Quality-Aware Distributed Data Delivery for Continuous Query Services

Buğra Gedik
CERCS, College of Computing
Georgia Institute of Technology
Atlanta, GA, USA
bgedik@cc.gatech.edu

Ling Liu
CERCS, College of Computing
Georgia Institute of Technology
Atlanta, GA, USA
lingliu@cc.gatech.edu

ABSTRACT

We consider the problem of distributed continuous data delivery services in an overlay network of heterogeneous nodes. Each node in the system can be a source for any number of data streams and at the same time be a consumer node that is receiving streams sourced at other nodes. A consumer node may define a filter on a source stream such that only the desired portion of the stream is delivered, minimizing the amount of unnecessary bandwidth consumption. By heterogeneous, we mean that nodes not only may have varying network bandwidths and computing resources but also different interests in terms of the filters and the rates of the data streams they are interested in. Our objective is to construct an efficient stream delivery network in which nodes cooperate in forwarding data streams in the presence of constrained resources. We formalize this distributed stream delivery problem as an optimization one by starting with a simple setup where the network topology is fixed and the node bandwidth characteristics are known. The goal of the optimization is to find valid delivery graphs with minimum bandwidth consumption. We extend this problem formulation to QoS-aware stream delivery, in order to handle the bandwidth constrained cases in which unwanted drops and delays are inevitable. We provide a classification of delivery graph construction schemes, and in light of this classification we develop pragmatic quality-aware stream delivery (QASD) algorithms. These algorithms aim at constructing efficient stream delivery graphs in a distributed setting, where global knowledge is not available and network characteristics are not known in advance. We introduce a set of evaluation metrics and provide experimental results to illustrate the effectiveness of our proposed algorithms under these metrics.

1. INTRODUCTION

Data streams, that are time ordered series of events or readings, are becoming increasingly common in today's data processing tasks. This is fostered by the proliferation of continuously changing on-line information sources, which in turn is boosted by the advances in web services, global communications, and sensing technologies. Continuous query (CQ) systems [22, 10, 16] that can evaluate standing queries over data streams have created a new paradigm in data management. The resulting Data Stream Management Systems (DSMSs) [1, 3, 9, 20] are expected to parallel, in the data stream domain, the success of traditional DBMSs in managing stored data. Examples of data streams include stock tickers in financial services, link statistics in networking and telecommunications, sensor network readings in environmental monitoring and emergency response, and satellite and live experimental data in scientific computing.

Recent advances in peer-to-peer (P2P) and Grid computing have stimulated more research on distributed CQ systems in which both data sourcing and processing are distributed among the nodes, such as GridDB [15], PIER [12], and StreamGlobe [14]. In these kinds of systems, a data stream sourced at one node is usually of interest to a number of other nodes that are the consumers of the stream. Supply of data streams to all interested nodes in a distributed CQ system constitutes the *distributed stream delivery problem*. Nodes in the system can source any number of data streams, and each stream has an associated schema. For example, a node can be the source for a scientific data stream from a satellite or a sensor data stream from a local sensor network deployment. A consumer node can subscribe to multiple streams and the source nodes that are producing these streams are assumed to be known to the consumer node (i.e., the discovery of data sources is a separate problem and is beyond the focus of this paper). A consumer node interested in receiving a stream may define a filter on the source stream so that only the desired portion of the stream is received and the bandwidth consumption is minimized.

To understand the critical factors that affect the performance of such a distributed data delivery network, we consider two extreme scenarios: (1) In a centralized stream delivery solution, each node can simply receive the stream directly from the source node. This solution, however, results in putting too much responsibility on the source node and may consume all of the bandwidth available to it. (2) On the other hand, having a sequential chain based solution in which a node forwards the stream it receives to a single other node, does not work either. The latter approach will cause high overall bandwidth consumption when the nodes are interested in non-overlapping parts of the streams. This analysis tells us that the communication bandwidth is a critical factor. As a result, we are interested in designing a stream delivery system in which all nodes receive their interested streams (possibly through multiple hops) and at the same time the solution is efficient in terms of the overall bandwidth consumption.

Building an optimal stream delivery configuration may not always be possible, mainly due to the insufficient network bandwidth resources, which is usually a consequence of high stream rates, large number of consumers, or low level of overlap in filter specifications. In such cases unwanted *drops* and *delays* are inevitable. Delays in stream delivery are introduced due to communication and processing delays. In general, larger number of forwarding steps tends to increase the delay. Drops are introduced due to insufficient bandwidth. For instance, a node may be forced to forward data at a rate higher than it can forward based on the available bandwidth, in which case the excessive load is shed by dropping tuples randomly from the stream.

SIGMOD 2006 June 27-29, 2006, Chicago, Illinois, USA
Copyright 2006 ACM 1-59593-256-9/06/0006 ...$5.00.

We assume that the nodes do not buffer the streams for the benefit of downstream nodes. This is a reasonable assumption since such buffering will only defer the time at which the drops will occur and does not change the fact that drops are inevitable when the resources are not sufficient to provide complete delivery of streams to all nodes. Nevertheless, in most CQ applications both drops and delays are tolerable [21, 2]. This analysis tells us that the application level quality of service (QoS) is another critical factor in designing effective stream delivery networks.

Figure 1 depicts a partitioning of the application space based on tolerance with respect to delays and drops. At one extreme, there are applications that require both timely and complete data delivery, such as real-time intrusion detection. On the other extreme, there are applications that can tolerate both delays and drops in data delivery, such as archiving stream statistics, e.g. storing average hourly precipitation data from a stream of per-minute precipitation readings. As a result, an effective stream delivery system should take into account the requirements of applications in terms of delay and drop. In this work, we capture these requirements by using quality-of-service (QoS) functions that are attached to filters. These functions are specified by the nodes when their filters are created, according to the requirements of the applications running on the nodes.

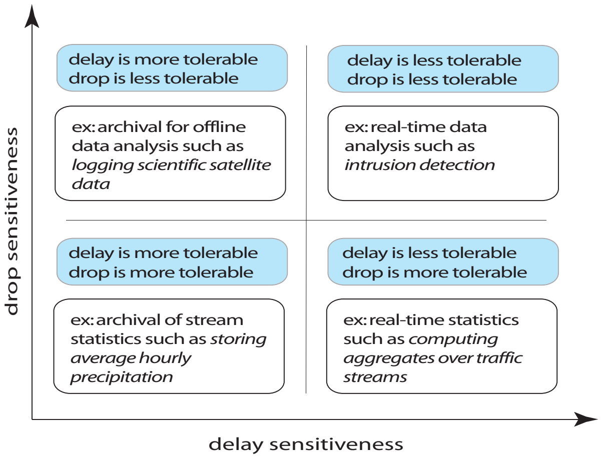

delay is more tolerable
drop is less tolerable
ex: archival for offline data analysis such as *logging scientific satellite data*
delay is less tolerable
drop is less tolerable
ex: real-time data analysis such as *intrusion detection*
delay is more tolerable
drop is more tolerable
ex: archival of stream statistics such as *storing average hourly precipitation*
delay is less tolerable
drop is more tolerable
ex: real-time statistics such as *computing aggregates over traffic streams*
drop sensitiveness
delay sensitiveness

Figure 1: Stream delivery applications

There are several challenges in building an effective stream delivery graph. First, we should be able to exploit the similarities in different filters in order to create delivery paths that are optimized in terms of grouping similar filters, thus having minimum bandwidth consumption. Second, we should reduce the amount of drops in stream delivery by taking into account the stream rates, the tuple sizes of filter applied streams, and the relation of these factors to the bandwidth resources of the nodes. Third, we should integrate different quality of service requirements into the process of constructing the distributed data delivery network, so that filters with less stringent QoS requirements can be located on resource constrained regions of the delivery graph. Last but not the least, we need efficient and practical algorithms that are on-line, distributed, and work in the absence of pre-knowledge about the resource availability of the nodes. This paper makes three major contributions toward addressing these challenges:

− We introduce important concepts related with filters, streams, and links, and use these concepts to formally define valid stream delivery graphs. This definition leads to the formulation of distributed stream delivery as an optimization problem, where the aim is to find valid delivery graphs with minimum bandwidth consumption. This formulation can be used to construct optimal stream delivery graphs for systems with fixed topology and known bandwidth characteristics.

− We extend our problem formulation to QoS-aware stream delivery in order to handle the bandwidth constrained cases where unwanted drops and delays are inevitable. This includes introducing per-filter QoS functions, formulating delay and drop rates for given delivery graph configurations, and modifying the definition of efficient delivery graphs to integrate QoS characteristics of the filters.

− We provide a classification of delivery graph construction schemes; and in light of this classification, we develop pragmatic quality-aware stream delivery (QASD) algorithms to construct efficient stream delivery graphs. These algorithms make no assumptions about the pre-knowledge of node bandwidth characteristics. We introduce a set of evaluation metrics and study the effectiveness of our algorithms under these metrics through simulation-based experiments.

2. DISTRIBUTED STREAM DELIVERY

In this section, we give an overview of our solution, describe the fundamentals of distributed stream delivery, and introduce the notation used.

2.1 Overview

At the core of the stream delivery problem, lies the assumption that the nodes cooperate in forwarding streams. However, a node will only receive and forward streams for which it has a filter registered. As a result, a node can receive a stream from either the source node, or from some other node that is already receiving the stream. Moreover, the stream can already be filtered when it is received by a node, with the condition that the filter registered by the node is *applicable over* the filters applied to the stream previously, i.e. the portion of the stream that the node is interested in receiving is a subset of what the node is receiving. This concept of filter applicability, together with the concept of *amortized size* which captures the bandwidth requirement of streams after a set of filters are applied on them, form a basis to build an effective stream delivery graph. They facilitate assessing the feasibility of a given delivery graph, as well as its effectiveness in terms of bandwidth consumption. Using these concepts, we formalize the problem as an optimization one in Section 3.

When the resources are insufficient to provide complete and timely stream delivery to every node, solutions which can gracefully degrade are needed. In order to support this, we attach a QoS function to each filter. This function specifies the quality of the stream delivery for its associated filter for given drop and delay values. With these functions in place, the aim is to maximize the overall quality of stream delivery which is defined as an aggregate of the QoS specifications of the filters. We formalize the QoS-aware stream delivery problem in Section 4.

