A General Algorithmic Model for Subscription Propagation and Content-based Routing with Delivery Guarantees

Yuanyuan Zhao yuanyuan@us.ibm.com

Sumeer Bhola sbhola@us.ibm.com

Daniel Sturman sturman@us.ibm.com

IBM T.J. Watson Research Center

Abstract

This paper examines the problem of subscription propagation and aggregation in content-based publish/subscribe systems. We present a general model for subscription propagation and content-based routing and the correctness criteria for it to support reliable delivery. This formal model is used to construct a generic subscription propagation algorithm for asynchronous content-based routing networks with redundant paths and failures. This algorithm does not require agreement between multiple paths in the network and hence is efficient and highly available.

We analyze the safety aspects of the generic algorithm and interpret many previously known algorithms as different encodings of the generic algorithm, with various space optimizations for different circumstances. The algorithm is applicable to both best-effort and reliable delivery of messages.

Keywords: publish/subscribe, content-based routing, redundant overlay networks, high availability, distributed algorithms

Submission category: Regular paper

Total pages: 10 + appendix

Not eligible for Best Student Paper Award

Can be considered as a brief announcement

Contact author:

Yuanyuan Zhao

IBM T.J. Watson Research Center

P.O. Box 704, Yorktown Heights, N.Y. 10598

Phone: 914-784-7176, Fax: 914-784-6807

Email: yuanyuan@us.ibm.com

1 Introduction

Content-based publish/subscribe messaging is a popular paradigm for building asynchronous distributed applications. A publish/subscribe system consists of publishers that generate messages and subscribers that register interest in all future messages matching the predicate specified in their subscription. The system, implemented as a network of routing brokers, is responsible for routing published messages to interested subscribers. Information providers and consumers are decoupled, since publishers need not be aware of which subscribers receive their messages, and subscribers need not be aware of the sources of the messages they receive.

Subscription propagation is a mechanism of propagating subscribers' interest throughout the broker network. This interest is usually expressed through Boolean expressions called *filters*. This allows brokers to filter out and withhold from sending messages to parts of the network where there are no interested subscribers. This functionality is hence very important for efficiency and scalability of content-based pub/sub systems. However, the task of designing a subscription propagation algorithm is greatly challenged by several factors, especially: 1) clients' requirement of strong service guarantees such as in-order, gapless delivery [1] (referred to as reliable delivery in the rest of the paper); 2) the existence of multiple routing paths between publishers and subscribers; 3) communication asynchrony, especially asynchrony among multiple redundant paths; 4) failures.

As a result of these challenges, most previous work on subscription propagation [2, 5, 6, 3, 12] did not provide a solution that guarantees the correctness of content-based routing and hence is not capable of supporting reliable delivery in the presence of failures and multiple paths. It is not until recently in [14] a set of subscription propagation algorithms were designed using virtual time vectors to guarantee correctness of routing decisions. We think this situation is due to lack of understanding of the fundamentals of the subscription propagation problem. There is no coherent theory of how subscription propagation algorithms should work in general. As a result, designing subscription propagation algorithms typically becomes isolated activities each dealing with different situations.

This paper explores the structure of the subscription propagation problem and defines a general model for subscription propagation and content-based routing. Under this model, we define the correctness criteria of subscription propagation and content-based routing and a set of sufficient conditions for supporting reliable delivery. The formal model allows us to construct a generic subscription propagation algorithm and can serve as a basis for analyzing existing subscription propagation algorithms and providing support for future algorithm design. The generic algorithm supports reliable delivery in the presence of multiple routing paths, broker and link failures, and communication asynchrony without requiring expensive distributed agreement between redundant paths. It provides high network availability and efficiency by allowing data message routing to choose any of the redundant paths. We also present a refinement of the generic algorithm that utilizes subscription aggregation. Many existing algorithms can be interpreted as specializations of the generic algorithm under different circumstances.

The rest of the paper is organized as follows. Section 2 describes a topology model of redundant routing networks and background information on content-based routing and reliable delivery. Section 3 describes the general model for subscription propagation and content-based routing and the correctness criteria of subscription propagation algorithms. Section 4 describes a generic subscription propagation algorithm and Section 5 discusses the generic algorithm extended with subscription aggregation. Section 6 describes a number of previously known algorithms as specializations of the generic algorithm. Section 7 discusses related work and we conclude in Section 8.

