Routing and Forwarding with Flexible Addressing

Leonid B. Poutievski, Kenneth L. Calvert, and James N. Griffioen

Abstract: **We present a new network-layer architecture that provides generalized addressing. The forwarding infrastructure is independent of the addressing architecture, so multiple addressing architectures can be used simultaneously. We compare our solution with the existing Internet protocols for unicast and multicast services, given the address assignment used in the Internet. By means of an extensive simulation study, we determine the range of parameters for which the overhead costs (delay, state, and network load) of our service are comparable to those of the Internet.**

Index Terms: **Addressing, clustering, forwarding, routing.**

I. INTRODUCTION

The current Internet architecture has several characteristics that are widely believed to limit its flexibility. Addressing, routing and forwarding are tightly intertwined, making it difficult to change or improve any one of them in isolation. Identity is tied to location, making it difficult to support advanced routing paradigms, such as publish-subscribe systems and mobile systems. Furthermore, the Internet Protocol provides few (and the current infrastructure implements even fewer) mechanisms for controlling or manipulating the paths followed by packets end-to-end. To circumvent these limitations, many have proposed new routing and forwarding services implemented as overlay networks [1]–[5]. Such solutions suffer from the overlay-to-underlay mapping problem, resulting in inefficiencies. For instance, the latency in overlays is often higher than the network-layer solution, because the path taken across overlay nodes (end systems) is rarely the shortest path at the network layer. These considerations suggest that investigating the benefits of redesigning the network layer may be worthwhile.

We are developing a new network-layer design that provides a routing and forwarding infrastructure that is, to a large degree, independent of the addressing architecture; in fact, multiple addressing architectures can be used simultaneously with the same forwarding architecture. This makes it possible to provide addressing modes, such as unicast and multicast, using a single forwarding subsystem. Indeed, the addressing architecture could evolve over time to create multiple addressing schemes, without changing the forwarding infrastructure.

In our new network layer datagram-oriented service, called *speccast*, each node is identified by a *node specification* that is not dictated by the node's location in the network and, in general, might be independent of the topology. Of course, the absence of any correlation between node specifications and the topology comes at a cost in terms of scalability. Each packet carries a *destination specification*, along with a partial *forwarding tree*. The latter describes the path(s) along which the network is to carry the packet, while the former defines the destinations that are supposed to receive the packet. Both are placed in the packet by the source. The routers in the network forward the packet along the indicated path(s), duplicating and delivering it to all nodes that match the destination specification. Because it is not possible for sources to know the entire network topology, the original forwarding tree, in general, is only an "outline" of the path. As the packet is forwarded through the network, routers along the way *refine* the tree, filling in gaps caused by the source's lack of knowledge using their own (more detailed) knowledge of the local topology and specifications. We posit a hierarchical link-state routing approach to go along with this forwarding mechanism.

Our approach offers a good deal of flexibility with respect to path selection: Sources may either delegate the question entirely to their service provider or keep the control. We believe such flexibility is not only appropriate, but necessary in an Internet in which the "cost" to originate a packet may vary substantially from node to node — cf. wireless sensor nodes to large web servers. Because paths can be cached and re-used, the effort required to select a path can be amortized over many packets. (As opposed to the current Internet, which amortizes the cost of all routing decisions over all packets equally.)

On the other hand, the major challenge of such a flexible service is scalability. Using simulation, we have compared our approach to existing Internet algorithms for two popular services (unicast and multicast) along several important dimensions of scalability: Forwarding state, delay (i.e., path stretch), and network load. It should be noted that network load is not traditionally considered when evaluating routing/forwarding schemes — at least not forwarding-plane load—because traditional algorithms consider only unicast, and relay a single copy of each packet. We, however, are interested in exploring new trade-offs; in particular we give away some "accuracy" of delivery in return for reduced amounts of state. That is, in order to reduce routing state requirements, we allow some packets to be delivered to parts of the network where they will ultimately be discarded. Our results show that for several kinds of topologies and specification assignments, our design scales comparably to the existing Internet routing/forwarding structure, given the address assignment used in the Internet.

The rest of this paper is organized as follows. In Section II, we present an overview of our routing and forwarding architecture. In Section III, we describe our simulation experiments while Section IV presents the results of our experiments. In Section V, we describe related work, and in Section VI, we present conclusions.

II. THE SPECCAST APPROACH

Manuscript received May 16, 2007.

This work supported in part by the US National Science Foundation under grants CNS-0626918, CNS-0435272, and EIA-0101242.

The authors are with the Laboratory for Advanced Networking, University of Kentucky, Lexington, KY 40506-0046, email: {leon, calvert, griff}@netlab.uky.edu.

In speccast, node and destination specifications can be defined in any one of many possible *specification languages*, because specifications (addresses) are independent of the routing and forwarding mechanism. For example, a specification could be a 32-bit number (i.e., an IPv4 address), or it could be a boolean predicate involving a set of attributes (e.g., "A and B or C and D"), or it could be a tree (e.g., "A with children B and C, where C has children D and E"). Consequently, the speccast forwarding algorithm, described in Section II-B, applies a language-specific "matching" function to determine which nodes satisfy (match) the specification.

If every node knows all node specifications and the entire topology, the originator of a packet can compare the specification to all nodes to determine destination(s), compute the path to all destination nodes, and place a source route (i.e., a *forwarding tree*) in the packet. Unfortunately, this clearly will not scale; it is unreasonable to assume that every node knows the entire network graph, and every node specification. Consequently, we need a way to aggregate the network state, so that hosts only need limited knowledge of topology and specifications.

One way to significantly reduce the network state is to keep aggregated, possibly imprecise state about nodes that are further away. A standard way to achieve this is by using a hierarchy of domains over the node set (as in Kleinrock and Kamoun [6]). In this case some nodes keep state about *clusters*, which are single nodes in the graph, representing connected subgraphs of the original graph. The price for this reduction in state is increased path length: Sources use the same path to all nodes in a cluster, which may not be optimal for each node in the cluster.

Network state in speccast consists of topological information (that describes connectivity) and semantic information (that describes specifications). Aggregating a subset of nodes into a cluster can potentially reduce state not only by hiding topology information (i.e., individual nodes and links within the cluster), but also by reducing semantic state (i.e., by aggregating information about what specifications are satisfied by nodes inside a cluster). To reduce semantic state, we replace node specifications within a cluster with a single abstracted *cluster specifications*, which may have a smaller representation. Since we allow cluster specification to be less precise, an abstracted specification indicates that a cluster "might" contain a node that matches the specification. This imprecision means that a source or intermediate router may incorrectly conclude that a cluster contains a node that satisfies a destination specification, when it does not. Thus, we reduce network state, but at the expense of possible *overdelivery* (i.e., bandwidth wasted by forwarding packets along paths that do not lead to destinations). Because forwarding state is reduced by hiding topology and by abstracting specifications, routers can fill in the missing parts using their own knowledge of their local topology and specifications — which, in general, is more detailed than the originator's because each node has detailed knowledge of its local area. To control this trade-off, we require that the specification language have a "knob" (parameter) that varies the semantic state (sizes of specifications) vs. the amount of overdeliveries. Such a parameter can be a maximum size of a cluster specification.