In practice, stream delivery graph construction algorithms are online and do not usually have access to knowledge about bandwidth resources of the nodes. In Section 5, we provide simple yet effective algorithms for constructing stream delivery graphs. We describe *feedback-based* algorithms that can learn resource availability of nodes on-the-fly as the delivery graph is constructed, as well as less complex algorithms that do not use feedback but still generate delivery graphs that are better than the naïve approaches.

2.2 Fundamentals and Notation

Nodes and Streams: We denote the set of nodes by P, defined as $\{p_i : 1 \leq i \leq |P|\}$, where p_i is the ith node. We denote the set of streams by S, defined as $\bigcup_{i=1}^{|P|} S_i$. Here S_i denotes the set formed by the streams that have node p_i as their *source*, defined as $S_i = \{s_i^j : 1 \leq j \leq |S_i|\}$, where s_i^j is the jth stream sourced at node p_i. Each stream is a time ordered (possibly unbounded) set of tuples of equal size, where size of a tuple in stream s_i^j is denoted by $\varsigma(s_i^j)$. Streams can have different rates and we denote the rate of stream s_i^j by $\lambda(s_i^j)$.

Filters: A node can register at most one filter on a stream. This filter is used to specify the interested portion of the stream. Filters can specify selection and projection operations over the tuples of a stream. We denote the set of filters by Q, defined as $Q = \bigcup_{s_i^j \in S} Q_i^j$. Here Q_i^j denotes the set of filters registered on stream s_i^j and is defined as $\{q_{i,k}^j : p_k \in P$ has a filter registered on $s_i^j\}$. In other words, $q_{i,k}^j$ is the filter registered by p_k on s_i^j. For notational convenience, in the rest of the paper, we will threat a set of filters registered on the same stream also as a filter. Concretely, a set $Q' \subset Q_i^j$ is a filter with the following properties: Its selection operator is a disjunction of the selection operators of the individual filters and its projection operator is a union of the projection operators of the individual filters. An empty set of filters $\{\}$ is assumed to suppress all tuples from a stream.

Filter Applicability: We say that a filter $Q' \subset Q_i^j$ is *applicable over* another filter $Q'' \subset Q_i^j$, denoted as $Q' \sqsubset Q''$, if and only if the selection and projection operators of Q' are as restrictive as or more restrictive than Q'''s selection and projection operators, respectively. Concretely, $Q' \sqsubset Q''$ means that if a tuple passes the filter Q', then it must also pass Q'' such that the resulting filtered tuple of Q'' contains at least the same set of attributes present in the filtered tuple of Q'.

Amortized Size: Size of a tuple after applying a filter $Q' \subset Q_i^j$ is denoted by $\varsigma(Q')$, whereas the selectivity (the fraction of tuples that pass the filter) of the filter is denoted by $\sigma(Q')$. Then we define the *amortized size* of a tuple after applying filter Q' as $\sigma(Q') \cdot \varsigma(Q')$, denoted by $\Phi(Q')$. It can be thought of as the size of a tuple after applying a filter, divided by (amortized over) the average number of tuples we need to push through the filter to pass one tuple.

Links: We denote the set of communication links between the nodes by L, defined as $\{l_{i,j} : p_i, p_j \in P, p_i$ and p_j are connected$\}$. We denote the capacity (bandwidth) of a link $l_{i,j}$ by $\kappa(l_{i,j})$. Although this way of modeling the connections between the nodes is appropriate for small scale systems with explicit knowledge of the *physical* links between the nodes, for large scale P2P overlays where any two node can connect to each other, it is more meaningful to model upload and download bandwidths of nodes. In this latter case there exists a *logical* link $l_{i,j}$ between any two nodes p_i and p_j. When the logical link model is considered, we use $\kappa^{\uparrow}(p_i)$ and $\kappa^{\downarrow}(p_i)$ to denote the upload and download bandwidths of node p_i and do not use link bandwidths for expressing bandwidth requirements. The formalizations in the rest of the paper apply to both physical and logical link models, except when stated otherwise.

Notation	Meaning
p_i	ith node
s_i^j	jth stream sourced at p_i
$\lambda(s_i^j)$	rate of stream s_i^j
$q_{i,k}^j$	filter of p_k on s_i^j
$Q' \sqsubset Q''$	Q' is applicable over Q''
$\Phi(Q' \in Q_i^j)$	amortized size of a Q' applied s_i^j tuple
$l_{i,j}$	physical or logical link between p_i and p_j
$\kappa^{\uparrow}(p_i), \kappa^{\downarrow}(p_i)$	upload and download bandwidths of node p_i
${}_{u,v}Q_i^j$	set of filters applied on s_i^j while being forwarded from p_u to p_v
${}_{u,v}x_{i,k}^j$	binary variable denoting whether $q_{i,k}^j$ is applicable over ${}_{u,v}Q_i^j$ or not
$\mathcal{C}_{i,j}^k$	QoS function for $q_{i,k}^j$
$r_{i,k}^j, d_{i,k}^j$	drop and delay experienced by $q_{i,k}^j$

Table 1: Some important notations used throughout the paper

3. BASIC PROBLEM FORMALIZATION

To formulate the distributed data delivery problem we have outlined so far, we first consider the class of network of nodes over which a valid stream delivery graph can be found. In the next section we will relax this assumption and extend the problem formulation to allow limited communication bandwidth. We list some of the important notations used throughout the paper in Table 1.

Constructing a stream delivery graph requires defining which nodes should forward which streams to which other nodes by applying which filters. To formalize this assignment problem, we define a set of binary variables and four rules on them. These rules should hold for a particular assignment of these variables so that the resulting delivery graph is valid.

Filter assignment variables: We define a binary variable ${}_{u,v}x_{i,k}^j$ if and only if $l_{u,v} \in L$ and $q_{i,u}^j, q_{i,v}^j, q_{i,k}^j \in Q_i^j$. Simply stated, we define ${}_{u,v}x_{i,k}^j$ if and only if nodes p_u and p_v are connected by a link and nodes p_u, p_v, p_k all have filters registered on s_i^j. When the variable ${}_{u,v}x_{i,k}^j$ takes the value of 1, it means that $q_{i,k}^j$ is applicable over the filter that is applied to the stream sent from p_u to p_v over the link $l_{u,v}$. Once the filter assignment variables are set, the filter applied to s_i^j when it is being sent from p_u to p_v over the link $l_{u,v}$ can be computed as $\{q_{i,k}^j : {}_{u,v}x_{i,k}^j = 1\}$, denoted by ${}_{u,v}Q_i^j$.

3.1 Valid Delivery Graphs

Certain conditions about filter assignment variables have to be satisfied, so that we have a *valid* stream delivery graph. Below, we list these four conditions.

(1) *Filter Satisfaction Condition* states that for a filter $q_{i,k}^j$ registered by node p_k on stream s_i^j, there should exist a link $l_{u,k}$ connecting p_u to node p_k such that $q_{i,k}^j$ is applicable over the set of filters applied to s_i^j while it is being forwarded to p_k from p_u. It also states that there should not be any other link on which p_k receives s_i^j. In other words, receiving a stream from multiple nodes and performing synchronization is not considered. Formally:

$$\forall q_{i,k}^j \in Q, \exists l_{u,k} \in L \text{ s.t.}$$
$$\Big(q_{i,k}^j \sqsubset {}_{u,k}Q_i^j \;\wedge\; \neg\exists \underset{v \neq u}{l_{v,k}} \in L \text{ s.t. } {}_{v,k}Q_i^j \neq \emptyset\Big)$$

(2) *Stream Forwarding Condition* states that, if a stream s_i^j is being forwarded on a link $l_{u,v}$ from node p_u to p_v, then either p_u is the source of the stream ($u = i$) or there exists a node p_w that is forwarding the stream to p_u over the link $l_{w,u}$ such that the set of filters applied on the stream while it is being forwarded to p_v from p_u is applicable over the set of filters applied on the stream while it is being forwarded to p_u from p_w. Moreover, p_v should have a filter $q_{i,v}^j$ registered on s_i^j. Formally:

$$\forall s_i^j \in S,\ \forall l_{u,v} \in L,\ \ {}_{u,v}Q_i^j \neq \emptyset \rightarrow$$
$$\Big(\big(u = i \,\vee\, \exists l_{w,u} \text{ s.t. } {}_{u,v}Q_i^j \sqsubset {}_{w,u}Q_i^j\big) \;\wedge\; q_{i,v}^j \in Q_i^j\Big)$$

(3) *Bandwidth Requirement Condition* for the physical model simply states that the bandwidth consumed on a link due to stream delivery should not exceed the link's capacity. Formally:

$$\forall \underset{u<v}{l_{u,v}} \in L, \kappa(l_{u,v}) \geq \sum_{s_i^j \in S} \Big(\lambda(s_i^j) \cdot \big(\Phi({}_{u,v}Q_i^j) + \Phi({}_{v,u}Q_i^j)\big)\Big)$$

Bandwidth requirement for the logical model simply states that the total bandwidth consumed by a node due to forwarding streams should not exceed the node's upload capacity, whereas the total bandwidth consumed due to receiving streams should not exceed the node's download capacity. Formally:

$$\forall p_u \in P,\ \kappa(p_u^{\downarrow}) \geq \sum_{l_{v,u}\in L}\ \sum_{s_i^j \in S} \Big(\lambda(s_i^j)\cdot \Phi({}_{v,u}Q_i^j)\Big)$$

$$\forall p_u \in P,\ \kappa(p_u^{\uparrow}) \geq \sum_{l_{u,v}\in L}\ \sum_{s_i^j \in S} \Big(\lambda(s_i^j)\cdot \Phi({}_{u,v}Q_i^j)\Big)$$

(4) *No Loop Condition* states that the setting of the filter assignment variables should not result in a loop in the delivery of streams, ensuring that filters are served from the source of a stream after possibly multiple forwarding steps:

$$\forall s_i^j \in S,\ \forall P' \subset \{p_k \in P : q_{i,k}^j \in Q\},$$
$$|\{p_u \in P' : \exists p_v \in P \text{ s.t. } {}_{v,u}Q_i^j \neq \emptyset\}| < |P'|$$

3.2 Effective Delivery Graphs

To define what we mean by an effective delivery graph, we introduce a cost function. Our cost function is the total bandwidth consumed in the network for delivering streams to all nodes. This can be calculated by summing up for each node and for each link the bandwidth consumed on that link for forwarding the stream, i.e., the rate of the stream times the amortized size of the stream after the filters associated with the link are applied. Thus, the aim is to find a valid delivery graph with the minimal cost. Formally, the objective is:

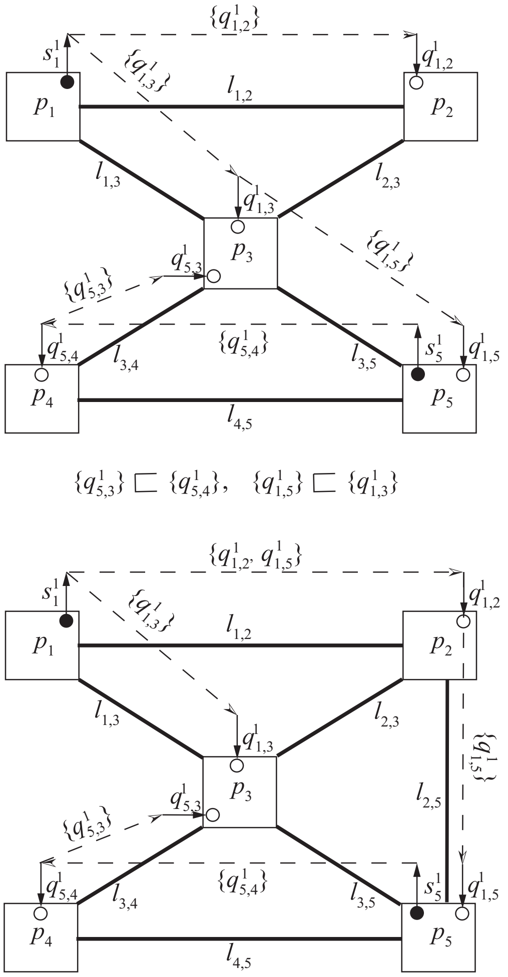
$$min \sum_{s_i^j\in S}\ \sum_{l_{u,v}\in L} \Big(\lambda(s_i^j)\cdot \Phi({}_{u,v}Q_i^j)\Big)$$

In effect, we have converted the problem into an assignment problem, where the aim is to assign values to binary variables ${}_{u,v}x_{i,k}^j$, such that the resulting assignment satisfies the above listed conditions and the cost function defined above is maximized. This problem can be shown to be an integer programming (IP) problem, although the conversion of the listed conditions to linear constraints is not straightforward. IP is known to be an NP-hard problem [17].

Example Delivery Graphs

Figure 2 shows two example stream delivery graphs. In the examples there are two streams, s_1^1 and s_5^1, and 5 filters that are $q_{5,3}^1$, $q_{5,4}^1$, $q_{1,2}^1$, $q_{1,3}^1$, and $q_{1,5}^1$. We have $q_{5,3}^1 \sqsubset q_{5,4}^1$ and $q_{1,5}^1 \sqsubset q_{1,3}^1$. In the first example, $q_{5,4}^1$ applied s_5^1 is served directly from p_5 to p_4, where $q_{5,3}^1$ applied s_5^1 is forwarded to p_3 from p_4, since $q_{5,3}^1$ is applicable over $q_{5,4}^1$. Note that this is better than serving $q_{5,3}^1$ applied s_5^1 to p_3 directly from p_5, because the link $l_{3,5}$ is already used to forward $q_{1,5}^1$ to p_5 from p_3 and in case we have $\kappa(l_{3,5}) < \lambda(s_1^1)\cdot\Phi(q_{1,5}^1) + \lambda(s_5^1)\cdot\Phi(q_{5,3}^1)$, this alternative is not valid. This shows that forwarding can help us to find valid delivery graphs. More importantly, it can also help decrease the overall bandwidth consumption. We illustrate this by comparing the way s_1^1 is delivered in the two examples. In the first example, $q_{1,3}^1$ applied s_1^1 is sent to p_3 from p_1 and $q_{1,3}^1$ is satisfied. Moreover $q_{1,5}^1$ applied s_1^1 is forwarded to p_5 from p_3 to satisfy $q_{1,5}^1$. This is possible since $q_{1,5}^1$ is applicable over $q_{1,3}^1$. Moreover, $q_{1,2}^1$ applied s_1^1 is forwarded to p_2 from p_1. In the second second example, an alternative is shown, where instead of forwarding $q_{1,5}^1$ applied s_1^1 from p_3, we forward it from p_2. In this alternative $\{q_{1,2}^1, q_{1,5}^1\}$ applied s_1^1 is sent to s_2 from s_1, so that $q_{1,5}^1$ can be satisfied from p_2. Note that the first example is more bandwidth efficient, since $\Phi(q_{1,2}^1) < \Phi(\{q_{1,2}^1, q_{1,5}^1\})$. The advantage of the first example comes from the fact that $q_{1,5}^1$ is applicable over $q_{1,3}^1$ and thus routing it through p_3 is more bandwidth efficient. If we did not have $q_{1,5}^1 \sqsubset q_{1,3}^1$, then we would have had to forward $\{q_{1,3}^1, q_{1,5}^1\}$ applied s_1^1 to p_3 from p_1 and the bandwidth efficiency of the two alternatives would have depended on the comparison between $\Phi(q_{1,2}^1) + \Phi(\{q_{1,3}^1, q_{1,5}^1\})$ and $\Phi(q_{1,3}^1) + \Phi(\{q_{1,2}^1, q_{1,5}^1\})$.

4. QUALITY-AWARE STREAM DELIVERY



$\{q_{5,3}^1\} \sqsubset \{q_{5,4}^1\}, \quad \{q_{1,5}^1\} \sqsubset \{q_{1,3}^1\}$

An alternative

Figure 2: Example stream delivery graphs

The problem formulation we have described so far only works for scenarios where a feasible solution, that is a valid stream delivery graph, can be found. If no such graph can be found, we are left without a solution that can deliver streams to all consumer nodes. To solve this, we extend our problem formulation to cover the scenarios where drops in the stream delivery graph, as well as delays, are tolerable and are captured by application-supplied QoS specifications. This extended formalization leads us to the quality-aware stream delivery problem. The main idea is to remove the bandwidth constraint so that the set of valid delivery graphs also includes the ones with drops in the received streams. Furthermore, the objective is also modified, so that the aim is to maximize the overall QoS in stream delivery. Since the QoS functions are defined based on drops and delays, this requires us to formulate the perceived drop and delay for each filter, for a given configuration of the stream delivery graph.

4.1 Extended Problem Formalization

Delays and Drops: We denote the delay introduced by a link (either physical or logical) $l_{i,j} \in L$ with $\delta(l_{i,j})$ and the average processing delay introduced by a node $p_u \in P$ with $\delta(p_u)$. The processing delay $\delta(p_u)$ may depend on the processing capacity of the node p_u and the amount of processing that it has to perform before forwarding streams. Recall that, node p_u may be applying filters on certain streams before forwarding them to other nodes. We do not attempt to model different components of the processing delay and their dependence on the stream delivery graph. For our simulation studies, we assume $\delta(p_u) = \delta^*, \forall p_u \in P$. The average delay on a stream tuple as perceived by a filter $q^j_{i,k} \in Q$ is denoted by $d^j_{i,k}$, whereas the average tuple drop rate perceived by $q^j_{i,k}$ is denoted by $r^j_{i,k}$. Tuple drop rate is a value in the range $[0, 1]$, where 0 denotes no drop and 1 denotes complete drop.

Quality of Service Specifications: We assume that each filter $q^j_{i,k}$ has an associated QoS specification function $\mathcal{C}^j_{i,k}$ that assigns a quality measure to stream delivery graph's performance with respect to satisfying the filter $q^j_{i,k}$ given the delay $d^j_{i,k}$ and the drop rate $r^j_{i,k}$ associated with $q^j_{i,k}$ in the stream delivery graph. We have $\mathcal{C}^j_{i,k}(r^j_{i,k}, d^j_{i,k}) \in [0, 1]$, where 0 denotes the worst quality and 1 denotes the best quality. The shape of the function $\mathcal{C}^j_{i,k}$ is filter dependent.

An example QoS function can be as follows:

$$\mathcal{C}^j_{i,k}(r, d) = \begin{cases} 1 & r \le 0.2,\ d \le 1 \\ 0.5 & r \le 0.2,\ d > 1 \\ 0.1/r & \text{otherwise} \end{cases}$$

This function specifies that the QoS is maximal (i.e., $= 1$) when the drop rate is less than or equal to 0.2 and the delay is less that or equal to 1 seconds. If the drop rate is less than or equal to 0.2 but the delay is higher than 1 seconds, then the QoS is set to 0.5. In the rest of the cases, the QoS is set to a value less than 0.5 depending only on the drop rate and being inversely proportional to it.

Updated Problem Definition: Since we no more impose the bandwidth requirement in forming the stream delivery graph, we penalize the high drop and delay in stream delivery by integrating the quality measure into our cost function, as follows:

$$max \sum_{q^j_{i,k} \in Q} \log \mathcal{C}^j_{i,k}(r^j_{i,k}, d^j_{i,k})$$

Filter satisfaction condition is met

Stream forwarding condition is met

No loop condition is met

In our new problem formulation, the goal is to maximize the mean (geometric) of the quality measures of different filters. The use of geometric mean in the cost function (i.e., maximizing the sum of logs) is aimed at penalizing solutions which result in very small or zero quality for certain filters.

With proper selection of QoS functions, this problem can be shown to be a binary quadratic programming problem. Quadratic programming is known to be an NP-hard problem [23]. The problem can be solved using popular optimization techniques such as simulated annealing. However, we are more interested in practical on-line solutions that are more in-line with the assumptions of a real-world system, where the details about the network characteristics are not globally known. We discuss these heuristic solutions, later in Section 5.

4.2 Delay and Drop Calculation

We now describe how to compute drop rate $r^j_{i,k}$ and delay $d^j_{i,k}$ values for a filter $q^j_{i,k}$ in a given stream delivery graph. For ease of exposition we use $f^j_{i,k}$ to denote the node that is forwarding a (filter applied) stream s^j_i to p_k. Note that in a valid stream delivery graph, based on the stream forwarding condition, there can not be more than one nodes that are forwarding parts of the same stream to a node. Thus, for $q^j_{i,k} \in Q$ we define $f^j_{i,k} = u$, such that ${}_{u,k}Q^j_i \neq \emptyset$.

