Path Optimization in Stream-Based Overlay Networks

Peter Pietzuch
Jeffrey Shneidman
Matt Welsh
Margo Seltzer
and
Mema Roussopoulos

TR-26-04

Computer Science Group
Harvard University
Cambridge, Massachusetts

Path Optimization in Stream-Based Overlay Networks

Peter Pietzuch, Jeffrey Shneidman, Matt Welsh, Margo Seltzer, Mema Roussopoulos
NOTE: Related Work under Blind Submission! Do NOT Distribute!
Harvard University Tech Report: TR26-04
hourglass@eecs.harvard.edu

Abstract

The emergence of sensor networks and distributed applications that generate data streams has created a need for Internet overlays designed for streaming data. Such stream-based overlay network (SBONs) consist of a set of Internet hosts that collect, process, and deliver stream-based data to multiple applications. A key challenge in the design and implementation of SBONs is efficient path optimization when mapping logical query streams to physical network hosts and paths. Suboptimal placements can induce poor utilization of network resources, leading to severe performance penalties, link saturation, and network hotspots. Our goal is to realize efficient stream placement that takes the physical topology of the Internet into account, thereby minimizing overall network utilization.

In this paper, we describe a novel, network-aware path optimization algorithm for stream-based overlay networks. Our approach is based on a spring relaxation model that operates in a metric space defined by the pairwise node latency within the SBON. This relaxation placement algorithm utilizes the underlying network resources efficiently, is capable of performing inter-stream optimization, and can be implemented in a scalable and decentralized way.

To evaluate its performance, we define a set of evaluation metrics for path optimization in terms of network utilization, application delay, and resource contention. Our simulation experiments with a realistic network topology show that relaxation placement approaches optimal network utilization without introducing an undue delay penalty or resource contention. Compared to an optimal placement strategy, relaxation placement causes only $15\,\%$ more network traffic on average, while adding a $24\,\%$ delay penalty for the application. We validate our simulation results with actual measurements involving 70 PlanetLab nodes.

1 Introduction

Recently, there has been much interest generated in building *data streaming applications* on the Internet. These applications typically involve querying, processing, and delivering real-time data from multiple distributed data sources, such as sensor networks, and making use of shared resources in the Internet to aggregate, filter, or multicast this data. Commonly-cited target applications include real-time processing of financial data streams [1, 2], continuous monitoring of Internet paths and system loads [15], and querying geographically diverse sensor networks [14, 18].

In these *data streaming applications*, queries tend to be simple but frequent, and data almost always needs to be transmitted to a location remote from the source. While any single streaming application might not stress the network infrastructure, unless each application deploys its own network, the aggregate load imposed by the myriad applications under consideration is likely to overwhelm any existing infrastructure. Therefore, we need new network infrastructure that is capable of efficiently supporting this class of applications. We call this network infrastructure a *stream-based overlay network* (SBON).

An SBON consists of some number of participants, possibly in different administrative domains, each of whom is willing and able to transmit data and implement a variety of stream operators (e.g., merging two streams, duplicating a single stream to facilitate transmission to multiple destinations, selecting elements from the stream that meet some criteria). In addition, some participants may implement application-specific operators.

Stream-based overlay networks introduce two fundamental technical challenges: First the SBON must perform a *path placement decision*, creating a mapping of a particular data stream and its operators onto a particular collection of participants. Second, the SBON must make this decision in a manner that makes efficient use of the underlying network. There is an inherent tension in this second challenge in that a placement that optimizes for the global network good may harm the performance (delay or response time) for a particular application. Thus, we must optimize network utilization within the constraints of tolerable application latency.

A number of distributed stream processing systems incorporate the notion of an SBON either explicitly or implicitly in their designs [10, 14, 15, 27]. These systems differ with respect to their scalability, fault tolerance, and query models. To date, however, few of these systems have attempted to address the path placement decision in a manner that takes the physical topology of the Internet into account. To the best of our knowledge, none of these systems have considered the placement decision as a whole, and instead focus on the placement or optimization of individual streams.

The contributions of this paper are as follows. First, we formalize the notion of a *stream-based overlay network* and the fundamental path placement decision that must be resolved in such a network. Second, we present a novel network-aware placement algorithm based on a spring relaxation model that operates in a metric space defined by the pairwise node latency within the SBON. The spring relaxation-based algorithm we propose has several attractive properties: it is fully decentralized, has low communication overhead, and it facilitates optimization across multiple data streams. We define a set of metrics that capture different aspects of network efficiency including network utilization, infrastructure utilization (*i.e.,* routers), and end-to-end latency. Finally, we present extensive simulation results that use these metrics to compare our algorithm to several others including IP multicast, random placement, and source/destination placement as well as an optimal placement. We also validate these simulation results with PlanetLab measurements.

We find that our spring-relaxation based algorithm achieves within $15\,\%$ of the optimal bandwidth-latency product (a metric that captures the amount of data in transit in the network)

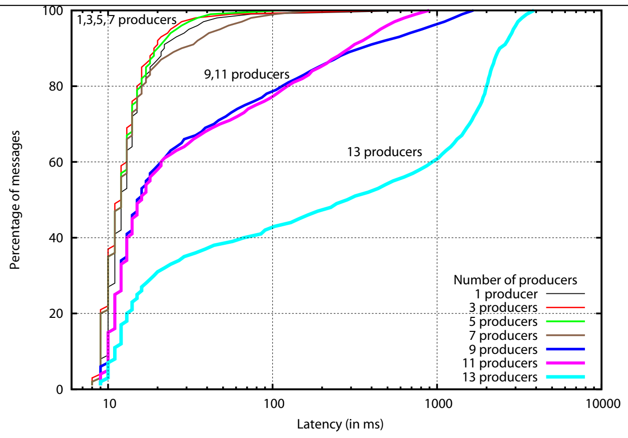


Figure 1: **Resource contention with centralized stream-processing on Planetlab.** *Each producer is sending* 32 KB *messages over a TCP connection at a rate of* 20 KB/s.

while imposing only a 24 % delay penalty. As a point of comparison, while IP Multicast achieves better latency results, it is only within 27 % of the optimal in terms of the bandwidth-latency product.

The rest of this paper is organized as follows. In Section 2, we motivate why SBONs are a necessary alternative to a traditional centralized data warehouse approach and specify the requirements for performing path placement in SBONs. In Section 3, we introduce our SBON system model, formalize the placement problem, and introduce our evaluation metrics. In Section 4, we present the Relaxation placement algorithm. In Section 5, we discuss both our simulation and deployment results. In Section 6, we place our work in the context of other related work and in Section 7, we discuss future work that is beyond the scope of this paper. We conclude in Section 8.

2 Motivation

The traditional approach to stream-based processing involves a centralized data warehouse that collects and processes real-time data [8]. While such an approach permits the use of extensive resource provisioning and replication to manage server load, it does not lead to an efficient use of resources because streams must be routed to the central warehouse for processing.

We illustrate the problems inherent to centralized stream-based processing with the Planetlab experiment in Figure 1, which shows the latency distribution of messages in a stream sent by several producers to a single consumer node. As can be seen from the graph, when more than five producers are simultaneously sending data directly to the consumer, contention in the system causes the observed application latency to increase. The contention worsens as more producers are added, increasing latency drastically.

Stream-based overlay networks (SBONs) offer a better approach by leveraging resources in the Internet to process and deliver stream data. Rather than sending data to a centralized store for processing, we believe that SBONs of the future could be used to run query operators on behalf of multiple concurrent queries, perhaps allowing the results of a particular computation to be shared across multiple users. In such a shared scenario, interesting path optimization problems arise. The essential task facing an SBON is *path placement*, whereby logical streams are

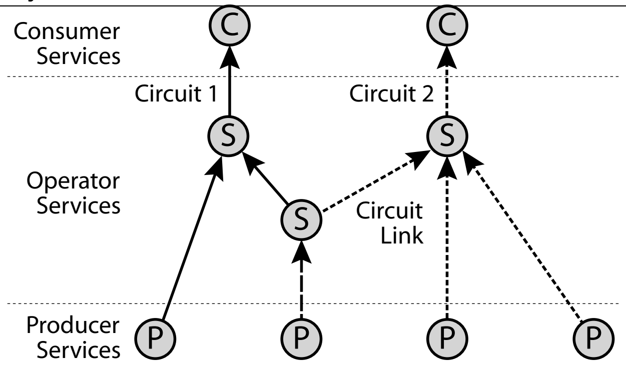


Figure 2: **Example of an SBON with two circuits that share a producer and operator service.**

mapped to physical resources (hosts and network links). Given a logical data stream consisting of zero or more data sources, one or more processing operators, and one or more destinations, the goal is to instantiate operators on nodes in the SBON to make best use of the physical network resources and maintain good performance.

Specifically, a good placement algorithm must satisfy three basic requirements. First, it must be *scalable* in that it can support a large number of concurrent streams, as well as a large number of network hosts acting as data producers, consumers, or intermediate nodes that execute query operators. This implies that the algorithm must be decentralized, not depend on global knowledge of network conditions and have low communication overhead. Second, path optimization should be *adaptive* with respect to changing network conditions such as load, latency, and link utilization, as well as changing stream characteristics and requirements. Third, the placement algorithm should yield *"good" placement*, where good is defined either in terms of global or application-specific metrics. For instance, it could minimize overall network utilization or load, or give placement priority to certain classes of streams. Similarly, the algorithm could incorporate end-application performance goals, such as maximum latency or jitter of streams. Regardless of the design goal, there should be clear criteria with which to evaluate competing algorithms.

3 Stream-based Overlay Networks

In this section, we introduce our model of a *stream-based overlay network* (SBON). A wide range of existing stream processing systems are examples of SBONs and are expressible within this model. We are creating this model because these systems share similar problems and can benefit from common problem-solving methodologies and analysis.

An SBON is an overlay network that streams data from one or more producers to one or more consumers, possibly via one or more operators that perform in-network processing. The basic model of an SBON is one of multiple *circuits* interconnecting multiple *services*, as shown in Figure 2. A circuit is a tree that specifies the relationship and identity of services in a data stream and corresponds to a query. Services that are part of a circuit are connected with *circuit links*. Note that a service identity may not initially specify a physical node for hosting that service; this *unplaced* service scenario is described shortly, but highlights the idea that a circuit can be a *logical* statement, before it becomes fully *realized* through placement decisions, tying it to physical network locations.