2 Topology, Content-based Routing & Reliable Delivery

We describe a topology model of redundant routing networks and content-based routing and reliable delivery in such networks.

2.1 Routing Topology

We adopt a topology model of spanning trees of *nodes* where each node includes multiple *brokers*, which are redundant and can work interchangeably. Trees are non-cyclic structures that simplify the task of loop-free routing and tree nodes with redundant brokers provide high availability. The concept of *virtual brokers* ([1]) allows the same physical broker to appear in more than one tree node while preserving an efficient implementation.

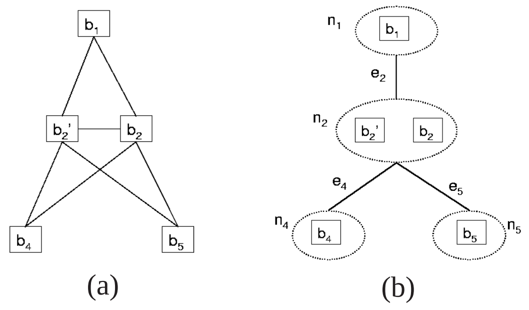

Figure 1: Redundant Routing Networks

We refer to a virtual broker where publishers connect as a publisher hosting broker (PHB) and a virtual broker where subscribers connect as a subscriber hosting broker (SHB). For simplicity, we discuss our work from the standpoint of one PHB. As a result, we can assume, without loss of generality, that SHBs reside in the leaf nodes of the tree; and there is only one PHB and the node it resides in is designated as the root of the tree. Because a client connects to one broker, we assume each leaf node contains one broker. For the purpose of this paper, we do not need to distinguish virtual from physical and refer only to brokers.

This topology model is sufficient to represent a large range of practical topologies and a large number of graphs as one can transform a graph (e.g., Figure 2.1(a)) with redundant paths into a topology (Figure 2.1(b)) under this model by grouping brokers into tree nodes and inter-broker links into tree edges.

We refer to where the PHB resides as upstream and direct the edges to point from upstream to downstream. We also refer to the edges of a tree node as the edges of brokers in the tree node.

2.2 Content-based Routing and Reliable Delivery

A valid implementation of content-based subscription can be one in which the PHB and intermediate brokers forward all published messages to SHBs, and only SHBs apply filtering. Such a solution will be a perfectly correct implementation, but it may waste considerable bandwidth sending messages that will be later discarded. Subscription propagation is an optimization which may result in fewer wasted messages being sent to SHBs in exchange for requiring the PHB and intermediate brokers to perform filtering and to acquire knowledge about subscription predicates. Since subscription propagation is an optimization, it should preserve the correctness properties of the original specification.

Data messages enter a pub/sub system through the PHB at the root of the redundant routing tree. The PHB maintains a stream for data messages published by a publisher. A data message flows downstream toward SHBs in the leaf nodes. When the message arrives at a broker, the broker performs content matching for each of its outgoing edges. If the data message does not match any subscription in the downstream of the edge, the data message can be filtered out for the downstream of the edge. Otherwise, the broker sends the data message to a broker in the child node on the other end of the edge. When the data message arrives at a SHB, the SHB finds the matching subscriptions and delivers the data message to the corresponding subscribers.

The data messages routed by a pub/sub system may be subject to various levels of service guarantees such as best effort or reliable delivery. Reliable delivery guarantees that for a subscription s and a published message stream, the pub/sub system finds a starting message and an ending message (upon the unsubscription request of s), and delivers all messages in the range of [starting point, ending point] that match s in an order conforming to the original stream. We require that the starting point be chosen within a finite amount of time if the system has bounded latency and runs without failure for sufficient amount of time. This excludes trivial algorithms such as one that delays choosing the starting point until it sees the unsubscription request, and delivers no message.

Providing reliable delivery is a challenging task in a content-based system deployed over a network with redundant paths. Due to content-based routing, gaps can not be detected by traditional methods such as publisher-assigned sequence numbers because each subscriber may request a completely unique sequence of messages to be delivered. Reliability in a content-based system hence requires brokers on the routing path to assist in gap detection ([1]).

Multiple paths, communication asynchrony and failures complicate subscription propagation as redundant brokers on alternative routes may have subscription information that is different from each other's. If messages from the same published stream are routed through those brokers, they are matched to different sets of subscriptions and gaps can appear in the message sequences delivered to subscribers.

