BridgeNet: An Adaptive Multi-Source Stream Dissemination Service Overlay

Xiaohui Gu, Zhen Wen, Philip S. Yu
IBM T. J. Watson Research Center, Hawthorne, NY 10532
{xiaohui, zhenwen,psyu}@ us.ibm.com

***Abstract*— Emerging stream processing applications such as on-line data analysis often need to acquire streaming information from geographically dispersed locations (e.g., different sensor networks). Different from conventional discrete data (e.g., messages), streaming data are often *time-varying* and *long-lived*, which provides both new challenges and opportunities for optimizing wide-area continuous information dissemination. In this paper, we present *BridgeNet*, a novel adaptive multi-source stream dissemination overlay network, which can efficiently collect streaming information from distributed locations and disseminate aggregated information to different stream consumers. BridgeNet provides a new *distributed cell tree* structure for multi-source stream aggregations, which can adaptively expand or contract itself in response to workload changes. In particular, BridgeNet performs *stream-pattern-based* cell tree adaptations, stream clustering, and overlay topology adaptations to deliver efficient stream dissemination without losing system stability. For failure resilience, BridgeNet provides light-weight backup schemes to achieve fast failure recovery. We have implemented a prototype of BridgeNet and conducted extensive experiments using both simulations and Planetlab deployment. The experimental results based on both synthetic workload and real data streams show that BridgeNet outperforms existing schemes for efficient multi-source stream dissemination.**

I. Introduction

Many real-world applications require on-line data analysis on continuous *time-varying* data streams, where data arrival rates can dynamically change over time. Examples of such data streams include stock prices, financial trading records, and sensor readings. Previous work has developed core data stream processing systems (e.g., [23], [17], [35]) to provide continuous query processing over dynamic data streams. However, stream sources are often dispersed at different distributed locations such as different sensor networks. Furthermore, applications often need to simultaneously access multiple data streams such as "tracking top ten largest vehicle traffic volumes among 100 major intersections" or "counting the number of servers among 1000 content servers whose access frequencies are larger than 1000 times/second". To fill the gap between distributed stream sources and different stream consumers, a multi-source stream dissemination (MSSD) system is highly desirable, which can (1) relieve stream sources and consumers from the burden of collecting, aggregating and disseminating various data streams over Internet; (2) perform in-network stream aggregation to reduce wide-area network traffic; and (3) avoid redundant aggregation and dissemination operations between a common set of stream sources and consumers. The goal of our research is to explore the design and implementation of the MSSD system, illustrated by Figure 1. Each MSSD session consists of two phases: (1) *aggregation phase* where data streams generated from geographically dispersed sources are merged into a single result stream via an aggregation tree; and (2) *distribution phase* where the result stream is distributed to different stream consumers via a distribution tree.

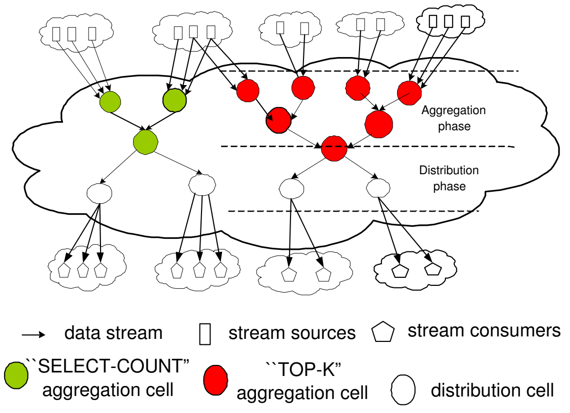

Fig. 1. Multi-source streaming information dissemination.

Previous data dissemination systems such as publish-subscription systems (e.g., SIENA [7], Gryphon [3], Sieve [15], and Kyra [6]) mostly concern about matching published information with subscriptions using selection predicates, and often deal with *discrete* data items such as messages and events. In contrast, our research focuses on the new challenges of delivering time-varying continuous data streams over Internet. First, streaming information are often *time-varying* where data items are continuously produced by different sources with fluctuating rates. This implies that the stream dissemination workload is likely to change during runtime. Thus, any static schemes will be either over-sufficient by wasting system resources, or under-sufficient by failing to meet workload requirements. Luckily, data streams are often *long-lived*, which allows the system to observe stream arrival patterns and perform meaningful adaptations. Second, wide-area stream dissemination needs to scale to a large number of geographically dispersed stream sources and consumers. Thus, the system needs to employ a decentralized and self-managed architecture to achieve scalability and efficiency. Third, data items of different streams often arrive in an asynchronous fashion. This property provides a new optimization opportunity for minimizing the aggregated rate of an aggregation node by clustering streams based on their arrival patterns.

To appear in Proc. of IEEE INFOCOM Mini-Symposium, Alaska, May, 2007(preprint).

In this paper, we present BridgeNet, a novel adaptive multi-source data stream dissemination service overlay. BridgeNet employs a set of *fully distributed*, *stream-pattern-based*, *adaptive* algorithms for disseminating multi-source data streams over Internet. BridgeNet explores the *long-lived* and *time-varying* features of data streams by tracking data arrival patterns of different streams. Based on the knowledge of data arrival patterns, BridgeNet can make informed adaptation decisions for provisioning and maintaining different MSSD service sessions and the underlying overlay mesh. Specifically, this paper makes the following contributions:

- We propose a new *distributed cell tree* structure that can adaptively expand or contract itself to meet dynamic stream processing workload requirements with a minimum aggregation tree. Thus, we can achieve both lower stream dissemination delay and higher system throughput than existing non-adaptive algorithms. The units of a cell tree, called cells, can also migrate from one host to another for reducing dissemination delay or improving load balancing.
- We present a set of *pattern-based* adaptation algorithms to achieve efficient stream dissemination without losing system stability. We apply time series analysis techniques to derive the frequencies of load variations at different aggregation node. Thus, we can perform meaningful adaptations during "stable period" when the workload does not fluctuate at high frequency. Further, we conduct stream clustering based on the correlations among the data arrival patterns of different streams. We strive to connect "complementary" streams with negative-correlated data arrival patterns to the same aggregation node to achieve low-variance aggregated workload.
- For failure resilience, we provide light-weight backup schemes to achieve fast failure recovery. Different from reactive failure recovery, proactive scheme maintains a few backups in advance for reducing failure recovery time for delay-sensitive stream dissemination services.
- We provide dynamic overlay topology maintenance algorithms with the goal of minimizing overlay stretch for current stream dissemination sessions. The basic idea is to make the overlay topology congruent with the connection requirements of current cell trees to avoid overlay-layer relay as much as possible. Thus, the neighbor set of each overlay node is dynamically selected based on the communication patterns of current stream dissemination sessions.
- We have implemented a prototype of the BridgeNet system and conducted extensive experiments using both simulations and wide-area network testbed PlanetLab [27]. The experimental results show the performance advantages of our approach compared to other alternative approaches.