Services are categorized into three classes: A (1) *producer service* strictly acts as a data producer for the SBON. A (2) *consumer service* may only receive data from the SBON. An (3) *operator service* processes data, both consuming and producing data streams. If a service is permanently associated with a physical network location, it is called a *pinned service*. Typical pinned services may include producer and consumer services; the network addresses for these services may be explicitly specified by the creator of a circuit. In contrast, an *unpinned service*, such as a relational join operator, can be instantiated at an arbitrary location in the SBON. Unpinned services can further be described as *placed* or *unplaced* based on whether they are currently associated with a physical network location. Unpinned services are placed by a *placement algorithm*. An unpinned, placed service can be re-placed by a later iteration of the algorithm if the network or stream characteristics change.

Our model of an SBON makes few assumptions about services in order to capture a wide range of existing stream-processing systems. In particular, no single query language or data model has been assumed. Instead we assume that information about circuits used in placement decisions can be accessed by the system. For example, this involves knowledge of network characteristics of the circuit links connected to services, such as bandwidth usage. These characteristics are either known statically or determined through run-time measurements.

3.1 Placement Problem

In this section, we formalize the placement problem for unpinned services that underlies data path optimization in an SBON. The placement optimization is motivated by four observations, which we will formalize in the following sections.

1. Unpinned, unplaced services must be placed on physical nodes; in other words, logical circuits must be realized.
2. Some placements are better than others. There is some cost function that reveals the "quality" of a placement decision.
3. The goal of the placement problem is an optimal node assignment for all unpinned services with respect to the cost function.
4. Optimal placement requires global knowledge and is *NP*-hard, so we seek an approximation algorithm that is scalable, adaptive, and yields "good" placement decisions, the criteria mentioned above.

Placement Requirement. We seek a placement Π that specifies where each of the m unpinned services $(s_1 \ldots s_m)$ in the SBON are located from the l possible physical nodes $(n_1 \ldots n_l)$. As a pedagogical device, one can think of Π as a matrix, where a 1 in a service's row entry indicates placement on the physical node corresponding to the column entry. In the absence of outside constraints, a physical node can support multiple placed services, as in this placement of three services onto two nodes:

$$\Pi = \left(\begin{array}{cccc} & n_1 & n_2 & n_3 \\ s_1 & 1 & 0 & 0 \\ s_2 & 1 & 0 & 0 \\ s_3 & 0 & 0 & 1 \end{array} \right) \qquad (1)$$

We require that every service be placed exactly once; rows of Π must sum to 1.

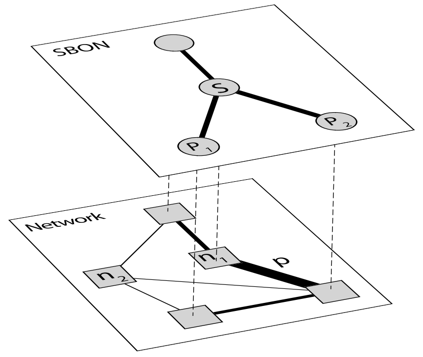


Figure 3: **Example of an inefficient circuit placement.** *The thickness of a link is proportional to the used bandwidth.*

Placement Quality. There are many possible placements Π. To evaluate a placement or to guide an optimization function to find the best placement Π, we must state a cost function of the following form: Given the set of circuits C, which includes the relationship and identity of all pinned and unpinned services, the topology T, which includes the network characteristics, and a specific placement Π, $\mathrm{f}_{\mathrm{cost}}$ evaluates to some scalar value k.

$$\mathrm{f}_{\mathrm{cost}}(C, T, \Pi) = k \qquad (2)$$

Intuitively, k is the quality of the placement decision Π as calculated by this cost metric. Sample cost metrics are given in Section 3.2.

Ideal Goal: Optimal Placement. Ideally, we seek an optimization algorithm that calculates the best possible placement Π. This can be expressed as an argument minimization function,

$$\mathrm{Place}_{\mathrm{opt}} = \arg\min_{\Pi} \left[f_{\mathrm{cost}}(C, T, \Pi) \right], \qquad (3)$$

subject to placing all services exactly once, and perhaps other contraints that specify the set of candidate nodes allowed for placement.

Realistic Goal: Approximation. $\mathrm{Place}_{\mathrm{opt}}$ finds the optimal placement for unpinned services for every circuit in the system. However, the assumption of global knowledge about all possible placements in the SBON is not realistic for a large-scale distributed system. In addition, if f_{cost} is non-trivial, then $\mathrm{Place}_{\mathrm{opt}}$ is *NP*-hard [19]. These two concerns makes the implementation of $\mathrm{Place}_{\mathrm{opt}}$ infeasible.

Therefore, the rest of this paper focuses on $\mathrm{Place}_{\mathrm{relax}}$, which is an approximation algorithm based on spring relaxation that uses two phases to approximate the optimal solution. In the next section, we consider interesting choices of f_{cost}, and then motivate our selection when describing the Relaxation algorithm in Section 4.

3.2 Cost Metrics

Cost metrics can be used both as part of an optimization function and to evaluate a placement. Our strategy in this paper is to use several metrics, which we describe below, to drive our evaluation.

Short of economic utility statements, it is unlikely that all of the participants in a network can reach agreement on the "right" optimization metric. Much existing work in operator placement has focused on allowing individual circuit owners to drive the optimization in their favor. In contrast, this paper places the optimization function in the hands of the network administrator, who may be better equipped to optimize for the "network good." The placement in Figure 3 gives an intuition for what would be an inefficient placement. The placement of service S on physical node n_1 causes link p to carry twice as much traffic, whereas node n_2 would not cause the same problem because of the direct links to the producer nodes. The metrics described in this section capture this type of global network efficiency, but also incorporate application delay penalty and resource contention.

3.2.1 Network Utilization

Bandwidth-Latency (BW-Lat) Product is the product of the bandwidth (in KB/s) used in a realized circuit link e times the latency (in ms) of that link, calculated over all realized circuit links E (computed from C, T, and Π) in the SBON:

$$\sum_{e \in E} \mathrm{BW}(e)\ \mathrm{Lat}(e) \tag{4}$$

The BW-Lat metric captures network utilization by computing the amount of data that is in transit in the network at any point in time. The rationale is that the less data is put into the network by circuits in the SBON, the more network capacity is available to other circuits and applications. As a result, the BW-Lat metric makes an assumption that high latency network links are more costly to use than low latency ones [33]. Often high latency over a network link indicates network congestion caused by popularity or long geographical distance that translates into higher operating cost. In both cases, reducing the utilization of such a costly network link is beneficial.

Bandwidth-Hop Count (BW-HC) Product is the product of the bandwidth (in KB/s) used in a realized circuit link e times the number of physical routing hops for that link, calculated over all circuit links E in the SBON:

$$\sum_{e \in E} \mathrm{BW}(e)\ \mathrm{HC}(e) \tag{5}$$

The BW-HC metric emphasizes the utilization of Internet routers and physical links by including the number of physical routing hops into the cost of streaming data through the circuit. This evaluation metric favors circuits that involve a small number of physical links, thus using a smaller fraction of the total bandwidth available in the network. Often the number of physical routing hops correlates with geographic distance. This metric assumes that routing information about the physical topology is available to get an accurate measure of hop count.

3.2.2 Delay Penalty

Delay Stretch is defined as the sum of the longest path delay under placement Π for a circuit C divided by the shortest path delay. The shortest path delay is the circuit realized by placing all services at the consumer node (Π_{cons}). Thus, using the function Delay to denote the longest path delay of circuit under a particular placement, we have:

$$\sum_{C} \frac{\mathrm{Delay}(C, T, \Pi)}{\mathrm{Delay}(C, T, \Pi_{\mathrm{cons}})} \tag{6}$$

For simplicity, we are ignoring the processing delay introduced by services. Delay stretch is a common metric for measuring the delay penalty that applications have to pay when using an application-level multicast network compared to an IP-level multicast implementation [6].

3.2.3 Resource Contention

Node Stress addresses resource contention in the SBON. Node stress is defined as the number of services hosted at a given node. Usually, each node has a maximum node stress that it can support.

Link Stress is defined as the total bandwidth sent through a physical link. When placing services in the SBON, a circuit link cannot be placed if the placement causes a physical link along the path to exceed its maximum capacity.

4 Relaxation Placement

In this section, we describe Relaxation placement, our network-aware path optimization algorithm. The main idea behind Relaxation placement is to partition placement into two phases. First, an unpinned service in a circuit is placed using a spring relaxation technique in a virtual *latency space* and then the solution is mapped back to the physical *network space*. Performing data path optimization in latency space has the advantage of naturally capturing global knowledge about latency in the network topology, without imposing a large overhead due to network probing. The choice of latency as the principal metric acknowledges that most of the placement cost functions described in Section 3.2 depend on latency.

Next, we explain the characteristics of latency space before presenting the spring relaxation model used for service placement within that space in Section 4.2. A description of the complete Relaxation placement algorithm is given in Section 4.3. In Section 4.4, we explain how Relaxation placement can perform cross-circuit optimization, adopting a global view of the optimization problem.

4.1 Latency Space

Recently, several proposals have been made for the computation of synthetic, n-dimensional *network coordinates* [20, 21] for physical nodes in the Internet. Network coordinates have the property that the Euclidean distance between two coordinates is a reasonable prediction for the communication latency between the corresponding network nodes. This enables large-scale distributed applications to make latency-conscious decisions without the overhead of probing the network directly with all-pairs ping measurements. Network coordinates can usually be calculated after probing the latency to only a small subset of nodes. That subset either consists of a set of well-known landmark nodes [21] or is randomly selected [11]. Since communication latency on the Internet violates the triangle inequality and changes dynamically over time, network coordinates can only provide an estimate of the true latency value. However, simulation results [12] suggest that network coordinates, even with low dimensionality, have a small predication error.

