Understanding Content-Based Routing Schemes

Antonio Carzaniga[†,‡] Aubrey J. Rembert[‡] Alexander L. Wolf[†,‡]

[†]Faculty of Informatics
University of Lugano
6900 Lugano, Switzerland

[‡]Department of Computer Science
University of Colorado
Boulder, Colorado 80309-0430

University of Lugano
Faculty of Informatics
Technical Report 2006/05
September 2006

Abstract

Content-based networking is a message-oriented communication service in which a message is delivered to all destinations that have declared a selection predicate matching the content of that message. Analogous to that of a traditional address-based network, a routing scheme in a content-based network defines the router-local matching, forwarding, and header functions that collectively realize the delivery function. Several such routing schemes have been proposed in the literature, but they have been evaluated only qualitatively or in simulation. In this paper we abstract from those previous results in an effort to place them in a general theoretical framework. This framework allows us to rigorously define notions of correctness, minimality, and complexity. In particular, we prove the correctness and characterize the complexity of two existing content-based routing schemes, propose a new latency-minimal scheme, and provide the results of a Monte Carlo simulation that serves as the basis for estimating the space requirements of any given scheme.

1 Introduction

Consider a network service in which receivers advertise predicates instead of traditional numeric addresses, and in which a message is delivered to all interested receivers whose predicates match the content of the message. This is the service provided by the so-called *content-based network* [6]. It can be seen as a generalized multicast communication service that does not rely on the existence or maintenance of group address spaces, but instead on the advertisement and propagation of message selection predicates. The term *content-based routing* refers to the distributed algorithm that is responsible for delivering messages from senders to interested receivers. This paper focuses on the structure of the routing state and the corresponding forwarding functions used to realize the algorithm within a particular *routing scheme*.

Several routing schemes have been proposed in the study of distributed publish/subscribe systems and on the specific subject of content-based networking [2, 3, 4, 5, 12, 18]. These schemes have been evaluated through simulations and through qualitative analyses. This paper abstracts from those previous results to place them in a general theoretical framework. In addition to gaining a better understanding of specific schemes, our goal is to characterize the general problem of content-based routing.

For traditional address-based networks, a series of ever-improving theoretical results have been obtained in the tradeoff between space and time of routing (so-called "compact" routing schemes [1, 9, 16, 19]). Space is generally measured in terms of packet headers in messages and routing state at processors, while time is given in terms of the computation required to select next-hop processors and the length (or cost) of network paths between senders and receivers. Because content-based networking involves concepts such as predicates and message content, rather than simple addresses, these results do not apply. Nevertheless, they give a basic structure in which to reason about space and time tradeoffs in routing, and we make use of this structure in our own analysis.

We first cast the problem of content-based routing in terms of a standard model of a network and its routing functions [16]. We then use this model to formulate: (1) a specialized correctness condition for a routing scheme that reflects the semantics of content-based networking; (2) a notion of minimality that allows us to define an ideal delivery and, therefore, an ideal content-based routing scheme, which delivers every message using a latency-minimal and cost-minimal delivery tree; and (3) a characterization of routing-state space complexity that can be tuned to the specifics of a given selection-predicate language and expected application workload.

2 Model of a Content-Based Network

We model a content-based network as a finite connected undirected graph $G = (V, E)$ where a vertex $v \in V$ represents a processor and an edge $e \in E$ represents a reliable bidirectional communication link between two processors. Without loss of generality, we assume that a processor acts both as a *host*, producing and consuming messages, and as a *router*, forwarding messages on behalf of other processors. We assume that processors are given numeric identifiers, so hereafter we let $n = |V|$ and $V = \{1, \ldots, n\}$.

A content-based network is also defined by the set $\mathcal{M}$ of allowable *messages*, and the set $\mathcal{P}$ of allowable selection *predicates* over $\mathcal{M}$. A predicate $p \in \mathcal{P}$ is a total Boolean function $p : \mathcal{M} \to \{0, 1\}$ that implicitly defines a set of messages $\mathcal{M}(p) = \{m | m \in \mathcal{M} \wedge p(m) = 1\}$. When there is no risk of ambiguity, we simplify our notation by using predicates to denote the sets of messages they define. So, we simply write p to denote $\mathcal{M}(p)$. We also write $m \in p$ and say that predicate p matches message m when $p(m) = 1$.

A message in a concrete implementation of a content-based network would typically consist of a tuple of attributes. For instance, the message [*alert* = congestion, *severity* = 3, *location* = highway1] may describe an event related to a highway traffic-control application. A selection predicate is typically modeled as a disjunction of conjunctions of constraints on attributes. For example, the predicate (*alert* = congestion $\wedge$ *severity* > 2) $\vee$ (*alert* = accident) would *match* the message above. In defining a general content-based networking model, we ignore the concrete structure and representation of messages and predicates.

We define the following common functions: A *neighbor function*, $N : V \to \mathbb{P}(V)$, such that $N(v)$ represents the neighbors of processor v. ($\mathbb{P}(X)$ denotes the power set of X.) A *distance function*, *dist* : $V \times V \to \mathbb{R}$, such that *dist*$(u, v)$ represents the length of the shortest path between processors u and v. The degree of a processor v is denoted by *deg*(v), and the maximum degree of any processor in the network is denoted by Δ. Each processor is responsible for collecting the interests of its hosted applications and delivering each matching message to those applications accordingly. We define a *predicate function*, *pred* : $V \to \mathcal{P}$ that associates each processor v with a predicate *pred*(v). The predicate *pred*(v) represents the interests of the hosted applications of v. Consistently, we say that a processor v is *interested* in a message m when $m \in$ *pred*(v).

2.1 Routing Scheme

We define a content-based routing scheme by adapting the standard model of Peleg and Upfal [16]. We assume that a processor v can send a *packet* to one of its neighbors u by invoking a "link-level" communication primitive. We assume that this primitive requires v to specify only the intended destination u. We also ignore the complexity of link-level communication primitives since its impact on routing is minimal and well understood. For the purpose of this paper, a packet c consists of a message $msg(c) \in \mathcal{M}$ and a header $hdr(c) \in \mathcal{H}$, where $\mathcal{H}$ is the set of allowable message headers defined by the routing scheme. A packet header contains information that may be helpful in delivering the message. (Obviously, other packet types might be used for the purpose of routing. In this paper we focus exclusively on message packets.)

In addition to $\mathcal{H}$, a content-based routing scheme defines a distributed algorithm in which each processor v implements the following functions: An *initial header function*, $\mathsf{Init}_v : \mathcal{M} \to \mathcal{H}$ that, given a new message m originating at v, produces an initial header $\mathsf{Init}_v(m)$. A *header function*, $\mathsf{Hdr}_v : \mathcal{H} \to \mathcal{H}$ that, given a header h produces a new header $\mathsf{Hdr}_v(h)$. A *forwarding function*, $\mathsf{Fwd}_v : \mathcal{H} \times \mathcal{M} \to \mathbb{P}(N(v))$ that, given a header h and a message m produces a subset $\mathsf{Fwd}_v(h, m)$ of the set of neighbors $N(v)$ of v.