3 General Model for Subscription Propagation and Content-based Routing

In this section, we introduce a general model for subscription propagation and content-based routing. We present the correctness criteria of subscription propagation and a set of sufficient conditions for correct content-based routing and reliable delivery.

3.1 Notations

In a pub/sub system, the brokers route messages for subscriptions that are submitted by subscribers at the leaf nodes and can stop routing messages for subscriptions that are unsubscribed. The sets of subscriptions that have been subscribed (denoted as $\mathbb{S}$) or unsubscribed (denoted as $\overline{\mathbb{S}}$) at any point of time in the whole system are monotonically increasing. This global knowledge is distributed among the leaf node brokers and can only be known to an oracle. We introduce these concepts for correctness reasoning. They are not maintained by brokers.

We assume a subscription has a unique identity and a filter in the form of a disjunction/set of conjunctions. This does not limit the power of our model as every Boolean expression can be transformed into disjoint normal form (DNF). We use the mathematical symbol λ to denote each individual conjunction and Λ to denote a conjunction set. We use operator $conj(s)$ to denote the conjunction set of a subscription s. Even though multiple subscriptions can have the same set of conjunctions, the identities are different. We also assume that a subscription can only enter the system through one leaf node in the spanning tree.

The routing brokers maintain local knowledge of $\mathbb{S}$ and $\overline{\mathbb{S}}$ in the form of subscription identities as well as a set of conjunctions for purpose of content matching and routing for these subscriptions. This local knowledge is accumulated as subscription changes are propagated. In an asynchronous system, this local knowledge usually lags behind the real global values at various degrees in different brokers. They are subsets of $\mathbb{S}$ and $\overline{\mathbb{S}}$.

Definition 3.1. *For broker b and its outgoing edge e, b's* **matching set of subscriptions** *$S(e)$ represents its knowledge of the set of client subscriptions that are downstream of edge e. Similarly, b's* **matching set of unsubscriptions** *$\overline{S}(e)$ represents its knowledge of the set of client unsubscriptions in the downstream of e.*

Definition 3.2. *For broker b and its outgoing edge e, b's* **matching set of conjunctions** *$\Lambda(e)$ are the set of conjunctions b maintains to match and route data messages to the downstream of e.*

As local knowledge, $S(e)$ and $\overline{S}(e)$ are subsets of $\mathbb{S}$ and $\overline{\mathbb{S}}$. It is not necessary for a broker to explicitly maintain the list of subscriptions in $S(e)$ and $\overline{S}(e)$ as shown in Section 6.

3.2 Correctness of Subscription Propagation

Intuitively, a subscription propagation algorithm is correct if the set of conjunctions ($\Lambda(e)$) each broker uses for matching and routing is sufficient for the set of subscriptions ($S(e) - \overline{S}(e)$) it routes message for. We now formally define this sufficiency requirement after introducing the concepts of *covering* of conjunctions, conjunction sets and subscription sets.

Definition 3.3. *Conjunction λ_2 covers λ_1, denoted as $\lambda_2 \succeq_c \lambda_1$, if and only if $M(\lambda_2) \supseteq M(\lambda_1)$, where $M(\lambda)$ is the set of all messages matching λ. Conversely, λ_1 is covered by λ_2, denoted as $\lambda_1 \preceq_c \lambda_2$.*

Definition 3.4. *Conjunction set Λ_2 covers Λ_1, denoted as $\Lambda_2 \sqsupseteq_c \Lambda_1$, if and only if $\bigcup_{\lambda_2 \in \Lambda_2} M(\lambda_2) \supseteq \bigcup_{\lambda_1 \in \Lambda_1} M(\lambda_1)$*

Definition 3.5. *Conjunction set Λ covers subscription set S if and only if $\Lambda \sqsupseteq_c \bigcup_{s_i \in S} conj(s_i)$.*

We use the same operator $\sqsupseteq_c$ for the covering relationship between a conjunction set and a subscription set.

The correctness of subscription propagation is an invariant each broker should maintain.