As in other hierarchical clustering schemes [6], nodes are grouped into nested, disjoint clusters. Since node specifications

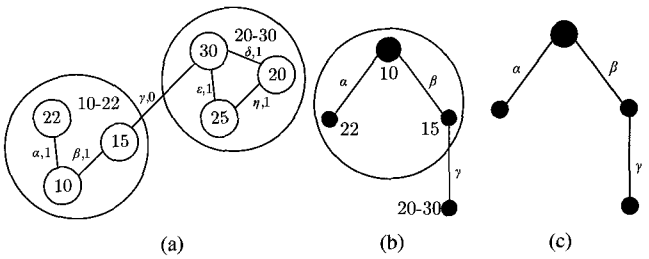


Fig. 1. (a) Topology and specifications, (b) topology view at node "10", and (c) forwarding tree at the source.

are not necessarily unique and might have no relation to topology and hierarchical clustering, we need some way to convey topological information. Since nodes are identified by specifications, we do not assign additional topological names to nodes and clusters. Instead, topological information can be specified by identifying links (i.e., channels between nodes), assigning every link a unique *link identifier* and a *link level*, determined by the link's "depth" in the cluster hierarchy — that is, the number of clusters that contain the link. The specific clustering hierarchy is independent of routing and forwarding algorithms presented here. We assume that the multi-level clustering hierarchy is created by the distributed self-organizing toposemantic clustering algorithm described in Section II-C.

By way of example, suppose that each node corresponds to a user, and the node specification represents the age of the user at that node. Destination and cluster specifications are ranges of ages. Two specifications match if corresponding ranges intersect. Destination specifications are also given by a range of values, meaning that speccast should deliver a packet to all nodes whose ages are within a given range. An example of the forwarding process is given in Section II-B. Fig. 1(a) illustrates a sample hierarchy. Each link has a label, where Greek letters are link identifiers and numbers are link levels. (Lower link level numbers correspond to crossing more link boundaries.) Each node label is a specification (in our example, an age). Node specifications "20", "25", and "30" are abstracted with a single specification "20–30" that represents a range of values. Similarly "10", "15", and "22" are abstracted to a cluster specification "10–22". Each node propagates a routing message (link state announcement (LSA)) containing its specification and links to the nodes in the same cluster. Similarly clusters propagate an LSA containing its *border links* (i.e., the links that connect a cluster to its neighboring clusters) and a cluster specification to peer clusters. In Fig. 1(a), the node with specification "15" transmits two LSAs. The first LSA contains ("15", $\{\beta, \gamma\}$) and is sent over link β and then, link α. The second LSA contains ("10–22", $\{\gamma\}$) and is sent over γ; its information is then propagated to other nodes in the right cluster. The topology view of node "10" in Fig. 1(b) shows a graph, where nodes correspond to clusters (nodes) in Fig. 1(a) for which routing messages have been received.

Another potential application of speccast use is *distributed interactive simulation* (DIS), used by the military. In DIS, each entity must have a view of the battlespace, which consists of information about other entities within the range of a given entity in the battlespace environment. To enable this, each entity must communicate its current state (location, velocity, orienta-

tion, etc.) to all entities interested in this information. The most efficient way to deliver a message to the group of "interested" receivers is to use multicast. In DIS, however, the number of groups is very large (at least one per entity), and delivery to a combination of groups is required. Speccast can represent the destination set efficiently using a destination specification. For example, an entity can be described by a set of values, e.g., $c = vehicle$, $x = 10$, $y = 20$, where x and y represent coordinates. Then, a packet with a destination specification, e.g., $(x \geq 5) \wedge (x \leq 15) \wedge (c = vehicle)$, is delivered to all nodes matching that specification.

A. Speccast Routing

We use a hierarchical link-state algorithm, which involves the generation and propagation of link state announcements (LSAs) and constructing a graph model of the network from received LSAs. Each node in cluster C that connects to nodes in neighboring clusters of C assembles and broadcasts an LSA describing C.

Our routing protocol differs from well known link-state hierarchical protocols, such as OSPF [7]. The link-state announcement in speccast is the external representation of a cluster that consists of topological and semantic information. An LSA's *topological* information is its set of cluster border links along with their levels. An LSA's *semantic* information is an abstracted cluster specification. As in OSPF, each node maintains a database of received LSAs, called a *link-state database*.

To obtain an LSA of a cluster C, LSAs corresponding to subclusters of C are "merged". Border links of C are obtained from a union of border links of subclusters of C, excluding links internal to C. The cluster specification of C is calculated by aggregating and abstracting specifications of subclusters.

B. Speccast Forwarding

The goal of the forwarding algorithm is to deliver a packet to each node that satisfies the destination specification contained in the packet header. Packets are forwarded using a loose source routing approach. Because a packet may have multiple destinations, the source route is a tree (called a *forwarding tree*). Thus each datagram header contains a destination specification and a forwarding tree.

The forwarding algorithm has three parts: Destination specification matching, route refinement (to extend, if necessary, the forwarding tree in a packet), and a forwarding step (to forward the packet according to the forwarding tree). First, if the current node specification matches the destination specification, the packet is delivered to the higher (application) layer. Next, the *route refinement* step is performed. Route refinement occurs when a packet has just entered a cluster: This situation can be detected by the incoming link's level. In the route refinement step, we calculate a forwarding tree of links inside the new cluster that connects the current node with (i) all sub-clusters inside the cluster that might satisfy the destination specification and (ii) all "next-hop" links in the partial forwarding tree (in case they are not directly connected to the node). Several strategies can be used to calculate this tree. In our simulations, we use Dijkstra's algorithm to calculate the shortest-path forwarding tree. This locally-calculated tree is merged with the forwarding tree in the packet. Finally, after the route refinement step is completed, the packet is forwarded to each child link of the root of the expanded forwarding tree.

Consider Fig. 1(a). Suppose that the node with specification "10" (call it s) needs to send a packet with a destination specification d = "20–24". The topology view of s is shown in Fig. 1(b). Its link-state database contains two clusters whose specification might match d: The node with specifications "22" and the cluster with specification "20–30". Node s builds the forwarding tree shown in Fig. 1(c). One copy of a packet is forwarded over link α to node "22". Another copy is forwarded over link β and then γ to the cluster (call it x) with a specification "20–30". Since x's graph model has more information (in particular, it contains a node whose specification matches the destination specification), the forwarding tree is extended with link δ leading to "20" (see Fig. 1(a)). When the packet arrives at "20" and "22", it is delivered to the application.

C. Speccast Clustering Algorithm

In previous work [8], we proposed a *toposemantic* clustering algorithm, which minimizes the total network state consisting of both topological (number of stored LSAs) and semantic (sum of sizes of specifications in link-state database) information. To avoid an excessive number of levels in the resulting hierarchy and thus a high delay stretch, we introduced the constraint $bf > 1$, where bf is a branching factor that determines the maximum number of child clusters.