In the rest of the paper we use the letter v to refer to the source of a message, and the letter u to refer to any other processor (i.e., router) that forwards m. For a message m originating at v, processor v creates a packet header $h = \mathsf{Init}_v(m)$, and sends a packet $c = \langle h, m\rangle$ to all the processors in $\mathsf{Fwd}_v(h, m)$. When forwarding an incoming packet c, processor u extracts the header $h = hdr(c)$ and the message $m = msg(c)$ from c, computes the set of next-hop processors $\mathsf{Fwd}_u(h, m)$, and forwards to all those processors a new packet $c' = \langle \mathsf{Hdr}_u(h), m\rangle$. This generic process applies to all routing schemes. Each particular routing scheme is defined completely by its processor-specific header and forwarding functions Init, Hdr, and Fwd.

2.2 Delivery Trees

Given a message m originating at processor v, a routing scheme S induces a directed *delivery tree* $D^S_{m,v}$ defined by all the packets transmitted and processed according to S in response to the injection of message m at processor v. A vertex d in a delivery tree $D^S_{m,v}$ is associated with a packet $pkt(d)$ and a processor $proc(d)$. If S causes two identical packets to be sent to the same processor, then $D^S_{m,v}$ will contain two distinct vertexes to represent the two events. Intuitively, $D^S_{m,v}$ is defined by the recursive application of the header and forwarding functions defined by S. A bit more formally, the root d_0 of $D^S_{m,v}$ is associated with packet $pkt(d_0) = \langle \mathsf{Init}_v(m), m\rangle$ and processor $proc(d_0) = v$. Then, each vertex d_x in $D^S_{m,v}$ associated with packet $pkt(d_x) = \langle h, m\rangle$ and processor $proc(d_x) = u$ such that $\mathsf{Fwd}_u(h, m) = \{w_1, w_2, \ldots, w_k\}$ has k children, $d_{x\cdot 1}, d_{x\cdot 2}, \ldots, d_{x\cdot k}$, associated with processors $proc(d_{x\cdot 1}) = w_1, proc(d_{x\cdot 2}) = w_2, \ldots, proc(d_{x\cdot k}) = w_k$ and with packets $pkt(d_{x\cdot 1}) = \langle \mathsf{Hdr}_{w_1}(h), m\rangle, pkt(d_{x\cdot 2}) = \langle \mathsf{Hdr}_{w_2}(h), m\rangle, \ldots, pkt(d_{x\cdot k}) = \langle \mathsf{Hdr}_{w_k}(h), m\rangle$, respectively.

Delivery trees serve as a basis to define the correctness and efficiency of a routing scheme S.

2.3 Correctness and Efficiency

Correctness is quite straightforward. S must be such that, for each message m and for each processor v, the delivery tree $D^S_{m,v}$ contains vertexes representing any and all processors interested in m. Formally, for every source processor $v \in V$, for every other processor $u \in V$ and every message m such that $m \in pred(u)$, there exists a vertex $d \in D^S_{m,v}$ such that $proc(d) = u$.

A good routing scheme should also be efficient. However, given the multicast nature of content-based routing, efficiency may be interpreted in a number of ways. A plausible goal is to minimize the total communication cost of each message. Assuming that the number of transmitted packets is a reasonable measure of communication cost, then a good scheme would minimize $|D^S_{m,v}|$ for each m and v. Another plausible definition of efficiency is one that interprets edge weights as link latencies, and that focuses on minimizing the delivery latency for each interested destination processor. Given a message m originating at processor v, a *latency-minimal* delivery tree $D^S_{m,v}$ is one that minimizes the length of the delivery path for each processor x interested in m.

We say that a routing scheme is *ideal* when it minimizes latency and total cost, in this order of priority. Specifically, for each message m and for each origin v, the delivery tree $D^S_{m,v}$ induced by the scheme is the smallest among all latency-minimal delivery trees. (Notice that the smallest tree might not be unique.)

2.4 Memory Requirements

Besides communication costs and latency, we are interested in characterizing the space complexity of content-based routing. We refer to the memory requirement of every element of a routing scheme with a generic function $M(\cdot)$ that denotes the number of bits necessary to represent a given data structure. Given a routing scheme S, we are interested in characterizing the total memory requirement

$$M(S) = \sum_{v \in V} (M(\mathsf{Init}_v) + M(\mathsf{Hdr}_v) + M(\mathsf{Fwd}_v))$$

A routing scheme would generally require processors to store processor identifiers as well as predicates. A processor identifier requires $\Theta(\log n)$ bits of storage. The requirements for predicates are less obvious. In the most general case, without any specific constraining model for messages and predicates, we should assume that the representation of a message $m \in \mathcal{M}$ would require $M(m) = \Omega(\log |\mathcal{M}|)$ bits, and that the representation of a predicate p, which itself represents a set of messages, would require $M(p) = \Omega(\min\{|\mathcal{M}|, |p| \log |\mathcal{M}|\})$ bits. However, in practice we should consider the information-theoretic requirements for predicates in the context of the given structures and distributions of messages and, more importantly, of predicates.

Moreover, predicates are often combined by routing schemes [4, 5], so it is useful to characterize the memory requirements of disjunctions (i.e., unions) of predicates. To do this, we define the *disjunction advantage*, under a given representation model, for two predicates p_1 and p_2 as $\alpha(p_1, p_2) = \frac{M(p_1 \vee p_2)}{M(p_1)+M(p_2)}$. Intuitively, the disjunction advantage indicates by how much the memory requirements are reduced when storing the disjunction of two predicates instead of the two individual predicates. Notice that in practice we can assume that $\alpha(p_1, p_2) \leq 1$. That is, there is never a penalty for combining predicates. This is because it is always possible to represent a disjunction $p_1 \vee p_2$ as a sequence $\{p_1, p_2\}$, with $M(p_1 \vee p_2) = (M(p_1) + M(p_2))$. Then, depending on the structure and representation of $\mathcal{P}$, it might be possible to achieve effective reductions (i.e., $\alpha(p_1, p_2) < 1$). For example, using a language based on expressions of constraints over message attributes that is common in publish/subscribe systems, the disjunction of $p_1 = (\textit{severity} > 2)$ and $p_2 = (\textit{severity} > 3)$ can be represented as $p_1 \vee p_2 = (\textit{severity} > 2)$.

We generalize the definition of disjunction advantage to an arbitrary number of predicates. Given a set of predicates $P = \{p_1, p_2, \ldots, p_n\}$, we define the advantage

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

3 Routing Schemes

We start by presenting a novel routing scheme, first in a simple formulation and then in an improved variant. We then model and analyze two other, existing schemes.

3.1 Per-Source Forwarding

The idea of this scheme, which we call *PSF*, is to represent a directed shortest-paths spanning tree T_v for each source processor v (i.e., T_v is the tree computed by Dijkstra's algorithm). We let $child_u(v)$ denote the set of children of processor u in the tree T_v. We also use the notation T_v/w to refer to the set of processors in the subtree of T_v rooted at w. That is, T_v/w contains w and all its direct and indirect descendants in T_v. Given a processor v and an edge (u, w), we denote by $T_v/(u, w)$ the set of *descendant processors via edge* (u, w) *in* T_v, which we define as

$$T_v/(u,w) = \begin{cases} T_v/w, & \text{if } w \in child_u(v) \\ \emptyset, & \text{otherwise.} \end{cases}$$

For example, in the network of Figure 1, $T_1/(6, 7)$ is the set $\{4, 7, 8, 11, 12\}$ while $T_1/(6, 9)$ is the empty set.

