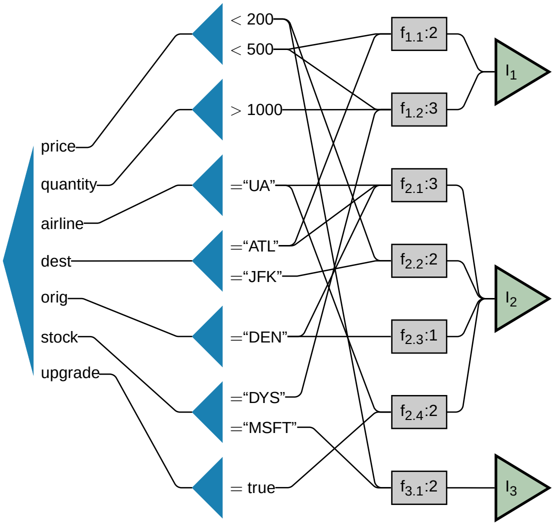


Counting Algorithm

message

airline = "UA"
fare = "T"
price = 248
orig = "DEN"
dest = "ATL"
upgrade = false

// local variables
excluded = $\{I_3, I_2, I_1\}$
output = $\{I_2, I_1\}$
counter[$f_{2.1}$] = 2/3
counter[$f_{2.4}$] = 1/2
counter[$f_{1.1}$] = 2/2
counter[$f_{1.2}$] = 1/3
counter[$f_{2.3}$] = 1/1



Evaluation

- C++ implementation
- Synthetic workloads
- Experiments on a 950Mhz computer with 512Mb

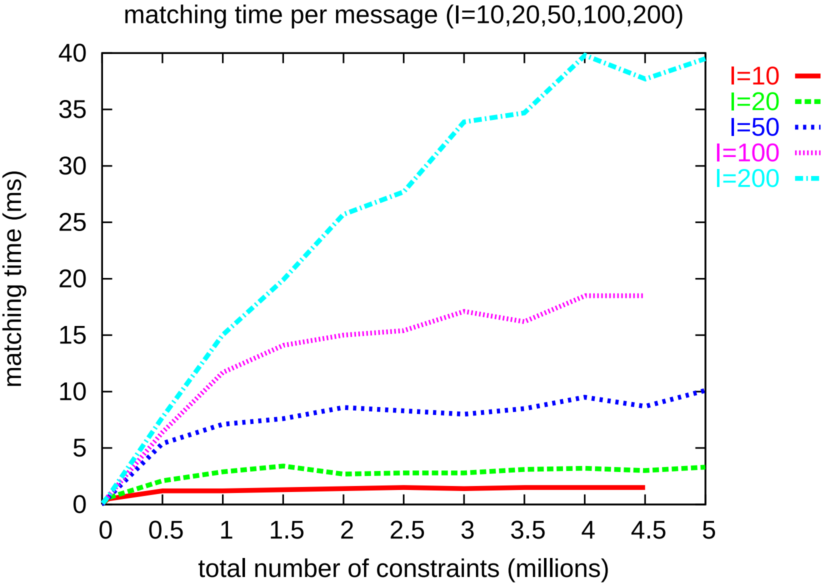
Evaluation

- C++ implementation
- Synthetic workloads
- Experiments on a 950Mhz computer with 512Mb
  - okay, remember that this was done in 2002
  - much better results today thanks to progress in CPU speeds

Workload Parameters

- Messages: 5–10 attributes
- Filters: 1–6 constraints
- Attributes and values: dictionary of 1000 words with Zipf distribution
- Operators
  - *integers*: 60% equality, 20% less-than, and 20% greater-than
  - *strings*: 35% equality, 15% prefix, 15% suffix, 15% substring, 10% less-than, and 10% greater-than
- Forwarding table: up to 5M constraints, from 2 interfaces to 1M interfaces

Main Results

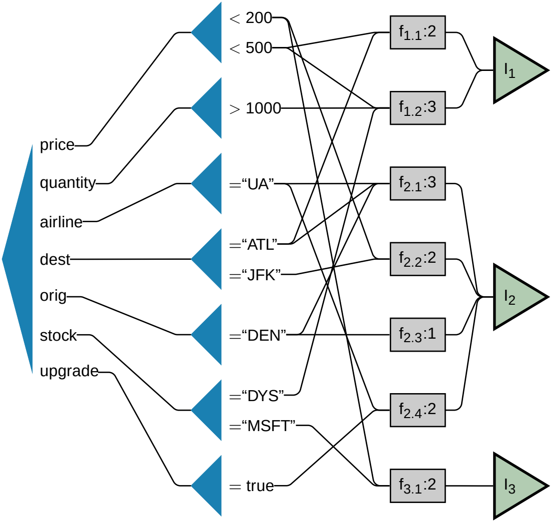


An Improvement

message

airline = “UA”
fare = “T”
price = 248
orig = “DEN”
dest = “ATL”
upgrade = false

// local variables
excluded = $\{I_3\}$
output = $\emptyset$

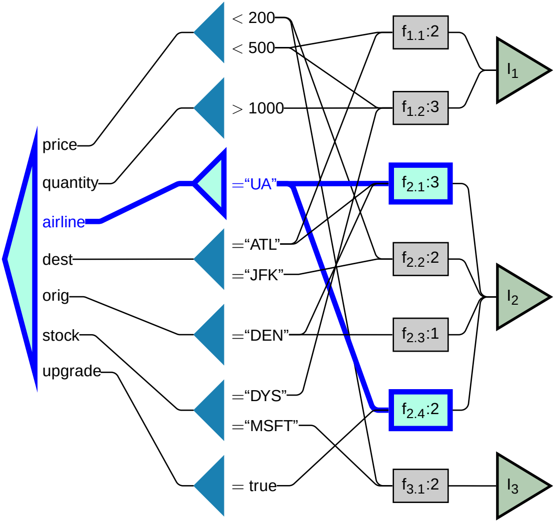


An Improvement

message

airline = "UA"
fare = "T"
price = 248
orig = "DEN"
dest = "ATL"
upgrade = false

// local variables
excluded = $\{I_3\}$
output = $\emptyset$
counter[$f_{2.1}$] = 1/3
counter[$f_{2.4}$] = 1/2

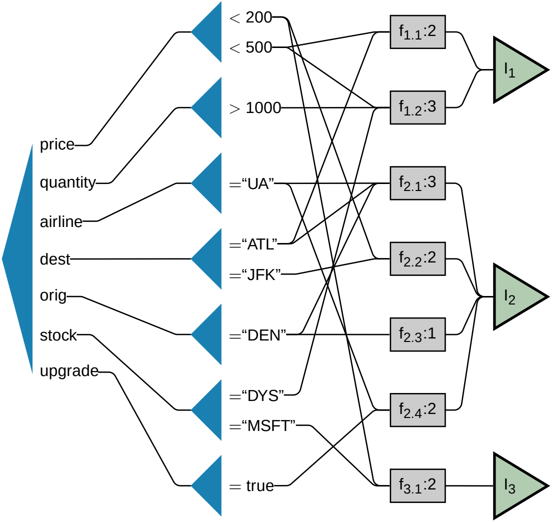


An Improvement

message

airline = "UA"
fare = "T"
price = 248
orig = "DEN"
dest = "ATL"
upgrade = false

// local variables
excluded = $\{I_3\}$
output = $\emptyset$
counter[$f_{2.1}$] = 1/3
counter[$f_{2.4}$] = 1/2