Delay computation is straight forward:

$$d^j_{i,k} = \begin{cases} 0 & \text{if } k = i \\ \delta(l_{u,k}) + \delta(p_u) + d^j_{i,u} & \text{otherwise} \end{cases}, \text{ where } u = f^j_{i,k}$$

Simply, we sum up the link and processing delays among the delivery path from the source p_i to sink p_k for filter $q^j_{i,k}$.

The computation of drop rate of filters in a given delivery graph is more involved, but is crucial to evaluate the quality of a given solution. The complexity comes from the fact that drop rate for a filter depends on other filters, since the bandwidth is shared with source streams of other filters. Here we provide two different formulations for drop rate, one for physical and one for logical model, that can be used to *iteratively solve* and calculate the drop values, $r^j_{i,k}$ for each $q^j_{i,k} \in Q$. The formulation is used to calculate $\overline{r}^j_{i,k}$, which is the fraction of tuples in the $q^j_{i,k}$ filtered stream s^j_i that reaches node p_k. We simply have $r^j_{i,k} = 1 - \overline{r}^j_{i,k}$.

$$\textit{For Physical model:} \quad \overline{r}^j_{i,k} = \overline{r}^j_{i,w} \cdot$$

$$min\left(1, \frac{\kappa(l_{w,k})}{\sum_{s^v_u \in S}\left(\lambda(s^v_u) \cdot \left(\begin{array}{c} \Phi({}_{w,k}Q^v_u) \cdot \overline{r}^v_{u,w} + \\ \Phi({}_{k,w}Q^v_u) \cdot \overline{r}^v_{u,k} \end{array}\right)\right)}\right)$$

$$, \text{ where } w = f^j_{i,k} \qquad (1)$$

If we have $w = f^j_{i,k}$ then $\overline{r}^j_{i,k}$ can be expressed as $\overline{r}^j_{i,w}$ times the *permeability* of the link $l_{w,k}$. The permeability of a link is the ratio of the number of stream tuples it successfully carries trough compared to the total number of stream tuples that are pushed through the link, with the assumption that any excessive load is randomly shed and shedding is uniform among different streams. For instance if a link has a capacity 2 units and we have three streams being pushed through it with data rates 1, 1, and 2 units, then the permeability of the link is $2/(1 + 1 + 2) = 0.5$. The assumption is that each stream observes the same drop rate, i.e. the link carries through 0.5 fraction of all three streams. To calculate the permeability of the link, in addition to the link capacity, we need to know the total rate of stream data being pushed through the link. The latter can be formalized as the denominator in Equation 1 for the link $l_{w,k}$, where $\lambda(s^v_u) \cdot \Phi({}_{w,k}Q^v_u)$ is the data rate for the ${}_{w,k}Q^v_u$ applied s^v_u forwarded from p_w to p_k. However, since the tuples of s^v_u may have been dropped earlier in the path, the actual data rate is adjusted by $\overline{r}^v_{u,w}$, i.e. by the fraction of stream tuples available to node p_w for forwarding them to node p_k. Given this derivation, $\overline{r}^j_{i,k}$ is calculated as given in Equation 1, by comparing the link capacity to the total rate of stream data being pushed through the link.

$$\textit{For Logical model:} \quad \overline{r}^j_{i,k} = \overline{r}^j_{i,w} \cdot$$

$$min\left(1, \frac{\kappa(p^{\uparrow}_w)}{\sum_{s^v_u \in S}\sum_{l_{w,z} \in L}\left(\lambda(s^v_u) \cdot \Phi({}_{w,z}Q^v_u) \cdot \overline{r}^v_{u,w}\right)}, \right.$$

$$\left. \frac{\kappa(p^{\downarrow}_k)}{\sum_{s^v_u \in S}\sum_{l_{z,k} \in L}\left(\lambda(s^v_u) \cdot \Phi({}_{z,k}Q^v_u) \cdot \overline{r}^v_{u,z}\right)}\right)$$

$$, \text{ where } w = f^j_{i,k} \qquad (2)$$

The derivation for the logical model is similar. Instead of link capacities, we employ upload and download bandwidths to compute $\overline{r}^j_{i,k}$. Concretely, if we have $w = f^j_{i,k}$, then $\overline{r}^j_{i,k}$ can be expressed as $\overline{r}^j_{i,w}$ times the permeability of logical link $l_{w,k}$. The permeability of link $l_{w,k}$ is the ratio of the number of stream tuples that reach p_k from p_w compared to the total number of stream tuples that are destined to p_k from p_w, with the assumption that any excessive load is randomly shed and shedding is fair among different streams. However the amount of shedding (drop) is not only dependent on the upload bandwidth of p_w, but also on download bandwidth of p_k. The total rate of stream data that p_w pushes to other nodes can be calculated as $\sum_{s^v_u \in S}\sum_{l_{w,z} \in L}\left(\lambda(s^v_u) \cdot \Phi({}_{w,z}Q^v_u) \cdot \overline{r}^v_{u,w}\right)$ and can be compared with the upload bandwidth of p_w, i.e. $\kappa(p_w^{\uparrow})$, to compute one upper bound for permeability of logical link $l_{k,w}$. Additionally, the total rate of stream data that is pushed to p_k from other nodes can be calculated as $\sum_{s_u^v \in S} \sum_{l_{z,k} \in L} \left(\lambda(s_u^v) \cdot \Phi({}_{z,k}Q_u^v) \cdot \overline{r}_{u,z}^v \right)$ and can be compared with the download bandwidth of p_k, i.e. $\kappa(p_k^{\downarrow})$, to compute another upper bound for permeability of link $l_{w,k}$. As shown in Equation 2, the minimum of these two bounds is used to calculate $\overline{r}_{i,k}^j$. In the rest of the paper, we base our discussions on the logical link model.

5. STREAM DELIVERY ALGORITHMS

In this section, we describe pragmatic algorithms for building stream delivery graphs in a distributed setting, where nodes do not have global knowledge about the network characteristics. We consider stream delivery graph construction as an assignment problem. Concretely, when a node has a filter defined on a stream, it contacts a *control point* that is responsible for that stream and requests to join the delivery graph, or more precisely to the subgraph formed by the nodes that have filters defined on the same stream and are already receiving the stream. The latter forms a tree, called the *delivery tree* of the stream. Upon receiving a request to join the tree, the control-point *assigns* a parent node to the requester node. The algorithm used by the control node to make this assignment, which we call the *construction* algorithm, plays a key role in the effectiveness of the overall stream delivery graph and is the main focus of this section.

Once a parent node is assigned to a requester node, the control-point checks its locally stored copy of the delivery tree to see whether an *expansion* is required or not. An expansion is required if the filter of the requester node is not applicable over all of the filters along the path to the root. We illustrate this with an example. Figure 3 shows an example delivery tree, where node p_5 requests to join the tree and is assigned p_3 as its parent. However, the stream that p_3 receives from its parent p_1 (which is also the root) has $q_{1,3}^1$ applied to it. In this case we have $q_{1,5}^1 \not\sqsubset q_{1,3}^1$, and thus ${}_{1,3}Q_1^1$ has to be expanded from $\{q_{1,3}^1\}$ to $\{q_{1,3}^1, q_{1,5}^1\}$ (see step (3) in the figure). Note that, expansion also involves the notification of the forwarder nodes that are incident upon the links on which expansion is performed, so that they can adjust the streams they are forwarding to their children. After the expansion is performed (possibly on more than one link), the assigned parent of the requester node is notified to serve the requested stream to the requester node (see step (4) in the figure).

5.1 Classification of Construction Algorithms

We classify the delivery graph construction algorithms along three dimensions:

Point of Control: In *source-controlled* construction, there is a one-to-one mapping between the source nodes and the control-points.

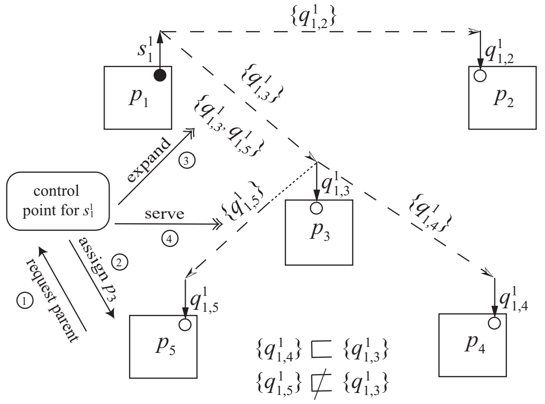


Figure 3: Example parent assignment and expansion

A control point only maintains information about the delivery trees rooted at a single source node and does not share this knowledge with other control points. In this case, the control point is assumed to be co-located with the source node. The source node can delegate the control-point responsibilities to one or more number of other nodes, although this aspect is not considered in this paper.

In *globally-controlled* construction, there is a centralized control-point that maintains knowledge about the complete delivery graph. Since a node can belong to multiple delivery trees with different root (source) nodes, having a control-point with global knowledge helps making better decisions during parent assignment. However, as opposed to source-controlled assignment in which control-points are co-located with the source nodes, in globally-controlled assignment an additional centralized control node is needed.

Source-controlled construction is the preferred point of control method due to its distributed nature. We use globally-controlled construction for comparison purposes, i.e., to assess the effectiveness of source-controlled construction algorithms in which the control-points (source nodes) lack the additional knowledge that is accessible to the global control-point.

Node Feedback: In *uninformed* construction, control points make their assignment decisions based solely on the current topology of the delivery graph and the characteristics of the filters defined by the nodes (including selectivity, amortized size, and QoS functions). Feedback from the nodes that are already part of the stream delivery graph is not used.

In *informed* construction, the control point receives feedback from the nodes, such as their current drop rates and perceived delays. This kind of feedback helps in making informed decisions about the suitability of assigning a certain node as a parent to the requester node. As a result, informed construction algorithms are expected to be more effective, compared to their uninformed counterparts.

Assignment Dynamics: In *non-adjusting* construction, the assignment of a parent node to the requester node is a one-time process and is not revised unless there is a node failure or departure in the delivery graph. This nature of the assignment process does not provide any opportunities to adjust the position of the node in the delivery graph to improve the overall efficiency of stream delivery.

In *adjusting* construction, the parent assignments of the nodes can be changed after the initial assignment is made. This is aimed at improving the graph by revising the initial assignments, in case they turn out to be ineffective. This requires feedback from the nodes and only applies to informed construction algorithms. However, these re-assignments have to be performed with care, since frequent adjustments may create overhead and may lead to unnecessary oscillations resulting in an unstable delivery graph.