For each source processor v, *PSF* annotates each edge (u, w) with the disjunction of the predicates of the processors in $T_v/(u, w)$. The representations of these annotated trees are distributed throughout the network, such that each processor u stores its local view of T_v for every source processor v. In particular, for each processor v (including $v = u$) and for each neighbor processor $w \in N(u)$, u associates a predicate with each edge (u, w) in T_v. The scheme also annotates the packet header with the source v of a message m so that each processor u receiving a packet containing m can forward it along the branches of T_v.

Formally, *PSF* consists of a function $F_u : V \times N(u) \rightarrow \mathcal{P}$ that maps a source processor v and a neighbor processor w to a predicate $F_u(v, w) = \bigvee_{x \in T_v/(u,w)} \mathit{pred}(x)$.

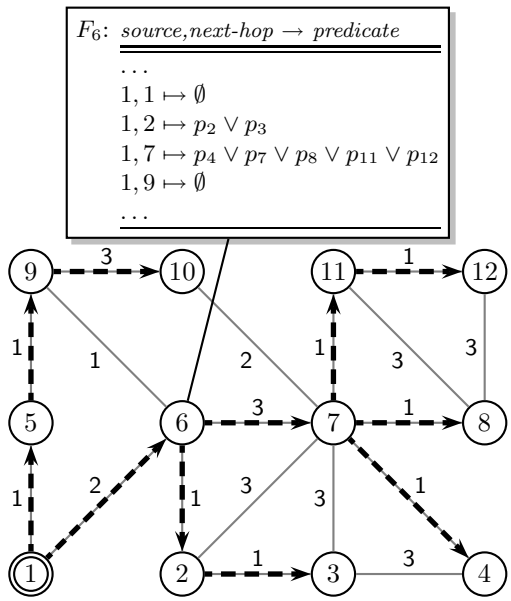

Figure 1: Forwarding in *PSF*

Figure 1 shows an example network. Thin lines represent network links, each of which has an associated distance. Thick dashed arrows represent the edges of the shortest-paths tree T_1 rooted at processor 1. The figure also shows the relevant fragment of a table representing the function F_6 of processor 6. (We use an abbreviated notation for the predicates associated with processors, where p_2 means $\mathit{pred}(2)$, p_3 means $\mathit{pred}(3)$, etc.)

Header and forwarding functions in *PSF* are defined as follows. Each processor creates initial headers with its own identifier and copies headers from input to output packets. So $\mathsf{Init}_x(m) = x$ and $\mathsf{Hdr}_x(h) = h$ for every processor x and for every message m. The forwarding function Fwd_u called on header h and message m reads the source processor v from the header h and outputs the set of next-hop processors $\mathsf{Fwd}_u(v, m) = \{w | m \in F_u(v, w)\}$. That is, u forwards a single message m originating at v to all next-hop processors w downstream from u on T_v such that the (u, w) edge in T_v is associated with a predicate p matching m. Processor u makes this forwarding decision using its local view of T_v.

PSF is a correct routing scheme. A full proof of correctness is beyond the scope of this paper. However, the intuition is that *PSF* forwards a message originating at v within the shortest-paths tree T_v rooted at v, based on the source identifier carried in the packet header. Within T_v, the message is forwarded only through edges that reach descendant processors interested in m. We can also show that *PSF* is an ideal scheme. This is because every source tree T_v contains only minimal paths by construction.

To analyze the complexity of *PSF*, we distinguish a set of active receivers $R \subseteq V$ and a set of active senders $S \subseteq V$, and we let $s = |S|$ and $r = |R|$. For simplicity, we assume that all receiver predicates have the same memory requirement M_p, and that all edges have constant weight.

In *PSF*, each sender processor $v \in S$ must store its identifier, with a total memory requirement of $s \log n$, and every processor $u \in V$ must store its function F_u. Collectively, the F_u functions represent s trees, one for each sender, where each edge is associated with a disjunction of the predicates of the receivers that are descendant processors via that edge in that tree. The memory requirement for the tree for source $v \in S$ is $\sum_{u \in V} M(F_u(v, \cdot))$, and therefore the total memory requirement for *PSF* is $M(\mathit{PSF}) = s \log n + \sum_{v \in S} \sum_{u \in V} M(F_u(v, \cdot))$.

We first analyze the memory requirements for the $F_u(v, \cdot)$ functions for a source v in the absence of any disjunction advantage (i.e., with $\alpha(\{\mathit{pred}(x) | x \in R\}) = 1$). The analysis is based on the observation that the predicate $\mathit{pred}(x)$ of a receiver processor x is associated with all the edges on the path from v to x. Therefore $\sum_{u \in V} M(F_u(v, \cdot)) = M_p \sum_{x \in R} \mathit{dist}(v, x)$. The total requirement for *PSF* is therefore

$$M(\mathit{PSF}) = s \log n + M_p \sum_{v \in S} \sum_{x \in R} \mathit{dist}(v, x)$$

Assuming a uniform distribution of senders and receivers, and denoting by d the average distance between any two processors in the network, we can write the memory requirement in the simpler form

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

The average distance d obviously depends on the network topology, and can be as high as $O(n)$. However, it is very small in many common cases. For example, Chung and Lu showed that in most cases the average distance d is $O(\log \log n)$ in power-law random graphs [8] such as the Internet [10] and other networks based on social connections.

The product of the number of senders and receivers sr, which is $O(n^2)$, is a crucial factor in the memory requirement of *PSF*. Intuitively, the number of senders s is a factor because the *PSF* scheme requires that each processor be able to distinguish every sender. This, in turn, is because each sender may induce a different shortest-paths tree. The number of receivers r is a factor because every receiver predicate is included in a disjunction of predicates with one or more edges in a shortest-paths tree, and therefore each receiver predicate contributes linearly to the overall memory requirement.

In practice, both these assumptions can be relaxed, effectively tightening the memory requirement of *PSF*. In the next section we present a variant of *PSF* based on the idea that processors do not need to distinguish all sources, thanks to the fact that source-rooted trees might be *indistinguishable*. Section 4 discusses the disjunction advantage α, and shows that adding predicates to a disjunction can incur a sublinear increase in memory requirements.

3.2 Improved Per-Source Forwarding

Depending on the network topology, source-rooted trees might partially or completely overlap. For example, for every edge $e \in E$ that is also a "bridge" (i.e., when removing e would partition the network), all the sources on one side of e should be indistinguishable from the viewpoint of every processor on the other side of e, and vice versa. This is because all paths from all sources on one side to all receivers on the other side would have to go through the bridge, and could use exactly the same subtree past the bridge.

This observation suggests an improvement on the basic *PSF* scheme. The idea is that if two sources v_1 and v_2 are *indistinguishable* from the viewpoint of processor u for the purpose of computing their entries in F_u, then u can collapse their two entries into a single one. As an extreme example, consider a network G consisting of a tree. In this case, every edge is a bridge and, therefore, from the viewpoint of each processor u all sources reachable through the same neighbor $w \in N(u)$ are indistinguishable. As a result, the domain of F_u for all processors u can be reduced from V to $N(u)$.

As it turns out, the existence of a bridge is only a sufficient condition, and there is a weaker condition that makes two sources indistinguishable. As an example, consider the network depicted in Figure 1. Processors 1, 5, and 9 are indistinguishable from the perspective of processor 6. We now detail an improved scheme that exploits indistinguishability of sources, and later present a tighter definition of indistinguishability.