Our algorithm performs bottom-up hierarchical clustering. It starts with each node in a separate cluster. At each step the algorithm selects a pair of neighboring clusters to merge. We distinguish two types of merging operations: *Push* and *Fuse*. *Push*(A, B) forms a new cluster N containing A and B as subclusters. *Fuse*(A, B) forms a new cluster N containing $A_0, A_1, \cdots, A_n$ and $B_0, B_1, \cdots, B_m$ as subclusters where the A_i's are subclusters of A and B_i's are subclusters of B; clusters A and B cease to exist.

Let $\mathsf{specRed}(A, B)$ denote the reduction of the semantic state due to merging: $\mathsf{specRed}(A, B) = \mathsf{specSize}(A) + \mathsf{specSize}(B) - \mathsf{specSize}(N)$. Let $\mathsf{sib}(A, B)$ denote the number of nodes in clusters that are siblings to A and B. We use an estimate of the reduced network state $H(A, B) = (\mathsf{specRed}(A, B) + \tau)\mathsf{sib}(A, B)$ as the metric for choosing which clusters to merge. In all the simulations we used very large τ ($\forall A, \tau > \mathsf{specSize}(A)$). Therefore, the goal is for all nodes to have approximately the same depth in the hierarchy, and specifications are considered only as a secondary criterion during clustering.

Initially, there is one cluster containing all nodes. The following steps are iterated as long as it is possible to reduce the amount of state:

1. Evaluate the heuristic H on every pair of neighboring clusters A and B such that the number of subclusters in the parent cluster of A and B exceeds bf.
2. Pick a pair of neighboring clusters A and B with a maximum state reduction $H(A, B)$. If the total number of subclusters of A and B does not exceed bf, or, A or B

contains only a single node, then $Fuse(A, B)$; otherwise, $Push(A, B)$.

D. Optimization: Filters

Abstracting away information in LSAs leads to overdelivery: Packets are sent to clusters that should not receive them. We propose to eliminate "overdeliveries" for flows of packets by maintaining additional "filter" state (similar to DVMRP [9]). Filters provide "unhiding" of specification information at specific nodes. The additional state and routing messages required to set up this state can be amortized over a flow of packets.

A filter at a node x that contains an LSA for a cluster c in its routing table can store a negative filter, encoded as a tuple (p, c), where p is a specification and c is a reference to a cluster c. The filter is interpreted to mean that no nodes in a cluster c satisfy specification p even though the advertised cluster specification of c matches specification p. Filters are checked during the route refinement step.

If a node in a cluster c receives a data packet with destination specification p (this might happen because of specification abstraction), but no nodes in c satisfy p, the border router can send a message back toward the sender to install a negative filter and stop future packets with specification p. The negative filter for a cluster c is sent back along the reverse path (visited links can be stored in the packet) to the node x that added c to the forwarding tree, and the filter is installed at node x.

In Section IV-B, we present results demonstrating the effect of filters by showing the amount of overdeliveries for the first packet (without filters) and also for subsequent packets assuming filters are installed. Per flow state needed to store filters is shown as well.

E. Optimization: Default Routes

A standard mechanism for state reduction used in the Internet is the usage of "default routes". For example, if a domain (call it a "stub" domain) is connected to the rest of the network through a single "provider" domain, nodes inside the stub domain need not keep specific state about the network outside that domain; instead, they keep only default routes. If the destination address does not match any local node, the packet is forwarded outside the domain where more information is available to continue forwarding.

A similar mechanism can be used with our approach. Instead of propagating all external LSAs to each node in a stub domain connected only to a single other domain, only a single "default" LSA is propagated. This LSA contains border links of the stub domain and a specification **True** that matches any specification. Once the packet leaves the cluster c with default routing, the refinement step is performed as if the packet had just been originated locally (except it is not forwarded back to c).

III. EVALUATION

We evaluate our routing and forwarding service by comparing it to IP unicast and multicast protocols currently in use in the Internet. Internet routing protocols are considered to be more or less scalable, and thus provide an excellent basis against which to compare the scalability of our network routing service. Our goal is to understand the (added) overhead incurred by supporting a powerful and flexible routing service — i.e., speccast. First, we compare the scalability of both systems for unicast addressing, then for multicast (for varying numbers of groups and varying group sizes), and finally, for unicast and multicast combined. We also study the relation between the specification assignment and speccast parameters.

A. Simulation Model

A.1 Simulated Routing Protocols

To evaluate the scalability of our architecture, we simulated our speccast approach and conventional Internet unicast protocols. For unicast Internet-like routing, we simulated OSPF [7] as the intra-domain routing protocol and BGP [10] as the inter-domain routing protocol. To compare with existing multicast approaches, we simulated PIM Sparse Mode (PIM-SM) [11] as the intra-domain routing protocol and MSDP [12] as the inter-domain routing protocol. In the description of our experiments and the graphs which follow, we refer to the set of all these Internet protocols as *IP*.

To obtain a fair comparison, our BGP implementation only selected the shortest autonomous system (AS) path and did not include support for policies that could prevent the shortest path from being selected. Our simulations assumed that each AS had several BGP speakers, one for each neighboring AS. When forwarding to the next AS in the AS path, the nearest exit (border router) to that AS is preferred. We also implemented "default route" mechanisms in both the IP and speccast simulations, which produced similar state savings for both protocols.

Our PIM-SM simulation assumed a single rendezvous point (RP) per AS for all groups. As defined by the MSDP protocol, our simulation delivered the first multicast packet sent to a particular group to every AS. ASs that have group members joined the RP of the source. Subsequent packets destined for the same multicast group were then propagated from the source to the local RP, then along the source tree to RPs in domains that had group members, and finally from RPs to group members.

A.2 Simulation Topology

We generated two different types of topologies using GT-ITM [13] and BRITE [14]. GT-ITM graphs were generated using transit-stub topologies (3 stub domains per transit node, where transit and stub domains have on average 8 nodes). Different graph sizes were obtained by increasing the number of transit domains; the number of extra transit-stub links was proportional to the graph size (4 per transit domain), and no stub-stub edges were added. BRITE graphs were generated using a top-down, 2-level structure: A top-level (AS) graph with power-law degree property was generated using the Barabasi method [15], while the topology within each AS was generated using the Waxman algorithm. Each AS was connected on average to 1.5 other ASs. All results presented here were calculated as an average over 5 different topologies.

A.3 Clustering

While nodes in our Internet simulation form clusters accord-

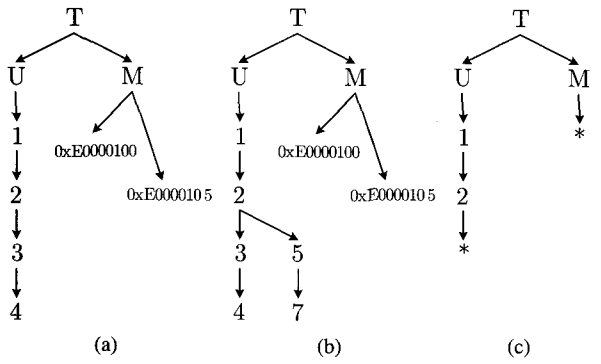


Fig. 2. Example of (a) a node specification, (b) a cluster specification, and (c) an abstracted cluster specification.

ing to AS boundaries determined by the topology generator, speccast uses a different clustering algorithm. In particular, we used the toposemantic algorithm, described in Section II-C.