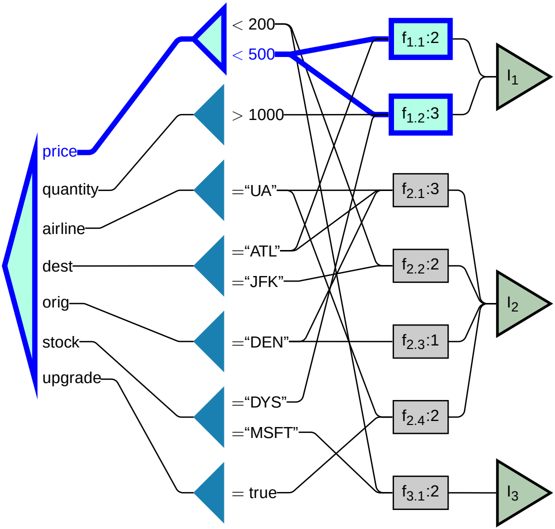


An Improvement

message

airline = “UA”
fare = “T”
price = 248
orig = “DEN”
dest = “ATL”
upgrade = false

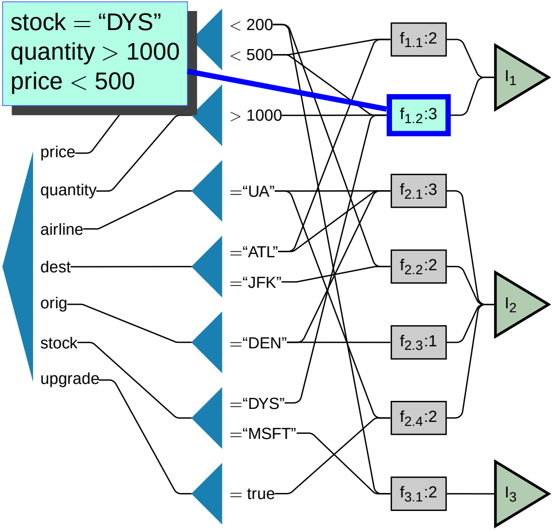
// local variables
excluded = $\{I_3\}$
output = $\emptyset$
counter[$f_{2.1}$] = 1/3
counter[$f_{2.4}$] = 1/2
counter[$f_{1.1}$] = 1/2
counter[$f_{1.2}$] = 1/3



An Improvement

message

airline $=$ "UA"
fare $=$ "T"
price $=$ 248
orig $=$ "DEN"
dest $=$ "ATL"
upgrade $=$ false



// local variables
excluded $= \{I_3\}$
output $= \emptyset$
counter[$f_{2.1}$] $=$ 1/3
counter[$f_{2.4}$] $=$ 1/2
counter[$f_{1.1}$] $=$ 1/2
counter[$f_{1.2}$] $=$ 1/3

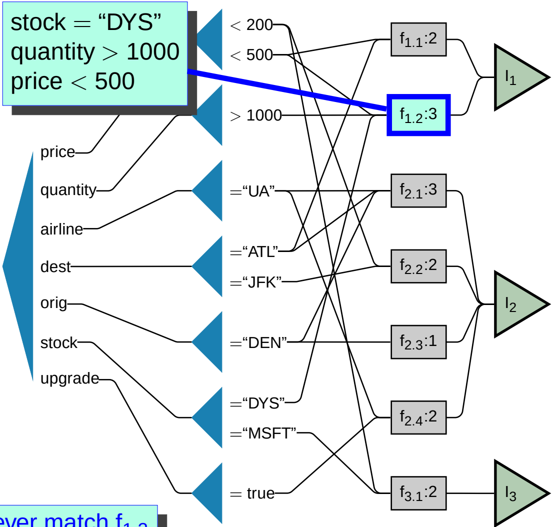
An Improvement

message

airline = “UA”
fare = “T”
price = 248
orig = “DEN”
dest = “ATL”
upgrade = false

// local variables
excluded = $\{I_3\}$
output = $\emptyset$
counter[$f_{2.1}$] = 1/3
counter[$f_{2.4}$] = 1/2
counter[$f_{1.1}$] = 1/2
counter[$f_{1.2}$] = 1/3

stock = “DYS”
quantity > 1000
price < 500



Useless—the message will never match $f_{1.2}$

Idea: Bloom Filters

We can use *Bloom filters* to represent set of names in a filter f and in a message m

Idea: Bloom Filters

We can use *Bloom filters* to represent set of names in a filter *f* and in a message *m*

message m	*filter f*
airline = "UA" fare = "T" price = 248 orig = "DEN" dest = "ATL" upgrade = false	stock = "DYS" quantity $>$ 1000 price $<$ 500
$B_m = [0110100011]$	$B_f = [0010010010]$

Idea: Bloom Filters

We can use *Bloom filters* to represent set of names in a filter f and in a message m

message m	*filter f*
airline = "UA" fare = "T" price = 248 orig = "DEN" dest = "ATL" upgrade = false	stock = "DYS" quantity > 1000 price < 500
$B_m = [0110100011]$	$B_f = [0010010010]$

If $B_m \not\supseteq B_f$ then we can immediately skip f (i.e., we don't bother maintaining a counter for f, and we don't look up f's interface, etc.)

Observations on Bloom Filters

- Bloom filters for filters (B_f) are computed statically with the forwarding table

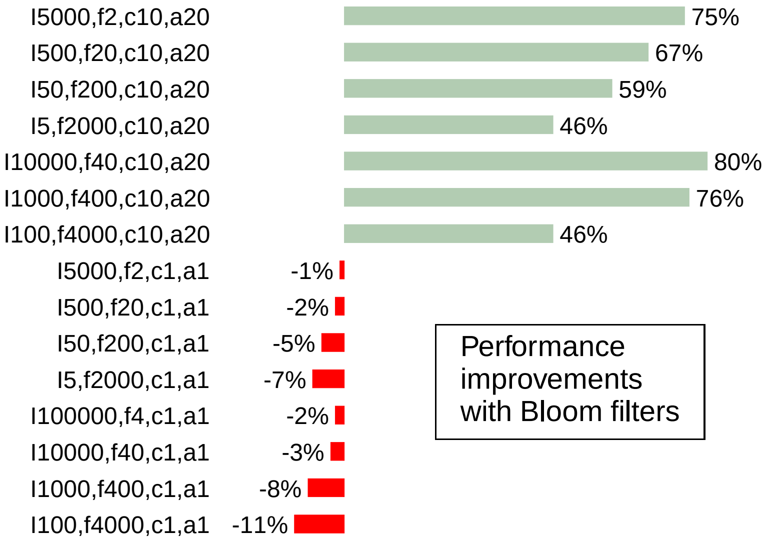
- B_m is computed dynamically, but we can use very simple hash functions

- The complexity of checking $B_f \subseteq B_m$ is $O(1)$
  - in C: `(Bf & Bm) == Bf`

- False positives do not affect correctness

- The idea works with messages with sets of attributes that do not always “cover” filters

Experimental Results



A Further Improvement

Observation:

- By excluding interfaces, we can short-circuit the evaluation of a message and speed up forwarding

A Further Improvement

Observation:

- By excluding interfaces, we can short-circuit the evaluation of a message and speed up forwarding

Idea:

- Compute a table of “selective” attributes
  - an attribute a is *selective* for an interface i if a must exist in a message m in order for m to match P_i
  - i.e., a must appear in every conjunct of the disjunct P_i
- Use the *selectivity table* to *exclude* interfaces from processing (*selectivity preprocessing*)

Selectivity Table Example

forwarding table

I_1	dest = "ATL" price < 500
	stock = "DYS" quantity > 1000 price < 500
I_2	airline = "UA" orig = "DEN" dest = "ATL"
	dest = "JFK" price < 200
	orig = "DEN"
	airline = "UA" upgrade = true
I_3	stock = "MSFT" price < 200