The plot in Figure 4 shows a 3-dimensional latency space for a 1550-node transit-stub topology with 10 transit domains and 150 stub domains. The topology was created with the Georgia Tech topology generator [34] and the network coordinates were

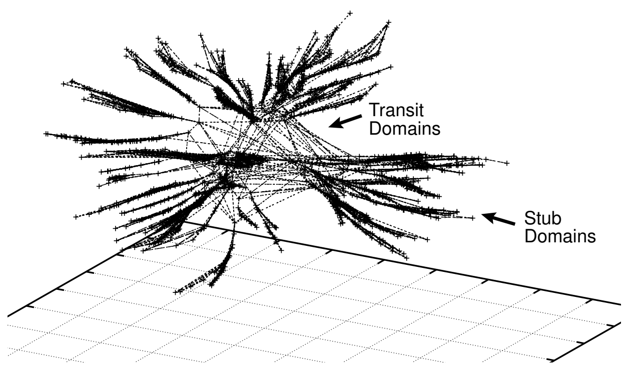


Figure 4: **Visualization of a 1550-node transit-stub topology in latency space.**

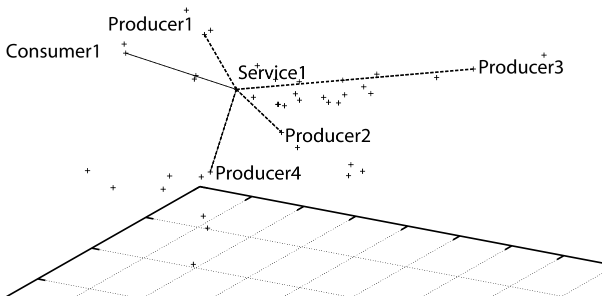


Figure 5: **Example of circuit placement in latency space.** *The circuit consists of* 4 *pinned producers,* 1 *unpinned service, and* 1 *pinned consumer, placed in* 3*-dimensional latency space with* 42 *nodes.*

calculated according to the Vivaldi algorithm [11, 12]. Physical nodes are indicated as points, and edges represent physical links between nodes. The length of an edge predicts the physical communication latency. The transit domains can be seen layered on top of each other at the center of the plot because they are densely interconnected, exhibiting low communication latency. Stub domains are radiating off the center as thin clusters because all outgoing traffic from a stub domain must be routed through one or more transit domains first.

Intuitively, an efficient placement location for an unpinned service in terms of network utilization is "on the routing path" between the pinned services in the circuit. This is captured by the BW-Lat and BW-HC cost metrics introduced in Section 3.2. If an unpinned service is not on the routing path, additional traffic is added to the network, thus increasing both products. The same argument can be made for service placement in latency space. Figure 5 depicts the placement of a single circuit in latency space. This placement is efficient because the unpinned service is located on the routing path between the pinned services, yielding the lowest possible network utilization.

4.2 Spring Relaxation Model

We compute the placement location for an unpinned service in latency space by solving the optimization problem using *spring relaxation*. In spring relaxation, the goal is to compute the rest lengths of a system of interconnected springs. Spring relaxation is a technique that can approximate solutions to minimization problems. For example, the Vivaldi algorithm uses spring relaxation to approximate network coordinates for physical nodes in the network. A spring is modeled as having an extension vector $\vec{s_i}$ and a spring constant k_i. The average force $\vec{F_i}$ experienced by a spring i is governed by

$$\vec{F_i} = \frac{1}{2} k_i \vec{s_i}. \tag{7}$$

A system of interconnected springs attempts to reach a low energy state by minimizing the sum of the potential energies E_i stored in the springs, according to

$$\arg\min_{\vec{s_i}} \sum_i E_i = \sum_i \vec{F_i}\vec{s_i} = \frac{1}{2} k_i \vec{s_i}^2. \tag{8}$$

This can be implemented efficiently with a decentralized algorithm, such that each spring is relaxed independently by moving it a small amount as desired, potentially affecting the extension of other springs in the system. After a number of such relaxation iterations, the system of springs will converge towards the low energy state. Note that the relaxation of an individual spring only requires local knowledge.

To solve the path optimization problem in latency space with spring relaxation, we model circuit links as springs. The spring constant equals the data rate transfered over that link, $k_e = \mathrm{BW}(e)$, and the spring extension comes from the latency, $s_e = \mathrm{Lat}(e)$, where $\mathrm{BW}(e)$ and $\mathrm{Lat}(e)$ are defined as in Section 3.2. Services are modeled as massless bodies between springs. Pinned services have a fixed location, whereas unpinned services can move freely. Expressing this setup as a cost function for the placement problem from Section 3.2, we obtain

$$f_{\text{cost}}(C, T, \Pi) = \sum_{e \in E} \left[\mathrm{BW}(e)\ \mathrm{Lat}(e)\right]^2, \tag{9}$$

This f_{cost} resembles the BW-Lat product cost metric that we would like to minimize. The additional squared exponent in the cost function above gives preference to certain placements over others that have the same cost under the BW-Lat metric.

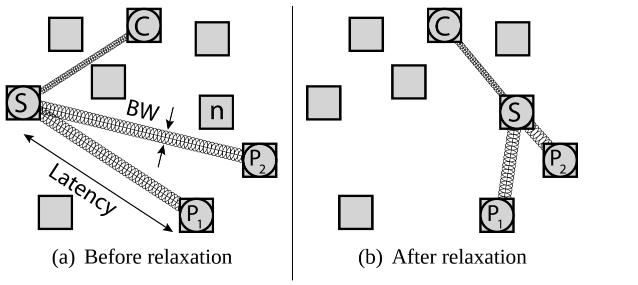


Figure 6: **Example of circuit placement using spring relaxation.** *The circuit is placed in* 2*-dimensional latency space. The thickness of a circuits link is proportional to the used bandwidth.*

Figure 6 illustrates the placement of a circuit in latency space using spring relaxation. In (a), the initial location of the unpinned service S "stretches" the circuit links to producers P_1 and P_2, which then causes the service to migrate to a better node n in (b). In general, a strong force pulling an unpinned service into a particular direction can either be caused by multiple circuit links or a single circuit link that uses high bandwidth.

Using spring relaxation to solve the path placement problem in SBONs has two advantages. First, the iterative nature of

```
1  updatePlacement(S):
2      $\vec{F} \leftarrow \vec{0}$
3      $\forall S_{parent} \in$ Parents($S$)
4          $\vec{F} \leftarrow \vec{F} + (\vec{S} - \vec{S}_{parent}) *$ BW($S$, $S_{parent}$)
5      $\forall S_{child} \in$ Children($S$)
6          $\vec{F} \leftarrow \vec{F} + (\vec{S} - \vec{S}_{child}) *$ BW($S_{child}$, $S$)
7      IF ($|\vec{F}| > F_{\text{thresh}}$) THEN
8          $\vec{S} \leftarrow \vec{S} + \vec{F} * \delta$
```

Figure 7: **Pseudo code for the decentralized relaxation placement algorithm**

spring relaxation allows placement decisions to adapt to changed network and circuit conditions. Second, the decentralized implementation of spring relaxation does not require coordination between services. This keeps the communication overhead low and opens up the possibility of cross-circuit optimization that has global impact in the SBON, as will be described in Section 4.4.

4.3 Algorithm

In this section, we explain how the Relaxation placement algorithm is used by an SBON and outline a decentralized implementation. Consider an SBON with its circuits. Each unpinned service executes the following two steps continuously. (1) The service *places* itself using spring relaxation in latency space with respect to the location of its neighbors in the circuit. After that, (2) it *maps* the computed location from latency space to a node in the physical network. If the new physical location differs from its current location, the service may migrate to the new node.

To make the placement decision adaptable, the placement and mapping steps are repeated continuously by all unpinned services in the SBON. This does not cause a large communication overhead because each service can perform its placement decision with local knowledge only after learning the network coordinates of its direct circuit neighbors. When the cost of migration from one node to another is less than the improvement in the cost metric, the service moves to the new node. This optimizes the data paths created in the SBON and adjusts them to changes in network and application conditions. Next, we describe the two steps in more detail.

Relaxation Placement. We show the pseudo code for the distributed Relaxation placement algorithm for circuits in Figure 7. The `updatePlacement` function is executed periodically by every unpinned service in the SBON. The current network coordinate of the service is $\vec{S}$. A new unpinned service that has been added to the SBON is assigned a provisional location in latency space, such as the node hosting the consumer service for the new circuit. The total force $\vec{F}$ is calculated by iterating over all parent and child services connected through input and output circuit links to service S and obtaining their current network coordinates. The force $\vec{F}$ is updated with the distance in latency space between the current coordinate $\vec{S}$ and the remote coordinates scaled by the bandwidth used by the circuit link (lines 3–6). The movement of the service through latency space is dampened by a factor δ. If the magnitude of the force vector $\vec{F}$ is larger than a force threshold F_{thresh}, the `updatePlacement` function updates the current network coordinate $\vec{S}$ (lines 7–8). The choice of the force threshold depends on the cost of service migration. After the service coordinate has been updated, the service is mapped to a physical node in the SBON.

Service Mapping. To place a service in the SBON, it is necessary to map the network coordinate of the computed placement location in latency space to a suitable physical node. Two considerations are important here. First, the mapping should choose a physical node with a network coordinate as close as possible to the placement coordinate. Second, the selected node and network path must have sufficient resources to support the new service and circuit links. Resources include node resources, such as the load in terms of CPU utilization (node stress) or memory consumption, and network resources, such as the available network capacity on the path of the circuit links (link stress).

We propose a simple scheme, in which we consider the k-closest physical nodes in latency space to the placement coordinate in order of proximity and select the first node for placement that has sufficient resources. This relies on a scalable directory mechanism for looking up network coordinates of existing nodes and identifying the closest one. A possible solution is to rely on the routing properties of a DHT, such as Pastry [24], which routes a message with a destination key to the node with the closest existing key in the system. Each physical node in the SBON stores its network coordinate in the DHT. An n-dimensional network coordinate can be mapped to a 1-dimensional DHT key with the help of a space-filling Hilbert function [25]. To find the closest existing network coordinate, a message is routed via the DHT with the placement coordinate as a key. The DHT then ensures that the message arrives at the closest existing node in the coordinate space.