5.2 Naïve Approaches

The first two algorithms we discuss are naïve and serve as baselines. They also represent extreme cases with respect to some important metrics, such as the overall bandwidth consumption, variance in the perceived delays of the nodes, and variance in the bandwidth consumed by the nodes for forwarding streams.

The CHAIN *and* BRUSH *Algorithms*

In the CHAIN algorithm, each delivery tree takes the form of a chain, where each node (except the leaf) has a single child node. In the BRUSH algorithm, each delivery tree takes the form of a brush, where every node (except the root) has the source node as its parent.

The total bandwidth consumed by CHAIN can be high, especially when the filters of the nodes that are down in the chain are not applicable over the filters of the nodes above. On the other hand, BRUSH has the minimum possible total bandwidth consumption. However, BRUSH has high variance, thus unfairness, in the forwarding respon-

Algorithm 1: Evaluation metric calculation for UNFA

```
EVALUNFA($p_k$, $q^j_{i,v}$)
{collect downstream bw. consumptions (A) and node depths (D)}
foreach $p_u \in T^j_i$
    if UNFAG {global assignment}
        $A[p_u] \leftarrow \sum_{T^y_x \in T} \sum_{p_z \in \Psi(T^y_x, p_u)} (\lambda(s^y_x) \cdot \Phi({}_{u,z}Q^y_x))$
    else if UNFAS {source-controlled assignment}
        $A[p_u] \leftarrow \sum_{T^y_i \in T} \sum_{p_z \in \Psi(T^y_i, p_u)} (\lambda(s^y_i) \cdot \Phi({}_{u,z}Q^y_i))$
    $D[p_u] \leftarrow \iota(p_u)$
{perform updates on $A$ and $D$ for the addition of $p_v$}
for $i \leftarrow 1$ to $i \leq \iota(p_k)$ {for each link on the way to the root}
    Let $p_a = \pi^i(T^j_i, p_k)$ {upstream node}
    Let $p_b = \pi^{i-1}(T^j_i, p_k)$ {downstream node}
    if $q^j_{i,v} \not\subset {}_{a,b}Q^j_i$ {perform expansion if needed}
        $A[p_a] \leftarrow A[p_a] + \lambda(s^j_i) \cdot (\Phi({}_{a,b}Q^j_i \cup \{q^j_{i,v}\}) - \Phi({}_{a,b}Q^j_i))$
$A[p_k] \leftarrow A[p_k] + \lambda(s^j_i) \cdot \Phi(q^j_{i,v})$ {for the node to be added}
$D[p_v] \leftarrow \iota(p_k) + 1$; $A[p_v] \leftarrow 0$ {for the requester node}
return $1/(Var(A) \cdot Var(D))$
```

sibilities of the nodes. This is because all of the forwarding is done by the source node. Obviously this approach can result in high drop rates. Conversely, CHAIN is more fair in terms of the amount of bandwidth consumed by nodes for fulfilling their forwarding responsibilities. However, for the CHAIN algorithm the variance of the delays perceived by the nodes is high, since the delivery tree has the maximum possible height. On the other hand, BRUSH results in lower and more uniform delays, since the nodes are one-hop away from the root.

5.3 The UNFA Algorithm

We now describe a more sophisticated construction algorithm that is called UNFA (uninformed, non-adjusting, fair assignment), and its two variations UNFAS and UNFAG that are source-controlled and globally-controlled, respectively. We first introduce some additional notation that will be used in the rest of this section.

We denote the delivery tree associated with stream s^j_i by T^j_i. The set of all delivery trees (that form the delivery graph) is represented by $T = \bigcup_{s^j_i \in S} T^j_i$. We use $\pi(T^j_i, p_u)$ to denote the parent node of p_u in T^j_i and $\pi^k(T^j_i, p_u)$ to denote the kth ancestor of p_u in T^j_i, where $\pi^0(T^j_i, p_u) = p_u$. The depth of a node p_u in a delivery tree T^j_i is denoted by $\iota(T^j_i, p_u)$, where $\iota(T^j_i, p_i) = 0$. And finally, the set of children of nodes of p_u in T^j_i is denoted by $\Psi(T^j_i, p_u)$.

The UNFA algorithm considers all possible parent nodes for a requester node and assigns the one that results in the highest *evaluation metric*. For a requester node that is interested in receiving s^j_i, the set of candidate parent nodes is $\{p_k : p_k \in T^j_i\}$. The evaluation metric used by the UNFA algorithm is $(Var(A) \cdot VAR(D))^{-1}$, where $VAR(D)$ is the variance in the depths of the nodes, and $VAR(A)$ is the variance in the bandwidth required for a node to forward the assigned streams to its children nodes. In the source-controlled version of the algorithm, that is UNFAS, only the bandwidth required for forwarding the streams that are sourced at p_i (i.e., S_i) are considered. This is because the control-point will only know about the delivery trees that are rooted at p_i. For a node $p_u \in T^j_i$, this is given by $\sum_{T^y_i \in T} \sum_{p_z \in \Psi(T^y_i, p_u)} (\lambda(s^y_i) \cdot \Phi({}_{u,z}Q^y_i))$. On the other hand, the globally controlled version, that is UNFAG, extends this to all forwarded streams. It is important to note that before $VAR(A)$ can be computed, the expansions that are needed in order to serve the requester node from a candidate parent p_k should be performed. Denoting the requester node by p_v and its filter on s^j_i by $q^j_{i,v}$, this requires to check the links on the path from p_k to the root on T^j_i and perform expansions when needed, i.e. when $q^j_{i,v}$ is not applicable over the filter that is applied to the stream forwarded on the link. The details of this process is given in Algorithm 1, which provides the pseudo code for calculating the evaluation metric for a candidate parent p_k, given the filter of the requester node $q^j_{i,v}$.

UNFA is referred to as fair, since it aims at minimizing the differences in both the forwarding responsibilities of the nodes and the differences in the delay perceived by the nodes. This nature of UNFA results in creating delivery trees that strike a balance between the CHAIN and BRUSH algorithms. Thus, it does not suffer from the deficiencies of the two elementary algorithms. However, UNFA is unable to make use of the QoS specifications of the filters, mainly because it is an uninformed algorithm. In the next section, we discuss more advanced informed construction algorithms.

5.4 The QASD Algorithm

We now introduce an informed and adjusting construction algorithm called QASD and its source-controlled and globally-controlled variations, called QASDS and QASDG, respectively.

Being an informed construction algorithm, QASD receives feedback from the nodes in the delivery graph. For a node p_u, this feedback includes three types of information: (1) For each delivery tree T^j_i that p_u participates in, the control point receives the drop rate experienced by p_u for the stream s^j_i, denoted by $r^j_{i,u}$. (2) Similarly, for each delivery tree T^j_i that p_u participates in, the control point also receives the delay experienced by p_u for the stream s^j_i, denoted by $d^j_{i,u}$. (3) Moreover, the control point receives the *shedding fraction* for each node p_u, denoted by z_u. Shedding fraction of a node is the fraction of tuples dropped when forwarding streams. Recall (see Section 4.2) that the shedding is performed uniformly among the forwarded streams and thus the same fraction applies to all streams forwarded at a node.

Besides the variables that are bound to feedback information from the nodes, the control point also maintains a derived variable for each node p_u, which is called the *slack bandwidth* and is denoted by B_u. Slack bandwidth is an estimate of a node's available bandwidth that can be used for forwarding purposes (upload bandwidth). Initially, we have $B_u = \infty, \forall p_u \in T$, since there is no information about the resources of the nodes, available to the control point. The slack bandwidth values are updated by the algorithm as the delivery trees are formed. The QASD algorithm works based on three heuristic principles. Before describing the algorithm in detail, we first list these principles:

1. Refrain from making an assignment that results in expansion in the delivery tree.

2. Refrain from making an assignment that depletes the slack bandwidth of a node.

3. Among the available parents that satisfy items 1 and 2, prefer the ones that result in better QoS for the node.

The first principle avoids assignments that cause expansions, for two reasons. First, expansions result in poor sharing of resources. In an ideal situation, a node will only receive the part of the stream it is interested in and will forward this stream or a subset of it to other nodes. Nodes carrying parts of streams they are not interested in constitutes overhead in bandwidth consumption. Second, expansion potentially affects multiple nodes along a path and carries a risk of depleting slack bandwidth of nodes that are closer to the root. Drops appearing at lower depths of the delivery tree are expected to have a more pronounced effect in the overall quality of stream delivery, due to larger number of downstream nodes affected from the drops. The second principle is an intuitive one, that avoids additional drops. An assignment that fills up the available slack bandwidth of the parent node will initiate shedding at the parent node and introduce drops. The third principle gives priority to parents that result in better QoS

Algorithm 2: QASD Algorithm

```
QASD($q^j_{i,v}$)
(1)  $V \leftarrow -1$ {best score};  $a \leftarrow -1$ {parent node index}
(2)  $W \leftarrow \lambda(s^j_i) \cdot \Phi(q^j_{i,v})$ {required bandwidth}
(3)  foreach $p_k \in T^j_i$ {each possible parent}
(4)      if $q^j_{i,v} \sqsubset q^j_{i,k}$ and $B_k \geq \overline{r}^j_{i,k} \cdot W$  {if applicable}
(5)          $V' \leftarrow \mathcal{C}^j_{i,v}(r^j_{i,k},\ \delta^+ + d^j_{i,k})$ {quality}
(6)          if $V' > V$ or $(V' = V$ and
                 $\text{EVALUNFA}_A(p_k, q^j_{i,v}) > \text{EVALUNFA}_A(p_a, q^j_{i,v}))$
(7)              $V \leftarrow V'$; $a \leftarrow k$ {better choice}
(8)  if $a \neq -1$ {if a parent is found}
(9)      Assign $p_a$ as temporary parent to $p_v$ in $T^j_i$
(10)     if $z_a = 0$ {if no shedding on $p_a$}
(11)         $B_a \leftarrow B_a - \overline{r}^j_{i,k} \cdot W$  {update the slack}
(12)         Assign $p_a$ as parent and return
(13)     else {if there is shedding}
(14)         Unassign $p_a$ as parent
(15)         if QASDG {globally-controlled}
(16)             $X \leftarrow \sum_{T^y_x \in T} \Big(\lambda(s^y_x) \cdot \overline{r}^y_{x,a} \cdot$
                     $\sum_{p_k \in \Psi(T^y_x, p_a)} \Phi(_{a,k}Q^y_x)\Big)$
(17)         else  {QASDS, source-controlled}
(18)             $X \leftarrow \sum_{T^y_i \in T} \Big(\lambda(s^y_i) \cdot \overline{r}^y_{i,a} \cdot$
                     $\sum_{p_k \in \Psi(T^y_i, p_a)} \Phi(_{a,k}Q^y_i)\Big)$
(19)         $B_k \leftarrow (1 - z_a) \cdot (X + W) - X$ {update the slack}
(20)         return QASD($q^j_{i,v}$) {find new parent}
(21) else {$a = -1$, no parent found}
(22)     Assign parent based on $\text{UNFA}_A$
(23)     foreach $p_k \in P$ {for each node}
(24)         if $z_k \neq 0$ then $B_k \leftarrow 0$
```

for the newly added node. Since the first two principles strive to make sure that the newly added node minimally effects the QoS of already existing nodes, the heuristic used in the third principle, which optimizes for the newly added node, is sound.