The indistinguishability relation is an equivalence relation defined for each processor u. We represent each equivalence class of indistinguishable sources through one of its elements, and we denote by $\overline{V}_u \subseteq V$ the set of representative sources. The improved *PSF* scheme, which we call *iPS*, uses a function $I_u : V \to \overline{V}_u$ that maps a source processor v to the representative $I_u(v)$ of its equivalence class. *iPS* also uses a function $F_u : V_u \times N(u) \to \mathcal{P}$ that is conceptually identical to F_u in *PSF*, although with a smaller domain. The Fwd_u function is similar to that of the *PSF* scheme, except that it uses the I_v function to compress the space of sources. Formally, $\mathsf{Fwd}_u = \{w | m \in F_u(I_u(v), w)\}$.

In some cases, an equivalence class of sources defined for processor u and containing processor v is the same for all processors $w \in T_v/u$. This happens whenever there is a bridge in the network, but it might also happen in other cases. For example, in the network of Figure 1, processors 1, 5, and 9 remain indistinguishable from the perspective of all the processors that are descendants of 6 in the source tree T_1 (i.e., processors 2, 3, 4, 7, 8, 11 and 12). In fact, processor 6 is also indistinguishable from the perspective of all its descendants. In such cases, it makes sense to rewrite the source header in the packet, replacing the original source v with the most-recently visited processor beyond which sources remain indistinguishable. This is to compress the domain of the I_u functions in all the descendants. It is for this purpose that the *iPS* scheme defines a header rewriting function $R_u : V_u \to \{0, 1\}$ for each processor u. R_u is used in Hdr_u to decide whether to rewrite the header. Specifically, $\mathsf{Hdr}_u(v) = v$ if $R_u(I_u(v)) = 0$, or $\mathsf{Hdr}_u(v) = u$ if $R_u(I_u(v)) = 1$.

We now define the condition that allows a processor u to combine the two or more sources. Recall that for each source v, the forwarding function F_u maps a neighbor processor w to a predicate $F_u(v, w)$ that is the disjunction of the predicates *pred*(x) of all processors x that are descendants of u in T_v via the (u, w) edge.

We say that two sources v_1 and v_2 are *indistinguishable* from the viewpoint of processor u if the following two conditions hold. First, $child_u(v_1) = child_u(v_2)$, meaning that every child of processor u on T_{v_1} is also u's child on T_{v_2}, and vice versa. Second, each child processor w has the same set of descendants on T_{v_1} and T_{v_2}. Formally, $\forall w \in child_u(v_1) : T_{v_1}/w = T_{v_2}/w$.

A precise characterization of the memory requirements of *iPS* would require a complex probabilistic analysis of the indistinguishability relation under various network topologies, and is beyond the scope of this paper. Regardless of its prevalence and value in the general case, our intuition is that indistinguishability would play an important role in practice because of the inherently hierarchical structure of communication networks.

Also, we note that the extreme example of a tree network mentioned above admits a fairly simple analysis. This case is particularly important as many routing schemes are based on tree covers. In this case, for an incoming packet c, the forwarding function decides exclusively on the basis of which neighbor processor forwarded the packet. Thus, $V_u = N(u)$, I_u is the identity function, and $R_u = 1$, so that the source address is always updated with the current processor u. The memory requirement for both I_u and R_u is zero. The requirement for F_u is $M(F_u) = O(rM_p\alpha(r/deg(u)))$. The total memory requirement for *iPS* in the case of an tree topology is

$$M(iPS) = O(nrM_p\alpha(\frac{r}{\Delta}))$$

3.3 Per-Interface Forwarding

We now describe a routing scheme based on per-processor forwarding functions whose domain is limited to node neighbors. We call this a *per-interface forwarding* function, and thus we call this scheme *PIF*. *PIF* is a variant of the *Combined Broadcast and Content Based (CBCB)* scheme [5]. Similar to the *PSF* routing scheme, *PIF* is based on directed shortest-paths spanning trees. Recall that the *PSF* scheme associates a predicate with each edge in a spanning tree T_v, where v is the source of a message. Thus, for each source processor v, *PSF*'s forwarding function F_u at processor u maps edge (u, w) to the predicate $F_u(v, w)$ representing the disjunction of all the predicates of the descendant processors in T_v via the (u, w) edge.

The *PIF* scheme is based on two ideas. The first is to assign to each edge (u, w) a single predicate that combines the predicates of the descendants via (u, w) in the spanning trees for all sources. Thus, *PIF* maintains a function $F^*_u : N(u) \rightarrow \mathcal{P}$ at each processor u that maps a neighbor $w \in N(u)$ to the disjunction over all sources v of the predicates of the descendants via (u, w) in T_v. Formally, reusing the definition of $F_u(v, w) = \bigvee_{x \in T_v/(u,w)} pred(x)$ given for *PSF*, we have

$$F^*_u(w) = \bigvee_{v \in V} F_u(v, w)$$

Figure 2 shows an example network where the *PIF* scheme is used. The figure shows the forwarding function F^*_6 of processor 6 and focuses on $F^*_6(9)$. The figure lists the descendants via edge $(6, 9)$ in a number of shortest-paths trees. For clarity, the figure does not show all shortest-paths trees, although these can be intuitively figured out with the given edge weights. Notice how the value of $F^*_6(9) = p_5 \vee p_9 \vee p_{10}$ is determined by the union of descendants $T_1/(6, 9) \cup T_2/(6, 9) \cup \ldots \cup T_{12}/(6, 9) = \{5, 9, 10\}$.

The intent of F^* is of course to forward messages along links associated with matching predicates. However, if one were to follow only the content-based forwarding defined by F^*, messages would most likely end up in routing loops and they would be duplicated indefinitely. For example, referring again to Figure 2, a message originating at processor 6 and matching p_7 and p_{10} would match both $F^*_6(9)$ and $F^*_6(7)$, and therefore it would be forwarded to 9 and to 7. The copy that reaches 7 would then be forwarded to 10, because it would certainly match $F^*_7(10)$. Similarly, the copy that reaches 10 by way of 9 would be forwarded to 7, etc. The second idea of *PIF* is to avoid loops for a message m by forwarding m only within the tree T_v of its originating process v. In order to do that, *PIF* must also store all per-source trees. This can be done by explicitly representing the $child_u$ function at each processor u.