Invariant 1. *For every broker b and its outgoing edge e, the $\Lambda(e)$ set of b covers $S(e) - \overline{S}(e)$, that is, $\Lambda(e) \sqsupseteq_c S(e) - \overline{S}(e)$. We call this the S-Λ invariant.*

3.3 Content-based Routing Algorithm

A data message m flows from the root to the leaves in the routing tree. It carries a *subscription header* that is initially set by the PHB equal to $S(e_i)$ it maintains for outgoing edge e_i. This header identifies the set of subscriptions the message should be matched against. We call this the *matching set of subscriptions* of m, denoted as $S(m)$. A broker b on the routing path of m can modify $S(m)$ by adding subscriptions (in b's matching set of subscriptions) to $S(m)$. Adding a subscription s to

$S(m)$ is subject to the requirement that $conj(s)$ must be covered by the conjunctions of subscriptions in the original $S(m)$.

Each tree node has one incoming edge, and one or more outgoing edges. When a broker b within a node receives m through its incoming edge e, it separates out the subscriptions in $S(m)$ that are relevant to each of its outgoing edges. This can be achieved by associating each subscription entity with information on where it subscribes. We call this a *projection* of the $S(m)$ set on the outgoing edge, and denote it as $S(m, e_i)$, where e_i is an outgoing edge.

The broker then checks whether it has knowledge of all the *active* subscriptions in $S(m, e_i)$. An *active* subscription is one that has not unsubscribed, i.e., not in $\overline{\mathbb{S}}$. Hence a *sufficiency test* is performed for $S(m, e_i) - \overline{\mathbb{S}}$.

Definition 3.6. *A broker b is sufficient for message m on outgoing edge e_i if $S(m, e_i) - \overline{\mathbb{S}} \subseteq S(e_i) - \overline{S}(e_i)$.*

As mentioned before, $\overline{\mathbb{S}}$ may only be known to an oracle. However, since $\overline{S}(e_i)$ is always a subset of $\overline{\mathbb{S}}$, $S(m, e_i) - \overline{\mathbb{S}} \subseteq S(m, e_i) - \overline{S}(e_i)$. The sufficiency test can be replaced by a stricter test at broker b: $S(m, e_i) - \overline{S}(e_i) \subseteq S(e_i) - \overline{S}(e_i)$.

If there is a subscription s such that $s \in S(m, e_i)$ but $s \notin S(e_i)$ and $s \notin \overline{S}(e_i)$, the sufficiency test fails for edge e_i. This indicates that broker b lacks information on s. In this case, b forwards m to a broker at the other end of edge e_i.

If the sufficiency test passes for edge e_i, broker b matches m against $\Lambda(e_i)$. If there is a match, m is forwarded to a broker at the other end of e_i. If there is no match, b may filter m out. As we mentioned before, providing reliable delivery in a content-based system requires brokers on the routing path to assist in gap detection, hence broker b needs to forward information on data message m to indicate that it is filtered out to distinguish it from a gap caused by message reordering or losses. We refer to a filtered out message as a *silence* message. A silence message usually incurs less communication overhead than the original data message m and may be combined with adjacent silence or data messages such as described in [1]. However, for the purpose of this paper, we do not discuss such optimizations.

We conclude that a correct content-based routing algorithm is one where each broker only filters out a message for an outgoing edge if the sufficiency test passes and there is no match. In support of reliable delivery, such a content-based routing algorithm forwards a *silence* message in place of a filtered message. We call such an algorithm *sufficiency-directed content-based routing*.

3.4 Providing Reliable Delivery

We first address the issue of selecting delivery starting/ending points of a subscription s.

Let b be the SHB of s. We define the delivery starting point of s as the first message m_1 that b receives such that $s \in S(m_1)$. We define the delivery ending point of s as the last message m_n with $s \in S(m_n)$ that b receives before it receives the unsubscription request of s.

Due to PHB failures, a message m (or a silence for m if filtered) after m_1 may be forwarded with $S(m)$ such that $s \notin S(m)$. This can also occur if a broker on the routing path fails or a message takes a different routing path as a routing broker can change $S(m)$ (Section 3.3). In this case, b discards m (or the silence) and waits for a retransmission of the message with the right $S(m)$.

We hence define *eventual monotonicity* on $S(m)$ of messages.

Definition 3.7. *The $S(m)$ set of messages of a published stream in range $[m_1, m_2]$ is eventually monotonic with regard to subscription s if the SHB of s eventually receives those messages (or corresponding silences) with $S(m)$ such that $s \in S(m)$.*

We then present a theorem of sufficient conditions for correct reliable delivery. We give an informal reasoning of the theorem in Appendix A.