The mapping of a placement location in latency space to a physical node will introduce a *mapping error* that depends on the distribution of nodes in latency space. In Section 5.5, we show that the mapping error stays within a small fraction of the network diameter for realistic network topologies.

4.4 Cross-circuit Optimization

An advantage of the Relaxation placement algorithm is that it naturally supports cross-circuit optimization decisions when circuits are interconnected with shared services. For this, the entire circuit graph can be viewed as a network of springs. The placement of an unpinned service in the SBON can then potentially affect the placement of any other service with transitively shared circuit links. In practice, placement effects will be more localized because the cost of service migration ensures that placement decisions do not create a domino effect in the system. Incorporating the required bandwidth of a circuit link in the cost function has the desired result that placement decisions for services with high data rates have a stronger global impact in the SBON.

To enable the sharing of services and links among circuits, a *circuit analyzer* must determine whether a new circuit overlaps with any circuits already existing in the SBON. If this is the case, stream processing and communication efforts can be reused, thus reducing resource usage in the SBON. A wide variety of techniques found in distributed query optimization [17] can be adopted to identify coverage among circuits.

To evaluate the potential of cross-circuit optimization with Relaxation placement, we implemented a simple circuit analyzer that determines whether producers are shared among circuits in the SBON. For each newly added circuit, the circuit analyzer checks whether any of the producers are already part of other circuits. When such a producer is identified, the structure of the involved circuits is altered by inserting a *multicast service*. A multicast service has a single input link from the producer and multiple outputs links connected to the services that share this producer. It can then be placed at a location closer to the consumers using the regular Relaxation placement algorithm described above in order to reduce the amount of network traffic generated. We evaluate the effect of multicast services on Relaxation placement in Section 5.3.1.

5 Evaluation

In this section, we present the evaluation of the Relaxation placement algorithm for solving the path optimization problem in an SBON. Our goal is to compare Relaxation placement to five other relevant placement algorithms using the metrics defined in Section 3.2. The results are obtained from both simulation and PlanetLab deployment.

In simulation, we are able to perform the full evaluation and compare each algorithm in terms of network utilization (BW-Lat and BW-HC metrics), application delay penalty (delay stretch), and resource contention (node and link stress). In addition, we examine cross-circuit optimization in terms of the BW-Lat metric using *multicast services*, as described in Section 4.4.

On PlanetLab, we compare Relaxation placement to four other algorithms because the fifth, IP Multicast placement, assumes complete knowledge of the physical topology. We explore the BW-Lat metric and an approximation to the BW-HC metric since the exact physical topology is unknown. We also examine application delay stretch. These experiments were performed by entering the full set of Planetlab ping times into our off-line simulator, which then output an experiment file with placement decisions. This file was used to deploy a real stream-processing system on Planetlab. We did not run resource contention experiments on Planetlab because of the side-effects this would have on other experiments running in this shared environment.

The remainder of this section is structured as follows: first, we describe the alternate placement algorithms; we then present our simulator setup in Section 5.2 and the simulation results in Section 5.3; this is followed by our PlanetLab setup in Section 5.4 and the corresponding results in Section 5.5.

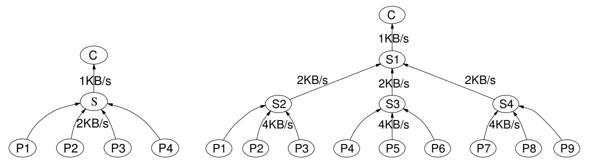
5.1 Other Placement Algorithms

In order to provide a comparative evaluation of the Relaxation algorithm described in Section 4, we implemented five additional placement approaches.

Optimal Placement chooses the best possible placement with respect to the BW-Lat cost metric after performing an exhaustive search over all possible placements, having global knowledge of all placement locations. The complexity of this algorithm is exponential in the number of unpinned services in the circuits. This means that results for Optimal placement can only be calculated for simple circuits.

IP Multicast Placement puts unpinned services at physical nodes that would be routers hosting branches of an IP multicast tree. In simulation, we calculate the IP multicast tree by taking the union of all the IP unicast routes from the producers to the consumers.

Producer Placement randomly picks one of the physical nodes hosting producer services for the circuit and places all unpinned services at this node. Such a placement algorithm often utilizes the network less than Consumer placement because it can better leverage any selectivity of operator services. Less



(a) Simple circuit (b) Complex circuit

Figure 8: **The two circuits used in experiments.**

data is sent through the circuit if it is processed and potentially discarded at a location close to the data producers.

Consumer Placement places all unpinned services at the physical node hosting the pinned consumer service. This placement algorithm corresponds to a centralized, data-warehouse approach to stream-processing, which is often used for current deployments of SBONs.

Random Placement picks a physical node for each unpinned service in the circuit at random. This placement algorithm is also equivalent to a placement strategy in which a DHT is used to hash an operator to a physical node in the network.

5.2 Simulation Setup

Since some of our evaluation metrics require the knowledge of the physical network topology, we decided to use a simulator to evaluate the performance of our path optimization algorithm. Following previous work on the evaluation of large-scale peer-to-peer systems [24], we built a discrete-event simulator to avoid the scalability issues of standard network simulators. Our simulator can place a large number of circuits on a physical network topology according to one of several placement algorithms and then compute a set of evaluation metrics.

Most of our experiments were carried out with a 1550-node transit-stub topology, which was generated by the Georgia Tech topology generator [34]. The topology has 10 transit domains with 5 nodes, each connected to 150 stub domains with 10 nodes on average. Routing tables for the topology were calculated with the help of the routing policy weights assigned to physical links by the topology generator. These weights are intended to reflect Internet routing policy. The network diameter of the transit-stub topology was $878\,\mathrm{ms}$.

The majority of experiments evaluated placing 1000 circuits in the SBON. Pinned services were randomly distributed across the network. Only a single producer service was hosted at any physical node, with producers being shared among circuits. Each circuit had its own consumer service. The experiments considered the two types of circuits shown in Figure 8. Both circuits aggregate data from more than two producers, which we believe is realistic for many applications. The *simple circuit* consists of 4 producers sending messages at a rate of $2\,\mathrm{KB/s}$ with 1 unpinned service S aggregating the incoming data and outputting $1\,\mathrm{KB/s}$. The *complex circuit* involves 9 producers and requires the placement of 4 unpinned services.

5.3 Simulation Results

Our experiments investigated the network utilization, delay penalty, and resource contention of the six placement algorithms. Next, we present and discuss the results for each of these in terms of our evaluation metrics from Section 3.2.

Algorithm	*Simple circuit*	*Complex circuit*
Optimal	1.00	—
IP Multicast	1.27	1.00
Relaxation	1.15	0.90
Producer	1.43	1.43
Consumer	1.60	1.32
Random	1.81	1.58

Table 1: **Ratio of BW-Lat product for different placement algorithms.**

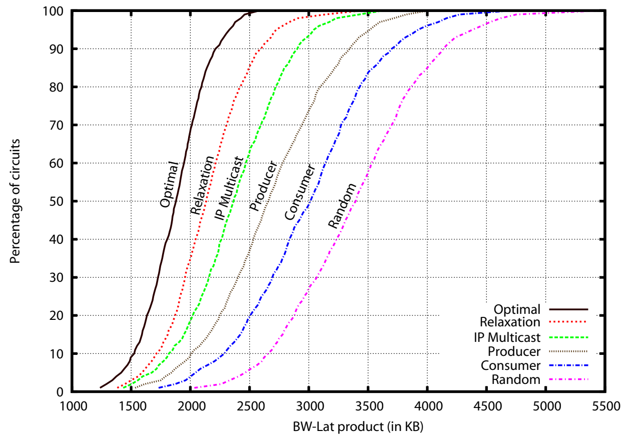


Figure 9: **Distribution of BW-Lat product for different placement algorithms.**

5.3.1 *Network Utilization*

Bandwidth-Latency Product. Our first evaluation metric is the BW-Lat cost that captures network utilization as the amount of data in transit. Table 1 shows the BW-Lat product as a ratio between Optimal placement and IP Multicast/Relaxation/Producer/Consumer/Random placements. The results give the average BW-Lat ratios of two separate experiments, placing 1000 simple and complex circuits in the 1550-node topology.

For the simple circuit case, Relaxation placement performs well, which is expected because BW-Lat is the cost function used by this placement algorithm in latency space. Relaxation placement is more efficient than IP Multicast placement because IP Multicast attempts to minimize routing hops and not necessarily routing latency. Therefore, Relaxation placement manages to find better placement nodes that are not on the IP routing path from the producer to consumer. The Producer placement algorithm does better than Consumer placement because, by placing the selective operator service on one of the producers, data from only 3 producers needs to be sent through the network to the operator service. Random placement exhibits the highest BW-Lat cost and Consumer placement is only marginally better.

With complex circuits, the Optimal placement algorithm is not feasible any more because of the complexity of the exhaustive search through all placement possibilities. Instead, we normalize the BW-Lat ratio against IP Multicast placement. Relaxation placement has the lowest BW-Lat cost. Producer placement does worse now than Consumer placement because the reduction of traffic by one producer is out-weighed by the bad placement decision of picking that producer for the placement of all services in the circuit.

The distribution of the BW-Lat product per circuit as a CDF is plotted in Figure 9. It illustrates the clear division between

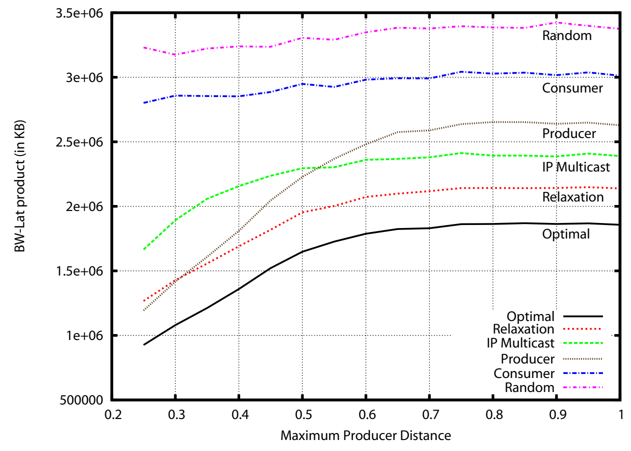


Figure 10: **BW-Lat product as a function of maximum producer distance (MPD) in circuit.**

the evaluated placement algorithms. Even for the small fraction of badly placed circuits, Relaxation placement manages to cause only 33 % more network traffic compared to Optimal placement. In the worst case, Producer placement results in 55 % more data.