The rest of the paper is organized as follows. Section II presents the system model and problem formulation. Section III presents the design and algorithms details of the BridgeNet system. Section IV presents a thorough experimental evaluation to show the benefits of our approaches. Section V discusses related work. Finally, Section VI concludes this paper.

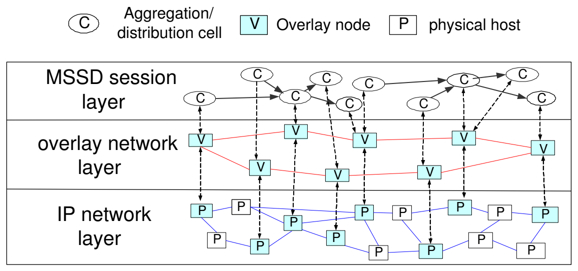

Fig. 2. Architecture of stream dissemination overlay.

II. System Model and Problem Formulation

The multi-source stream dissemination service provides a "bridging" mechanism connecting different stream sources and stream consumers dispersed in the wide-area network. BridgeNet adopts an overlay-based approach to wide-area stream dissemination for failure resilience [1] and quality-of-service (QoS) management [18], illustrated by Figure 2. BridgeNet consists of cooperative overlay nodes (v_i) connected via application-level virtual links (e_i). Different multi-source stream dissemination (MSSD) sessions are dynamically provisioned on top of the shared overlay mesh. Data stream processing (i.e., aggregation or distribution) is often resource-intensive, which can exceed the resource capacity (e.g., network bandwidth, CPU, memory) of a single host. Thus, BridgeNet performs both stream aggregation and stream distribution hierarchically using a set of overlay nodes connected into an aggregation tree[1] or a distribution tree. The aggregation tree and distribution tree are connected via a common root node called *rendezvous point* [5]. The distribution phase is similar to previous content distribution services, which can be delivered using overlay multicast trees (e.g., [20], [9]). In contrast, the aggregation tree is connected with multiple, distributed, time-varying stream sources, which demands stream-oriented, adaptive schemes to construct the aggregation tree[2].

The aggregation phase performs continuous data stream aggregation using an application-specified n-way continuous aggregation function[3] $f(S_1, ...S_n)$. The goal of the aggregation phase is to perform data summarization and filtering to reduce the stream dissemination workload. Many aggregation function has such load reduction capability, such as (1) TOP-K function that keeps track of k hot-spot content servers with highest access frequencies; and (2) SELECT-COUNT function that first selects the values of distributed sensor readings according to a select predicate and then calculates the mean of those selected values. We use S_i to denote a data stream that consists of a sequence of data items denoted by $s_i \in S_i$. In reality, many

[1]We assume that the aggregation function can be decomposed into a set of smaller aggregation functions, each of which only processes a subset of input streams. This is generally true for most common aggregation functions such as TOP-K, MEAN, UNION, SUM, and COUNT.

[2]Although this paper focuses on the aggregation phase, the scheme can also be applied to the distribution phase if necessary.

[3]In this paper, we limit ourselves to the case of performing a uniform multi-way aggregation function in each MSSD session.

data streams (e.g., sensor readings, stock trading records) are time-varying, where data can arrive in a bursty fashion. Thus, the aggregation workload can dynamically change over time. Any static aggregation would be either over-sufficient wasting resources or under-sufficient causing degraded service quality. Thus, we need to dynamically adapt the aggregation tree based on the data arrival patterns of different stream sources. We use data arrival time series $A_i, 1 \leq i \leq k$ to describe the arrival pattern of the stream S_i, which is called the *signature* of S_i. The data arrival time series consist of a sequence of time-stamped number $a_{i,k} \in A_i$ that denotes the number of data items produced by S_i during the k-th time epoch. We maintain a moving window of data arrival time series $A_i = \{a_{i,1}, ..., a_{i,w}\}$ to represent the current signature of the stream S_i.

Each overlay node can provide stream aggregation and distribution processing functions for multiple stream dissemination sessions according to its resource capacity. The stream aggregation/distribution functions are performed by the aggregation/distribution cells (C_i). All the stream processing cells share the resources of one physical host proportionally. Each aggregation cell has multiple input ports for receiving input streams and one output port for generating the result streams. The aggregation computation is performed periodically at the end of each time epoch. Let $S_i[t, t+1]$ denote the set of stream items arrived during the time period $[t, t+1)$. The aggregation cell performs the aggregation function over all the non-empty stream set[4] (i.e., $S_i[t, t+1] \neq \emptyset$). In contrast, each distribution cell has one input port for receiving the aggregated stream. The distribution cell simply replicates each input into multiple copies that are sent to different down-stream cells or end-users.

Each overlay node v_i is connected with a number of other overlay nodes called neighbors via overlay links. Each overlay link e_i is mapped into a network path in the underlying IP network. The number of neighbors to which an overlay node is connected is called the degree of the overlay node. Logically, if there is a data stream flowing from v_i to v_j, v_j should be the neighbor of v_i in the overlay mesh. For QoS management, each overlay node needs to periodically monitor the conditions of its neighbors and adjacent overlay links such as the load conditions of its neighbors and the network delay/bandwidth of its adjacent overlay links. However, for scalability, each overlay node can only select a limited number of nodes as its neighbors since a fully connected mesh can incur excessive maintenance overhead. When two overlay nodes (e.g., v_2 and v_3 in Figure 2) are not directly connected, the stream transmission between them has to go through an overlay path consisting of multiple overlay links. Thus, an overlay mesh with node degree constraint can cause degraded network QoS (e.g., longer delay) than the original IP path, which is called *overlay stretch* [20].

BridgeNet aims at achieving the following design goals to meet the new challenges of disseminating *time-varying streaming information* over Internet: (1) *scalability* where the system must scale well in the presence of many stream sources and stream consumers; (2) *timeliness* where data streams should be delivered to all stream consumers in a timely fashion (i.e., with minimum dissemination delay[5]; and (3) *failure resilience* where the system should be able to quickly recover the failures of overlay nodes or links to provide continuous stream disseminations. Essentially, BridgeNet addresses the following major problems to achieve the above design objectives:

Problem 1: How to maintain a minimum aggregation tree[6] *to meet the dynamic workload requirements of an MSSD session?*

Problem 2: How to place aggregation cells on different overlay nodes to achieve best dissemination delay and load balancing?

Problem 3: When to trigger system adaptations to achieve efficiency without losing system stability?

Problem 4: How to quickly recover the failure of an MSSD session with minimum service disruption?

Problem 5: How to efficiently adapt the overlay topology to achieve minimum overlay stretch for current MSSD sessions?