With F^*_u and $child_u$, the header and forwarding functions in *PIF* are defined as follows. Init_x and Hdr_x are identical to those of *PSF*, with each processor creating initial headers with its own identifier and copying headers from input to output packets. So $\mathsf{Init}_v(m) = v$ and $\mathsf{Hdr}_u(h) = h$ for all processors v and u, and for every message m. The forwarding function Fwd_u, which is also similar to the corresponding function in *PSF*, reads the source processor $v = hdr(c)$ from the header of the incoming packet, as well as the message $m = msg(c)$, and outputs the set of next-hop processors $\mathsf{Fwd}_u(v, m) = \{w | w \in child_u(v) \wedge m \in F^*_u(w)\}$. That is, u forwards a message

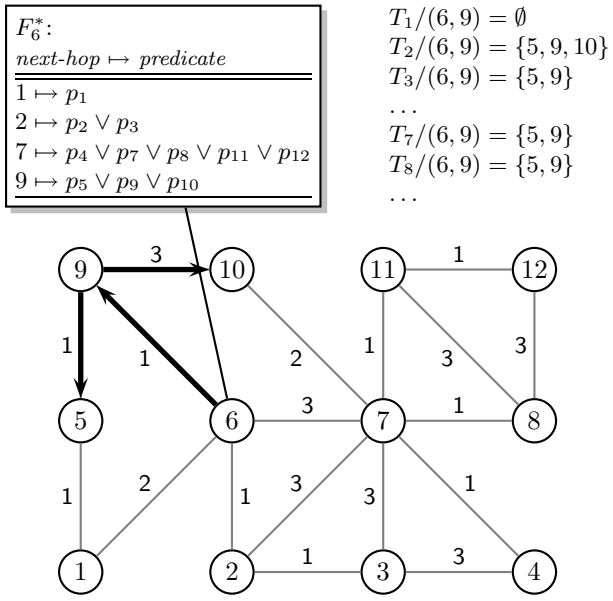

Figure 2: Forwarding Function in *PIF*

m originating at v through all edges (u, w) that are both (1) descendants of u in T_v and (2) associated with a predicate $F_u^*(w)$ matching m.

It is easy to see that *PIF* is correct. In fact, we can prove the correctness of *PIF* by showing that the delivery tree $D_{m,v}^{PSF}$ of *PSF*, which we proved correct, is always a subtree of the delivery tree $D_{m,v}^{PIF}$ of *PIF*. Intuitively, the reason is that the predicate $F_u^*(w)$ that *PIF* associates with edge (u, w) is the combination of all the source-specific predicates $F_u(\cdot, w)$ that *PSF* associates with (u, w). Therefore, for each source v and message m, if *PSF* forwards m along $(u, w) \in T_v$, then *PIF* will also do that.

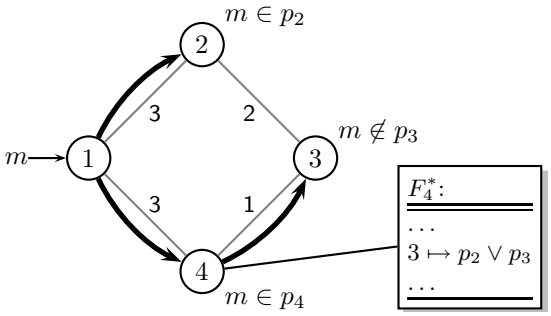

Figure 3: Non-Ideal Delivery Tree in *PIF*

It is also easy to see that *PIF* is not an ideal scheme using a counter-example. Figure 3 shows one such counter-example. Message m originating at processor 1 is correctly delivered to processors 2 and 4, which are interested in it, but it is also sent to processor 3, which is not interested in it. *PIF* sends m from 4 to 3 because edge $(4, 3)$ is on the shortest-paths tree T_1, and is associated with predicate $F_4^*(3) = p_2 \vee p_3$ that matches m. $F_4^*(3)$ happens to include p_2 because processor 2 is a descendant processor via edge $(4, 3)$ on the shortest-paths of T_4 rooted at 4.

The *PIF* scheme requires that each processor u store the F_u^* function and the $child_u$ function. The requirement for F_u^* is $M(F_u^*) = O(rM_p\alpha(r/deg(u)))$, where r is the number of receivers, M_p is the size of a receiver predicate (assumed constant for simplicity), and $\alpha(r/deg(u))$ is the disjunction advantage of r predicates combined into $deg(u)$ disjunctions. The requirement for the $child_u$ function is $M(child_u) = O(sdeg(u) \log n)$ where s is the number of senders. The total requirement for *PIF* is

$$M(PIF) = O(n(s\Delta \log n + nrM_p\alpha(\frac{r}{\Delta})))$$

3.4 Per-Receiver Forwarding

In this section, we describe a routing scheme that uses only per-receiver forwarding information. This scheme is a simplification of the *Distance-Vector / Dynamic Receiver Partitioning (DV/DRP)* protocol [11]. The main idea of DV/DRP is to compute the set of interested processors for a message m at the originator processor. The set of destinations is attached as a header to the message, and is used at every hop to forward the message to all the destinations. Whenever a message needs to be forwarded to two or more next-hop processors, the set of destination is partitioned according to their respective next-hop. This forwarding process is called *dynamic receiver partitioning*, therefore we call this scheme *DRP*.

The two main ingredients of the *DRP* scheme, stored at every processor v, are the *pred*$(\cdot)$ function and the unicast routing information necessary to reach every other processor in the network. Specifically, every processor v stores a table representing the *pred* function plus a table representing a function *unicast*$_v : V \to V$ such that *unicast*$_v(w)$ is v's neighbor on the shortest path from v to w.

The Init_v function takes a message m, computes the set of interested processors $W = \{w | m \in \textit{pred}(w)\}$, and then, if $W \neq \emptyset$, computes a partition function $H_W : N(v) \to W$ that maps each neighbor of v to a subset of W. H_W defines a partition in the sense that $\bigcup_{u \in N(v)} H_W(u) = W$ and $\forall x, y : x \neq y \Rightarrow H_W(x) \cap H_W(y) = \emptyset$. The H_W function is such that $\forall w \in W :$ *unicast*$(w) \in H_W(u)$. Intuitively, H_W partitions the set of receivers W by grouping them by next hop. The output of the Init_v function is a packet $\langle H_W, m \rangle$ containing the partition H_W and the message m.

At an intermediate processor u, the Hdr_u function does almost exactly what Init_u would do. The difference being that Hdr_u does not compute the destination set using message m and the *pred*$(\cdot)$ function. Instead, Hdr_u extracts the partition function $H_{W'}$ from the packet header and uses it to compute the destination set $W = H_{W'}(u)$. If $W \neq \emptyset$ then Hdr_u proceeds to compute its partition function H_W and to assemble its output packet exactly as in Init_u.

The forwarding function Fwd_u computes the set of next-hop destinations the same way that Init_u and Hdr_u compute the partition H_W of the destination set W. In particular, Fwd_u extracts the partition function H_W from the header. If $H_W(u)$ is undefined, then this means that u is the origin processor, and therefore Fwd_u returns the domain of H_W. Otherwise, Fwd_u reads $W = H_W(u)$ and returns the set $\{x | \exists w \in W \wedge x = \textit{unicast}(w)\}$.