In the previous experiment, the producers in the SBON were randomly assigned to physical nodes in the network. However, for some applications it is more likely that producers in a circuit are heavily clustered because, for example, they represent sensor networks that are in physical proximity to each other. To address this, the graph in Figure 10 investigates how the sum of the BW-Lat product is influenced by the clustering of producers in the circuit.[1] The maximum producer distance (MPD) is defined as the fraction of the network diameter that limits the maximum distance between any two producers in a circuit.

As can be seen from the plot, an efficient placement strategy can result in a lower BW-Lat product when the producers are highly clustered in the circuits. In that case, Producer placement is also a viable placement strategy because it places unpinned services within that cluster. As the MPD increases, Relaxation and IP Multicast placement give better performance.

Bandwidth-Latency Product with Multicast Services. As explained in Section 4.4, *multicast services* can improve the network utilization when producers are shared among circuits. We examined an extension to Relaxation placement, called *Relaxation-MC*, which has a circuit analyzer for adding multicast services on demand. For comparison, we also considered an extension of IP Multicast placement, called IP Multicast-MC, in which multicast services are also added to shared producers. Unlike Relaxation-MC, IP Multicast-MC does not optimize services globally. This means that the location of placed services is never reconsidered, unless they are added to a new circuit.

The plot in Figure 11 shows how the BW-Lat product is affected by multicast services when more circuits are added to the SBON. It compares the two versions of Relaxation and IP Multicast placement with and without multicast services against the baseline of Optimal placement (without multicast services). Initially, as few producers are shared, multicast services add an overhead. However, after about 250 circuits are added, multicast

[1]Note that the y-axis of this plot (and others) is not zero-based. This is to emphasize the differences between the placement algorithms, acknowledging that no algorithm can perform better than Optimal placement.

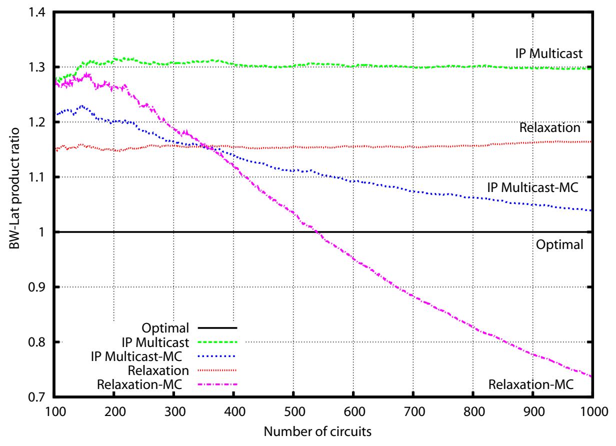


Figure 11: **Ratio of BW-Lat product for placement with multicast services as a function of number of circuits.**

Algorithm	*Simple circuit*	*Complex circuit*
Optimal	1.00	—
IP Multicast	1.12	1.00
Relaxation	1.17	1.10
Producer	1.38	1.60
Consumer	1.56	1.48
Random	1.76	1.77

Table 2: **Ratio of BW-HC product for different placement algorithms.**

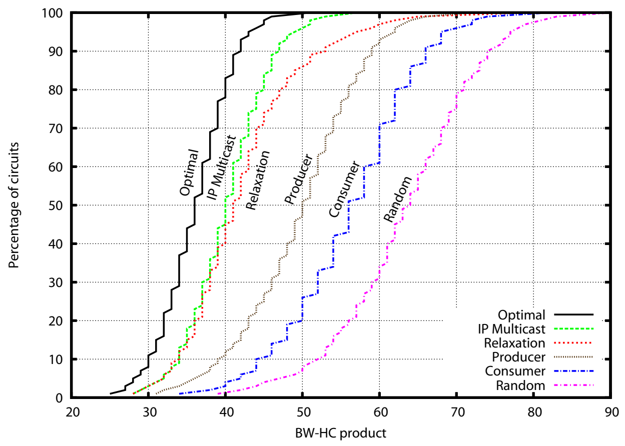


Figure 12: **Distribution of BW-HC product for different placement algorithms.**

Algorithm	*Simple circuit*	*Complex circuit*
Consumer	1.00	1.00
Optimal	1.13	—
IP Multicast	1.00	1.00
Relaxation	1.24	1.44
Producer	1.75	2.58
Random	1.76	2.58

Table 3: **Delay stretch for different placement algorithms.**

services substantially improve the efficiency of the system. The Relaxation-MC algorithm performs better than IP Multicast-MC because it is capable of globally optimizing the placement of all services in the system, whereas IP Multicast cannot revisit placement decisions and thus is restricted to local placement on a per circuit basis. Because producers and consumers are randomly distributed in the network, the Relaxation-MC algorithm slowly causes all multicast services to move into the transit domains. This makes produced streams easily accessible to all consumers.

Bandwidth-Hop Count Product. The second network utilization metric we evaluated is the BW-HC product, which quantifies the impact of a placement on routers and network links. Similar to the BW-Lat metric, we express the BW-HC product as a ratio between Optimal placement and IP Multicast/Relaxation/Producer/Consumer/Random placements in Table 2. Note that Optimal placement still optimizes for the BW-Lat product as before. The results give the average BW-HC ratios for the placement of 1000 simple and 1000 complex circuits.

A noticeable feature is that, unlike for the BW-Lat product, IP Multicast now gives better results than before because it attempts to minimize the number of physical routing hops in the placement path of the circuit. Even though Relaxation placement does not optimize for the BW-HC product directly, it performs only marginally worse than IP multicast and substantially better than Producer, Consumer, and Random placement. The saving obtained by Relaxation placement is even larger for the complex circuit, where bad placement decisions have bigger impact.

The plot in Figure 12 shows the distribution of the BW-HC product per circuit as a CDF. Using this metric, IP Multicast placement performs close to the Optimal placement algorithm. For about 40 % of the circuits, Relaxation placement is able to find a similarly efficient placement as IP Multicast placement. However, in a small number of cases, Relaxation placement favors additional hops to reduce the latency of the overall path, and thus shows worse behavior.

5.3.2 *Delay Penalty*

Delay penalty is concerned with the delay overhead of placing services in-network, which is important to most applications using circuits in a SBON. In general, the lowest delay on the longest path from any producer to the consumer in the circuit is achieved when all unpinned services are placed on the consumer node, resulting in the Consumer placement algorithm. We express delay penalty as *delay stretch*, which normalizes the longest path delay with respect to delay of Consumer placement.

Table 3 shows the average delay stretch after placing 1000 simple and 1000 complex circuits in our 1550-node transit-stub topology. As expected, the table reveals that Consumer and IP Multicast placement have optimal delay stretch because they place services only along IP routing paths. However, Relaxation and Optimal placement introduce only a small average delay penalty of 24 % and 13 %, respectively. Producer and Random placement perform worst because they add a random delay overhead. The delay penalty for complex circuits is higher because more unpinned services need to be placed. This effect is especially visible for Producer and Random placements, which both exhibit the highest delay penalties.

Figure 13 shows the distribution of the delay stretch after placing 1000 simple circuits. An interesting observation is that both Optimal and Relaxation placement achieve a lower delay stretch than Consumer placement for about 15 % of the circuits. This can be explained by the fact that the routing weights in transit-stub topology reflect the real-world, in which IP routing paths are sometimes sub-optimal in terms of latency. Optimal and Relaxation placement are then able to reduce the latency by

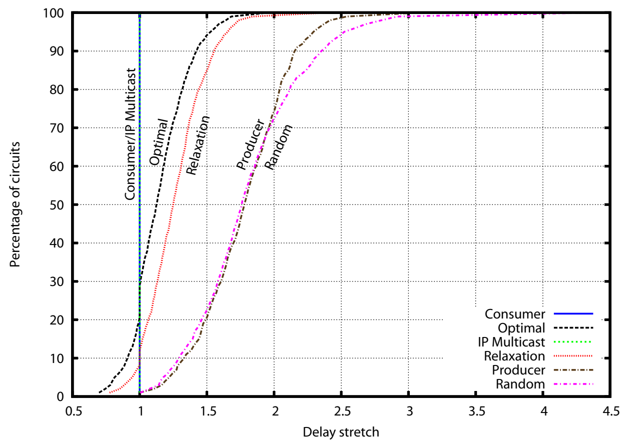


Figure 13: **Distribution of delay stretch for different placement algorithms.**

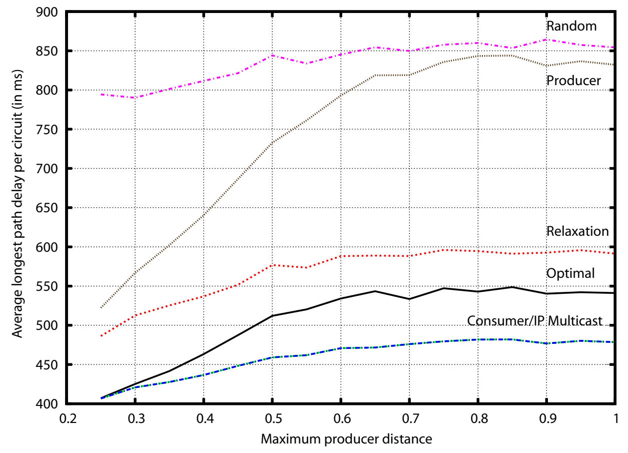


Figure 14: **Average longest path delay as a function of maximum producer distance in circuit.**

choosing a different overlay routing path with a higher hop count but lower latency. Producer and Random placement have a long tail of bad placement decisions with a large delay stretch.