III. Design and Algorithms

In this section, we present the design and algorithm details of the BridgeNet system including (1) pattern-based cell adaptation triggering algorithm; (2) complementary stream clustering algorithm for efficiently distributing workload among different cells; (3) decentralized cell tree adaptation algorithms; (4) failure resilience management schemes; and (5) dynamic overlay mesh topology configuration algorithm.

A. Stable Cell Tree Adaptation Triggering

At the beginning of a session, the cell tree contains only the root cell that is instantiated on the rendezvous host[7]. During runtime, the cell tree can dynamically expand or contract itself via cell splitting or merging to adapt to stream processing workload changes. One major challenge is to achieve good tradeoff between adaptability and stability. If the system responds to every workload fluctuation, the cell tree may be frequently expanded and contracted back and forth, which makes the system highly unstable. To address the problem, we propose a *pattern-based adaptation triggering* (PAT) algorithm to achieve stable cell tree adaptations by observing the workload variation patterns. A cell adaptation action is only triggered during "stable period" when the workload does not fluctuate at high frequency. We use a moving window of time series $L_i = \{l_{i,0}, ..., l_{i,N-1}\}$ to denote the time-varying load levels of a cell C_i, where the value of

[4]If there are multiple data items in $S_i[t, t+1]$, the aggregation cell first perform a merge operation (e.g., union or mean) over all the data items arrived in the stream buffer based on the definition of the aggregation function.

[5]The dissemination delay includes both stream processing delay and network transmission delay.

[6]By minimum, we mean the aggregation tree consists of a minimum number of aggregation cells, which can lead to minimum resource consumptions and lowest dissemination delay.

[7]The rendezvous host is selected as follows: We first create a set of cells to connect with all stream sources, which then concurrently run a multicast tree algorithm to get the average delay of the paths to all the stream consumers. The root of the multicast tree with minimum average delay is selected as the rendezvous host. The goal of the above selection algorithm is employ a best multicast tree for the distribution phase.

```
Procedure: Complementary-Stream-Clustering (CSC)
input: $m$ points $p_1, ...p_m$
output: two clusters $N_1$, $N_2$
1. Select two distant points $p_{c_1}$ and $p_{c_2}$ as two initial centroids
2. while changes of centroids are larger than a certain threshold
3.     for all the other points $p_k, 1 \leq k \leq m, k \neq c_1, c_2$
4.         calculate distance $d(p_k, N_1) = cor(A_k, \sum_{p_z \in N_1} A_z) + 1$
5.         calculate distance $d(p_k, N_2) = cor(A_k, \sum_{p_z \in N_2} A_z) + 1$
6.         insert $p_k$ to the cluster with smaller distance
7.     calculate new centroids $p'_{c_t} = \sum_{p_z \in N_t} A_z / |N_t|, t = 1, 2$
```

Fig. 3. Complementary stream clustering.

$l_{i,k}$ denotes the k-th sampled load value[8]. Our load variation evaluation is based on Discrete Fourier Transform (DFT). The Fourier transform represents the original signal (i.e., load time series) as a linear combination of the complex sinusoids $s_f(n) = \frac{e^{2\pi ikn/N}}{\sqrt{N}}, i = \sqrt{-1}$. The DFT of a load time series $\{l_{i,0}, ...l_{i,N-1}\}$ consists of a vector of complex numbers:

$$L(f_k) = \sum_{n=0}^{N-1} l_{i,n} e^{-i2\pi kn/N}, 0 \leq k \leq N-1, \quad (1)$$

where $f_k = 2\pi k/N$ denotes the k-th frequency. Thus, the Fourier coefficients represent the amplitude of each of these sinusoids. We can then identify the dominant frequencies by calculating the signal power at each frequency. We evaluate the fluctuation degree of load time series by examining whether its dominant Fourier coefficients (i.e., the top-k frequencies that carry most of the signal energy) fall into high frequency range. Thus, the cell adaptation is triggered only when the dominant frequencies of load time series are below a certain frequency threshold. For example, the PAT algorithm can dampen adaptation triggering when the cell workload frequently changes around the threshold value.

B. Complementary Stream Clustering

We propose a *complementary stream clustering* (CSC) algorithm to group different input streams based on their data arrival patterns. Using maintained stream data arrival time series $A_i = \{a_{i,1}, ..., a_{i,w}\}$, we can calculate arrival rate variance $var(A_i)$ of a single stream S_i, and arrival rate covariance $cov(A_i, A_j)$ between two streams S_i and S_j as follows,

$$var(A_i) = \frac{1}{w}\sum_{k=1}^{w}(a_{i,k} - \frac{1}{w}\sum_{k=1}^{w} a_{i,k})^2 \quad (2)$$

$$cov(A_i, A_j) = \frac{1}{w}\sum_{k=1}^{w} a_{i,k}a_{j,k} - (\frac{1}{w}\sum_{k=1}^{w} a_{i,k})(\frac{1}{w}\sum_{k=1}^{w} a_{j,k}) \quad (3)$$

Based on the signatures of the two streams S_i and S_j, we calculate the statistical data arrival correlation $cor(A_i, A_j)$ between S_i and S_j as follows,

$$cor(A_i, A_j) = \frac{cov(A_i, A_j)}{\sqrt{var(A_i)}\sqrt{var(A_j)}} \quad (4)$$

The value of $cor(A_i, A_j)$ is in the range of $[-1, 1]$. The arrival patterns of two streams S_1 and S_2 are called "complementary" if A_1 and A_2 have negative correlation (i.e., $cor(A_i, A_j)$ is close to -1). We strive to connect complementary streams to the same aggregation cell to achieve smooth aggregated workload with minimum variance. When a splitting action is triggered, the cell needs to decide which input streams to keep and which input streams to offload to the new cell. We perform the stream clustering using a modified K-means (K=2) clustering algorithm [14]. Suppose the cell C_i has m input streams $S_1, ..., S_m$. We can construct a weighted graph G_c where each node p_i represents a stream S_i with signature A_i and the value of an edge denotes the distance between two nodes. To group negative correlated streams into one cluster, we define the distance between two nodes as $d(p_i, p_j) = cor(A_i, A_j) + 1$. Thus, $d(p_i, p_j) = 0$ when A_i and A_j are negative correlated and $d(p_i, p_j) = 2$ when A_i and A_j are positive correlated. Figure 3 shows the pseudo-code of the CSC algorithm that includes the following steps: (1) randomly select one point p_i to represent the centroid of one cluster and then select the other point p_j that has the largest distance from p_i to represent the centroid of the other cluster; (2) assign all the other points to the closest cluster[9]; (3) calculate the mean values of the two clusters to represent their new centroid points; (4) repeat step (2) and (3) until the change of centroid values is smaller than a pre-defined threshold.

C. Cell Tree Adaptation Algorithms

The cell tree can be dynamically adjusted during a session in three ways: (1) cell tree expansion via cell splitting; (2) cell tree contraction via cell merging; and (3) cell tree self-optimization via cell migration. Figure 4 shows the pseudo-code of the major cell tree adaptation algorithms.