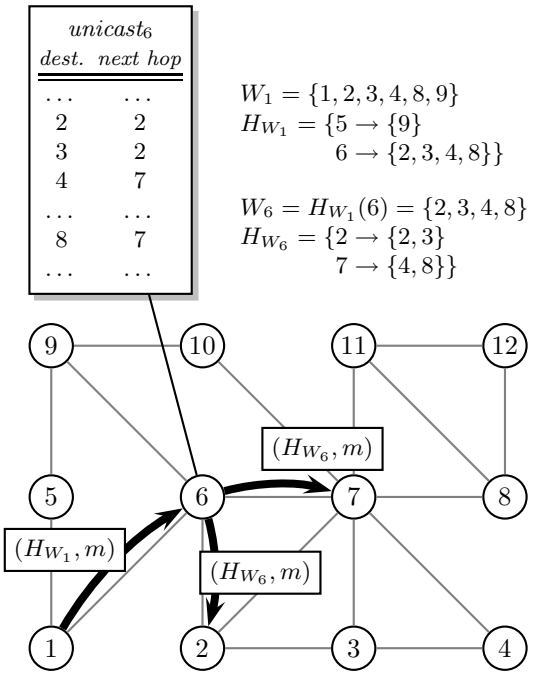

Figure 4: Dynamic Receiver Partitioning

The example of Figure 4 illustrates the *DRP* scheme. Processor 6 receives a packet $\langle H_{W_1}, m \rangle$. The Hdr_6 function extracts the destination set assigned to processor 6 by the H_{W_1} partition. The result is $W_6 = H_{W_1}(6) = \{2, 3, 4, 8\}$. The *unicast*$_6$ function, which is represented as a table associated with processor 6 in Figure 4 maps

the destinations $\{2, 3, 4, 8\}$ to next-hop processors 2 and 7. Specifically, destination sets $\{2, 3\}$ and $\{4, 8\}$ map to next-hop processors 2 and 7, respectively. This defines the partition H_{W_6}. Processor 6 assembles a packet $\langle H_{W_6}, m\rangle$ and proceeds to send it to processors 2 and 7. (Notice that an implementation of *DRP* could save header space and simplify the processing function by assembling a specific packet for each next-hop processor. In this example, processor 6 would send a packet $\langle\{2, 3\}, m\rangle$ to processor 2 and a packet $\langle\{4, 8\}, m\rangle$ to processor 7.)

It is easy to show that *DRP* is a correct routing scheme, assuming its unicast functions are correct. *DRP* is also ideal if and only if shortest paths between any pair of processors are unique. As a counter-example, consider the network topology of Figure 4. Suppose process 5 sends message m to $W = \{7, 8, 11\}$, and that the shortest path from 5 to 8 goes through 9 while the shortest path from 5 to 11 goes through 1, and they all go through 7. In this case, processor 7 will see at least two packets of the delivery tree $D^{DRP}_{m,5}$.

The analysis of the memory requirement of *DRP* is very simple. For each processor u, the unicast function requires $M(\textit{unicast}_u) = O(r \log n)$ bits, while the *pred* function requires $M(\textit{pred}) = rM_p$ bits at each sender. The total requirement is

$$M(\textit{DRP}) = O(nr \log n + srM_p)$$

Notice that the above analysis is based on a realization of the *unicast* function that guarantees minimal paths. However, this function is essentially a pluggable module of *DRP*. Therefore, relaxing the minimality requirement (i.e., by allowing the unicast function to "stretch" paths by a factor $k > 1$) one could use any one of the many proposed unicast schemes with much lower memory requirements.

4 Disjunction Advantage

In this section we provide a probabilistic analysis of the disjunction advantage α, defined in Section 2, as a way to characterize its expected value and, thereby, characterize its effect on the complexity of routing. Recall that α depends on the structure and distribution of messages and predicates.

4.1 α in a General Data Model

We begin by characterizing α in the very general case, where messages are simply elements of a universe of messages $\mathcal{M}$ and predicates are generic subsets of $\mathcal{M}$. This case is an abstraction of any content-based networking data model. In fact, it corresponds to the general model defined in Section 2.

For the purpose of this analysis, we assume that $\mathcal{M}$ is a finite set. In addition, we make two uniformity assumptions intended to simplify the characterization of α. First, we limit the analysis to predicates of uniform size, that is, we consider only predicates $p \subseteq \mathcal{M}$ of a fixed cardinality $|p| = k$. Second, we only compute the disjunction advantage for an entire set of predicates. In practice, given a set of predicates $p_1, p_2, \ldots p_k$, we are interested in the disjunction advantage

$$\alpha = \frac{M(p_1 \cup p_2 \cup \ldots \cup p_n)}{M(p_1) + M(p_2) + \ldots + M(p_n)}$$

where $p_i \subseteq \mathcal{M}$ and $|p_i| = k$.

Let $P = p_1 \cup p_2 \cup \ldots \cup p_n$. Given that a generic predicate p can be represented with $M(p) = p \log |\mathcal{M}|$ bits, we have $\alpha = \frac{|P|}{nk}$, and the expected value of α is

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

where $\mathrm{E}(|P|)$ is the expected size of the union of n random sets of size k. The probability that a uniform random message m is not in a predicate p of size $|p| = k$ is $\Pr[m \leftarrow \mathcal{M}; m \notin p] = 1 - \frac{k}{|\mathcal{M}|}$. Therefore, the probability that a message m is in the union of n such predicates is

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

which defines the expected size of P and then the expected disjunction advantage

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

This analysis is based on a uniform distribution of predicates, which leads to the highest expected values of α. Any other distribution would clearly favor one set of $\mathcal{M}$ over others, and therefore it would increase the probability of collisions between predicates, leading to lower (i.e., better) values of α.

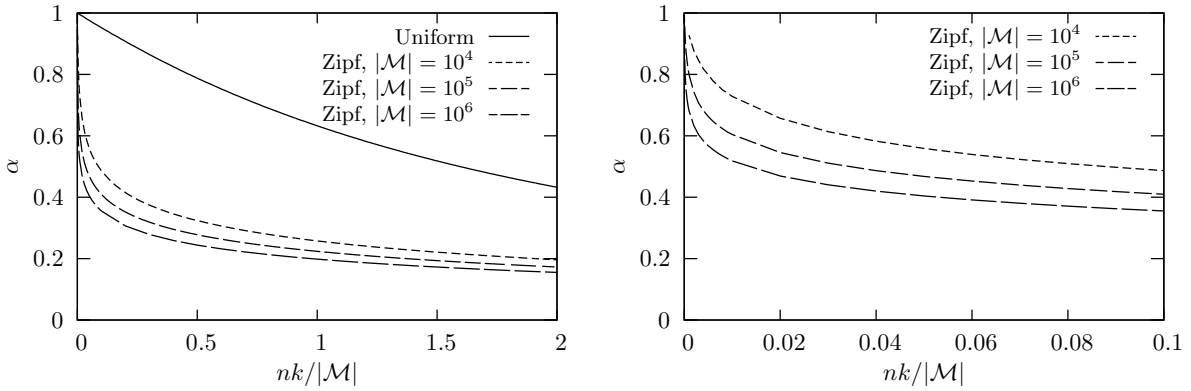

Figure 5: Disjunction Advantage in the General Data Model

Figure 5 shows a characterization of the disjunction advantage α under a uniform and Zipf probability distribution for message occurrences in predicates. This characterization was obtained through Monte Carlo simulations over universes of 10^4, 10^5, and 10^6 messages. Predicates were obtained by selecting messages from a uniform probability distribution, as well as a Zipf distribution with exponent 1. The first graph in the figure shows α for both uniform and Zipf distributions over a wide range of the ratio $nk/|\mathcal{M}|$, while the second graph focuses on the Zipf distribution, zooming in on the range $0 < \frac{nk}{|\mathcal{M}|} \leq 0.1$.

The behavior of the Uniform curve is exactly as predicted by the analytical expression of $\mathrm{E}(\alpha)$ obtained above. The most interesting result is that of the Zipf curves, which confirm the intuition that a realistic non-uniform distribution of predicates would yield a much better reduction even for lower values of the ratio $\frac{nk}{|\mathcal{M}|}$. For example, combining only five predicates each one selecting one out of one hundred messages would result in a space reduction of about 50%.