Selectivity Table Example

forwarding table

I_1	dest $=$ “ATL” price $<$ 500
	stock $=$ “DYS” quantity $>$ 1000 price $<$ 500
I_2	airline $=$ “UA” orig $=$ “DEN” dest $=$ “ATL”
	dest $=$ “JFK” price $<$ 200
	orig $=$ “DEN”
	airline $=$ “UA” upgrade $=$ true
I_3	stock $=$ “MSFT” price $<$ 200

selectivity table

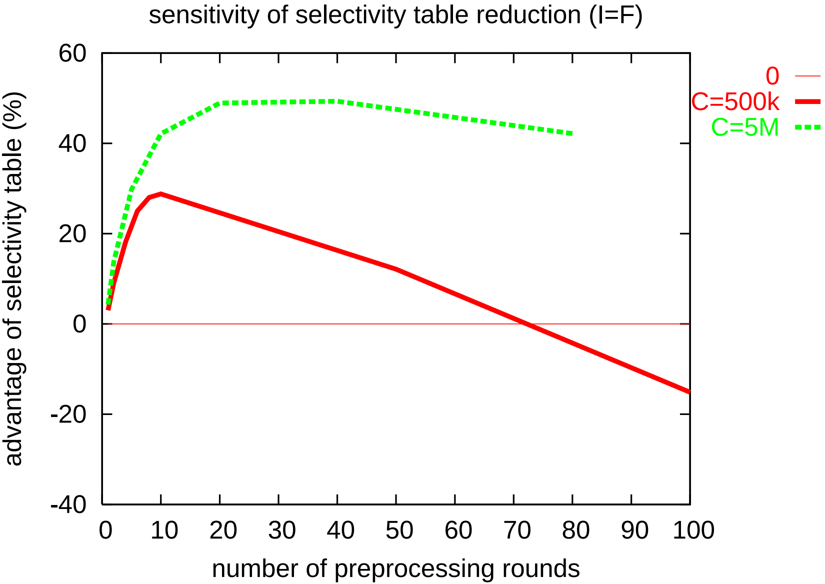
price	I_1, I_3
stock	I_3

Selectivity Preprocessing

```
map<Name, set<Interface>> selectivity_table
int preprocessing_rounds

proc preprocess(Message msg, set<Interface> exclude) {
  int rounds ← preprocessing_rounds
  foreach ⟨attribute,selectivity⟩ in selectivity_table {
    if rounds = 0
      return exclude
    rounds ← rounds − 1
    if attribute ∉ msg {
      exclude ← exclude ∪ selectivity
      if |exclude| = total_interface_count
        return exclude
    }
  }
  return exclude
}
```

Sensitivity to Preprocessing



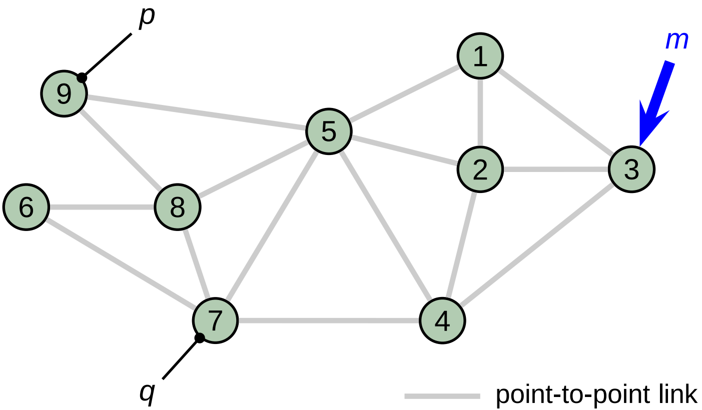
Summary of C-B Forwarding

- Focus on performance and scalability
- Predicate index with lookup based on iteration over the input message
- Novel ideas
  - short-circuit evaluation of disjunctions
  - use *Bloom filters* to exclude conjunctions
  - use (absence of) *selective attributes* to exclude entire disjunctions
- Experiments show good absolute performance and a synergistic behavior of our optimizations
- Software and documentation available at http://www.inf.unisi.ch/carzaniga/cbn/

Part III

Theory of Content-Based Routing

Content-Based Routing



- Where and how to forward *m*?
- Based on which kind of routing information?

Theory of Content-Based Routing

- State of the art
  - a number of concrete routing protocols (including ours)
  - validation through simulation
  - focus on the exchange of routing information

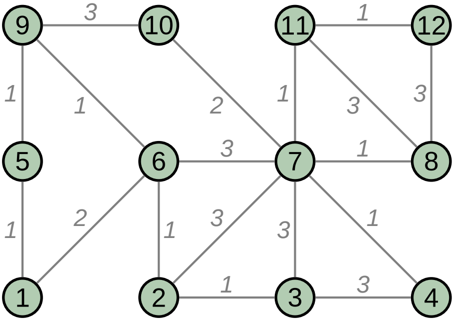
Theory of Content-Based Routing

- State of the art
  - a number of concrete routing protocols (including ours)
  - validation through simulation
  - focus on the exchange of routing information
- New research: *theoretical foundations of content-based routing*
  - provable properties of a protocol
  - properties of content-based routing
    - i.e., properties of *any* protocol
  - focus on routing *state* (i.e., memory complexity)

Research Plan

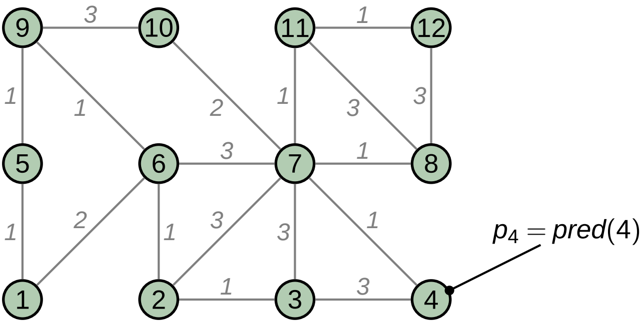
- Models
  - network model
  - general model of content-based routing (forwarding)
  - model of routing information and its space complexity
- Analysis of specific routing protocols
  - upper bounds for the space complexity of content-based routing
- New improved routing protocols
  - design of light-weight content-based routing protocols
- General analysis
  - lower bounds

Content-Based Network Model



- $CBN = (V, E, weight, \mathscr{M}, \mathscr{P}, pred)$
  - $v \in V$ 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

Content-Based Network Model



- $CBN = (V, E, weight, \mathscr{M}, \mathscr{P}, pred)$
  - $v \in V$ 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
  - $\mathscr{M}$ is a set of *messages*
  - $\mathscr{P}$ is a set of *predicates*; $p \in \mathscr{P}$ is a function $p : \mathscr{M} \to \{0,1\}$
  - $pred : V \to \mathscr{P}$ associates a processor $v \in V$ to a predicate $p \in \mathscr{P}$

Content-Based Routing Scheme

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

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

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

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; $m \in \mathscr{M}$
  - $h = hdr(c)$ is a *header*; $h \in \mathscr{H}$
  - a scheme defines $\mathscr{H}$, 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

Per-Process Routing Functions

For each processor v

Per-Process Routing Functions

For each processor v

- *Initial header function*

$$\mathbf{Init}_v : \mathscr{M} \rightarrow \mathscr{H}$$

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

Per-Process Routing Functions

For each processor v

- *Initial header function*

$$\mathbf{Init}_v : \mathscr{M} \rightarrow \mathscr{H}$$

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

- *Header (rewriting) function*

$$\mathbf{Hdr}_v : \mathscr{H} \rightarrow \mathscr{H}$$

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

Per-Process Routing Functions

For each processor v

- *Initial header function*

$$\mathbf{Init}_v : \mathscr{M} \rightarrow \mathscr{H}$$

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

- *Header (rewriting) function*

$$\mathbf{Hdr}_v : \mathscr{H} \rightarrow \mathscr{H}$$

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

- *Forwarding function*

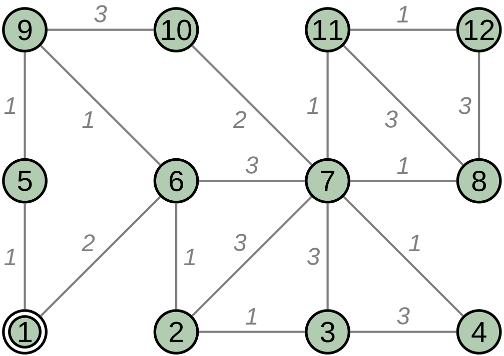
$$\mathbf{Fwd}_v : \mathscr{H} \times \mathscr{M} \rightarrow \mathbb{P}(\mathit{neighbors}(v))$$