1) Cell Tree Expansion: The workload of an MSSD session can dynamically increase when stream sources become more active by producing data at higher rates. In response, the cell tree expands itself via cell splitting to utilize more hosts, illustrated by Figure 5. When the PAT algorithm triggers a cell to split (i.e., dominant load fluctuation frequencies are lower than certain threshold and the mean load $\overline{L}_i = \sum_{k=0}^{N-1} l_{i,k}/N \geq \Theta$, where Θ denotes the load constraint for C_i.), we group input streams of C_i into two clusters using the CSC algorithm. One group of streams remains connected to C_i while the other group is connected to the new cell. If the overloaded cell C_i is not the root (e.g., C_3 in Figure 5), it splits itself into two cells, one of which remains on the current host v_i and the other cell is instantiated on one of the neighbors of v_i. The new cell becomes a sibling of C_i, which is also connected to the parent of C_i. If the overloaded cell is the root (e.g., C_0 in Figure 5), the splitting process consists of two steps. First, the overloaded root cell creates a new cell C_j and transfer all of

[8]The load metric is a configurable parameter, which can be bandwidth/CPU/memory requirements, or a composite metric combining different resource cost.

[9]Note that our distance calculation is a bit different from the traditional k-mean algorithm since we need to calculate the correlation between the current stream and the aggregated workload of all streams in one cluster. If a point has the same distance to both clusters, we select the cluster with the smaller size.

```
Procedure: Cell-Tree-Expansion
1. while $\exists C_i$ with input $\mathcal{S} = \{S_1, ...S_n\}$ is overloaded
2.  if $C_i$ is the root cell
3.      create two new cells $C_k$ and $C_m$
4.      split $\mathcal{S}$ into two groups $\mathcal{S}_1$ and $\mathcal{S}_2$ using CSC algorithm
5.      connect $\mathcal{S}_1$ to $C_k$ and $\mathcal{S}_2$ to $C_m$
6.      make $C_k$ and $C_m$ two children of $C_i$
7.  else if $C_i$ is not the root cell
8.      create one new cell $C_k$
9.      split $\mathcal{S}$ into two groups $\mathcal{S}_1$ and $\mathcal{S}_2$
10.     $\mathcal{S}_1$ remain connected to $C_i$ and $\mathcal{S}_2$ is connected to $C_k$
11.     make $C_k$ the sibling of $C_i$

Procedure: Cell-Tree-Contraction
1. while $\exists$ $C_i$ with parent $C_p$ is underloaded
2.  for $\forall$ siblings of $C_i$, select $C_j$ that
3.      can handle the combined workload (Equ. 5)
4.      and is mostly complementary to $C_i$ (Equ. 6)
5.  merge $C_i$ and $C_j$ into one cell $C_k$
6.  if $C_p$ has only one child $C_k$
7.      merge $C_p$ and $C_k$

Procedure: Cell-Migration
1. for $\forall$ neighbor host $v_k$ of $C_i$
2.  if $v_k$ can accommodate $C_i$
3.      calculate reward value $\mathcal{R}(v_k)$ (Equ. 7)
4. select neighbor $v_b$ with the largest reward
5. create a new cell $C_i'$ on $v_b$
6. replace $C_i$ with $C_i'$ in the cell tree
```

Fig. 4. Distributed cell tree adaptation algorithms.

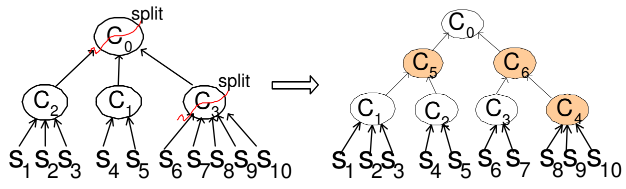
split
C_0
C_2
C_1
C_3
split
S_1 S_2 S_3 S_4 S_5 S_6 S_7 S_8 S_9 S_{10}
C_0
C_5
C_6
C_1
C_2
C_3
C_4
S_1 S_2 S_3 S_4 S_5 S_6 S_7 S_8 S_9 S_{10}

Fig. 5. Distributed cell tree expansion.

its children to C_j. This new cell is instantiated on the least-loaded neighbor of the current host and becomes the only child of the root cell. Second, the new cell C_j checks the local resource availability and executes the non-root splitting algorithm if the input workload from its children exceeds its processing capacity. All the new cells created by the splitting operations become the children of the root cell. The cell splitting algorithm can be executed in a recursive manner. The newly spawned sibling cell can still be overloaded by the allocated workload. Thus, the new sibling cell needs to continue to split itself until the workload of each new sibling cell meets its processing capacity. The sibling cell splitting can also trigger the parent cell to split since the number of its children increases. The parent cell can split itself to generate its new siblings for sharing the workload, which will then trigger the grandparent cell to split. When the splitting process propagates to the root cell, the height of the cell tree will be increased by one.

2) Cell Tree Contraction: When input streams become less active (i.e., slower data arrivals), the cell tree can dynamically contract itself via cell merging to reduce resource consumption and improve service quality (e.g., smaller delay). When the PAT algorithm triggers a cell to split (i.e., dominant load fluctuation frequencies are lower than certain threshold and mean load $\overline{L}_i \leq \Theta/2$, where Θ denotes the load constraint for C_i.), we need to select one best sibling cell to merge with C_i according to their load time series. Let $L_i = \{l_{i,0}, ..., l_{i,N-1}\}$ and $L_j = \{l_{j,0}, ..., l_{j,N-1}\}$ denote the load time series of the cell C_i and its sibling C_j. We first examine whether the combined workload of C_i and C_j is within the load constraint of C_j, denoted by Θ_j. A cell C_j is said to be a combinable cell for C_i if the following condition holds:

$$\frac{1}{N}\sum_{k=0}^{N-1}(l_{i,k} + l_{j,k}) < \Theta_j \tag{5}$$

Among all the combinable sibling cells, we select the one whose input stream arrivals are most complementary to C_i for low-variance aggregated workload. Suppose the cell C_i has k input streams whose signatures are denoted by $A_{i,1}, ..., A_{i,k}$ and C_j has m input streams with data arrival signatures $A_{j,1}, ..., A_{j,m}$. We calculate the correlation between C_i and C_j as follows :

$$cor(C_i, C_j) = \frac{1}{km} \cdot \sum_{x=1}^{k}\sum_{y=1}^{m} cor(A_{i,x}, A_{j,y}) \tag{6}$$

The sibling cell that has the smallest correlation value is selected as the best complementary cell to merge with. The merging of C_i into C_j is performed by connecting the children of C_i with C_j, and then deleting C_i from the cell tree. Similar to the cell splitting process, cell merging can also be recursive. First, the merged cell can still be under-loaded, which can continue trigger the merging operation. Second, the merging process can trigger the parent cell to merge since the number of its children decreases. The parent cell can then merge itself with its sibling cells, which may trigger the grandparent cell to merge. When the merging process propagates to the root cell, all children of the root cell merge with each other into one cell. The only child of the root is then merged into the root cell, which reduces the height of the cell tree by one.