Theorem 1. *Reliable delivery of a subscription s can be guaranteed if the following conditions are satisfied:*
1. (Correctness of subscription propagation) Every broker in the pub/sub system maintains S-Λ invariant.
2. (Correctness of content-based routing) Routing brokers use sufficiency-directed content-based routing.
3. If there is a sufficiently long period of time for which the system runs without failure, newly published messages (or silences) start to arrive at the SHB of s with $S(m)$ such that $s \in S(m)$ unless the system fails again. Delivery starting and ending points of s can be chosen in the aforementioned way.
4. The system guarantees eventual monotonicity of $S(m)$ for messages in [starting point, ending point] and the

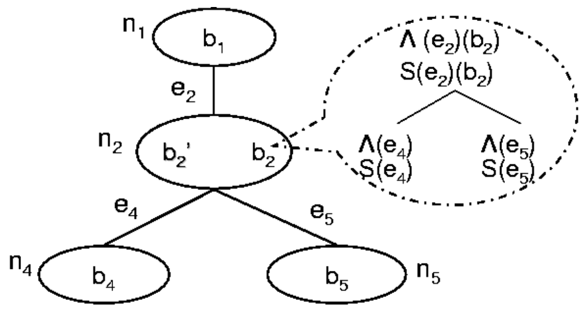

Figure 2: Binary Redundant Spanning Tree

SHB only accepts the transmission of messages with monotonic $S(m)$.

In summary, we have addressed the issues of correct content-based routing and selecting delivery starting/ending points. We also note here that eventual monotonicity can be guaranteed using a liveness scheme such as negative acknowledgement (indicating s is needed to be in $S(m)$) and retransmission. In next two sections, we discuss the issue of maintaining S-Λ invariant.

4 Generic Subscription Propagation Algorithm

Pub/sub systems are usually dynamic with subscriptions entering and leaving. As a result, the members in the $S(e)$, $\overline{S}(e)$ and $\Lambda(e)$ sets change. In this section, we describe a generic subscription propagation algorithm. We focus on the safety aspect, that is, the maintainence of S-Λ invariant in such a dynamic environment.

We first describe the information brokers maintain and exchange during the subscription propagation process and then how this information is computed.

Without loss of generality, we consider a directed redundant spanning tree where every non-leaf node has exactly two children. Figure 2 shows a partial topology that is of interest for examining broker b_2, which resides in node n_2 with a redundant peer b_2'. Node n_2 is connected to two of its children through edge e_4 and e_5 and parent node n_1 through edge e_2.

4.1 Subscription Information Maintenance and Exchange

For broker b_2 to route messages to subscribers that are downstream of edge e_4 and e_5, b_2 maintains the following information:

- Set of *conjunctions* $\Lambda(e_4)$ and $\Lambda(e_5)$ for matching messages before sending on edges e_4 and e_5;
- Set of *subscriptions* $S(e_4)$ and $S(e_5)$;
- Set of *unsubscriptions* $\overline{S}(e_4)$ and $\overline{S}(e_5)$.
- Presumed knowledge of the state for edge e_2 maintained by a broker (e.g., b_1) in b_2's upstream node n_1. This includes the presumed conjunction set, $\Lambda(e_2)(b_2)$, the presumed subscription and unsubscription sets $S(e_2)(b_2)$ and $\overline{S}(e_2)(b_2)$. Note that we have included (b_2) in the suffix to distinguish the presumed state from the actual state.

In an asynchronous distributed system, there is no guarantee that the real state in b_1 is exactly as b_2 presumes. This state is computed at b_2 as follows:

$$S(e_2)(b_2) = S(e_4) \cup S(e_5) \tag{4.1}$$
$$\overline{S_2}(e)(b_2) = \overline{S}(e_4) \cup \overline{S}(e_5) \tag{4.2}$$
$$\Lambda(e_2)(b_2) = \Lambda(e_4) \cup \Lambda(e_5) \tag{4.3}$$

When subscription changes happen, the SHB generates information about the change (described in Section 4.2). This information propagates upstream and enters broker b_2 through the reverse direction of its outgoing edges e_4 and e_5. The propagation message, e.g. on edge e_4, includes the following items

- $\Delta S(e_4)$ represents the new subscriptions;
- $\Delta\overline{S}(e_4)$ represents the subscriptions that have been unsubscribed;
- $\Delta\Lambda(e_4)$ represents the conjunctions that should be added or removed if $\Delta S(e_4)$ and $\Delta\overline{S}(e_4)$ are to be applied. The elements of $\Delta\Lambda(e_4)$ take the form of "$+\lambda$" or "-λ";
- constraint set $C(e_4)$ of Boolean expressions represents the conditions on $S(e_4)$ and $\overline{S}(e_4)$ at b_2 in order for it to apply the aforementioned changes.