A.4 Node and Destination Specifications

Although speccast allows many possible types of specifications, our simulations use a simple specification language that is capable of representing unicast and multicast addresses of the kind used in our IP simulation. A node (or destination) specification is represented by a tree of labels. The root label is always **True** (**T**). The root typically has two children (labels): A label **U** for a branch representing unicast addresses, and a label **M** for a branch representing multicast addresses. An example node specification is shown in Fig. 2(a).

An IP address is represented by the **U** label followed by a path of 4 labels, where each label corresponds to a single byte value. For example, the IP address 1.2.3.4 is converted to the specification shown as a **U** branch in Fig. 2(a).

IPv4 multicast addresses can also be represented in this language. For example, the multicast addresses 224.0.1.0 (0xE0000100) and 224.0.1.5 (0xE0000105) can be represented by an **M** branch as shown in Fig. 2(a).

Aggregation of cluster specifications is achieved by merging paths corresponding to node specifications. An example of an aggregated cluster specification is shown in Fig. 2(b). A specification can be further abstracted by replacing any branch in the specification tree with a wildcard. A specification tree can be reduced to a given number of tree nodes *lim* (*maximum specification size*) by performing a breadth first search, leaving the first *lim* nodes, and replacing all other branches with wildcards. In the simulation we use a distinct specification size limit for the unicast branch (lim_u) of a specification and for the multicast branch (lim_m) of the specification. Given $lim_u = 2$ and $lim_m = 0$, the specification in Fig. 2(b) is abstracted to the specification in Fig. 2(c).

Two specifications *match* if every branch of the destination tree overlaps with some branch of the node specification tree. Two branches overlap if all elements are equal up to the last element or up to the wildcard (*) label in one of them.

A.5 Node Specification Assignment

For the GT-ITM topologies, IP address assignment was provided by the topology generator. It produced addresses dependent on node locations in the transit-stub topology. For the BRITE topology, we performed our own, similar address assignment. Unicast specifications in speccast were obtained by converting IP addresses to our specification language.

For multicast we considered both a *high locality assignment*, where group members are placed close to each other, and a *random assignment* where group members are placed randomly through the network.

A.6 Network Traffic

To simulate unicast network traffic we implemented a traffic model in which a source and destination node were randomly selected. Similarly, for multicast traffic destination multicast groups were selected randomly. A packet was then sent from the selected source with the destination specification carried in the packet. This procedure was repeated to create traffic load across the network for the duration of the simulation.

A.7 Metrics

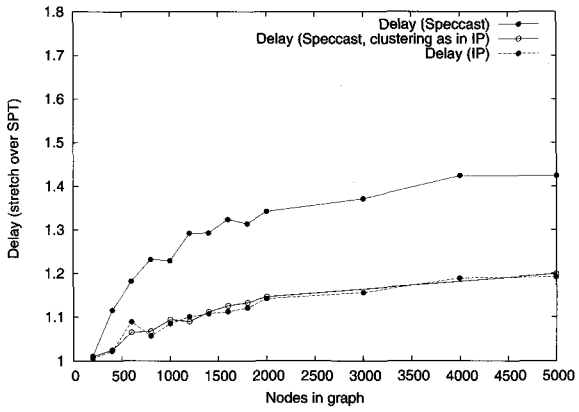
We define the *delay cost* of a routing solution to be the sum, over all nodes d satisfying p, of the number of edges traversed on the path from source n to d, when that solution is used. We define *delay stretch* to be the ratio of the algorithm's measured delay cost versus the minimum possible delay cost (i.e., the shortest path tree (SPT)).

Load is defined as the number of links over which a message is forwarded en route to its destination(s). This includes all links over which the message is forwarded, even if some links do not lead to any node that satisfies p. We define the *network load ratio* to be the ratio of the total number of edges crossed by a message to the number of links in the shortest-path tree from the source n to all nodes satisfying p. (Note that the shortest path tree does not necessarily yield the smallest network load as we have defined it. It is therefore possible for a solution to achieve network load ratios less than one. This effect has been observed before in "shared tree" multicast protocols such as CBT and PIM-SM [16].)

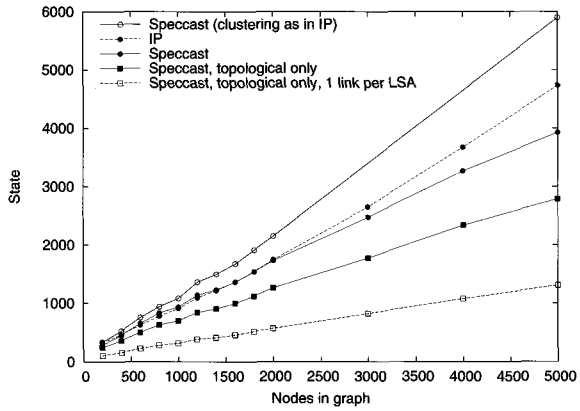
We measure the *network state* of a solution as the average amount of information stored at each node. For speccast we also separately compute the *topological state*, which consists of the network state without counting specifications. For multicast flows of packets we count *per-flow state* and *per-flow number of routing messages*. For speccast these metrics are used to charge for negative filters. For IP these metrics are used to charge for PIM-SM source trees.

A.8 Simulation Parameters

Finally, our speccast service is parameterized by three values: The *branching factor (bf)* used to construct the clustering hierarchy, the *maximum size of a cluster specification* of a unicast part (lim_u), and that of a multicast part (lim_m). The branching factor determines the depth of the hierarchy. As the branching factor increases, so does the size of the topological state that must be maintained at each node; at the same time, delay and load decrease. The maximum cluster specification size defines the amount of information loss that occurs when specifications are merged together. Truncating the specification produces less



(a)



(b)

Fig. 3. Unicast scalability, GT-ITM topology, speccast, $bf = 8$, $lim_u = 15$: (a) Delay and (b) per node state.

precise specifications which leads to overdelivery as described in Section II. This trade-off was studied in our previous work [8].

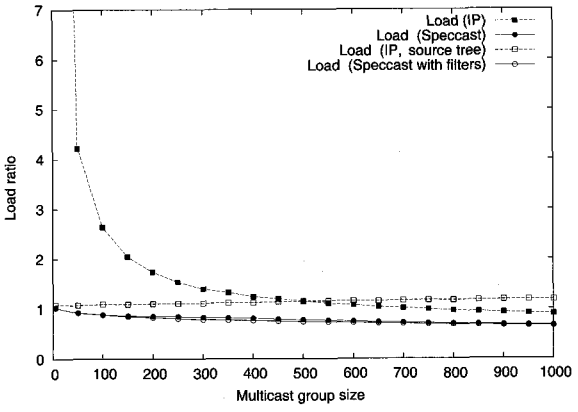
IV. RESULTS

A. *Unicast Results*

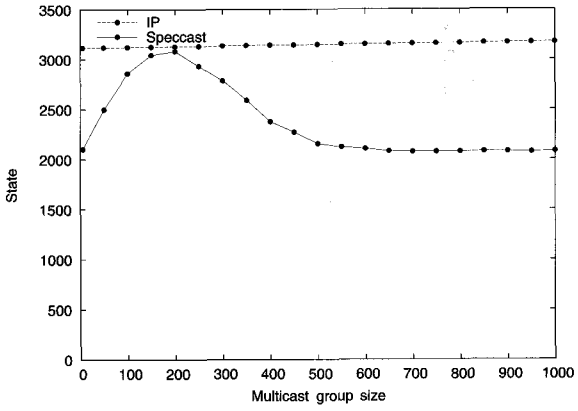
First, we compare speccast to the standard unicast service offered by the Internet today. Similar results were obtained for both GT-ITM and BRITE topologies. We present results for the GT-ITM topology in Fig. 3. BRITE topology results were similar and more results can be found in [17].