3) Cell Migration: BridgeNet supports runtime cell migration to continuously optimize the performance of the cell tree. We can migrate a cell C_i from a host v_i to one of the neighbors v_j using different criteria such as (a) shorter dissemination delay; and (b) lower workload. Different criteria can lead to different host comparison results. BridgeNet allows upper-level applications to prioritize different criteria for customized decision-making. For illustration, let us assume that criteria (a) has higher priority than (b). Each cell C_i periodically probes its neighbor hosts to decide whether migration should be triggered. Let us assume C_i is located at v_i that has k neighbors $v_{n_1}, ..., v_{n_k}$. In the probing message, the cell C_i sends the addresses of its parent cell v_p and children cells $v_{c_1}, ..., v_{c_z}$ to each of the neighbor hosts $v_{n_j}, 1 \leq j \leq k$, and asks v_{n_j} to calculate a reward function $R(v_{n_j})$ quantifying its change to the MSSD dissemination delay[10] if we migrate C_i from v_i to v_{n_j}. Let $D(v_i, v_j)$ denote the dissemination delay from v_i to v_j and N_t denote the number of cells included in the subtree with root $v_{c_t}, 1 \leq t \leq z$. Note that the link

[10]The dissemination delay of an MSSD session is defined as the average stream dissemination delay from all stream sources to all consumers.

$v_{c_t} \rightarrow v_i$ or $v_{c_t} \rightarrow v_{n_j}$ is used by $\sum_{t=1}^{z} N_t + 1$ paths and the link $v_i \rightarrow v_p$ or $v_{n_j} \rightarrow v_p$ is used by $(\sum_{t=1}^{z} N_t + 1)$ paths in the aggregation tree.

$$\mathcal{R}(v_{n_j}) = (D(v_i, v_p) - D(v_{n_j}, v_p)) \cdot (\sum_{t=1}^{z} N_t + 1) + \\ (D(v_{c_t}, v_i) - D(v_{c_t}, v_{n_j})) \cdot \sum_{t=1}^{z} \cdot N_t, 1 \leq j \leq k \quad (7)$$

The cell C_i selects the best neighbor that has the largest positive reward value and enough resources for hosting C_i. If there are multiple neighbors with similar reward value, C_i selects the least-loaded one. For stability, cell migration is triggered only when the neighbor host significantly outperforms the current host. To achieve smooth migration, we first create a new cell C_i' on the selected neighbor host and connects C_i' to the parent and children of C_i in the cell tree. In the meantime, we still use C_i to serve the current MSSD session. When C_i' finishes the setup, the children of C_i is notified to send their output streams to C_i'. The old cell C_i is then deleted.

D. Failure Resilience Management

BridgeNet performs proactive replication-based failure recovery to tolerate fail-stop failures. Different from reactive failure recovery, proactive scheme maintains a number of backups in advance for reducing failure recovery time. Each cell on the cell tree, called the primary, maintains a number of backup replicas on different hosts, called the buddy list. The locations of the replicas can be decided based on different pre-defined policies (e.g., using neighbor hosts for localized replica maintenance or using remote hosts for tolerating region failures). During runtime, the actual data streams are not sent to all backups. Instead, the primary only sends periodical measurement probes to its replicas for monitoring their liveness and performance. If any replica becomes unavailable or unqualified, the primary cell finds another host to create a new replica. When replicas stop receiving the heartbeat messages (i.e., the periodical probes) from the primary, they assume that the primary fails. Replicas then execute an election algorithm to reach a consensus on which replica should take over based on a pre-defined election criteria (e.g., smallest host identifier). The elected replica then contacts the parent and the children of the failed primary cell that are told to drop the connection to the failed primary and connect to the new primary cell. The number of replicas represents the trade-off between failure resilience and replication overhead. However, the higher-level cells in the cell tree are more important than the lower-level cells since they are responsible for aggregating the output streams of those lower-level cells. Thus, we adopt a differentiated replication scheme to maintain more replicas for higher-level cells in the cell tree.

We now briefly describe how BridgeNet handles system churns (i.e., dynamic node departures/arrivals). When a peer wants to join BridgeNet, it is first incorporated into the overlay mesh by an out-of-band bootstrap mechanism [20]. The peer then selects a few hosts provided by the bootstrap service as neighbors and also requests a few other nodes to add itself as a neighbor. After the peer successfully joins the overlay mesh, it can be selected to instantiate aggregation cell, distribution cell or backup cell. When a peer v_i leaves the system without pre-notice (i.e., crash/disconnection), the system first needs to repair the overlay mesh and updates membership lists on other live peers. The system can repair the partitioned mesh by adding more overlay links at partitioned peers [20]. If v_i also provides a primary cell C_i, the departure of v_i will trigger dynamic failure recovery to repair the cell tree with a replica of C_i. If v_i only acts as a backup for a primary mixer C_i, the departure v_i will cause C_i to create a new backup cell.

E. Overlay Topology Adaptation

The goal of our overlay topology adaptation algorithm is to minimize the overall overlay stretch for current MSSD sessions under the overlay node degree constraint, which is briefly described as follows: We define that a host v_j is "stream-bounded" with the host v_i if the connection from v_i to v_j is used by at least one cell tree. We define the bounding degree of the host v_j to v_i as the number of the MSSD sessions that include the connection $v_i \rightarrow v_j$ as their tree links[11]. Suppose each overlay node v_i can have at most d out-bound and in-bound neighbors. The overlay node v_i keeps track of the MSSD sessions that have streams flowing into and out of it. The host v_i maintains a set of hosts $V = \{v_1, ...v_m\}$ that have the largest bounding-degrees to v_i. For example, let us consider a host v_j that is not included in the neighbor set of v_i. If v_j has a higher bounding degree than one of the existing neighbors of v_i, and v_j can accept an extra inbound neighbor, v_j is added into the out-bound neighbors of v_i if the out-bound neighbor set of v_i is not full. Otherwise, v_j replaces an existing neighbor v_k of v_i that has the lowest bounding degree with v_i. The host v_k then deletes v_i from its inbound neighbor set. The intuition behind our approach is that the overlay mesh topology should be congruent with the topologies of current cell trees to achieve minimum overlay stretch. In other words, if $v_i \rightarrow v_j$ frequently appears in current cell trees, v_i and v_j should be direct neighbors in the overlay mesh.