The delay stretch is also influenced by the clustering of producers in a circuit. The plot in Figure 14 shows how the longest path delay in a circuit is affected by the clustering of producers, as expressed by the MPD. The results show that if producers are highly clustered ($\mathrm{MPD} = 0.25$), Consumer and Optimal placement perform similarly well because unpinned services that are placed on one of the producers are likely to be on the routing path to the consumer. This effect is confirmed by the low longest path delay achieved by Producer placement when the MPD is low. Irrespectable of the MPD, Relaxation placement manages to obtain a good compromise in terms of delay penalty.

5.3.3 Resource Contention

Node Stress. The experiment in Figure 15 investigates the distribution of placed services (i.e. the load) across the physical nodes in a 600-node transit-stub topology with 3 transit domains and 72 stub domains. In (a), the topology is shown in latency space with the transit domains at the center. (b) Random placement of 1000 simple circuits does not create any load peaks because of the uniform placement of unpinned services. However, (c) Optimal placement identifies that placing unpinned ser-

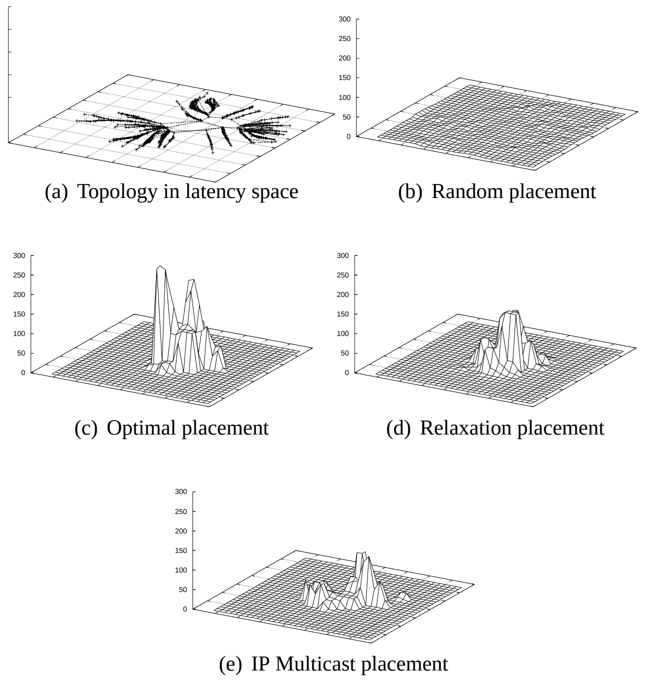


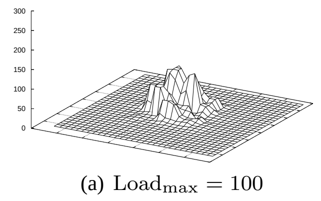
Figure 15: **Load maps for different placement algorithms.**

vices in transit domains results in more efficient network utilization. When consumers and producers are randomly distributed in the network, most traffic must flow through the transit domains. Therefore, placing services there does not consume additional network bandwidth. The 3 transit domains in the network topology are visible as peaks in the load map. (d) Relaxation and (e) IP Multicast placement also prefer transit domains for service placement, although not as strongly as Optimal placement.

As mentioned before, any efficient placement algorithm must cap the number of unpinned services that can be placed at a physical node. The plots in Figure 16 show how the concentration of service placement in transit domains caused by Relaxation placement can be reduced through a threshold on the maximum number of services, $\mathrm{Load}_{\max}$, that can be hosted at a physical node. As the load threshold is lowered, more services are placed sub-optimally outside the transit domains and move into the stub domains, reducing load at the center of the topology.

Of course, the impossibility of placing unpinned services at the desired physical nodes reduces the efficiency of the placement algorithm. In Figure 17, the graph illustrates how a maximum load threshold $\mathrm{Load}_{\max}$ affects the network efficiency of Relaxation placement. Efficiency is expressed as the BW-Lat product after placing 1000 simple circuits in a 1550-node transit-stub topology. As can be seen from the plot, the loss in efficiency starts increasing substantially after a low load threshold of about 40 services. However, the placement efficiency still remains better compared to IP Multicast, which is included for comparison.

Link Stress. Another network resource that may prevent the placement of a circuit link on a network path is link stress. We defined link stress as the total amount of traffic that traverses a physical link. Figure 18 shows the distribution of link stress as a CDF in a 1550-node transit-stub topology after creating 1000 simple circuits for different placement algorithms. The figure shows that Optimal, IP Multicast, and Relaxation placement cre-



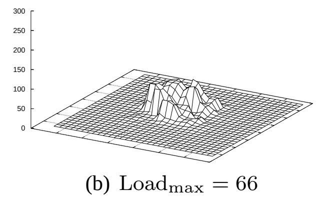
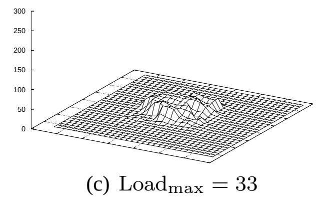
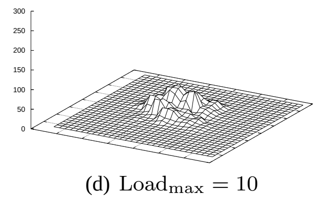




Figure 16: **Load maps for relaxation placement with different load thresholds** $\mathrm{Load}_{\max}$**.**

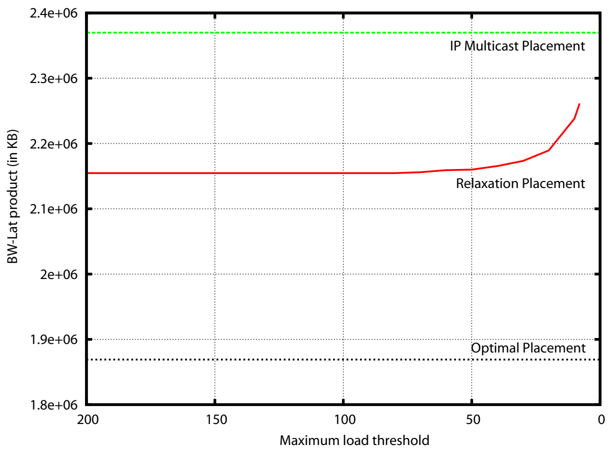


Figure 17: **BW-Lat product for Relaxation placement as a function of load threshold** $\mathrm{Load}_{\max}$**.**

ate overall less link stress through better utilization of the underlying network. For Relaxation placement, $80\,\%$ of all physical links use less than $30\,\mathrm{KB/s}$, compared to $46\,\mathrm{KB/s}$ with Consumer placement.

A lower maximum link stress with a particular placement algorithm directly translates into a larger number of circuits that can be handled by the SBON. To illustrate this point, Table 4 shows the maximum number of circuits that can be added to an SBON under a particular placement algorithm when there is a cap on the bandwidth usage of all physical links in the network. If all physical links in the network topology are restricted to $\mathrm{Link}_{\max} = 400\,\mathrm{KB/s}$, Random placement is able to place only $69\,\%$ of the circuits placed by Optimal placement. Relaxation placement achieves a much higher rate of $95\,\%$.

Algorithm	*Num. of circuits*
Optimal	844
IP Multicast	717
Relaxation	799
Producer	675
Consumer	680
Random	585

Table 4: **Number of placed circuits with maximum link stress threshold,** $\mathrm{Link}_{\max} = 400\,\mathrm{KB/s}$**.**

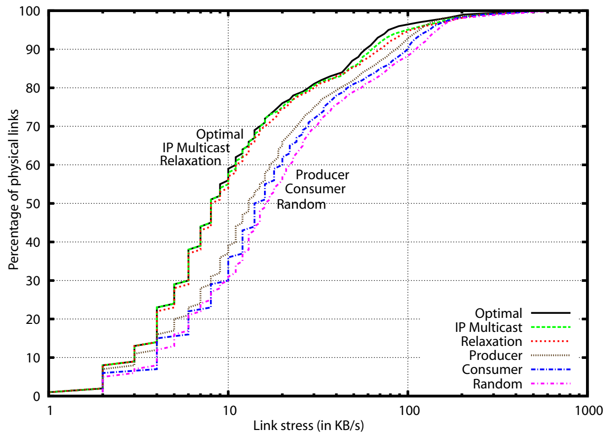


Figure 18: **Distribution of link stress for different placement algorithms.** *Note that the x-axis uses a log scale.*

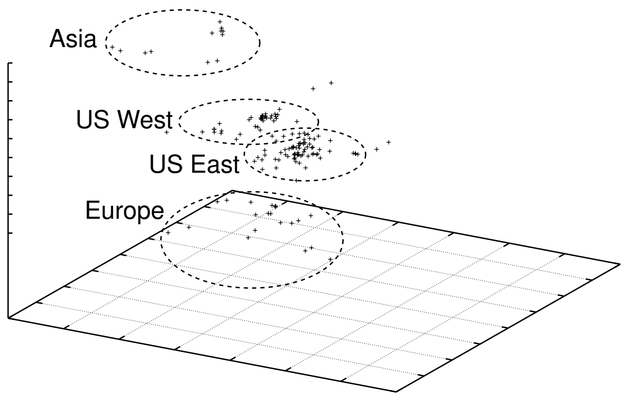


Figure 19: **Visualization of latency space for PlanetLab nodes.**

5.4 PlanetLab Setup

To validate the results obtained from our simulator on a transit-stub topology, we evaluated the different placement algorithms with an SBON deployed on *PlanetLab* [31]. PlanetLab is a global, distributed test-bed that currently encompasses 434 machines located on 5 continents. To give an idea for the PlanetLab topology, the plot in Figure 19 depicts 147 PlanetLab nodes in latency space created from all-pairs ping measurements between nodes [30]. As shown by the annotations, four clusters of nodes can be identified that correspond to nodes located on the US East coast, the US West coast, in Europe, and in Asia. The network diameter for the PlanetLab topology is $2074\,\mathrm{ms}$.