Speccast topological state (bottom curves in Fig. 3(b)) and delay (Fig. 3(a)), which are independent of specifications, scale well compared to the Internet. Load and semantic state depend on a particular specification language and specification assignment. For unicast service using our tree-based specification language, speccast achieves smaller state and slower state growth than IP at the expense of a slightly higher delay and load.

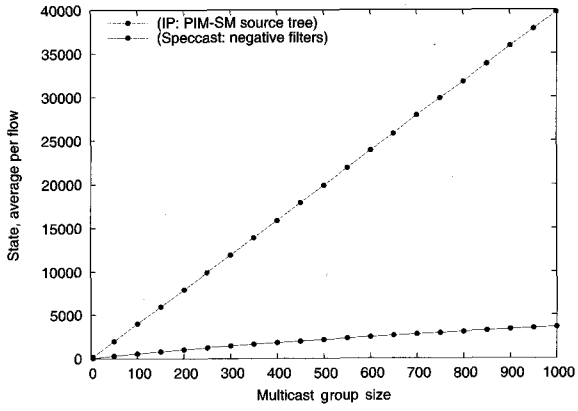
Fig. 3(b) shows that speccast topological state grows more slowly than IP state. This is because speccast has more levels in its clustering hierarchy. Speccast clustering is built with a branching factor of 8, so when the number of nodes in our simulation is greater than 512, the speccast hierarchy has approximately 4 to 5 levels as compared to 2 levels (AS clusters) in the



(a)



(b)



(c)

Fig. 4. Multicast, BRITE topology, 3000 nodes, $bf = 8$, $lim_m = 6$, number of groups = 10, random assignment: (a) Load, (b) per node state, and (c) per flow state.

IP clustering. When IP-like clustering is used for speccast, topological state grows at the same rate as IP (see the curve marked "Speccast (clustering as in IP)" in Fig. 3(b)). More levels in the hierarchy give speccast slower growth of state at the expense of an increased delay. In Fig. 3(a), we see that delay for IP and for speccast with IP clustering is almost the same, and both are lower than the delay of speccast with toposemantic clustering.

In Fig. 3(b), by comparing speccast topological state and speccast topological state where we only counted 1 link per LSA, we can see that a significant amount of speccast topological state (more than half) is caused by multiple links in LSAs.

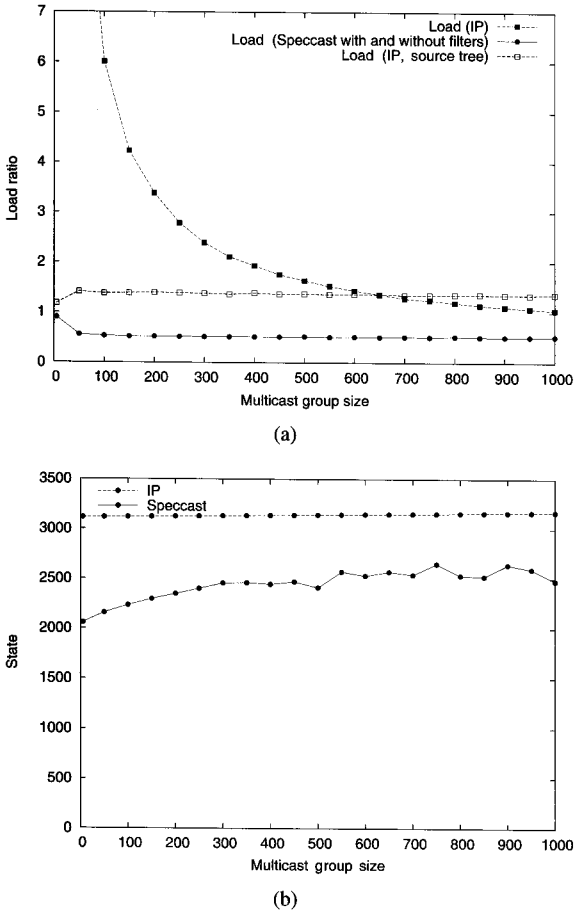


Fig. 5. Multicast, BRITE topology, 3000 nodes, $bf = 8$, $lim_m = 6$, number of groups = 10, high locality assignment: (a) Load and (b) per node state.

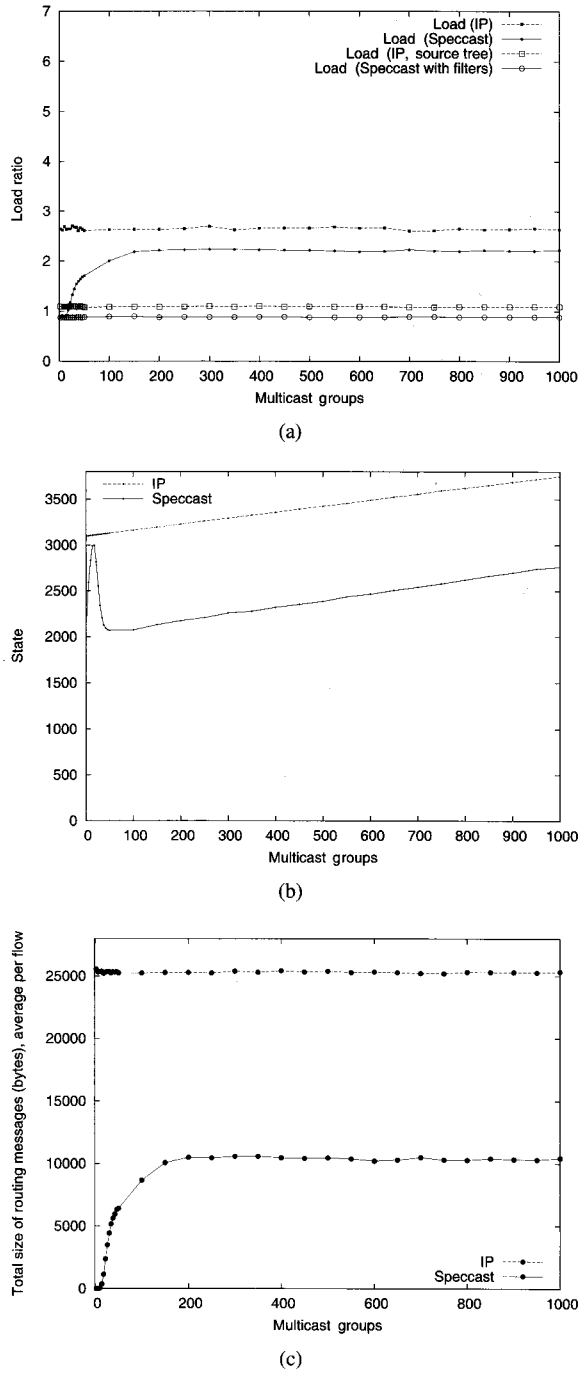


Fig. 6. Multicast, BRITE topology, 3000 nodes, $bf = 8$, $lim_m = 6$, group size = 100, random assignment: (a) Load, (b) per node state, and (c) sum of sizes of routing message per flow.