IV. EXPERIMENT EVALUATION

We have implemented a prototype of the BridgeNet system and evaluated its performance on both simulation testbed and PlanetLab Internet testbed [27] using a range of synthetic stream workloads and real sensor data streams [12].

A. Simulation Results

In the simulation testbed, all BridgeNet algorithms are fully implemented. Only underlying network is simulated to enable easy control. The simulator performs packet-level, discrete-event network simulation emulating packet routing and fine-grained resource allocation in the overlay network. The simulator uses the Internet topology generator Inet-3.0 [31] to generate a 5120 node power-law graph to represent the IP network, and then randomly selects [200,1000] nodes

[11]The bounding degree can also be weighted by the bandwidth requirement of each MSSD session for the connection from v_i to v_j.

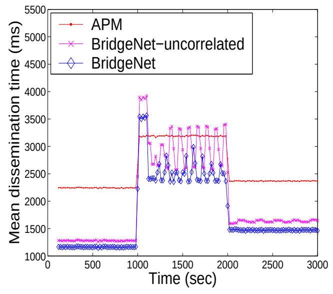

Fig. 6. Dissemination delay comparison under time-varying streams.

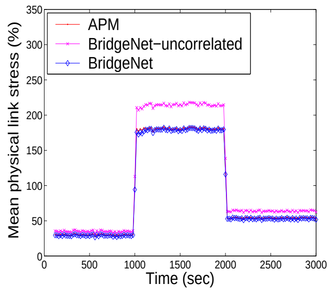

Fig. 7. Link stress comparison under time-varying streams.

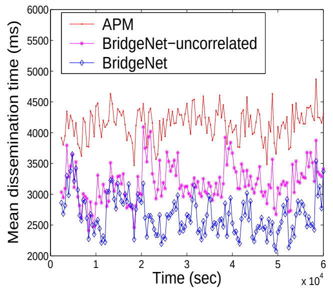

Fig. 8. Dissemination delay comparison under real sensor data streams.

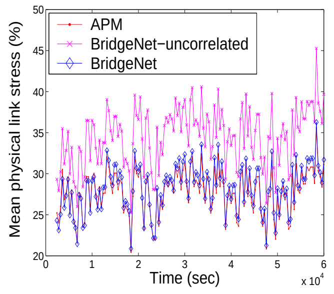

Fig. 9. Link stress comparison under real sensor data streams.

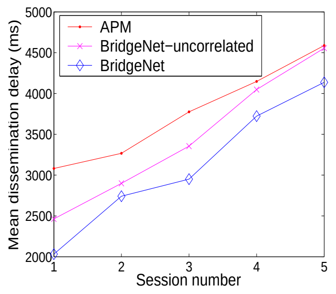

Fig. 10. Performance comparison under concurrent sessions.

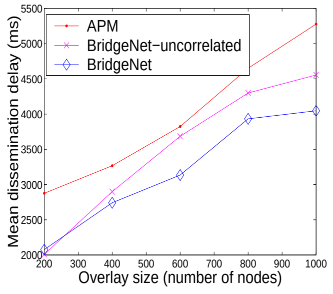

Fig. 11. Performance comparison under different overlay sizes.

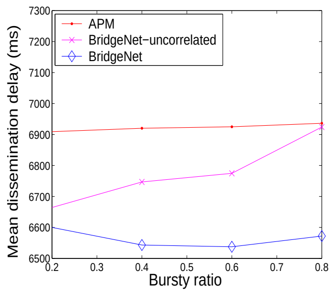

Fig. 12. Performance comparison under different stream burstiness.

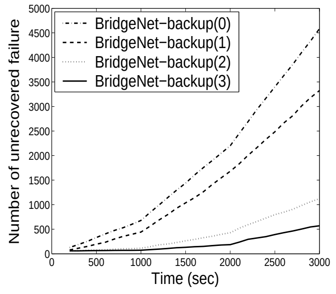

Fig. 13. Backup failure recovery under system churn.

as overlay nodes[12]. Each overlay node is connected to [3, 5] other nodes as neighbors to form the overlay mesh. Our simulator can simulate queueing delay at overlay nodes but not the IP network queueing delay. The resource and performance of overlay nodes and network links are uniformly distributed within certain range to reflect system heterogeneity. We compare our algorithm with several existing schemes: (1) *aggregation plus multicast (APM)* approach uses existing tree construction algorithm [20], [9] that does not perform adaptations in response to stream workload changes; and (2) *BridgeNet-uncorrelated* algorithm that performs cell tree adaptations but does not consider correlations among different stream arrival patterns.

We first compare the performance of different algorithms using a synthetic time-varying stream workload. Each stream source produces a fluctuating stream where its mean rate periodically alternates between a high rate and a low rate. Within each period, the inter-arrival time follows an exponential distribution with a mean set to the current rate. We simulate different inter-stream correlations by randomly assigning a start phase (i.e., high rate or low rate) to each stream. Each simulation run lasts 3000 seconds. We start two MSSD sessions in the system, each of which includes 50 stream sources and stream consumers whose locations are randomly distributed in the system. Each simulation run starts from a light workload where the high rate and low rate are 20kps and 5kbps, respectively. At time 1000, we increase the workload by raising the high/low rates to 100/50 kbps, and at time 2000, we decrease the workload by setting the high/low rates to 30/15kbps. Figure 6 shows the mean stream dissemination delay achieved by different algorithms under the above workload. BridgeNet outperforms APM by as much as 50% by employing adaptive cell trees. BridgeNet performs better than its uncorrelated version, which shows correlation-aware stream clustering is effective, especially under heavy workload. Figure 7 shows the mean physical link stress results. Each measurement is averaged over the physical links used by the stream dissemination sessions. The results show that BridgeNet does not increase physical link stress by employing smaller trees, and the correlation-aware stream clustering can also reduce link stress under heavy workload condition. We then repeat the above experiments with real sensor data streams [12], illustrated by Figure 8 and Figure 9. Each sensor data item carries a time-stamp, along with a set of measurement values such as humidity, temperature, light and voltage values. Each stream source reproduces sensor readings with varying rates based on the time-stamps recorded in the trace files. Again, the results show that BridgeNet consistently achieves much better performance than other alternatives because of its pattern-based adaptation and optimization capabilities.