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

Per-Source Forwarding (PSF) Scheme

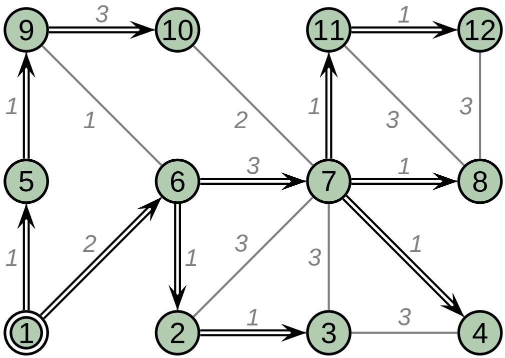
Per-Source Forwarding (PSF) Scheme

- Idea



Per-Source Forwarding (PSF) Scheme

- Idea
  - per-source spanning trees T_v



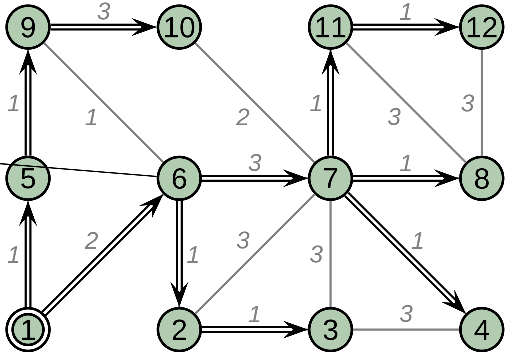
Per-Source Forwarding (PSF) Scheme

- Idea
  - per-source spanning trees T_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 $\rightarrow$ *predicate*

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



- Headers are used to store the source of a message

$$\mathscr{H} = V$$

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

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

PSF Scheme

- Headers are used to store the source of a message

$$\mathscr{H} = V$$

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

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

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

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

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

Analysis of PSF

- It is easy to prove that *PSF is correct*
  - correctness not yet defined, but quite straightforward

Analysis of PSF

- It is easy to prove that *PSF is correct*
  - correctness not yet defined, but quite straightforward

- How “expensive” is PSF?

- How much memory does it require?
  - focus on the *total memory requirement*

Analysis of PSF

- It is easy to prove that *PSF is correct*
  - correctness not yet defined, but quite straightforward

- How "expensive" is PSF?

- How much memory does it require?
  - focus on the *total memory requirement*

- Preliminary additional definitions
  - $M(X)$ denotes the memory requirements of a function or set X
    - e.g., processor v uses $M(\mathbf{Hdr}_v)$ bits to represent its **Hdr** function
  - $n = |V|$, therefore $M(v) = O(\log n)$
  - $S \subseteq V$ is a given set of senders, with $s = |S|$
  - $R \subseteq V, R = \{u \in V | pred(u) \neq \emptyset\}$, is the set of receivers, $r = |R|$

Memory Requirements of PSF

$$M(PSF) = \sum_{u \in V} \big(M(\mathbf{Init}_u) + M(\mathbf{Hdr}_u) + M(\mathbf{Fwd}_u)\big)$$

Memory Requirements of PSF

$$M(PSF) = \sum_{u \in V} \big(M(\mathbf{Init}_u) + M(\mathbf{Hdr}_u) + M(\mathbf{Fwd}_u) \big)$$

- $\mathbf{Init}_v$ must store v, so

$$M(\mathbf{Init}_u) = O(\log n)$$

- **Hdr** has zero memory requirements

Memory Requirements of PSF

$$M(PSF) = \sum_{u \in V} \left(M(\mathbf{Init}_u) + M(\mathbf{Hdr}_u) + M(\mathbf{Fwd}_u) \right)$$

- $\mathbf{Init}_v$ must store v, so

$$M(\mathbf{Init}_u) = O(\log n)$$

- **Hdr** has zero memory requirements
- The memory requirement of **Fwd** boils down to that of F

$$M(\mathbf{Fwd}_u) = M(F_u)$$

- The total memory requirement of **Fwd** is the sum of the memory requirements of each per-source tree

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

Memory Requirements of PSF (2)

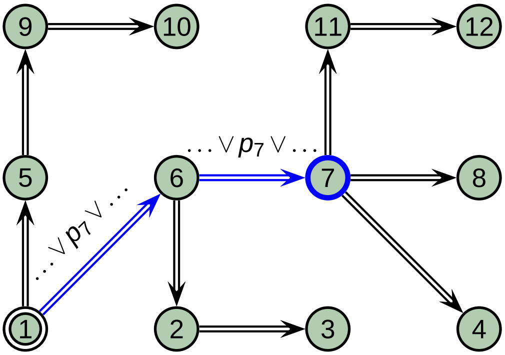
- Memory requirement of a source-rooted tree T_v

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

Memory Requirements of PSF (2)

- Memory requirement of a source-rooted tree T_v

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

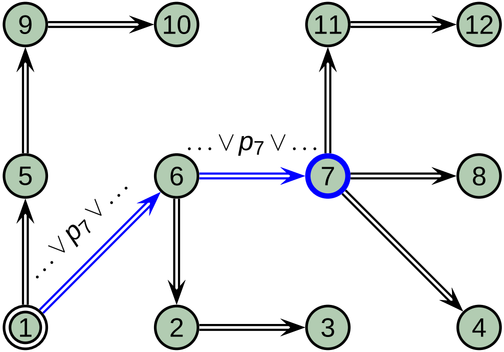


notation:
$p_7 = pred(7)$

Memory Requirements of PSF (2)

- Memory requirement of a source-rooted tree T_v

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



notation:
$p_7 = pred(7)$

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

Memory Requirements of PSF (3)

- Total memory requirement for PSF

$$M(PSF) = \sum_{v \in S} \left(\log n + \sum_{x \in R} M(pred(x)) distance(v, x) \right)$$

Memory Requirements of PSF (3)

- Total memory requirement for PSF

$$M(PSF) = \sum_{v \in S} \left(\log n + \sum_{x \in R} M(pred(x))distance(v,x) \right)$$

- 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

Memory Requirements of PSF (3)

- Total memory requirement for PSF

$$M(PSF) = \sum_{v \in S} \left(\log n + \sum_{x \in R} M(pred(x))distance(v,x) \right)$$

- 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

$$M(PSF) = s \log n + srM_p d$$

Memory Requirements of PSF (3)

- Total memory requirement for PSF

$$M(PSF) = \sum_{v \in S} \left(\log n + \sum_{x \in R} M(pred(x)) distance(v,x)\right)$$

- 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

$$M(PSF) = s \log n + srM_p d$$

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

Memory Requirements of PSF (3)

- Total memory requirement for PSF

$$M(PSF) = \sum_{v \in S} \left(\log n + \sum_{x \in R} M(pred(x)) distance(v,x) \right)$$