This is the price that we pay for allowing source nodes and some intermediate routers to have more than one path to choose from.

Unicast specifications add only a small amount of state to the speccast topological state since these addresses have hierarchical structure and are closely related to the topology. For the same reason there is almost no wasted bandwidth for the GT-ITM topology, so the delay stretch in Fig. 3(a) coincides with the load ratio, and thus negative filters are not used. The choice of the maximum specification size (lim_u) is evident from the experiment in Section IV-D.

B. Multicast Resu'ts

In Figs. 4 and 5, we vary the multicast group size from 5 to 1000 while keeping the number of groups fixed (10 groups) and a fixed graph size of 3000 nodes. For a given parameter setting for speccast, we see the same scalability properties as the IP solution. While the only way to change the trade-offs in the IP solution is by switching between protocols (e.g., replace PIM with DVMRP), it is possible to achieve different trade-offs in speccast by adjusting the knobs. Let us look at each metric in more detail.

First, for a small group size we notice that speccast state is lower than IP state because, unlike IP, speccast does not use unicast specifications, while IP multicast requires unicast. Then, in Fig. 4(b), speccast state grows with increasing multicast group size until it reaches saturation, when in most clusters the number of multicast groups per cluster almost reaches lim_m. When the number of groups per cluster becomes greater than lim_m, the precise information about multicast membership is replaced with a wildcard. After saturation, the number of clusters with wildcards increases, and therefore the amount of speccast state decreases as well. High locality assignment exhibits slower growth of state (Fig. 5(b)). Average delay stretch, which does not depend on the number of groups, is lower for Speccast (1.15) than for IP (1.7), since in PIM-SM/MSDP approach, the first multicast packet in a flow is forwarded via at least one RP. For

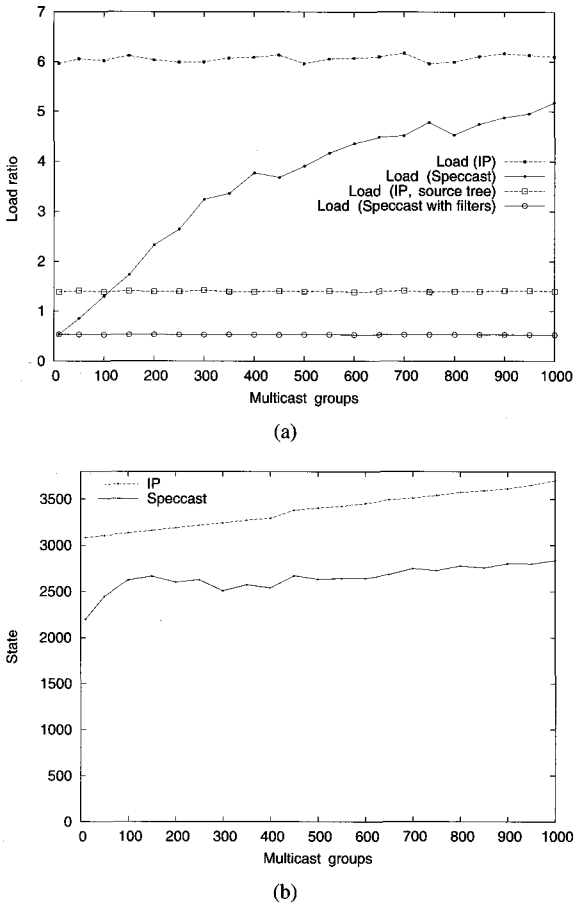


Fig. 7. Multicast, BRITE topology, 3000 nodes, $bf = 8$, $lim_m = 6$, group size = 100, high locality assignment: (a) Load and (b) per node state.

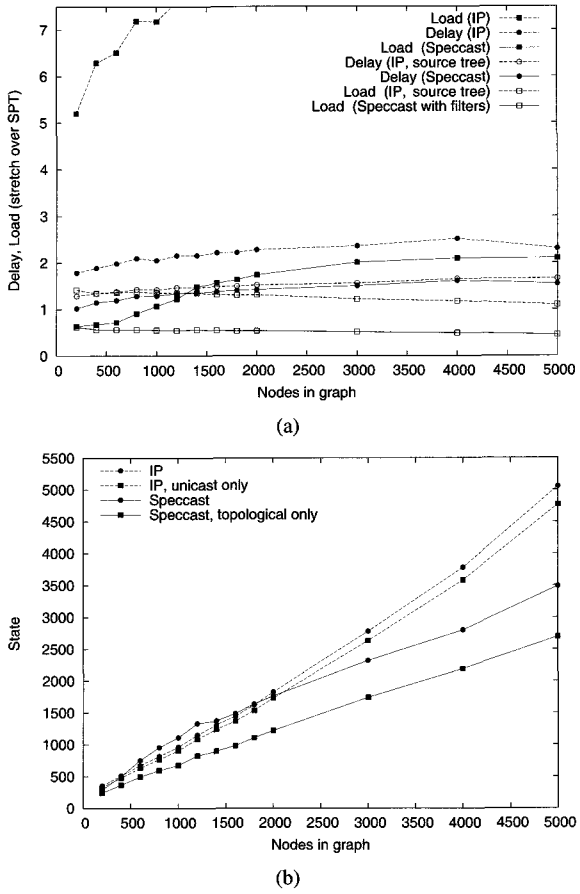


Fig. 8. Unicast and multicast, GT-ITM topology, $bf = 8$, $lim_u = lim_m = 8$, multicast, group size = 5%, groups = 5%, high locality: (a) Delay and load, and (b) per node state.

small numbers of groups, the load ratio (Figs. 4(a) and 5(a)) is significantly better for Speccast. The small amount of overdeliveries appears after the saturation. Load ratio decreases with increasing group size since the load of the SPT increases with growing group size. Load ratio is higher for a high locality assignment (Fig. 5(a)), since the load of SPT for this assignment is smaller, because group members occur in closer proximity than in a random assignment.

For a flow of packets, the PIM-SM/MSDP solution uses a source tree, which reduces the delay stretch to 1.1, which is very close to the delay stretch of speccast, although speccast can achieve similar delay even for the first packet. Speccast with negative filters shows lower load than a PIM-SM source tree (Figs. 4(a) and 5(a)) for flows of packets. Per-flow state (shown in Fig. 4(c)) and the number of per-flow routing messages grow much slower for speccast than for IP. This is because PIM-SM/MSDP uses positive state while speccast uses negative state. Speccast requires only one filter per negative branch, while PIM-SM/MSDP keeps state at every node of the forwarding source tree.