We then compare the performance of different algorithms under increasing number of concurrent sessions, shown by Figure 10. This experiment uses the synthetic dynamic streams with the mean high/low rates set as 100/50kbps. The values are measured when the system reaches its steady state. We observe that BridgeNet can employ a small cell tree under light workload condition to reduce the dissemination delay by 33% compared to APM. Under heavy workload, BridgeNet can adaptively increase the cell tree to utilize more overlay

[12]Because the simulator performs detailed packet-level emulations for resource-intensive data stream dissemination, 1000-node stream overlay is currently the largest-scale that can be executed within reasonable time on our server host with 3G HZ CPU and 1G RAM.

system size	200	400	600	800	1000
algorithm overhead (MB)	0.7	1.6	2.4	3.3	3.9
stream data (MB)	238	478	718	957	1197

TABLE I
BRIDGENET ALGORITHM OVERHEAD.

nodes. We now test our algorithms under different system sizes, illustrated by Figure 11. We gradually increase the system size from 200 nodes to 1000 nodes. We start two MSSD sessions on the system, each of which has 40 stream sources and stream consumers whose locations are randomly distributed. Given a constant node degree, larger system sizes can have bigger overlay stretch due to an increasing number of hops between every two nodes. We observe that BridgeNet still consistently performs the best, especially under large-scale overlay systems with more optimization opportunities. We now evaluate the effect of stream burstiness on the performance of our algorithms. We define a bursty ratio metric $\theta, 0 \leq \theta \leq 1$. The high rate and low rate are calculated by $(1+\theta)r_i$ and $(1-\theta)r_i$, respectively. The larger the bursty ratio, the more fluctuating the stream is. As expected, the correlation-based complementary stream clustering algorithm is most effective under highly bursty stream workloads.

We now evaluate the backup failure recovery schemes of BridgeNet under system churn where a number of peers dynamically leave or join the system, illustrated by Figure 13. The algorithm "BridgeNet-backup(k)" means that we maintain on average k backup cells for each primary. The system randomly selects a number of departure nodes every 20 seconds according to a specified churn rate. During each 3000-second simulation run, we start from a low-churning system with 10% churn rate δ (i.e., 10% of total system nodes randomly leave the system[13], then increase the churn rate to 20% at time 1000, and further increase the churn rate to 30% of all nodes at time 2000. The system repairs overlay mesh partition by randomly adding neighbors to the peers with few neighbors left. The Y-axis of Figure 13 shows the accumulated number of failures that cannot be recovered by the maintained backups. We observe that the system can withstand high system churn by just maintaining a few number of backup cells.

Finally, Table I shows some algorithm overhead measurements of the BridgeNet system under different sizes of overlay networks, which mainly includes overlay neighbor monitoring overhead, cell tree adaptation overhead, and backup maintenance overhead. BridgeNet generally has very low overhead since it only requires localized information to perform fully distributed adaptation and optimization algorithms.

B. Planetlab Results

To evaluate the feasibility and performance of our approach under real Internet environment, we have implemented an initial prototype of the BridgeNet system and tested it on the Planetlab Internet testbed [27]. The BridgeNet prototype is a multi-threaded distributed software system written in about 20K lines of Java code. Our experiments used about

[13]Some nodes will be dynamically added back to the system to keep the number of live nodes in the system at a constant level of $(1-\delta)\cdot N$.

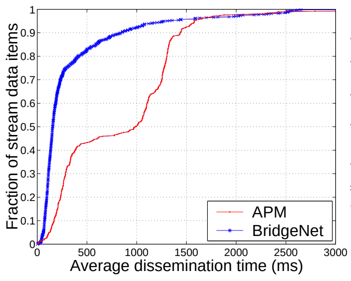

Fig. 14. Performance comparison on the Planetlab.

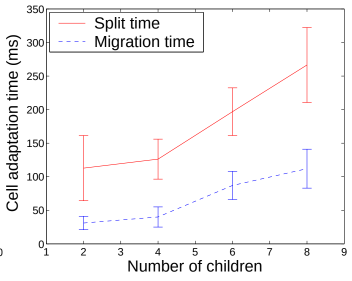

Fig. 15. Cell tree adaptation time on the Planetlab.

50 PlanetLab hosts that spread across US. In the experiments, half of nodes are configured as stream sources and the other half of nodes are configured as stream consumers. Figure 14 shows the cumulative distribution of data dissemination time between all pairs of stream sources and stream consumers. The dissemination time measured on PlanetLab includes the stream processing and queueing delays at both overlay hosts and Internet connections. In Figure 14, we observe that the cumulative distribution of data dissemination time of the BridgeNet approach stays left of APM distribution, which means a larger proportion of data dissemination time is lower than that of the APM's dissemination time. We also measured the distributed cell tree adaptation time on the PlanetLab, shown by Figure 15. We first measured the time of migrating an internal cell tree node from one Planetlab host to another host. The migration time includes the time to select the best neighbor as the new hosting place for the cell, and the time to modify the distributed cell tree structure using a set of tree update messages. For example, all the children of the current cell will be notified with its new location and send confirmation messages back to the current cell when they complete the update. Thus, the migration time grows with the children number of the migrated cell. Figure 15 shows the mean and standard deviation time for migrating an internal cell with 2 to 8 children cells from one host to a neighbor host in the overlay network. The splitting operation is similar to the migration operation with an extra time of computing the stream partitions and sending more cell tree update messages. Generally, the cell tree adaptation time is about tens to hundreds of milli-seconds, which is acceptable by long-lived stream dissemination services.

V. RELATED WORK

Our work is related to previous content-based publish/subscription (pub/sub) systems such as SIENA [7], Gryphon [3], Sieve [15], and Kyra [6]. Recent work has explored the problem of providing peer-to-peer pub/sub services (e.g., [36]). Other recent work has extended pub/sub systems with composite event subscriptions [28], [25], [10]. The pub/sub systems mostly concern about matching published information with subscriptions using selection predicates, and often deal with *discrete* data items such as messages and events. The Bistro [11] system addresses the problem of efficient data (file) collection by calculating an optimal data transfer schedule. In contrast, our work focuses on the problem of efficiently aggregating and disseminating long-lived, time-varying data streams from multiple given data sources that match users' interest. Those data sources can be discovered using previous matching algorithms.

Featured by flexibility and easy deployment, various overlay networks have been proposed for content distributions, such as resilient routing overlay [1], multicast overlays (e.g., [20], [9], [4], [24]) and reliable/high bandwidth stream overlays [30], [22], [8]. A flurry of research work (e.g., [32], [34], [13]) has devoted to studying the problem of constructing efficient overlay topologies. Zhuang et al. studied the failure detection problem in the overlay networks [37]. BridgeNet is orthogonal to the above work, and can benefit from previous schemes to provide wide-area MSSD services.

Distributed data stream processing has recently drawn much research attention [2], [33], [21], [19]. However, most of them focuses on distributed query processing instead of data stream dissemination. Some recent work [29], [26], [16] has addressed the stream dissemination problem, which mostly addresses efficient content-based filtering for distributing single-source streams. In contrast, our work focuses on the problem of *multi-source* stream dissemination and provides *stream-pattern-based adaptations* to both dissemination trees and underlying overlay topology.