When b_2 receives the propagation message, it evaluates $C(e_4)$. If $C(e_4)$ is not satisfied, it ignores the message. If $C(e_4)$ is satisfied, b_2 applies the Δ's to $S(e_4)$, $\overline{S}(e_4)$ and $\Lambda(e_4)$. Applying $\Delta S(e_4)$ and $\Delta\overline{S}(e_4)$ is to take the set union with $\S(e_4)$ and $\overline{S}(e_4)$ respectively. Applying $\Delta\Lambda(e_4)$ means adding a conjunction "$+\lambda$" or removing a conjunction "-λ" if it is in $\Lambda(e_4)$.

After applying the Δ's, broker b_2 computes subscription changes for brokers in its upstream node n_1. The items include $\Delta S(e_2)$, $\Delta\overline{S}(e_2)$, $\Delta\Lambda(e_2)$ and $C(e_2)$ and are computed from the changes to the presumed state $S(e_2)(b_2)$,$\overline{S}(e_2)(b_2)$ and $\Delta\Lambda(e_2)(b_2)$. If $\Delta\Lambda(e_2) = \emptyset$, broker b_2 has the option not to propagate the subscription further. Otherwise, b_2 sends the subscription message to b_1 through the reverse direction of its incoming edge e_2. The subscription message may also be routed to other brokers (if any) in n_1 following any path provided by the underlying routing topology. For instance, if b_1 has a redundant peer connected, it may receive the subscription message from b_1. However, our algorithm do not rely on any synchrony between the brokers in a node.

4.2 Computing Subscription Change for an Upstream Broker

This section describes the computation of incremental subscription changes at broker b_2 for brokers in its upstream node n_1.

Computing $\Delta S(e_2)$ and $\Delta\overline{S}(e_2)$ Since each subscription and its corresponding unsubscription enter the spanning tree at only one point, $S(e_4) \cap S(e_5) = \emptyset$ and $\overline{S}(e_4) \cap \overline{S}(e_5) = \emptyset$. Using Formula 4.1 and 4.2, we have $\Delta S(e_2) = \Delta S(e_4)$ and $\Delta\overline{S}(e_2) = \Delta\overline{S}(e_4)$.

Computing $\Delta\Lambda(e_2)$ and $C(e_2)$ Broker b_2 computes incremental change $\Delta\Lambda(e_2)$ from $\Delta\Lambda(e_4)$ and b_2's current presumed state of an upstream broker - $\Lambda(e_2)(b_2)$. Depending on the aggregation scheme used, this may generate one or more constraints. Constraint $C(e_2)$ includes these new constraints and the constraints in $C(e_4)$. We do not restrict the generic algorithm to any specific aggregation scheme. Instead, we only require that the aggregation scheme computes constraint set $C(e_2)$ and $\Delta\Lambda(e_2)$ such that if a broker initially satisfies the S-Λ invariant and applies $\Delta\Lambda(e_2)$ only when the constraints are satisfied, the broker maintains the S-Λ invariant.

An aggregation scheme can be one that does not aggregate the conjunctions. The constraint set is always $\emptyset$. In such a system, the subscription changes received at SHBs are flooded to all brokers without change and brokers maintain all subscription conjunctions. We describe a more sophisticated aggregation scheme in Section 5.

5 Subscription Propagation with Aggregation

The aggregation of $\Lambda(e_4)$, $\Lambda(e_5)$ and $\Delta\Lambda(e_4)$ at broker b_2 produces the conjunction changes $\Delta\Lambda(e_2)$ and constraints $C(e_2)$ that are part of the incremental subscription change the broker sends upstream. In this section, we describe an abstract scheme for aggregating these conjunctions based on their covering relationships. Our purpose is to provide a foundation for analyzing the family of covering-based aggregation schemes.