As a recent survey has shown [5], the PlanetLab topology is not an unbiased cross-section of the Internet but rather has an emphasis on well-provisioned, educational networks. Nevertheless, its wide-area topology allows a qualitative verification of our simulator results. In general, efficient network utilization is important on PlanetLab because it is heavily shared among many parties so that efficient path optimization of an SBON deployed on PlanetLab directly benefits other applications. One of the challenges of PlanetLab experiments to evaluate path optimization is that the physical network topology is not known. Although preliminary work to map the PlanetLab exists [23], we needed to perform our own real-time measurements that infered aspects of the PlanetLab topology. Circuits on PlanetLab were

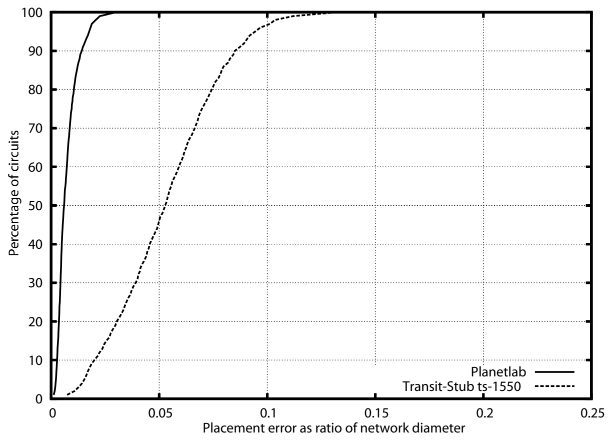


Figure 20: **Distribution of placement error when mapping from latency space to physical space.**

created with a distributed stream-processing infrastructure.[2]

The setup for our PlanetLab experiments was as follows. We used our simulator to build the Vivaldi latency space using all-pairs ping measurements. We then placed 1000 simple circuits off-line according to different placement strategies, with pinned services randomly distributed across the network. The simulator output an XML file, which then controlled the deployment of the stream-processing system on PlanetLab and set up circuits according to the calculated placement. Only one circuit was active in the system at a time. The quality of the placement was evaluated through measurements on Planetlab using *Scriptroute* [28] (for network utilization) and the stream-processing application itself (for delay penalty). We had no restriction on the number of services running on a node, but limited the candidate set of physical nodes to those running ScriptRoute and for which all-pairs ping data was available.

5.5 Planetlab Results

Before comparing Relaxation placement to other placement algorithms on Planetlab, we realized that its performance depends on the error introduced when mapping a placement from latency space to nodes in physical space, as explained in Section 4.3. To compare the suitability of Relaxation placement for the transit-stub and Planetlab topologies, Figure 20 plots the distribution of the *mapping error*. The mapping error is defined as the fraction of the network diameter by which a physical node's network coordinate used for placement differs from the desired coordinate.

The plot shows that the mapping error for the Planetlab topology is smaller than for the transit-stub topology. This is understandable because the transit-stub topology in latency space has large volumes between stub domains that do not contain any nodes. However, even the worst case mapping error for the transit-stub topology is below 13 %. The Planetlab topology is more applicable to Relaxation placement as nodes are more uniformly distributed.

5.5.1 Network Utilization

Bandwidth-Latency Product. The first PlanetLab experiment verifies the simulation results about network utilization in terms of the BW-Lat product. For this, we measured the latency of circuit links between PlanetLab nodes using Scriptroute. The CDF of the measurements is shown in Figure 21. The results reflect our simulated predictions in Figure 9. Relaxation placement achieves a reduction of 19 % in the amount of network traffic for 90 % of the circuits when compared to Producer placement.

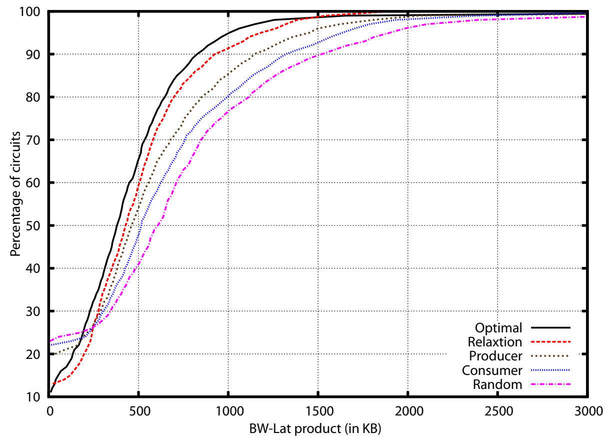


Figure 21: **Distribution of BW-Lat product as measured on PlanetLab.**

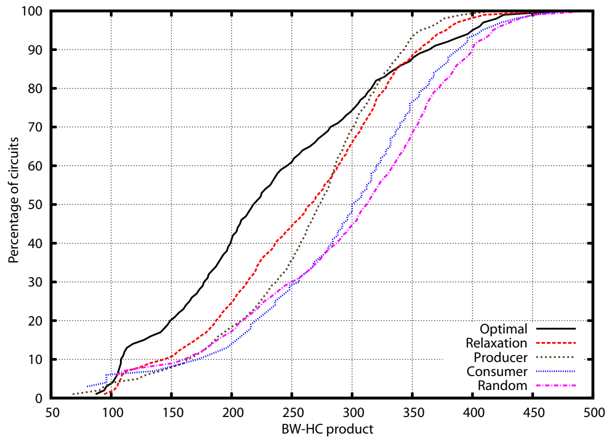


Figure 22: **Distribution of BW-HC product as measured on PlanetLab.**

Bandwidth-Hop Count Product. We also considered the effect of our placement algorithms on the BW-HC product on PlanetLab. To approximate the true physical hop count between two nodes on PlanetLab, we used Scriptroute to collect 16,000 traceroutes between 118 PlanetLab nodes over several days. Although this is an approximation of physical hop count since not all routers are visible at the IP-level and routing paths change, we believe that is a reasonable indicator of true hop count.

The CDF plots for circuits with 6 producers in Figure 22 show that Relaxation placement performs better than Consumer placement. It is interesting to see that Producer placement achieves a lower BW-HC product than simulation. A possible explanation for this is that, since most institutions only host 2 PlanetLab nodes, placing a service on a remote node almost always causes the path to leave the current *autonomous system* (AS). However, routing to another autonomous system incurs a much higher overhead in terms of hop count when compared to intra-AS routing. To verify this, we ran this experiment with 6 instead of 4 producers per circuit, which reduces the efficiency of Producer placement.

[2]Details omitted to preserve anonymity.

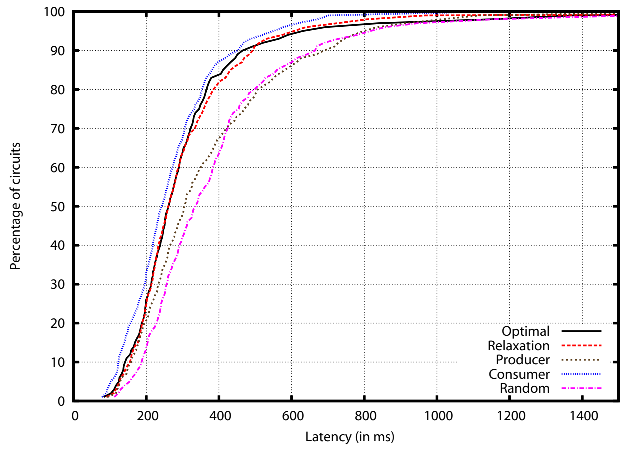


Figure 23: **Distribution of delay stretch as measured on PlanetLab.**

5.5.2 *Delay Penalty*

The final PlanetLab experiment explores the application-observed delay penalty of Relaxation placement. In this experiment, we determine the longest path latency from a consumer service to any of the producers in a circuit. The longest path delay is obtained by a simple stream-processing application, which measures the round-trip time of time-stamped messages through a circuit. The CDF of the average longest path delay can be seen in Figure 23. As expected, Consumer placement results in circuits with lowest average latency. The plot has a very heavy tail; the longest delay, equal to the network delay plus application processing time, is around 6 seconds for Random placement. We also observe that for most circuits, the cost of Relaxation placement, in terms of delay penalty, is not high. In fact, Relaxation placement performs comparably to Consumer placement, and out-performs Random and Producer placements.

5.5.3 *Summary*

Even though the physical topology of PlanetLab is not available, we believe that the results from our PlanetLab measurements confirm the performance evaluation in the simulator with a transit-stub topology. For three cost metrics (BW-Lat, BW-HC, and delay stretch), the evaluated placement algorithms perform as expected. This means that an SBON deployed on PlanetLab that uses Relaxation placement can achieve substantial savings in terms of network utilization, while keeping the application delay penalty low.

6 Related Work

In this section, we give an overview of work related to path optimization in stream-based overlay networks.

Distributed stream processing systems, such as *Medusa* [10] and *Borealis* [1], are faced with the path optimization problem when placing stream operators. Data paths in Medusa are controlled by contracts for load management [4] that take node properties into account but are not network-aware. The work on *network-aware query processing* [3] for Borealis is closest to ours. Here, operators are either placed at the consumer side, at the producer side, or in-network on the DHT routing path, depending on the network bandwidth usage for a query. Applications can also specify delay constraints on the placement path in the DHT. By performing in-network placement in latency space, our approach has more freedom in choosing a good placement node without having the restriction of following DHT routing paths. In addition, Relaxation placement supports dynamic, global path optimization across circuits and considers node and link stress. Other stream processing systems, such as *GATES* [9], avoid the problem of data path optimization by supporting only pre-defined locations with pinned services in the SBON. This leaves the burden of efficient data path selection with the system administrator. The work on *mesh-based XML routing* [27] uses an SBON for content-based routing of XML documents. XML routers create an overlay mesh for streaming data to clients. However, the topology of the mesh does not reflect the structure of individual queries in order to use the underlying network resources efficiently. *DistCED* [22] is a distributed system for pattern detection in message streams. Pattern detectors in the overlay network are placed at distributed nodes according to placement heuristics called distribution policies, minimizing different performance metrics. However, no evaluation of the system is provided.