- 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

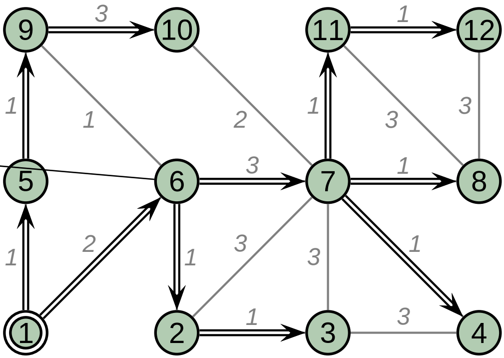
$$M(PSF) = s \log n + srM_p d$$

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

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

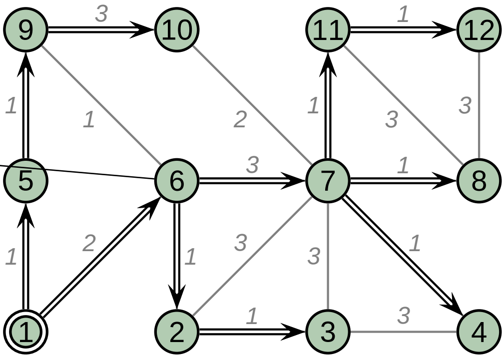
Assumptions on $M(F_u)$

F_6: annotations for proc. 6
source,next-hop $\rightarrow$ *predicate*
$\ldots$
$1,1 \mapsto \emptyset$
$1,2 \mapsto p_2 \vee p_3$
$1,7 \mapsto p_4 \vee p_7 \vee p_8 \vee p_{11} \vee p_{12}$
$1,9 \mapsto \emptyset$
$\ldots$



Assumptions on $M(F_u)$

F_6: annotations for proc. 6
source,next-hop $\rightarrow$ *predicate*
$\ldots$
$1,1 \mapsto \emptyset$
$1,2 \mapsto p_2 \vee p_3$
$1,7 \mapsto p_4 \vee p_7 \vee p_8 \vee p_{11} \vee p_{12}$
$1,9 \mapsto \emptyset$
$\ldots$



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

Assumptions on $M(F_u)$

F_6: annotations for proc. 6

source,next-hop $\rightarrow$ *predicate*

...

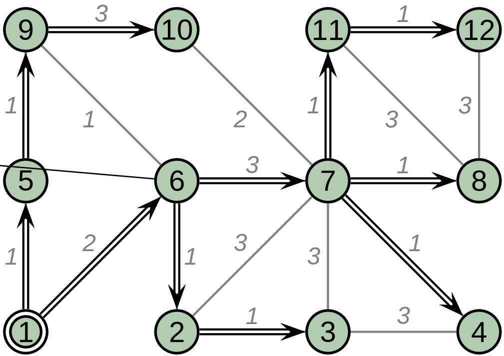
$1,1 \mapsto \emptyset$

$1,2 \mapsto p_2 \vee p_3$

$1,7 \mapsto p_4 \vee p_7 \vee p_8 \vee p_{11} \vee p_{12}$

$1,9 \mapsto \emptyset$

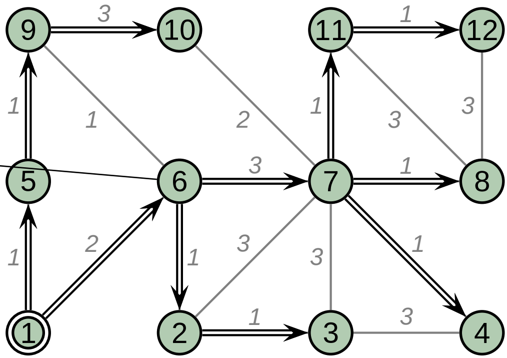
...



- The previous analysis and results assume that, e.g., $M(p_2 \vee p_3) = M(p_2) + M(p_3)$
- In general, $M(p_2 \vee p_3) \leq M(p_2) + M(p_3)$

Assumptions on $M(F_u)$

F_6: annotations for proc. 6
source,next-hop $\rightarrow$ *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 and results assume 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., $p_2 = (\text{port} > 1000) \wedge (\text{port} < 3000)$ and $p_3 = (\text{port} > 2000) \wedge (\text{port} < 4000)$ can be combined in the disjunction $p_2 \vee p_3 = (\text{port} > 1000) \wedge (\text{port} < 4000)$

Disjunction Advantage

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

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

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

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

Disjunction Advantage

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

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

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

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

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

- Can we quantify α?

α in a Generic Predicate Model

- A predicate $p \in \mathscr{P}$ is a subset of a finite universe of messages $\mathscr{M}$, therefore $M(p) = p \log |\mathscr{M}|$

α in a Generic Predicate Model

- A predicate $p \in \mathscr{P}$ is a subset of a finite universe of messages $\mathscr{M}$, therefore $M(p) = p \log |\mathscr{M}|$

- 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

α in a Generic Predicate Model

- A predicate $p \in \mathscr{P}$ is a subset of a finite universe of messages $\mathscr{M}$, therefore $M(p) = p \log|\mathscr{M}|$

- 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

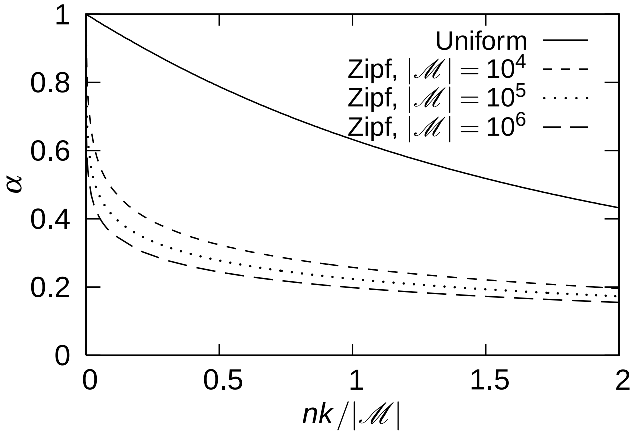
$$\Pr[m \in P] = 1 - \left(1 - \frac{h}{|\mathscr{M}|}\right)^n \approx 1 - e^{-\frac{nh}{|\mathscr{M}|}}$$

expected size of P and then the expected disjunction advantage

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

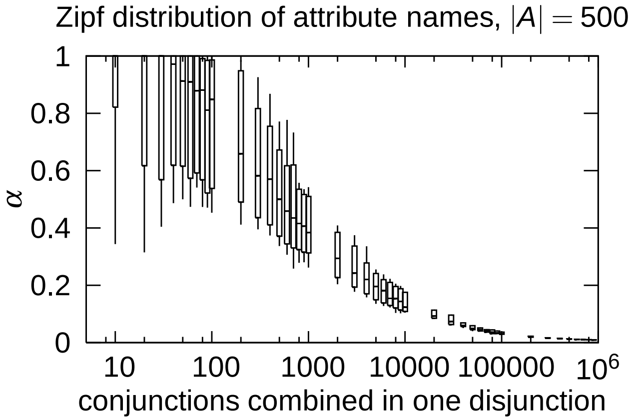
α in a Generic Predicate Model (2)

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



α in a Specific Predicate Model (1)

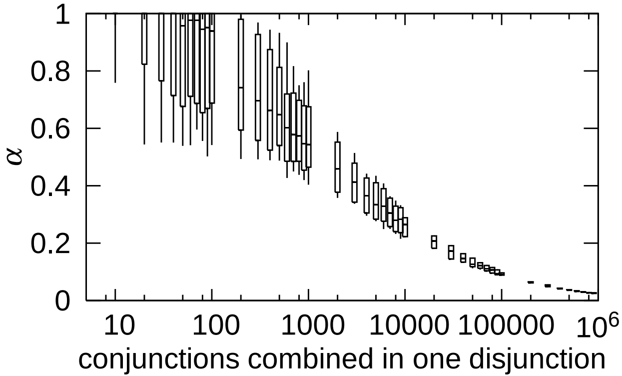
- Monte Carlo simulation
- Disjunctive normal form of attribute constraints



α in a Specific Predicate Model (2)

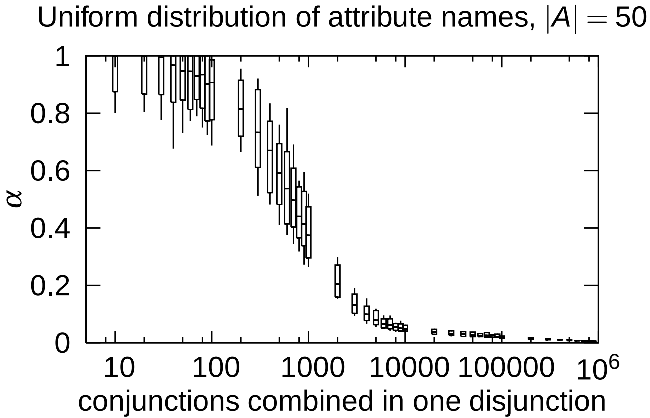
- Monte Carlo simulation
- Disjunctive normal form of attribute constraints