VI. Conclusion

In this paper, we have presented BridgeNet, a novel self-adaptive multi-source data stream dissemination overlay system. The design of the BridgeNet system centers around the two unique features of data stream dissemination: (1) data streams are *time-varying*, where stream sources can produce data items at fluctuating rates; and (2) data streams are *long-lived*, which allows the system to collect meaningful stream arrival patterns for efficient and stable adaptations. To the best of our knowledge, this is the first work that studied adaptive dissemination of time-varying data streams collected from multiple distributed locations. Specifically, this paper makes the following contributions: (1) a new distributed cell tree structure that can adaptively contract or expand itself in response to workload changes; (2) stream-pattern-based adaptation algorithms for both cell trees and overlay topology; (3) light-weight backup schemes to achieve failure-resilient stream dissemination. We have implemented a prototype of the BridgeNet system that is evaluated using both extensive simulation testbed and PlanetLab wide-area network testbed. The experimental results based on both synthetic workloads and real sensor data streams show that BridgeNet can always achieve better performance than existing schemes under time-varying stream workloads. The prototype implementation on the PlanetLab shows the feasibility of our approach where the cell tree adaptations can be quickly performed in wide-area networks.

References

[1] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient Overlay Networks. *In SOSP*, 2001.

[2] M. Balazinska, H. Balakrishnan, and M. Stonebraker. Contract-Based Load Management in Federated Distributed Systems. *In NSDI*, 2004.

[3] G. Banavar and et al. An Efficient Multicast Protocol for Content-Based Publish-Subscribe Systems. *In ICDCS*, 1999.

[4] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. *Proc. of ACM SIGCOMM, Pittsburgh, PA*, 2002.

[5] L. F. Cabrera, M. B. Jones, and M. Theimer. Herald: Achieving a Global Event Notification Service. *In HotOS-VIII*, 2001.

[6] F. Cao and J. P. Singh. Efficient Event Routing in Content-based Publish-Subscribe Service Networks. *In Infocom*, 2004.

[7] A. Carzaniga and A. Wolf. Forwarding in a content-based network. *In SIGCOMM*, 2003.

[8] M. Castro, P. Druschel, A-M Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: High-Bandwdith Multicast in Cooperative Environments. *In ACM SOSP*, 2003.

[9] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *JSAC, vol. 20(8)*, 2002.

[10] W. Chen, Z. Ge, J. Kurose, and D. Towsley. Optimizing Event Distribution in Publish/Subscribe Systems in the Presence of Policy-Constraints and Composite Events. *In ICNP*, 2005.

[11] W. C. Cheng, C.-F. Chou, L. Golubchik, S. Khuller, and Y.-C. Wan. Large-scale Data Collection: a Coordinated Approach. *In Infocom*, 2003.

[12] A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong. Model-Driven Data Acquisition in Sensor Networks. *In VLDB*, 2004.

[13] J. Fan and M. H. Ammar. Dynamic Topology Configuration in Service Overlay Networks: A Study of Reconfiguration Policies. *In Infocom*, 2006.

[14] K. Fukunaga. Introduction to Statistical Pattern Recognition. *Academic Press*, 1990.

[15] S. Ganguly, S. Bhatnagar, A. Saxena, S. Banerjee, and R. Izmailov. A Fast Content-based Data Distribution Infrastructure. *In Infocom*, 2006.

[16] B. Gedik and L. Liu. Quality-Aware Distributed Data Delivery for Continuous Query Services. *In SIGMOD*, 2006.

[17] The STREAM Group. STREAM: The Stanford Stream Data Manager. *IEEE Data Engineering Bulletin, 26(1):19-26*, 2003.

[18] X. Gu, K. Nahrstedt, R. Chang, and C. Ward. QoS-Assured Service Composition in Managed Service Overlay Networks. *In ICDCS*, 2003.

[19] X. Gu, P. S. Yu, and K. Nahrstedt. Optimal Component Composition for Scalable Stream Processing. *In ICDCS*, 2005.

[20] Y. h. Chu, S. G. Rao, S. Seshan, and H. Zhang. Enabling Conferencing Applications on the Internet using an Overlay Multicast Architecture. *In ACM SIGCOMM*, 2001.

[21] R. Huebsch, B. Chun, J. M. Hellerstein, B. T. Loo, P. Maniatis, T. Roscoe, S. Shenker, I. Stoica, and A. R. Yumerefendi. The architecture of PIER: an internet-scale query processor. *In CIDR*, 2005.

[22] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh. *In ACM SOSP*, 2003.

[23] S. Krishnamurthy and et al. TelegraphCQ: An Architectural Status Report. *IEEE Data Engineering Bulletin, 26(1):11-18*, 2003.

[24] G.-I. Kwon and J. W. Byers. Reliable Overlay Multicast with Loosely Coupled TCP Connections. *In Infocom*, 2004.

[25] G. Li and H.-A. Jacobsen. Composite Subscriptions in Content-based Publish/Subscribe Systems. *In Middleware*, 2005.

[26] O. Papaemmanouil and U. Cetintemel. Semcast: Semantic multicast for content-based data dissemination. *In ICDE*, 2005.

[27] L. Peterson, T. Anderson, D. Culler, and T. Roscoet. A Blueprint for Introducing Disruptive Change in the Internet. *In SIGCOMM HotNets Workshop*, 2002.

[28] P. Pietzuch, B. Shand, and J. Bacon. A Framework for Event Composition in Distributed Systems. *In Middleware*, 2003.

[29] S. S. Shyamshankar and D. K. Ramamritham. An Efficient and Resilient Approach to Filtering and Disseminating Streaming Data. *In VLDB*, 2003.

[30] A. C. Snoeren, K. Conley, and D. K. Gifford. Mesh-Based Content Routing using XML. *In SOSP*, 2001.

[31] J. Winick and S. Jamin. Inet3.0: Internet Topology Generator. *Tech Report UM-CSE-TR-456-02 (http://irl.eecs.umich.edu/jamin/)*, 2002.

[32] R. H. Wouhaybi and A. T. Campbell. Phenix: Supporting Resilient Low-Diameter Peer-to-Peer Topologies. *In Infocom*, 2004.

[33] Y. Xing, S. B. Zdonik, and J.-H. Hwang. Dynamic Load Distribution in the Borealis Stream Processor. *In ICDE*, 2005.

[34] A. Young, J. Chen, Z. Ma, A. Krishnamurthy, L. Peterson, and R. Y. Wang. Overlay Mesh Construction Using Interleaved Spanning Trees. *In Infocom*, 2004.

[35] S. Zdonik and et al. The Aurora and Medusa Projects. *IEEE Data Engineering Bulletin, 26(1)*, 2003.

[36] C. Zhang, A. Krishnamurthy, and R. Wang. Combining Flexibility and Scalability in a Peer-to-Peer Publish/Subscribe System. *In Middleware*, 2005.

[37] S. Q. Zhuang, D. Geels, I. Stoica, and R. H. Katz. On Failure Detection Algorithms in Overlay Networks. *In Infocom*, 2005.