In the next experiment (Figs. 6 and 7), we increase the number of multicast groups while keeping the size of each group constant (100 group members).

Since the maximum cluster specification size is fixed, when the number of groups is small almost all semantic state can fit without being abstracted (Figs. 6(b) and 7(b)), thus, the load is very low (Figs. 6(a) and 7(a)). When the number of groups gets larger, state is abstracted more, which results in a higher load that becomes similar to the one in the IP solution. The spike in Fig. 7(b) has the same explanation as in the previous experiment. The number of per-flow routing messages (Fig. 6(c)) grows with increasing load to prevent false positives with negative filters. Delay and per-flow state remain the same for both solutions since these metrics are independent of the number of groups.

C. Unicast and Multicast Together

In the final experiment (Fig. 8), we combine unicast and multicast services to compare our solution to IP. Traffic was divided to 80% of unicast and 20% of multicast. To keep multicast group sizes and number of groups proportional to the number of nodes, we scale these two parameters with the graph size. The number of multicast groups is 5% of the number of nodes and the group size is 5% of the number of nodes.

Speccast demonstrates a smaller amount and slower growth of state (Fig. 8(b)) and better delay and load parameters (Fig. 8(a)). Note that even the described speccast specification assignment allows much more than just unicast and multicast services. By making small changes to the specification language and by defining other types of destination specifications, it is possible, for example, to provide scoped multicast, where a packet is delivered to multicast group members only inside a desired subnet, or a packet can be delivered to a union or an intersection of mul-

ticast groups.

D. Specification Assignment vs. Speccast "Knobs"

In the next experiment (Fig. 9), we quantify the trade-offs for different levels of *locality*, which is a correlation between node specifications and nodes' locations in a topology. We start with our initial high locality assignment (in which nodes in the same part of the graph have similar addresses), then we pick random pairs of nodes and "swap" their specifications. The number of swapped pairs is written as the percentage of all nodes in the network. More "swaps" lead to lower locality and 50% of swapped specifications gives mostly random assignment of specifications.

For small values of lim_u, the state in Fig. 9(b) is approximately the same for all levels of locality, since the maximum specification size is reached at almost all clusters. This is compensated by extra load (Fig. 9(a)), which is higher for more random assignments (up to 4 times higher for a 1000 node graph).

E. Architecture Discussion

Although our approach could be used to mimic an Internet hierarchy and naming, it was not designed to be compatible with IP. However, it could conceivably be deployed as an interior gateway protocol. Border routers would need to translate between IP packets and speccast. To be practically deployable, many details not considered here would have to be fleshed out.

The computations required to manipulate links are as efficient as in other link-state protocols. The potentially expensive part of our solution is related to manipulating specifications, especially specification matching in the forwarding process. The specification matching is not performed by each router in the forwarding path, but only when a packet enters a new cluster, and the number of specifications for matching is logarithmic in the number of nodes in that cluster. The default route optimization reduces the amount of computation, by moving it to border routers, that could also perform caching and preprocessing. Although theoretically, speccast allows very general specification languages that require solving a satisfiability problem for each specification matching, in practice, we expect more efficient specification languages, similar to the one used in this paper.

V. RELATED WORK

We were not the first to propose this kind of routing/forwarding separation. Yang proposed an alternative architecture, called NIRA, which was designed to work in an ISP context, similar to today's. A major difference is that Speccast focuses on flexibility of node specification assignment, rather than service provider-based addressing.

Several projects have tried to circumvent the limitations imposed by the current network-level architecture (i.e., IP) by reimplementing routing and forwarding services as an overlay network. For example, the *Internet indirection infrastructure (i3)* [1] allows applications to insert a forwarding state at (overlay) routers to define new unicast, multicast, and mobile routing services. The system utilizes an underlying distributed hash table network (such as Chord [18]) to create rendezvous points for

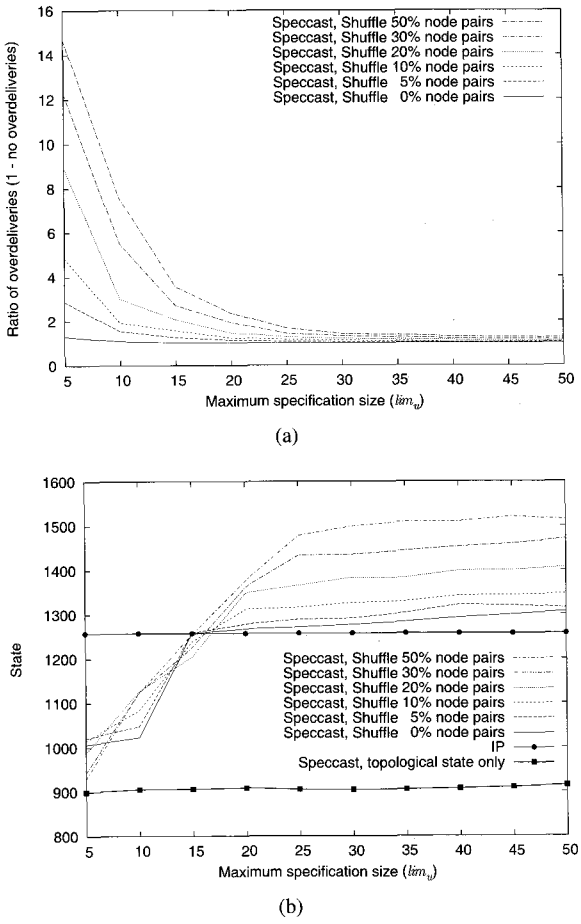


Fig. 9. Trade-offs under changing locality, BRITE topology, 1000 nodes: (a) Load and (b) per node state.

end systems. *Publish-subscribe* systems represent another class of applications that define their own routing/forwarding algorithms [3]–[5], [19]. In these systems certain nodes "publish" information that is consumed by nodes interested in that information ("subscribers"). A publish-subscribe overlay distribution network is then defined to carry data from publishers to subscribers. Unfortunately, all of the above approaches suffer from the overlay-to-underlay mapping problem, resulting in overlay networks that do not make efficient use of the underlying network.

There is also recent work on distributed hash tables implemented on a network layer [20], [21]. In these systems, randomly-chosen numbers are used as specifications. Speccast allows many possible specification languages and achieves a much lower delay, which is independent of specifications.

A notion similar to specification reduction through aggregation in speccast is used in the content-based networking algorithm by Carzaniga *et. al.* [3], [22]. The content-based networking algorithm propagates filters using the underlying "broadcast" layer (assuming it exists). Filters describing several nodes can be shared when the filter of one node "covers" a filter of another (in our terms, if all nodes that satisfy one node's specification satisfy another node's specification) and paths to these nodes from some other node intersect. Our network layer service does not require the overhead needed to maintain a separate underlying routing protocol. Also, our system provides much

greater control to the source, not just by giving more power in describing destinations, but also by allowing it to choose among available paths. Moreover, our system supports specification *abstraction*, which is not supported by the content-based networking service.