Zipf distribution of attribute names, $|A| = 5000$



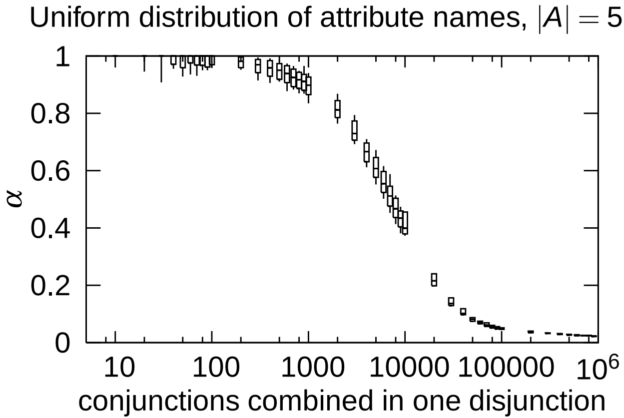
α in a Specific Predicate Model (3)

- 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



Summary

- Goal: *theoretical foundations of content-based routing*

Summary

- Goal: *theoretical foundations of content-based routing*
- Analysis of 4 routing protocols (3 existing, 1 new and improved)

scheme	*space complexity*	*delivery function*
PSF	$srM_pd + s\log n$	correct, minimal
iPS	$nrM_p\alpha(\frac{r}{\Delta})\ldots$	correct, minimal
PIF	$n^2rM_p\alpha(\frac{r}{\Delta}) + M(broadcast)$	correct, non-minimal
DRP	$rsM_p + M(unicast)$	correct, non-minimal

Summary

- Goal: *theoretical foundations of content-based routing*
- Analysis of 4 routing protocols (3 existing, 1 new and improved)

scheme	*space complexity*	*delivery function*
PSF	$srM_pd + s\log n$	correct, minimal
iPS	$nrM_p\alpha(\frac{r}{\Delta})\ldots$	correct, minimal
PIF	$n^2rM_p\alpha(\frac{r}{\Delta}) + M(broadcast)$	correct, non-minimal
DRP	$rsM_p + M(unicast)$	correct, non-minimal

- Definition and characterization of the α *reduction factor*
  - general model (analytical characterization)
  - common concrete model (Monte Carlo simulations)

Summary

- Goal: *theoretical foundations of content-based routing*
- Analysis of 4 routing protocols (3 existing, 1 new and improved)

scheme	*space complexity*	*delivery function*
PSF	$srM_pd + s\log n$	correct, minimal
iPS	$nrM_p\alpha(\frac{r}{\Delta})\ldots$	correct, minimal
PIF	$n^2rM_p\alpha(\frac{r}{\Delta}) + M(broadcast)$	correct, non-minimal
DRP	$rsM_p + M(unicast)$	correct, non-minimal

- Definition and characterization of the α *reduction factor*
  - general model (analytical characterization)
  - common concrete model (Monte Carlo simulations)
- Future work
  - more protocols and perhaps lower bounds

Part IV

Security in Content-Based Networking

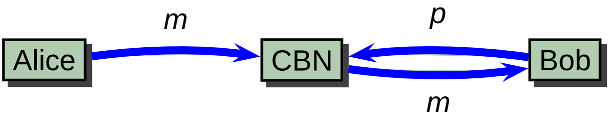
Security

Security

- Easy part
  - authentication
  - privacy with a trusted network
  - *e.g., traditional e-mail or web security*
- Difficult part
  - privacy in the presence of an untrusted network
  - *we want the network to route information on the basis of message content and receiver interests. But we do not want the network to learn anything about message content and receiver interests.*

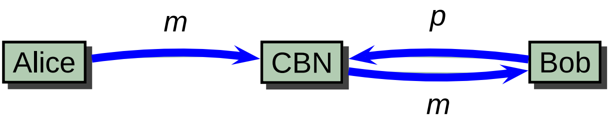
Security Model

- Simplified content-based communication scenario



Security Model

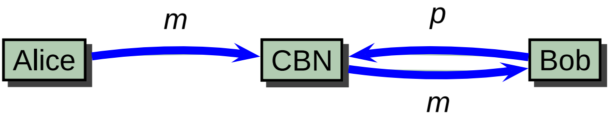
- Simplified content-based communication scenario



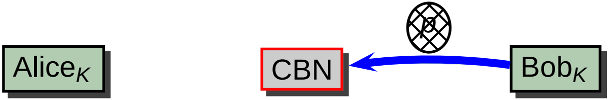
- Objectives

Security Model

- Simplified content-based communication scenario

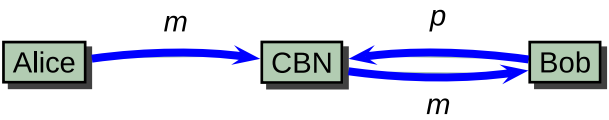


- Objectives

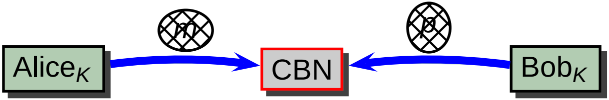


Security Model

- Simplified content-based communication scenario

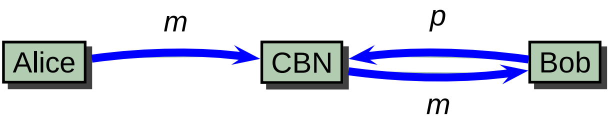


- Objectives

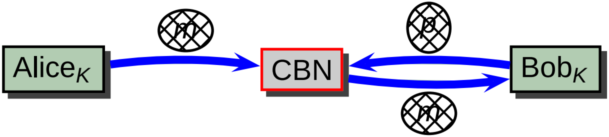


Security Model

- Simplified content-based communication scenario



- Objectives



Approximate Solutions

- *Goup anonymity*
  - receivers “hide” behind a trusted *proxy*
  - limited security

- *Overly generic predicates*
  - Bob declares a p' covering p
  - limited security (similar to group anonymity)

- *Obfuscation*
  - p is given as an “obfuscated” executable
  - incompatible with efficient routing protocols
  - limited security (dubious security for p, no security for m)

- *Computing over encrypted data*
  - either very inefficient or very limited

Approach

- Encode p and m with some K-dependent function(s)

$$
\begin{array}{ccc}
m & & p \\
\downarrow X_K & & \downarrow Y_K \\
m' & & p'
\end{array}
$$

Approach

- Encode p and m with some K-dependent function(s)

$$\begin{tikzcd} m \arrow[r, leftrightarrow] \arrow[d, "X_K"] & p \arrow[d, "Y_K"] \\ m' \arrow[r, leftrightarrow] & p' \end{tikzcd}$$

  - such that $p(m) \Leftrightarrow p'(m')$

Approach

- Encode p and m with some K-dependent function(s)

$$\begin{tikzcd}
m \arrow[r, leftrightarrow] \arrow[d, "X_K"] & p \arrow[d, "Y_K"] \\
m' \arrow[r, leftrightarrow] & p'
\end{tikzcd}$$

  - such that $p(m) \Leftrightarrow p'(m')$
  - and such that p' and m' do not reveal anything about p and m

Approach

- Encode p and m with some K-dependent function(s)

$$\begin{tikzcd}
m \arrow[r, leftrightarrow] \arrow[d, "X_K"] & p \arrow[d, "Y_K"] \\
m' \arrow[r, leftrightarrow] & p'
\end{tikzcd}$$