Distributed databases partition data across multiple nodes. The placement of query operators is then often driven by the location of the data [17]. In *IrisNet* [14], semi-structured data from multiple sensor networks is partitioned hierarchically among nodes. Stream queries contact all nodes that are relevant to the query [13]. Our approach is complementary by installing queries at suitable locations and streaming data to operators. The location of operators (and corresponding relational tables) in *PIER* [15], a distributed database built on top of a DHT, is determined through hashing. Such a random distribution has good load-balancing properties but uses the underlying network resources inefficiently. *Mariposa* [29] is a distributed database system that follows a market-based approach with economic techniques for a decentralized implementation of the query optimization problem. However, global network costs, such as network utilization, are not addressed. More recent efforts on *rate-based query optimization* [32] and intelligent *partitioning of streams* [26] are directly applicable to our work when making local placement decisions for services.

Content distribution networks build an overlay network for efficiently disseminating data to many parties, which requires a network-aware selection for the location of multicast nodes. *Narada* [33] is an application-level multicast (ALM) system, which builds a multicast tree out of an overlay mesh. The authors introduced the issue of network efficiency and defined metrics to quantify resource usage, which are similar to ours. The overlay mesh is optimized dynamically but cannot be optimized at the granularity of a single circuit as in our approach because the mesh is shared among applications. *Overcast* [16] is an ALM scheme, which maximizes total bandwidth for content distribution but does not deal with global network utilization. Publish/subscribe systems, such as *Scribe* [6, 7], are built on top of a DHT and rely on hashing for routing path selection.

7 Future Work

This paper has introduced a novel path placement scheme that is able to optimize an SBON with cross-circuit dependencies. We are currently working on a fully decentralized implementation of Relaxation placement that is deployable on PlanetLab. With this, we will explore cross-circuit optimization, which is supported by our current algorithm, on a global scale. In addition, we plan to investigate how physical network and logical circuit dynamics affect the performance of Relaxation placement. This involves the introduction of a mechanism for service migration, which is aware of the cost of migration.

Our analysis has assumed a fairly homogenous logical circuit structure. We are also interested in finding more representative circuits in existing SBON systems, particularly when there are cross-circuit dependencies. This will allow us to show the advantages of intelligent path optimization for a real stream-based application. We would also like to augment an existing SBON with the Relaxation path placement algorithm to show the wide applicability of this work.

8 Conclusions

In this paper, we have identified the problem of path optimization in stream-based overlay networks and presented a novel, decentralized placement algorithm, which is based on spring relaxation in a virtual latency space. Our placement algorithm is network-aware without imposing unnecessary probing or communication overhead, can adapt to changing network conditions, and supports optimization decisions across circuit boundaries. We have evaluated Relaxation placement in terms of network utilization, application delay, and resource contention in simulation and validated the results with experiments on PlanetLab.

Our results show that network-aware path optimization can result in substantial savings of network resources in overlay networks. Efficient use of resources when deploying stream-based overlay networks on shared networking infrastructures, such as PlanetLab, is essential to achieve scalability and cooperation between different applications. Our results further indicate that efficient network utilization is possible without sacrificing application metrics, such as delay, or creating resource contention in the network. We have also demonstrated the advantages of service placement in the Internet back-bone, which results in considerable improvements in terms of network efficiency.

References

[1] D. Abadi, Y. Ahmad, H. Balakrishnan, et al. The Design of the Borealis Stream Processing Engine. Technical Report CS-04-08, Brown University, July 2004.

[2] D. Abadi, D. Carney, U. Cetintemel, et al. Aurora: A New Model and Architecture for Data Stream Management. *VLDB Journal*, 12(2), Aug. 2003.

[3] Y. Ahmad and U. Çetintemel. Network-Aware Query Processing for Stream-based Applications. In *Proc. of the 30th Int. Conf. on Very Large Data Bases (VLDB'04)*, Toronto, Canada, Aug. 2004.

[4] M. Balazinska, H. Balakrishnan, and M. Stonebraker. Contract-Based Load Management in Federated Distributed Systems. In *Proc. of NSDI'04*, San Francisco, CA, Mar. 2004.

[5] S. Banerjee, T. G. Griffin, and M. Pias. The Interdomain Connectivity of PlanetLab Nodes. In *Proc. of the Passive and Active Measurement Workshop (PAM'04)*, Antibes Juan-les-Pins, France, Apr. 2004.

[6] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. Scribe: A Large-scale and Decentralized Application-level Multicast Infrastructure. *IEEE Journal on Selected Areas in Communication (JSAC)*, 20(8), Oct. 2002.

[7] M. Castro, M. Jones, et al. An Evaluation of Scalable Application-Level Multicast using Peer-to-peer Overlay Networks. In *Proc. of INFOCOM'03*, San Francisco, CA, Feb. 2003.

[8] S. Chandrasekaran, O. Cooper, A. Deshpande, M. Franklin, et al. TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. In *Proc. of the 1st Biennial Conf. on Innovative Data Systems Research (CIDR'03)*, Asilomar, CA, Jan. 2003.

[9] L. Chen, K. Reddy, and G. Agrawal. GATES: A Grid-Based Middleware for Processing Distributed Data Streams. In *Proc. of the 13th Int. Symp. on High-Performance Dist. Comp. (HPDC-13)*, Honolulu, Hawaii, June 2004.

[10] M. Cherniack, H. Balakrishnan, M. Balazinska, et al. Scalable Distributed Stream Processing. In *Proc. of the 1st Conf. on Innovative Data Sys. Research (CIDR'03)*, Asilomar, CA, Jan. 2003.

[11] R. Cox, F. Dabek, F. Kaashoek, et al. Practical, Distributed Network Coordinates. In *Proc. of the 2nd Workshop on Hot Topics in Networks (HotNets-II)*, Cambridge, MA, Nov. 2003.

[12] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A Decentralized Network Coordinate System. In *Proc. of the ACM SIGCOMM'04 Conference*, Portland, OR, Aug. 2004.

[13] A. Deshpande, S. Nath, P. B. Gibbons, and S. Seshan. Cache-and-query for Wide Area Sensor Databases. In *Proc. of the Int. Conference on Management of Data (SIGMOD'03)*, 2003.

[14] P. B. Gibbons, B. Karp, Y. Ke, S. Nath, and S. Seshan. IrisNet: An Architecture for a World-Wide Sensor Web. *IEEE Pervasive Computing*, 2(4), Oct. 2003.

[15] R. Huebsch, J. M. Hellerstein, N. Lanham, et al. Querying the Internet with PIER. In *Proc. of 29th International Conference on Very Large Data Bases (VLDB'03)*, Berlin, Germany, Sept. 2003.

[16] J. Jannotti, D. K. Gifford, K. L. Johnson, et al. Overcast: Reliable Multicasting with an Overlay Network. In *Proc. of the 4th Symposium on Operating Systems Design and Implementation (OSDI'00)*, San Diego, CA, Oct. 2000.

[17] D. Kossmann. The State of the Art in Distributed Query Processing. *ACM Computing Surveys*, 32(4), Dec. 2000.

[18] J. Ledlie, J. Shneidman, M. Welsh, M. Roussopoulos, and M. Seltzer. Open Problems in Data Collection Networks. In *Proc. of the SIGOPS Euro. Workshop*, Leuven, Belgium, Sept. 2004.

[19] L. F. Mackert and G. M. Lohman. R* Optimizer Validation and Performance Evaluation for Distributed Queries. In *Proc. of the 12th International Conference on Very Large Data Bases (VLDB'86)*, Kyoto, Japan, Aug. 1986.

[20] T. E. Ng and H. Zhan. Predicting Internet Network Distance with Coordinates-Based Approaches. In *Proc. of INFOCOM'02*, New York, NY, June 2002.

[21] M. Pias, J. Crowcroft, S. Wilbur, S. Bhatti, and T. Harris. Lighthouses for Scalable Distributed Location. In *Proc. of the 2nd Int. Workshop on P2P Systems (IPTPS'03)*, Berkeley, CA, Feb. 2003.

[22] P. R. Pietzuch, B. Shand, and J. Bacon. Composite Event Detection as a Generic Middleware Extension. *IEEE Network Mag.*, 18(1):44–55, January/February 2004.

[23] A.-M. Popa. An Investigation of Real Internet Topologies and AS Relationships. Master's thesis, University of the Federal Armed Forces, Munich, Germany, 2004.

[24] A. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems. In *Proc. of the 3rd International Conference on Middleware (Middleware'01)*, pages 329–350, Heidelberg, Germany, Nov. 2001.

[25] H. Sagan. *Space-Filling Curves*. Springer-Verlag, 1994.

[26] M. A. Shah, J. M. Hellerstein, S. Chandrasekaran, and M. J. Franklin. Flux: An Adaptive Partitioning Operator for Continuous Query Systems. In *Proc. of the 19th International Conference on Data Engineering (ICDE'03)*, Bangalore, India, Mar. 2003.

[27] A. C. Snoeren, K. Conley, and D. K. Gifford. Mesh-Based Content Routing using XML. In *Proc. of the 18th ACM Symposium on Operating Systems Principles*, Banff, Alberta, Canada, Oct. 2001.

[28] N. Spring, D. Wetherall, and T. Anderson. Scriptroute: A Public Internet Measurement Facility. In *Proc. of the 4th USENIX Symp. on Internet Tech. and Sys. (USITS'02)*, Seattle, WA, Mar. 2003.

[29] M. Stonebraker, P. M. Aoki, W. Litwin, et al. Mariposa: A Wide-Area Distributed Database System. *VLDB Journal*, 5(1), 1996.

[30] J. Stribling. All-Pairs-Pings for PlanetLab. http://www.pdos.lcs.-mit.edu/˜strib/pl_app/, Sept. 2004.

[31] The Planetlab Consortium. http://www.planetlab.org, 2004.

[32] S. D. Viglas and J. F. Naughton. Rate-based Query Optimization for Streaming Information Sources. In *Proc. of the Int. Conference on Management of Data (SIGMOD'02)*, June 2002.

[33] Yang-Hua Chu and Sanjay G. Rao and Hui Zhang. A Case for End System Multicast. In *Proc. of ACM SIGMETRICS'00*, pages 1–12, Santa Clara, CA, June 2000. ACM.

[34] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee. How to Model an Internetwork. In *Proc of IEEE Infocom'96*, volume 2, pages 594–602, San Francisco, CA, Mar. 1996.