4.2 α in a Specific Data Model

We now analyze the disjunction advantage in a specific and realistic case, where messages and predicates conform to the structure and semantics defined by a common data description/query language. In this scenario, messages consist of a set of named attributes, and predicates are defined by disjunctive normal form (DNF) expressions of Boolean constraints on attribute values [6]. For purposes of the analysis, we use only integer and string attribute values, and constraints defined by a relational operator and a comparison value. An example of a message and predicate of this form appears at the beginning of Section 2.

We define the *size* of a DNF predicate to be the number of constraints it contains; in the example at the beginning of Section 2, the size of the predicate is 3. It is easy to see that the disjunction of two predicates may admit a representation smaller than the sum of the sizes of the representations of the individual predicates. For example, a simple reduction algorithm would remove redundant constraints within conjunctions and redundant conjunctions within a predicate. In particular, a conjunction of two constraints $p = (c_1 \wedge c_2)$ can be reduced to $p = (c_1)$ if $c_1 \Rightarrow c_2$. Similarly, a disjunction $p = c_1 \vee c_2$ can be reduced to $p = (c_1)$ if $c_2 \Rightarrow c_1$. For instance, the representation of $(x > 10 \wedge y = 100) \vee (y > 0)$ could be reduced to $(y > 0)$. Other, more sophisticated reduction algorithms are of course possible. However, our analysis of α is based on the algorithm outlined above, which is simple and efficient.[1]

We compute α through Monte Carlo simulation. In each iteration we generate a set of random predicates. We then compute the reduced disjunction of the predicates, and divide the number of constraints in the reduced disjunction by the total number of constraints in all the original predicates.

Generating random DNF expressions of Boolean constraints involves several parameters. To make this analysis tractable and meaningful in the context of this paper, we make some assumptions on several aspects of the distribution of predicates, and focus only on a few independent variables. In particular, we generate predicates

[1] A concrete implementation of this algorithm is available on-line at http://www.cs.colorado.edu/serl/cbn/forwarding/.

consisting of one conjunction of 1, 2, 3, 4, 5, or 6 constraints, with probability 10%, 40%, 30%, 10%, 5%, and 5%, respectively. The constraints are generated by choosing an attribute name from a set A of names, with probability distribution D. Each name in A is associated with either string or integer constraints in equal proportions. Constraints are chosen by selecting a relational operator and a comparison value. Relational operators for strings are chosen with the following distribution: *equality* with probability 40%, and *prefix*, *suffix*, *substring*, *not-equal* and lexicographical ordering relations *less-than* and *greater than*, all with probability 10%. Comparison string values are selected from a distribution of the 200 most-common meaningful terms extracted from a large corpus of news reports [13]. Integer constraints are formed with the relational operators *equals*, *less-than*, *greater-than*, and *not-equals*, with probabilities 40%, 25%, 25% and 10%, respectively. Integer comparison values are uniform random variables between -100 and 100.

The free variables we focus on in our analysis are the probability distribution of attribute names D, the total number of attribute names $|A|$, and the number of predicates combined in one disjunction. For each number of predicates, we compute α over several predicate sets. We examine the minimum, 5^{th} percentile, median, 95^{th} percentile, and maximum values of α as a function of the number of predicates.

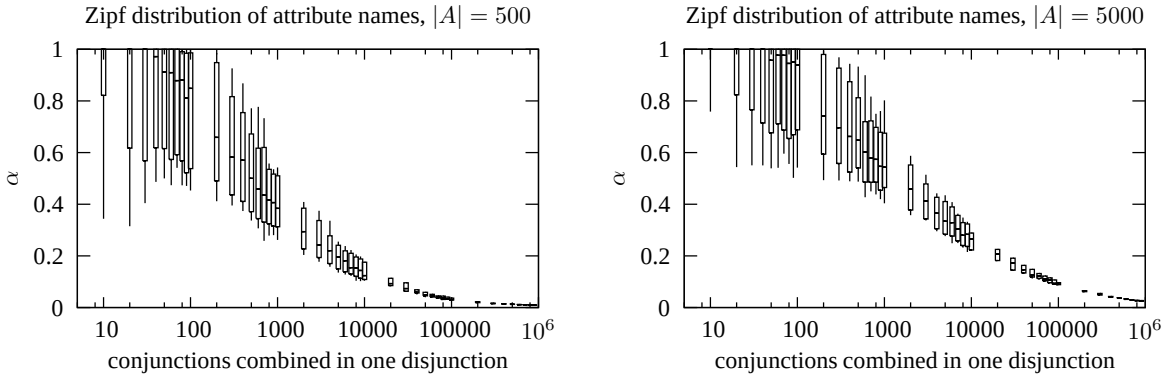

Figure 6: Disjunction Advantage Under a Zipf Distribution for Attribute Names

Figure 6 shows the results obtained using a Zipf distribution over a set of 500 and 5000 attributes, respectively. This scenario is intended to represent an open, wide-area network used by several diverse applications and users. In this scenario, the vocabulary of predicates, although vast, would conform to a "popularity" power-law distribution such as Zipf.

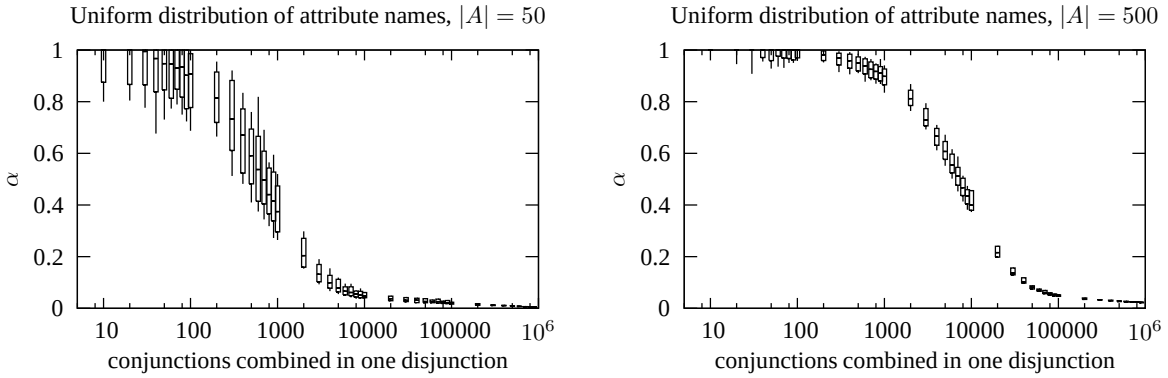

Figure 7: Disjunction Advantage Under a Uniform Distribution for Attribute Names

Figure 7 shows the results obtained using a uniform distribution over a set of 50 and 500 attributes, respectively. This scenario represent a single (or a few) large-scale applications (e.g., file sharing or network monitoring), where the vocabulary of constraints would be relatively small, but also more uniform in its distribution.

At a high level, our analysis confirms the intuition that the combination of more and more predicates yields better and better reductions, with α becoming progressively smaller. Also as expected, non-uniform distributions

of attribute names over the same total number of attribute names result in better reduction factors. The analysis additionally shows that the reduction factor is highly variable for small numbers of predicates (a few tens or hundreds of conjunctions), but then stabilizes for larger sets of predicates. A somewhat negative result is that the α reduction is not immediately effective, especially for large and/or uniform sets of attribute names. In fact, none of the given scenarios shows a consistent reduction of $\alpha = 0.5$ or better for less than 100 predicates, and in the worst case of 500 uniformly distributed attributes, α goes below 0.5 only when 7000 or more predicates are combined.