  - such that $p(m) \Leftrightarrow p'(m')$
  - and such that p' and m' do not reveal anything about p and m

- Method: encoding using *Bloom filters*

Bloom Filters

- Compact data structure
- Efficient set membership test
- Probabilistic result
  - false positives are possible
  - although (hopefully) improbable

Bloom Filters

- Compact data structure

- Efficient set membership test

- Probabilistic result
  - false positives are possible
  - although (hopefully) improbable

- *One-way information compression through hash functions*

Definitions

- $U = \{x_1, x_2, \ldots\}$ 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 $H : U \rightarrow \{0, 1, \ldots, m-1\}$

- $B(x)$ is computed as follows $B \leftarrow \emptyset$

  for $i \leftarrow 1 \ldots k$

  $\quad B[h_i(x)] \leftarrow 1$

Using Bloom Filters

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

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

  i.e.,
  $B \leftarrow \emptyset$
  foreach $x \in S$
      for $i \leftarrow 1 \ldots k$
          $B[h_i(x)] \leftarrow 1$

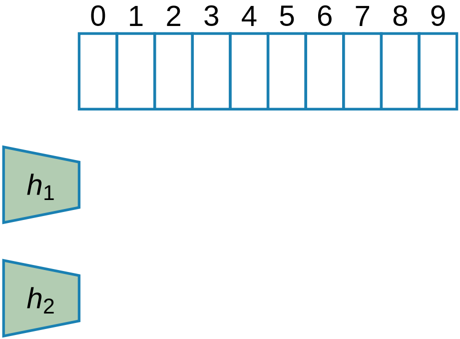
- Testing $x \in S$ amounts to testing $B(x) \subseteq B(S)$

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

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

Example

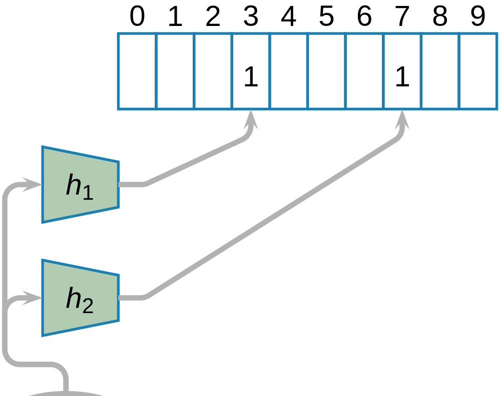
U is the universe of character strings; $k = 2$; $m = 10$



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

Example

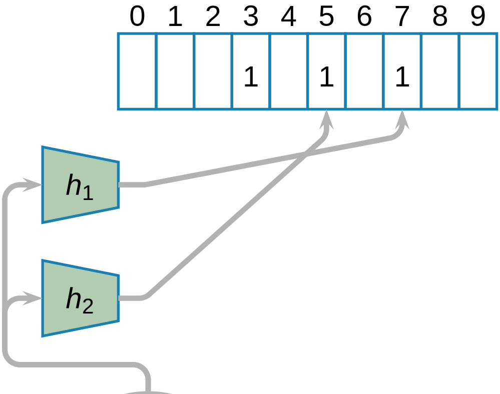
U is the universe of character strings; $k = 2$; $m = 10$



$S = \{$“ciao”, “foo”, “bar”$\}$

Example

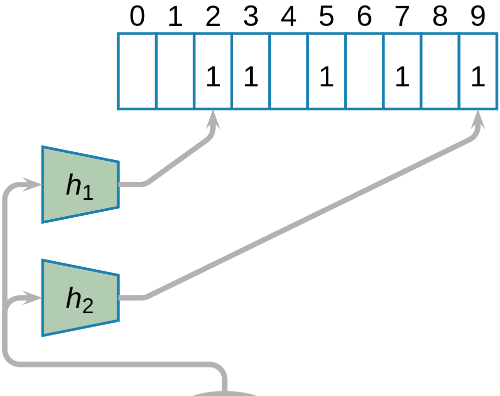
U is the universe of character strings; $k = 2$; $m = 10$



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

Example

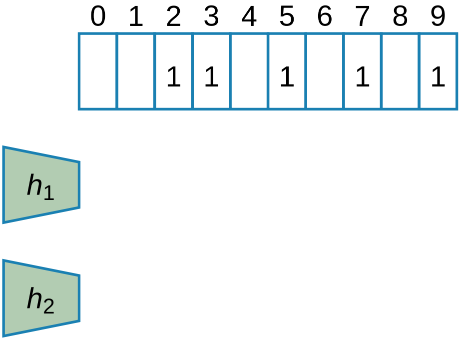
U is the universe of character strings; $k = 2$; $m = 10$



$S = \{$“ciao”, “foo”, “bar”$\}$

Example

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

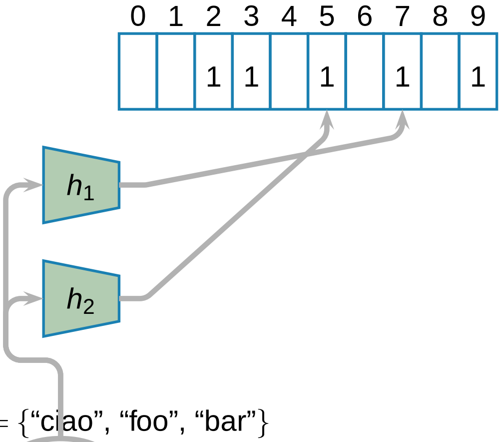


$S = \{$“ciao”, “foo”, “bar”$\}$

Test:

Example

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

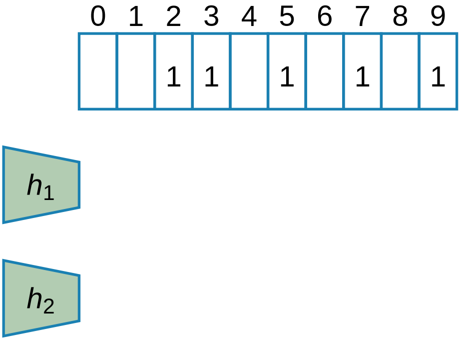


$S = \{$“ciao”, “foo”, “bar”$\}$

Test: “foo”

Example

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

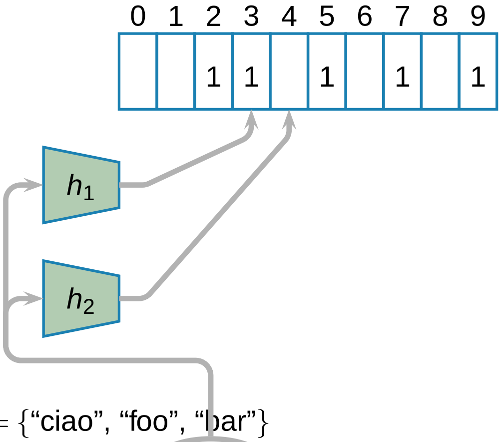


$S = \{$“ciao”, “foo”, “bar”$\}$

Test: “foo”: *yes*

Example

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

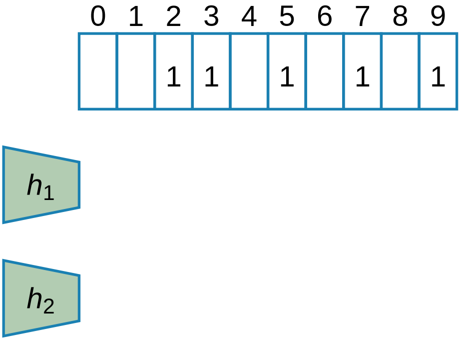


$S = \{$"ciao", "foo", "bar"$\}$

Test: "foo": *yes*, "abc"