Algorithmic Details

When a new node p_v with a filter $q^j_{i,v}$ is to join the delivery tree T^j_i, the following steps take place with the QASD algorithm. First, all nodes in T^j_i that have compatible filters with $q^j_{i,v}$ and have enough slack bandwidth to accommodate p_v are considered as potential parents. A node $p_k \in T^j_i$ can accommodate p_v if its slack bandwidth is larger than or equal to the drop experienced by p_k on s^j_i times the bandwidth requirement of $q^j_{i,v}$, and thus formally stated $B_k \geq r^j_{i,k} \cdot W$, where $W = \lambda(s^j_i) \cdot \Phi(q^j_{i,v})$ is the bandwidth requirement of $q^j_{i,v}$. If the set of potential parents is non-empty, then the one with the best evaluation score is selected. The evaluation score used is the expected quality of service for the node p_v. When p_k is considered as a parent, then the evaluation score is $\mathcal{C}^j_{i,v}(r^j_{i,k},\ \delta^+ + d^j_{i,k})$. Here, δ^+ is the average delay between p_k and its children nodes in any of the delivery trees. If no such children node exists, δ^+ is taken as the average delay between two nodes in the system. If there are several parent nodes that result in equally good QoS for the node p_v, then the one which results in better balance in the forwarding responsibilities of the nodes is selected as the parent node. This latter selection is very close to using the EVALUNFA procedure described before, except that the variance in the node depths are not considered. We name this variation of the UNFA algorithm as UNFA_A. In summary, when there are several parent nodes that provide equally good QoS, the assignment is made by using the UNFA_A algorithm by considering only these parents.

Once the parent node is selected, say p_a, it is assigned to p_v as a temporary parent. Then the shedding fraction z_a is observed. If we have $z_a = 0$, then the assignment is made final and B_a is updated by decreasing it by $r^j_{i,k} \cdot W$. Otherwise, the slack bandwidth is inaccurate and thus should be updated. Moreover p_v should be assigned to a different parent. In this case, we update the slack bandwidth B_a using the new shedding fraction z_a. The concrete details of updating the slack bandwidth is given in Algorithm 2, between lines 14 and 19. Note that the slack bandwidth can be properly computed only for the globally-controlled version of the algorithm (QASDG), whereas it can only be approximated for the source-controlled version (QASDS). In the latter case, the slack bandwidth computation results in a lower bound and thus the algorithm becomes conservative. Once the slack bandwidth for p_a is updated, the algorithm is run from the beginning (with a recursive call) to locate a new parent node for p_v.

In the first place, if the set of parents that have compatible filters and sufficient slack bandwidth turns out to be the empty set, then we switch to best-effort mode and make the assignment based on the UNFA_A algorithm by considering all nodes in the delivery tree. Once the assignment is made, no further parent adjustments are performed for the node p_k. For nodes whose shedding fractions have changed to a non-zero value as a result of this assignment, the slack bandwidths are set to 0.

Discussions

Due to its adjusting nature the QASD algorithm may go over several rounds involving temporary parents, before assigning a final parent to a node. As a result of this, the average number of rounds executed before making a final assignment is an important metric in assessing the overhead of the QASD algorithm, compared to the alternative approaches discussed in Section 5.2. Despite this runtime overhead (which turns out to be very small as it will be discussed later in Section 6), the QASD algorithm is expected to construct more effective delivery graphs, since it makes use of the feedback information received from the nodes to make informed decisions.

There are further issues to be discussed in the context of stream delivery in distributed CQ systems. These include handling of joins, departures, failures, as well as adapting to significant changes in node resources. In this paper, we do not discuss these issues and leave them as future work.

6. EXPERIMENTAL EVALUATION

We report a number of simulation-based experimental results and study the impact of various parameters on the cost and effectiveness of stream delivery, using different construction algorithms. Before presenting our findings, we first introduce the set of evaluation metrics used and describe the details of our experimental setup.

6.1 Evaluation Metrics

We use three main evaluation metrics to study the performance of the construction algorithms described in this paper. These metrics are:

Overall Service Quality: This metric is used to assess the quality of the stream delivery graph formed by a construction algorithm. It is defined as the geometric mean of the QoS values for the filters, as formulated in Section 4.1. Note that QoS value for a filter is dependent on the filter specific QoS function and the actual drop and delay values perceived by the node defining the filter. The latter two are computed based on the derivations given in Section 4.2.

Total Bandwidth: This metric is used to assess the cost of stream delivery. It is the total amount of bandwidth required to achieve stream delivery without introducing any drops, for a given delivery graph. The formulation given in Section 3.2 is used to calculate it.

Fairness in Forwarding: This metric is used to assess the fairness of a construction algorithm in terms of the forwarding responsibilities it assigns to different nodes. It is defined as the variance in the amortized forwarding bandwidths of nodes. For a node, the latter is defined as

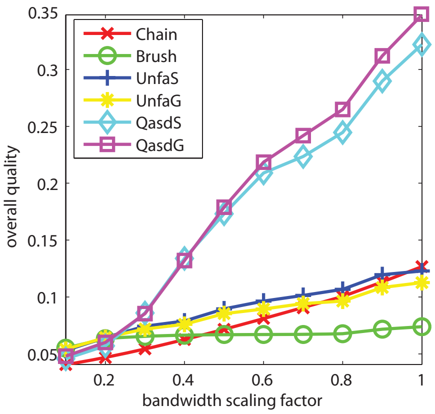


Figure 4: Impact of node bandwidth resources on overall quality of stream delivery

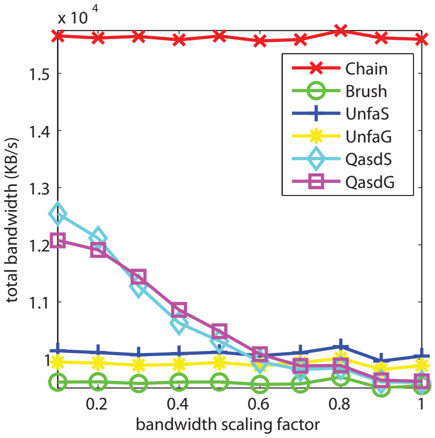


Figure 5: Impact of bandwidth resources on total bandwidth consumed in stream delivery

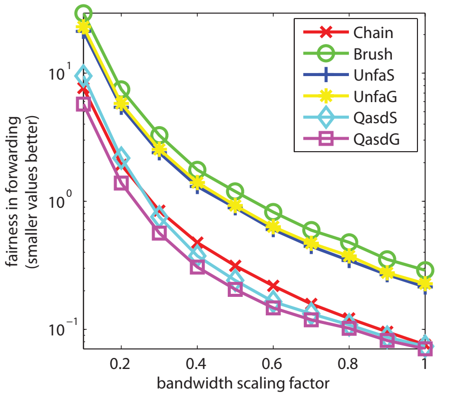


Figure 6: Impact of node bandwidth resources on fairness in stream forwarding responsibilities

parameter	default value or range
number of nodes	100
number of streams	20
number of filters per stream	30
link delay range	$[10, 500]$ msec
QoS drop threshold range	$[0.0, 0.2]$
QoS delay threshold range	$[1.0, 5.0]$ sec
number of filter blocks	10
number of filter types	4
stream bandwidth range	$[50, 100]$ KByte/sec
node download bandwidth range	$[1.0, 5.0]$ Mbit/sec
node upload bandwidth range	$[0.5, 2.5]$ Mbit/sec

Table 2: Simulation Parameters

the total bandwidth it needs for forwarding purposes divided by the node's available upload bandwidth. Smaller values for this metric imply increased fairness in forwarding responsibilities.

In addition to these three main metrics, we also measure other interesting statistics, such as the variance in perceived delays and the average number of assignment rounds for the QASDS and QASDG algorithms.

6.2 Experimental Setup and Parameters

For each simulation run a random resource setup is generated, followed by a random workload generation. Each experiment is run 100 times and the average results are reported. The resource configuration involves setting the number of nodes (100 by default), a link delay range which is used to select a random delay for each node pair ($[10, 500]$ msec by default), and node upload/download bandwidth ranges that are used to randomly assign bandwidth resources to nodes ($[1.0, 5.0]$ Mbit/sec for download and $[0.5, 2.5]$ Mbit/sec for upload by default). The workload configuration involves setting the number of streams (20 by default), number of filters per stream (30 by default), stream bandwidth range that is used to randomly set the flow size of streams ($[50, 100]$ KByte/sec by default), and finally the configuration of filter characteristics. An important filter characteristic is the QoS function. In the experiments, a common template is used for the QoS functions of the filters, with randomly assigned delay and drop thresholds (selected from the range $[1.0, 5.0]$ sec and $[0.0, 0.2]$ by default, respectively). Denoting these thresholds by η_d and η_r, for delay and drop respectively, the common QoS template is as follows:

$$\mathcal{C}(r,d) = \begin{cases} 1 & r \le \eta_r,\ d \le \eta_d \\ 0.5 & r \le \eta_r,\ d > \eta_d \\ 0.5 \cdot \eta_r / r & \text{otherwise} \end{cases}$$

Two remaining workload configuration issues related to filters is their applicability relationship and defining the size of a filtered stream. To handle this, we define a number of filter types (4 by default). Each filter type represents a random subset of 10 blocks and filter applicability is tested based on set containment. The size of a filtered stream is simply the size of the stream multiplied by the fraction of blocks existent in the block set of the filter. The disjunction of filters is emulated by set union operation over filter blocks.