Another theoretical work that studied the trade-off between stretch and network state was in the area of compact and interval routing. Initial results were reported by Peleg [23], which were later improved on by Thrup and Zwick [24]. The authors proposed a forwarding algorithm and an algorithm-dependent label assignment scheme (in our terms "node specifications") to optimize the trade-off. We do not control/limit address assignment and explore different trade-offs by allowing several copies of a packet to be present in a network.

Nimrod [25] describes a powerful scalable routing architecture based on hierarchical, link-state, and source routing approaches. Although this architecture could be extended to perform speccast-like routing, the disadvantage is the required resolution before forwarding of a datagram packet: A query-response protocol is needed to obtain the source route to the destination. In contrast, speccast performs on-the-fly route refinement to reduce the delay.

PNNI [26] is a system developed for ATM networks that also uses hierarchical, link-state and source routing ideas. A major difference is that addresses in PNNI can only be aggregated. Our system explores specification state for load trade-off by allowing specification abstraction and the forwarding algorithm which is different from longest prefix matching.

Our early work in this area described a different implementation of *speccast service* [27] where destinations were also described by a specification carried in a packet. We proposed a very simple specification language, but demonstrated the power and flexibility of using speccast as the network-level service on which a wide range of applications could be efficiently built. However, the solution did not scale as well to Internet-sized networks.

VI. CONCLUSIONS

We have presented a novel network-layer routing service that provides separation between topological and semantic information. In our solution, we investigated a novel point in the routing/forwarding solution space — one that allows trading the possibility of *overdeliveries* (increased network load) for reduced state requirements and reduced dependence of node specifications (addressing) on location. We compared the scalability of the proposed architecture with traditional Internet unicast and multicast services and found them comparable. Our system can be extended to implement other services besides unicast and multicast by giving a new specification language.

REFERENCES

[1] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana, "Internet indirection infrastructure," in *Proc. ACM SIGCOMM*, 2002.

[2] Y. H. Chu, S. G. Rao, and H. Zhang, "A case for end system multicast (keynote address)," in *Proc. ACM SIGMETRICS*, 2000.

[3] A. Carzaniga and A. L. Wolf, "Forwarding in a contentbased network," in *Proc. ACM SIGCOMM*, 2003.

[4] G. Banavar, T. Chandra, B. Mukherjee, R. E. S. J. Nagarajarao, and D. C. Sturman, "An efficient multicast protocol for content-based publish-subscribe systems," in *Proc. IEEE Int'l Conf. Distributed Computing Syst.*, 1999.

[5] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley, "The design and implementation of an intentional naming system," in *Proc. ACM SOSP*, vol. 20, 1999.

[6] L. Kleinrock and F. Kamoun, "Hierarchical routing for large networks: Performance evaluation and optimization," *Computer Networks*, vol. 1, no. 3, pp. 155–174, Jan. 1977.

[7] J. Moy, "OSPF version 2," RFC 2328, Apr. 1998.

[8] L. Poutievski, K. L. Calvert, and J. N. Griffioen, "Toposemantic network clustering," in *Proc. IEEE GLOBECOM*, Nov. 2006.

[9] D. Waitzman, C. Partridge, and S. Deering, "Distance vector multicast routing protocol," RFC 1075, Nov. 1988.

[10] Y. Rekhter *et al.*, "A border gateway protocol 4," RFC 4271, Jan. 2006.

[11] D. Estrin *et al.*, "Protocol independent multicast-sparse mode (PIM-SM): Protocol specification," RFC 2362, June 1998.

[12] B. Fenner and D. Meyer, "Multicast source discovery protocol (MSDP)," RFC 3618, Oct. 2003.

[13] E. Zegura and K. Calvert, "Georgia Tech Internet topology models," [Online]. Available: http://www.cc.gatech.edu/projects/gtitm

[14] A. Medina, A. Lakhina, I. Matta, and J. Byers, "BRITE: Universal topology generation from a user's perspective," in *Proc. Int'l Workshop on Modeling, Analysis and Simulation of Computer and Telecommun. Syst.*

[15] A.-L. Barabasi and R. Albert, "Emergence of scaling in random networks," ArXiv Condensed Matter e-prints, Oct. 1999.

[16] K. Calvert, R. Madhavan, and E. Zegura, "A comparison of two practical multicast routing schemes," Tech. Rep. GIT-CC-94/25, College of Computing, Georgia Institute of Technology, 1994.

[17] L. Poutievski, "Speccast: Toward a more general network layer," Ph.D. thesis, University of Kentucky, 2007.

[18] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proc. SIGCOMM*, 2001.

[19] A. Carzaniga, D. Rosenblum, and A. L. Wolf, "Achieving scalability and expressiveness in an Internet-scale event notification service," in *Proc. ACM Symp. Principles of Distributed Computing*, 2000.

[20] M. Caesar, T. Condie, J. Kannan, K. Lakshminarayanan, and I. Stoica, "ROFL: Routing on flat labels.," in *Proc. SIGCOMM*, 2006.

[21] M. Caesar, M. Castro, E. B. Nightingale, G. O'Shea, and A. Rowstron, "Virtual ring routing: Network routing inspired by dhts," in *Proc. SIGCOMM*, 2006, pp. 351–362.

[22] A. Carzaniga, M. J. Rutherford, and A. L. Wolf, "A routing scheme for content-based networking," in *Proc. IEEE INFOCOM*, 2004.

[23] D. Peleg and E. Upfal, "A trade-off between space and efficiency for routing tables," *J. ACM*, vol. 36, no. 3, pp. 510–530, 1989.

[24] M. Thorup and U. Zwick, "Compact routing schemes," in *Proc. ACM Symp. Parallel Algorithms and Architectures*, 2001.

[25] I. Castineyra, N. Chiappa, and M. Steenstrup, "The Nimrod routing architecture," RFC 1992, Aug. 1996.

[26] The ATM Forum, "Private network-network interface specification version 1.0 (PNNI 1.0)," Mar. 1996.

[27] L. Poutievski, K. L. Calvert, and J. N. Griffioen, "Speccast," in *Proc. IEEE INFOCOM*, 2004.

Leonid B. Poutievski was born in Perm, Russia, in 1978. He received a B.S. in Computer Science and Applied Mathematics from Perm State University (Russia) in 1999. Since then, he studied in the Computer Science department of the University of Kentucky, where he received his M.S. degree. At the same time, he worked in the Laboratory for Advanced Networking at the University of Kentucky. He received his Ph.D. from the University of Kentucky in 2007.

Kenneth L. Calvert is a professor and chair of the Department of Computer Science at the University of Kentucky, where he does research on the design and implementation of flexible network services. His interests include network architectures and protocols, programmable network services, secure multicast, and abstract models of network topology. He holds a Ph.D. from the University of Texas at Austin (1991), a M.S. from Stanford University (1980), and a Bachelor's degree from M.I.T. (1979). From 1979 to 1984, he was with Bell Telephone Laboratories, and from 1991 to 1998, he was with the College of Computing at Georgia Tech.

James N. Griffioen is a professor in the Department of Computer Science and is the director of the Laboratory for Advanced Networking at the University of Kentucky. His research interests include network protocol design, network services, distributed file and storage systems, and network access to digital collections. He received his M.S. and Ph.D. from Purdue University in 1988 and 1991, respectively.