Notice, however, that the base predicates used in this analysis are intentionally small and simple, consisting of only one conjunction. In a realistic wide-area network scenario, where each processor represents a router for an entire subnet, the predicate associated with a processor is likely to consist of hundreds of the base predicates used here.

5 Related Work

Although there has been substantial work on developing concrete solutions to the content-based routing problem [7, 11, 5, 2, 18, 3], and even some attempt at quantitatively comparing early versions of such schemes [14], this paper is the first presentation of a comprehensive, analytical characterization of the correctness, minimality, and complexity of content-based routing schemes. Therefore, there is little work that can be meaningfully related to ours.

The network and routing models described in this paper are based on the general framework laid out by Peleg for traditional address-based routing schemes [15]. Although we use that framework, the semantic differences between the address-based and content-based services lead to notions of correctness, minimality, and complexity that differ substantially. For example, in traditional address-based routing, it is recognized that storing full forwarding tables at each processor is inefficient because they would grow linearly with the size of the network. Therefore, the goal in traditional schemes is to make the forwarding table as compact as possible, while keeping the message delivery paths bounded to some multiple of the shortest paths; in the literature, this is called the *stretch* factor. An optimal solution is one that properly optimizes the trade off between stretch and routing table size. This led to an investigation of so-called "compact" routing schemes and their associated theoretical properties [1, 9, 16, 19].

In contrast, content-based networking is a multicast service requiring potentially many delivery paths for a given message, one for each interested receiver. The nature of "stretch" is therefore inherently different and must account for this multiplicative effect. Moreover, the forwarding-table minimization problem is dominated by the nature of the predicates and messages, and only to a lesser extent affected by storage of path information. This led us to introduce the disjunction advantage for content-based routing schemes, a notion that has no direct analog in address-based schemes. (One could view the simplification of predicates meeting at a particular processor in the network as a sort of "content-based subnetting", where processors far from a message destination would likely apply general predicates and those closer would apply more specific ones. But notice that such subnets would result from the particular constellation of predicates advertised by receivers, which arises dynamically from the behavior of applications making use of the network and not through some sort of static configuration of that network.)

Rajvaidya and Almeroth [17] provide a practical evaluation of multicast routing schemes based on data collected from actual deployments. They characterize the size of the connected hosts in multicast groups and the stability of the multicast routes across the topology. Their evaluation identifies some problems with multicast routing and suggests how multicast routing should evolve. The results of the Rajvaidya and Almeroth study do not provide much insight into content-based routing. While the two services share the idea that a single sender might communicate with arbitrary numbers of (anonymous) receivers, the similarity ends there. Multicast routing is based on group address spaces and, thus, largely reduces to a special form of address-based routing. It is a service more appropriate for relatively stable definitions of groups and group interests (usually defined by the sender, rather than the receivers) and for receivers whose membership in groups is also relatively stable.

6 Conclusion

Although there has been a growing body of work in experimental treatments of content-based networking, to our knowledge this is the first attempt at an analytical characterization of broad classes of content-based routing schemes. We have provided a theoretical framework in which to model correctness, minimality, and complexity.

The utility of the framework was demonstrated by capturing the essence of several superficially similar, yet practically distinct schemes. Building on these first concrete steps, we intend to study additional routing schemes, as well as to obtain tighter bounds on complexity.

Acknowledgements

This work was supported in part by the US National Science Foundation and US Army Research Office under agreement numbers ANI-0240412 and DAAD19-01-1-0484.

References

[1] M. Arias, L. J. Cowen, K. A. Laing, R. Rajaraman, and O. Taka. Compact routing with name independence. In *Proceedings of the Fifteenth Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '03)*, pages 184–192. ACM Press, June 2003.

[2] G. Banavar, T. D. Chandra, B. Mukherjee, J. Nagarajarao, R. E. Strom, and D. C. Sturman. An efficient multicast protocol for content-based publish-subscribe systems. In *The 19th IEEE International Conference on Distributed Computing Systems (ICDCS '99)*, pages 262–272, Austin, Texas, May 1999.

[3] F. Cao and J. P. Singh. Efficient event routing in content-based publish-subscribe service networks. In *IEEE Conference on Computer Communications (INFOCOM '04)*, pages 929–940, Hong Kong, China, Mar. 2004.

[4] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, Aug. 2001.

[5] A. Carzaniga, M. J. Rutherford, and A. L. Wolf. A routing scheme for content-based networking. In *IEEE Conference on Computer Communications (INFOCOM '04)*, pages 918–928, Hong Kong, China, Mar. 2004.

[6] A. Carzaniga and A. L. Wolf. Forwarding in a content-based network. In *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '03)*, pages 163–174, Karlsruhe, Germany, Aug. 2003.

[7] R. Chand and P. A. Felber. A scalable protocol for content-based routing in overlay networks. In *Proceedings of the Second IEEE International Symposium on Network Computing and Applications (NCA '03)*, pages 123–130. IEEE Computer Society, 2003.

[8] F. Chung and L. Lu. The average distances in random graphs with given expected degrees. *Proceedings of the National Academy of Sciences*, 99(25), Dec. 2002.

[9] T. Eilam, C. Gavoille, and D. Peleg. Compact routing schemes with low stretch factor. In *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing (PODC '98)*, pages 11–20. ACM Press, June 1998.

[10] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the Internet topology. In *SIGCOMM '99: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, pages 251–262, Cambridge, Massachusetts, United States, Aug. 1999.

[11] C. P. Hall, A. Carzaniga, J. Rose, and A. L. Wolf. A content-based networking protocol for sensor networks. Technical Report CU-CS-979-04, Department of Computer Science, Univ. of Colorado, Aug. 2004.

[12] P. Ji, Z. Ge, J. Kurose, and D. Towsley. Matchmaker: Signaling for dynamic publish/subscribe applications. In *11th IEEE International Conference on Network Protocols (ICNP '03)*, pages 222–233, Nov. 2003.

[13] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. Rcv1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397, Apr. 2004.

[14] G. Mühl, L. Fiege, F. C. Gärtner, and A. P. Buchmann. Evaluating advanced routing algorithms for content-based publish/subscribe systems. In A. Boukerche, S. K. Das, and S. Majumdar, editors, *The Tenth IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2002)*, pages 167–176, Fort Worth, TX, USA, October 2002. IEEE Press.

[15] D. Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM, 2000.

[16] D. Peleg and E. Upfal. A trade-off between space and efficiency for routing tables. *Journal of the ACM*, 36(3):510–530, July 1989.

[17] P. Rajvaidya and K. Almeroth. Analysis of routing characteristics in the multicast infrastructure. In *IEEE Conference on Computer Communications (INFOCOM '03)*, pages 1532–1542, San Francisco, California, USA, Mar. 2003.

[18] A. C. Snoeren, K. Conley, and D. K. Gifford. Mesh-based content routing using XML. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles (SOSP '01)*, pages 160–173. ACM Press, Oct. 2001.

[19] M. Thorup and U. Zwick. Compact routing schemes. In *Proceedings of the Thirteenth Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '01)*, pages 1–10. ACM Press, July 2001.