The simulation parameters and their default values or ranges are given in Table 2 for ease of reference.

6.3 Results

We now present our results on the comparative study of the six construction algorithms, namely BRUSH, CHAIN, UNFAS, UNFAG, QASDS, and QASDG. We investigate the impact of changing various simulation parameters that alter the workload characteristics or resource availability, on the evaluation metrics introduced.

Impact of Bandwidth Availability

We first study the impact of bandwidth availability on the three evaluation metrics, for different algorithms. Note that increasing the rate or tuple size of streams have a similar effect with decreasing the bandwidth availability of nodes. Thus, the results from this part can also be used to reason about the impact of stream bandwidth. To change the node bandwidth availability, we multiply our upload and download bandwidth ranges by a bandwidth scaling factor.

Figure 4 plots overall quality as a function of bandwidth, for scaling factors less than or equal to 1. We make many interesting observations from this figure. First, QASD-based algorithms show remarkably better performance, compared to all other alternatives. The improvement in performance is more prominent when the bandwidth availability is higher and reaches up to 180% improvement compared to the best alternative when the scaling factor is 1. This is due to the fact that QASD respects delay and drop requirements of filters. It also makes better use of the bandwidth resources, when they are available, through the feedback-based construction algorithm it employs. Second, we observe that QASDG is superior to QASDS (around %5 better), even though the difference between the two approaches is in-

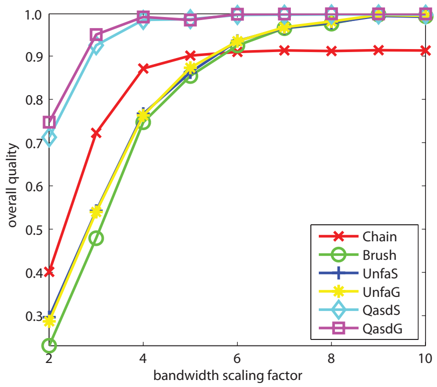


Figure 10: Impact of node bandwidth resources on overall quality of stream delivery (including perfect delivery scenarios)

significant compared to their advantage over other alternatives. The slightly less performance of QASDS stems from the inaccuracy it has in estimating the slack bandwidths (recall Section 5.4).

Third, we see from the graphs in Figure 4 that UNFAS has a slightly better performance than CHAIN, although the latter has a steeper curve and catches the former when the scaling factor reaches 1. The poor performance of CHAIN for very low levels of bandwidth availability can be attributed to its inability to make good use of filter similarity. Since it has chain-like delivery trees, most of the nodes closer to the source are forced to forward the complete stream, resulting in drops when bandwidth is not available and hurting the performance for all of the downstream nodes. However, as the bandwidth availability increases the drops decrease and thus the performance increases. If all nodes have enough bandwidth to forward and receive the full stream without drops, CHAIN is effective. On the other hand, UNFAS can still introduce drops when all nodes have enough bandwidth to forward and receive the full stream without drops. This is because UNFAS creates bushy trees in which nodes may be forced to serve more than one children nodes. Since UNFAS does not use any feedback, such multiple forwardings per node can introduce excessive drops.

Even though it may seem that CHAIN is a better choice compared to UNFAS in terms of overall quality when bandwidth is not extremely limited, Figure 4 only tells half of the story. If we look at Figure 10, which plots the overall quality for larger scaling factors, we see that the best overall quality achievable by CHAIN is 0.9, whereas UNFAS achieves the optimal value of 1. This is because of the excessive delay CHAIN introduces in stream delivery. Figure 10 also shows that QASD-based algorithms sustain their superior performance for larger bandwidth scaling factors, and reach to the optimal overall quality value of 1, before other alternatives.

Fourth, we observe that UNFAG performs slightly worse than UNFAS. This points out that the heuristics used in UNFA-based algorithms are more effective when they are applied separately on delivery trees instead of applying them globally on the complete delivery graph. Finally, we observe that BRUSH is only competitive when the bandwidth availability is extremely low or when bandwidth is abundant. This is the complete opposite behavior of CHAIN.

Figure 5 plots the total bandwidth requirement for stream delivery without drops as a function of bandwidth scaling factor, for different algorithms. We make three observations from the figure. First, as expected, BRUSH and CHAIN form the lower and upper bounds for the total bandwidth requirement. Second, we see that all approaches except QASD-based ones, make no adjustments in their delivery graphs to adapt to the changing bandwidth resource availability. This is because they do not use feedback information. Third, and most interestingly, we see that QASD starts with a total bandwidth requirement that is in the middle between BRUSH and CHAIN, and improves to the lower limit achieved by BRUSH as the scaling factor reaches to 1. The increased total bandwidth requirement of QASD for low bandwidth availability scenarios enables it to provide increased overall quality. On the other hand, QASD shows perfect total bandwidth requirement characteristics when bandwidth resources are abundant.

Figure 6 plots the fairness in forwarding as a function of bandwidth scaling factor, for different algorithms. The y-axis is in logarithmic scale. We observe that increasing bandwidth availability improves the fairness for all algorithms. It is also seen from the figure that BRUSH performs the worst and UNFA-based algorithms better but close to it. It is no surprise that CHAIN performs good, since every node except the leaf has only a single children node in a delivery tree. It may seem unexpected that the QASD-based algorithms perform better than CHAIN. However, if we recall that the fairness is defined based on amortized forwarding responsibility that takes into consideration the bandwidth resources of the nodes, the performance of QASD is justified. Assigning more responsibility to a node with higher resources is considered fair. Thus, QASD outperforms all alternatives.

Impact of # of Nodes

We study the impact of altering the number of nodes while keeping the number of streams and filters fixed, on the evaluation metrics for different algorithms. The number of nodes is decreased from its default value of 100 to the extreme case where there are 31 nodes and thus every node has a filter on every stream. Figure 7 plots the overall quality as a function of number of nodes, whereas Figure 8 studies the total bandwidth. The CHAIN algorithm is removed from Figure 8 due to its high total bandwidth requirement. The results for overall quality closely parallel the results form the bandwidth availability study. The QASD algorithms show up to 180% improvement over their best competitors. In terms of total bandwidth requirement, QASD algorithms perform close to the optimal scenario represented by BRUSH. It is also interesting to observe from Figure 8 that the improved overall quality provided by UNFAS over UNFAG comes at the cost of slight increase in total bandwidth requirement.

Fairness in Perceived Delays

We study the fairness in terms of delays perceived by the filters. We alter the link delay range used in the simulation by multiplying it with a delay scaling factor, and measure the resulting variance in perceived delays, for different algorithms. The results are plotted in Figure 9. Since BRUSH forms delivery trees in which every node is one hop away from the source, it has the smallest variance score. QASDS performs the second best, followed by UNFAS and CHAIN. As expected, CHAIN has very poor scalability due to its worst case structure with regard to delay distribution. Since BRUSH is not a real contender when overall quality is considered, we can say that QASDS shows better fairness in perceived delays than any viable alternative algorithm we have tested. Globally controlled QASD and UNFA variations have same results with their source-controlled counterparts and are not included in Figure 9.

Impact of # of Filter Types

We study the impact of altering the number of filter types on the evaluation metrics, for different algorithms. A larger number of filter types makes it less likely that a given filter is same with or applicable over any other filter. Thus, it puts a heavier burden on the resources and seeks for effective delivery graphs with good filter grouping.

Figure 11 plots the overall quality in stream delivery as a function of number of filter types. We see that the quality degrades as the number of filter types increases. A good property of QASD-based algorithms is that, they keep their advantage throughout the whole range

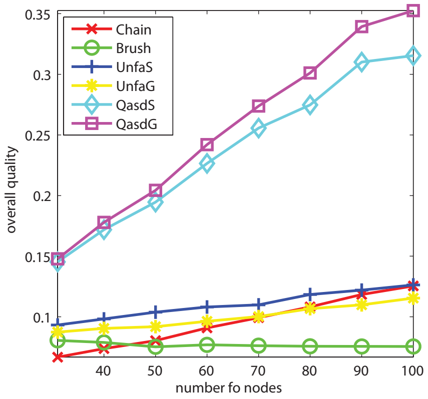


Figure 7: Impact of number of nodes on overall quality of stream delivery

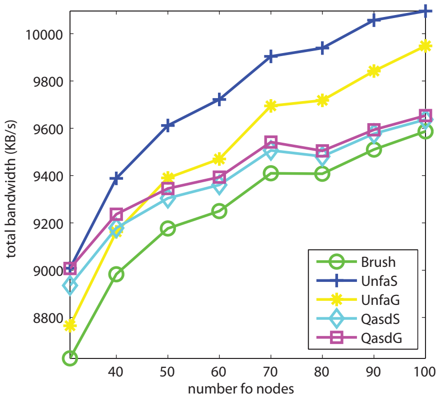


Figure 8: Impact of number of nodes on total bandwidth consumed in stream delivery

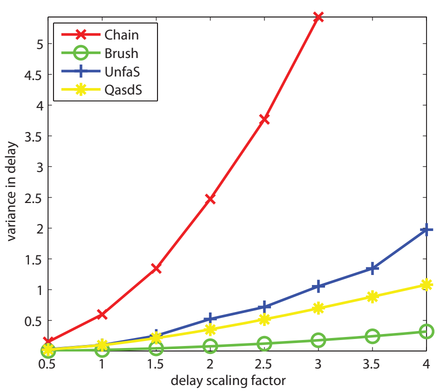


Figure 9: Impact of link delays on fairness in perceived delays in stream delivery

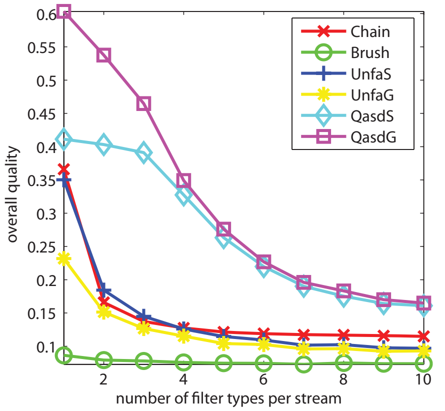


Figure 11: Impact of number of filter types on overall quality of stream delivery

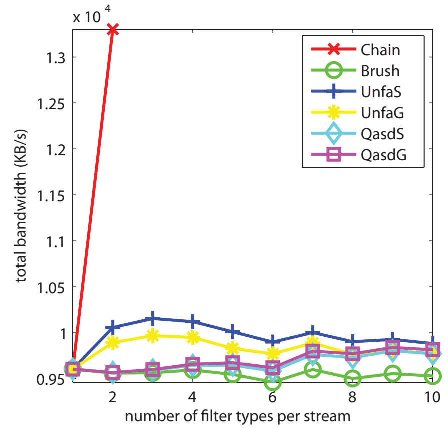


Figure 12: Impact of number of filter types on total bandwidth consumption in stream delivery

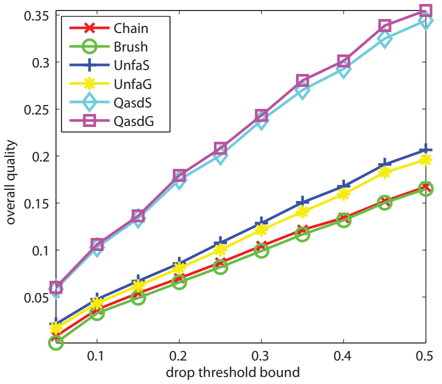


Figure 13: Impact of drop threshold on overall quality of stream delivery

of number of filter types. QASDG reaches up to around 200% improvement over the best competitor, where this improvement reduces to around 30% as the number of filter types reaches 10, and stabilizes thereafter. Although we see that QASDS performs close to QASDG most of the time, its performace degrades especially when the number of filter types is 1 or 2. However, it keeps its strong advantage over non-feedback based alternatives even for the case of 2 types of filters (around 100% better than the best competitor). This advantage drops to 10% for the single filter type per stream case.