5.1 Aggregating Conjunctions

The fact that covering conjunctions matches all messages of their covered conjunctions allows a broker to withhold from propagating changes of conjunctions that are covered by one or more conjunctions that the broker has already propagated and still guarantees correct content filtering. That is, $\Delta\Lambda(e)$ (for any edge e) needs to include only conjunctions that are not covered. For the same reason, a broker only needs to maintain the non-covered conjunctions in its conjunction set $\Lambda(e)$.

As previously described, broker b_2 applies $\Delta\Lambda(e_4)$ to obtain a new state of $\Lambda(e_2)(b_2)$ and $\Delta\Lambda(e_2)$ is computed as the conjunction change in $\Lambda(e_2)(b_2)$. We organize the conjunctions in $\Lambda(e_2)(b_2)$ into a DAG for easy representation of the covering relationships. The DAG nodes are conjunctions with edges drawn from a covering conjunction (a parent) to each of its covered conjunction (a child). The roots of the DAG are the set of conjunctions that are not covered and hence should be propagated. We examine the changes in the DAG resulted from applying $\Delta\Lambda(e_4)$. Incremental change $\Delta\Lambda(e_2)$ and $C(e_2)$ thus represent the change to the root set and the conditions under which it happened.. The initial value of $\Delta\Lambda(e_2)$ is an empty set $\emptyset$, whereas the initial value of $C(e_2)$ is set to $C(e_4)$ because the computation is based on b_2 satisfying constraints $C(e_4)$.

We first introduce a notation of $sub(\lambda)$ as the set of subscriptions in $S(e) - \overline{S}(e)$ over all edges at a broker such that $\lambda \in conj(s)$ for each s in this set.

Adding a Root Conjunction Figure 3(a) illustrates the case of adding a conjunction λ that will be a root in the new conjunction DAG of $\Lambda(e_2)(b_2)$.

The new root conjunction λ may cover zero or more conjunctions. As a newly created root in the DAG of $\Lambda(e_2)(b_2)$, "$+\lambda$" should be propagated and thus belongs

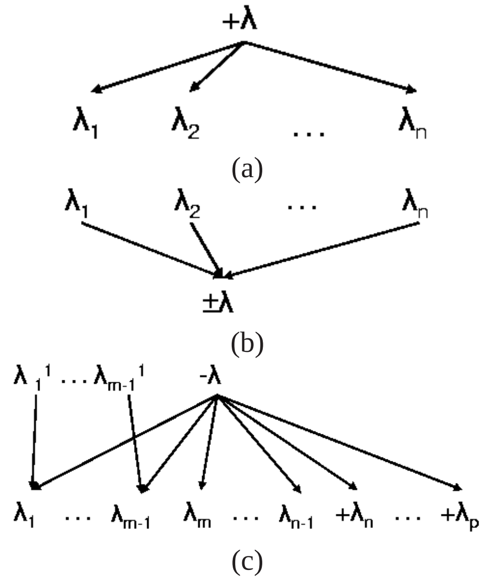

Figure 3: Applying Incremental Changes $\Delta\Lambda_m$ in b_2

to $\Delta\Lambda(e_2)$. Since λ is propagated, no new constraint is created.

Adding or Removing a Non-Root Conjunction Figure 3(b) describes the cases when a non-root conjunction is added or removed while it is covered by existing conjunctions that will continue to exist in the new DAG of $\Lambda(e_2)(b_2)$. Since λ is not a root conjunction, it does not need to be propagated and thus this results in no change to $\Delta\Lambda(e_2)$.

On the other hand, the withholding of λ is based on the existence of at least one of its covering conjunctions. Since the constraints represent requirement on an upstream broker's matching sets of subscriptions and unsubscription, this is to require the existence of at least one subscription in $\bigcup_{i=1}^{n} sub(\lambda_i)$. As a result, the following constraint is added to $C(e_2)$

$$\bigvee_{s\in\bigcup_{i=1}^{n} sub(\lambda_i)} s \in S(e_2) - \overline{S}(e_2) \qquad \text{(I)}$$

Removing a Root Conjunction Figure 3(c) describes the case when an existing root conjunction λ is removed and λ is not covered by any new conjunction that is added to the DAG. Conjunction $\lambda_{1..n-1}$ represent the set of conjunctions that are directly covered by λ. Conjunction $\lambda_{1..m-1}(m \leq n)$ represents those conjunctions in $\lambda_{1..n-1}$ that are also covered by other existing conjunctions. For example, λ_1 is covered by λ_1^1. For simplicity, we only show one covering conjunction for each of $\lambda_{1..m-1}$. Conjunction $\lambda_{n..p}$ represent newly added conjunctions that are covered by λ.