Example

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

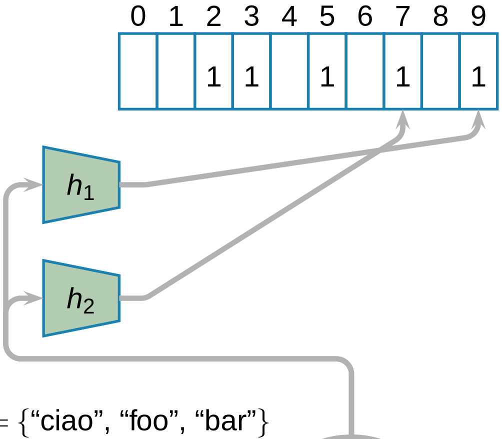


$S = \{$“ciao”, “foo”, “bar”$\}$

Test: “foo”: *yes*, “abc”: *no*

Example

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

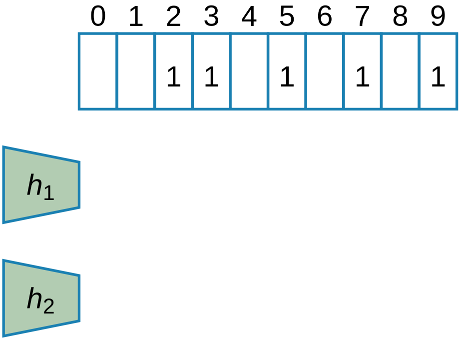


$S = \{$“ciao”, “foo”, “bar”$\}$

Test: “foo”: *yes*, “abc”: *no*, “xyz”

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)

Idea

$$m \xleftrightarrow{\;p(m)\;} p$$

Idea

$$\begin{tikzcd}
m \arrow[rr, leftrightarrow, "p(m)"] \arrow[d, "X_S"'] & & p \arrow[d, "X_P"] \\
S_m \arrow[rr, leftrightarrow, "S_m \supseteq S_p"] & & S_p
\end{tikzcd}$$

1. Reduce p and m into sets of strings, S_p and S_m, such that $p(m) \Rightarrow S_p \subseteq S_m$

Idea

$$\begin{tikzcd}
m \arrow[rr, leftrightarrow, "p(m)"] \arrow[d, "X_S"'] & & p \arrow[d, "X_P"] \\
S_m \arrow[rr, leftrightarrow, "S_m \supseteq S_p"] \arrow[d, "B^K"'] & & S_p \arrow[d, "B^K"] \\
B_m^K \arrow[rr, leftrightarrow, "B_m \supseteq B_p"] & & B_p^K
\end{tikzcd}$$

1. Reduce p and m into sets of strings, S_p and S_m, such that $p(m) \Rightarrow S_p \subseteq S_m$
2. Represent S_p and S_m with Bloom filters, B_p^K and B_m^K (the K superscript means that the Bloom filters use keyed cryptographic hash functions, with key K)

Predicate Encoding

1. Constraint encoding

   name$=$*value* (*equality* constraint)

   name$=$ *any* (*existence* constraint)

   e.g., idmef_version$>$2 $\rightarrow$ "$\exists$idmef_version", idmef_version$=$2 $\rightarrow$ "idmef_version$=$2"

Predicate Encoding

1. Constraint encoding
   name$=$*value* (*equality* constraint)
   name$=$ *any* (*existence* constraint)
   e.g., idmef_version$>$2 $\rightarrow$ "$\exists$idmef_version", idmef_version$=$2 $\rightarrow$ "idmef_version$=$2"

2. A *filter* f is encoded with a set $S_f = S_f^{=} \cup S_f^{\exists}$, representing the union of equality and existence constraints

Predicate Encoding

1. Constraint encoding
   name$=$*value* (*equality* constraint)
   name$=$ *any* (*existence* constraint)
   e.g., idmef_version$>$2 $\rightarrow$ "$\exists$idmef_version", idmef_version$=$2 $\rightarrow$ "idmef_version$=$2"

2. A *filter* f is encoded with a set $S_f = S_f^{=} \cup S_f^{\exists}$, representing the union of equality and existence constraints

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

Message Encoding

1. Every attribute *name*=*value* is encoded with two strings $s^{=}$ =“*name*=*value*” and $s^{\exists}$ =“*name*”

$$\text{idmef\_version}=2 \rightarrow \begin{cases} s^{=} = \text{“idmef\_version=2”} \\ s^{\exists} = \text{“}\exists\text{idmef\_version”} \end{cases}$$

Message Encoding

1. Every attribute *name*$=$*value* is encoded with two strings $s^{=} =$"*name*$=$*value*" and $s^{\exists} =$"*name*"

   $$\text{idmef\_version}{=}2 \rightarrow \begin{cases} s^{=} = \text{"idmef\_version=2"} \\ s^{\exists} = \text{"}\exists\text{idmef\_version"} \end{cases}$$

2. A message $m = \{a_1, a_2, \ldots, a_n\}$ is therefore encoded with a set $S_m = S_m^{=} \cup S_m^{\exists}$

“Encoded” Matching

Given P’s encoding $B_P = \{B_{f_1}, B_{f_2}, \ldots\}$, and m’s encoding B_m, we define the *Bloom-encoded matching* relation $m \prec_B P$ as follows:

$$m \prec_B P \Leftrightarrow \exists f \in P : B_f \subseteq B_m$$

Observations

- Matching an encoded message m with an encoded filter f amounts to testing inclusion of two Bloom filters
  - in C, this may be done with `(Bm & Bf) == Bf`
- The covering relation $f \prec g$ works exactly the same way
  - `(Bf & Bg) == Bg`

Summary

- Authentication and Integrity
  - traditional methods

- Privacy in the presence of an untrusted network
  - approach: encoding messages and filters
  - method: Bloom filters

- Ongoing research
  - efficient representation and processing of large sets of Bloom filters
    - *ideas: BDDs*

Summary

- Authentication and Integrity
  - traditional methods

- Privacy in the presence of an untrusted network
  - approach: encoding messages and filters
  - method: Bloom filters

- Ongoing research
  - efficient representation and processing of large sets of Bloom filters
    - *ideas: BDDs*

- http://www.inf.unisi.ch/carzaniga/cbn/

Part V

Conclusions

Conclusions

- Content-based communication is an exciting research area

Conclusions

- Content-based communication is an exciting research area
- Several interesting open problems
  - routing
  - service interface
  - theoretical framework
  - middleware design
  - design and engineering of applications
  - security and privacy
  - sensor networks
  - ...

Conclusions

- Content-based communication is an exciting research area
- Several interesting open problems
  - routing
  - service interface
  - theoretical framework
  - middleware design
  - design and engineering of applications
  - security and privacy
  - sensor networks
  - ...
- Several disciplines
  - networking
  - algorithms
  - systems
  - software

Content-Based Communication: The *Network* Underneath Event Processing

Antonio Carzaniga

Faculty of Informatics
University of Lugano

April 2008

http://www.inf.unisi.ch/carzaniga/

References

- A. Carzaniga and A. L. Wolf. **"Content-Based Networking: A New Communication Infrastructure."** In *NSF Workshop on an Infrastructure for Mobile and Wireless Systems*, LNCS 2538, Scottsdale, Arizona, Oct. 2001. Springer-Verlag.
- A. Carzaniga and A. L. Wolf. **"Forwarding in a Content-Based Network."** In *Proceedings of ACM SIGCOMM 2003*, Karlsruhe, Germany, Aug. 2003.
- A. Carzaniga, M. J. Rutherford, and A. L. Wolf. **"A Routing Scheme for Content-Based Networking."** In *Proceedings of IEEE INFOCOM 2004*, Hong Kong, China, Mar. 2004.
- A. Carzaniga, A. J. Rembert, and A. L. Wolf. **"Understanding Content-Based Routing Schemes."** *Technical Report 2006-05*, Faculty of Informatics, University of Lugano, Sep., 2006.
- C. P. Hall, A. Carzaniga, and A. L. Wolf. **"DV/DRP: A Content-Based Networking Protocol For Sensor Networks."** *Technical Report 2006-04*, Faculty of Informatics, University of Lugano, Sep., 2006.