Figure 12 plots the total bandwidth requirement for stream delivery without drops, as a function of number of filter types. We observe that QASD based algorithms start with a minimal bandwidth requirement (as dictated by BRUSH) when the number of filter types per stream is small (up to 3), and slightly increases as the number of filter types increases. However, the bandwidth requirement of QASD-based algorithms do not exceed the bandwidth requirement of UNFA-based algorithms. The total bandwidth requirement of QASD-based algorithms approach to that of UNFA-based ones as the number of filter types increases to 10, but are still within the 5% of the best achievable. The complete line for the CHAIN algorithm is not plotted due to its high total bandwidth requirement.

Impact of QoS Function Drop Thresholds

We study the impact of changing the drop threshold upper bound used in the simulation to configure the QoS functions, on the overall quality of the stream delivery. Recall that a drop threshold is randomly selected from a range ($[0, 0.2]$ by default) and is used in configuring the QoS functions. We alter the default value of 0.2 that was used as the upper bound for the drop threshold. In the experiments presented in this part, we also set the bandwidth scaling factor to 0.5, in order to observe the behavior under limited bandwidth scenarios.

The results are plotted in Figure 13 for different algorithms. The increase in overall quality with increasing thresholds is expected. Even though the absolute improvement of QASD-based algorithms over their best competitors seem to increase with increasing thresholds, percentage-wise their advantage drops from 150% improvement to 75% improvement as the drop threshold upper bound reaches from 0.1 to 0.5. The latter is intuitive, because when the filters are more tolerable towards delays, the competitive advantage of QASD over other alternatives lessens. Trying to avoid drops becomes less of an advantage in these cases.

scaler	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
source	1.41	1.42	1.37	1.28	1.20	1.14	1.10	1.07	1.05	1.04
global	1.31	1.33	1.30	1.23	1.16	1.10	1.08	1.05	1.04	1.03

Table 3: Impact of node bandwidth resources on number of assignment rounds

Assignment Round Analysis for QASD

Recall from Section 5.4 that QASD may need to execute a number of rounds before making a final parent assignment, where each round involves making a temporary assignment and observing the result. However our results listed in Table 3 show that, with the exception of very limited bandwidth scenarios (bandwidth scaling factor ≤ 0.4), the average number of rounds does not exceed 1.2. Average number of rounds is very close to 1 (< 1.05) when the bandwidth scaling factor is 1, which still represents a bandwidth limited scenario by simulation design (an overall quality of 0.35 at best with QASDG). We also see from Table 3 that QASDG performs slightly better than QASDS, the biggest difference taking place when the bandwidth scaling factor is 0.1, at which point QASDG requires around 1.3 number of rounds whereas QASDS requires around 1.4 rounds.

7. RELATED WORK

Recent literature includes several studies on different aspects of distributed data stream management in networked environments. These include query optimization [11], load balancing [24], high availability [13], and fault tolerance [4]. Interestingly, with the exception of [19], there is no study of stream delivery problem in the literature.

StreamGlobe [19] is a P2P stream delivery network in a Grid environment, that performs cooperative forwarding of streams with support for in-network filtering. This is along the similar lines of the system model assumed in this work. However, the work presented in [19] optimizes the stream delivery graph without taking into account resource availability in terms of network bandwidth and link delays. As shown in this paper, in the face of limited resources this may result in unwanted drops and delays in stream delivery, than can be avoided with a feedback based stream delivery graph construction algorithm. To our knowledge, our work is the first in-depth study of the stream delivery problem in this context, which considers resource limited scenarios and supports QoS-aware stream delivery.

Our work is to some extent related to research in publish/subscribe systems, as well as application-level multicast. However, most of the publish/subscribe systems, such as SemCast [18], Grypon [5], or Siena [6], are designed for networks that contain a set of infrastructure nodes (brokers) responsible for event forwarding and/or filtering. Our work builds upon a different system model, one based on P2P principles, in which nodes that are consumers of a stream cooperate among themselves to accomplish stream delivery. This system model is more suitable for large-scale distributed stream processing systems where end nodes of the system are query processors that need access to distributed streams for executing their local CQs. This P2P nature of the system model is shared with many of the works on application-level multicast, such as Scribe [8] and SplitStream [7]. However, none of these works dealt with QoS semantics of CQ applications that can tolerate some level of delays and drops in stream delivery.

It is also interesting to mention that, although not discussed in this paper, the problem of source discovery can be handled in our system using distributed hash table based techniques employed by some of the application-level multicast systems [8].

8. CONCLUSION

With rapid advances in sensing and wireless communication technologies, continuous data stream delivery in distributed overlay networks has gained increasing interest over the past few years. We presented and formulated the problem of distributed data stream delivery for continuous query services as an optimization problem. We first identified the critical system parameters that need to be taken into account in the optimization algorithms through the study of the basic formulation of our problem under the simplistic assumption that a valid stream delivery graph can always be found. Then we relaxed our basic definition and formulation of the stream delivery problem to handle the resource limited scenarios where unwanted drops and delays in stream delivery are unavoidable. We promoted the use of quality-aware stream delivery techniques to better optimize the overall system performance. By attaching QoS functions to filters, we made quality-aware stream delivery possible. These QoS functions specify the level of quality resulting from introducing certain amount of delays and drops in serving a filter, enabling us to construct delivery graphs in which resources are better utilized to handle the delay and drop sensitive filters. In addition to introducing several baseline algorithms, we presented feedback-based QASDS and QASDG algorithms that make use of simple yet effective heuristics and are easy to implement in practice. We demonstrated through extensive simulations that QASD-based algorithms can provide high overall quality in stream delivery with small bandwidth consumption and good balance in forwarding responsibilities.

9. REFERENCES

[1] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, R. Motwani, I. Nishizawa, U. Srivastava, D. Thomas, R. Varma, and J. Widom. STREAM: The stanford stream data manager. *IEEE Data Engineering Bulletin*, 26, 2003.

[2] B. Babcock, M. Datar, and R. Motwani. Load shedding for aggregation queries over data streams. In *IEEE ICDE*, 2004.

[3] H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, E. Galvez, J. Salz, M. Stonebraker, N. Tatbul, R. Tibbetts, and S. Zdonik. Retrospective on Aurora. *VLDB Journal*, 2004.

[4] M. Balazinska, H. Balakrishnan, S. Madden, and M. Stonebraker. Fault tolerance in the borealis distributed stream processing system. In *ACM SIGMOD*, 2005.

[5] G. Banavar, T. D. Chandra, B. Mukherjee, J. Nagarajarao, R. E. Strom, and D. C. Sturman. An efficient multicast protocol for content-based publish-subscribe systems. In *IEEE ICDCS*, 1999.

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

[7] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth content distribution in cooperative environments. In *ACM SOSP*, 2003.

[8] 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 Communications (JSAC)*, 20(8), 2002.

[9] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, V. Raman, F. Reiss, and M. A. Shah. TelegraphCQ: Continuous dataflow processing for an uncertain world. In *CIDR*, 2003.

[10] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. Niagaracq: A scalable continuous query system for internet databases. In *ACM SIGMOD*, 2000.

[11] N.-A. Q. P. for Stream-based Applications. Yanif ahmad and ugur cetintemel. In *VLDB*, 2004.

[12] 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.

[13] J.-H. Hwang, M. Balazinska, A. Rasin, U. Cetintemel, M. Stonebraker, and S. B. Zdonik. High-availability algorithms for distributed stream processing. In *IEEE ICDE*, 2005.

[14] R. Kuntschke, B. Stegmaier, A. Kemper, and A. Reiser. StreamGlobe: Processing and sharing data streams in grid-based p2p infrastructures. In *VLDB*, 2005.

[15] D. T. Liu and M. J. Franklin. The design of GridDB: A data-centric overlay for the scientific grid. In *VLDB*, 2004.

[16] L. Liu, C. Pu, and W. Tang. Continual queries for internet scale event-driven information delivery. *IEEE Transactions on Data and Knowledge Engineering*, July/August 1999.

[17] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, July 1998.

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

[19] B. Stegmaier, R. Kuntschke, and A. Kemper. StreamGlobe: Adaptive query processing and optimization in streaming p2p environments. In *Internatinal Workshop on Data Management for Sensor Networks*, 2004.

[20] Streambase systems. http://www.streambase.com/, May 2005.

[21] N. Tatbul, U. Cetintemel, S. Zdonik, M. Cherniack, and M. Stonebraker. Load shedding in a data stream manager. In *VLDB*, 2003.

[22] D. Terry, D. Goldberg, D. Nichols, and B. Oki. Continuous queries over append-only databases. In *ACM SIGMOD*, 1992.

[23] S. A. Vavasis. Quadratic programming is in NP. *Information Processing Letters*, 36(2), 1990.

[24] Y. Xing, S. Zdonik, and J.-H. Hwang. Dynamic load distribution in the borealis stream processor. In *IEEE ICDE*, 2005.