As a removed root, λ does not belong to the new DAG of $\Lambda(e_2)(b_2)$ and "$-\lambda$" should be added to $\Delta\Lambda(e_2)$. Because $\lambda_{m..n-1}$ become new root conjunctions, we add "$+\lambda_{m..n-1}$" to $\Delta\Lambda(e_2)$. Note that newly added root conjunction $\lambda_{n..p}$ will be handled as described in "Adding a Root Conjunction". To summarize, $\Delta\Lambda(e_2)$ is updated as follows:

$$\Delta\Lambda(e_2) \leftarrow \Delta\Lambda(e_2) \cup \{-\lambda\} \cup \bigcup_{i=m}^{n-1}\{+\lambda_i\}$$

Due to an upstream broker b_1 not maintaining covered conjunctions, all conjunctions of subscriptions in b_1 that were covered by λ should be restored if they are not covered by other conjunctions in $\Lambda(e_2)$. Adding "$+\lambda_{m_{n-1}}$" to $\Delta\Lambda(e_2)$ partly satisfies this requirement. However, if there is a subscription s in b_1 that is unknown to b_2 such that one or more conjunctions of s are covered by λ, then the upstream broker may violate the S-Λ invariant. To guarantee that such a case does not happen, we make sure there is no subscription in b_1 that is unknown to b_2 through the following constraint

$$S(e_2) - \overline{S}(e_2) \subseteq S(e_2)(b_2) - \overline{S}(e_2)(b_2) \qquad \text{(II)}$$

The withholding of $\lambda_{1..m-1}$ in Figure 3(c) is based on the existence of at least one of their covering conjunctions (represented by variable λ_t in the following formula) in broker b_1. This is represented by the following constraint for *each* λ_i where $i = 1..m-1$:

$$\bigvee_{s\in\left(\bigcup_{(\lambda_t \succeq_c \lambda_i)} sub(\lambda_t)\right)} s \in S(e_2) - \overline{S}(e_2) \qquad \text{(III)}$$

This type (III) of constraints are the same as type I. They are only caused by different situations.

As we describe in Section 5.2, constraints of type I II and III are sufficient to guarantee the S-Λ invariant. For efficiency of content-based routing, it is essential to guarantee that if a broker b_1 passes the sufficiency test against a message m for an outgoing edge e_2, b_1 will filter m if m does not match any subscription in $S(e_2) - \overline{S}(e_2)$. That is, subscription propagation needs to prevent from adding conjunctions that do not belong to any subscription in $S(e_2) - \overline{S}(e_2)$. This is represented by the following constraints for each λ_i in $\lambda_{m..n-1}$ that is added to $\Delta\Lambda(e_2)$:

$$\bigvee_{s\in sub(\lambda_i)} s \in S(e_2) - \overline{S}(e_2) \qquad \text{(IV)}$$

5.2 Operation and Constraint Properties

If a pub/sub system only uses the aforementioned operations and constraints, every broker will maintain the S-Λ

invariant. The following theorem is a formal statement of this property. Its proof appears in Appendix B.

Theorem 2. *For any broker b and an outgoing edge e in a redundant spanning tree, if the $S(e)$, $\overline{S}(e)$ and $\Lambda(e)$ sets of b initially satisfy the S-Λ invariant and are only modified by the operations and constraint type I, II and III described in Section 4.2, the broker maintains S-Λ invariant.*

The use of type IV constraints guarantees that the broker does not maintain extra conjunctions other than those needed for performing content matching for the subscriptions it maintains. The following theorem is a formal statement of this property. Its proof appears in Appendix C.

Theorem 3. *We consider broker b and its outgoing edge e in a network using operation and all types of constraints described described above. For every conjunction $\lambda \in \Lambda(e)$, there exists a subscription s such that $\lambda \in conj(s)$ and $s \in S(e) - \overline{S}(e)$.*

6 Subscription Propagation Protocols

In this section, we describe several existing practical protocols as encodings of the generic algorithm.

6.1 Single-Path Spanning Tree Algorithm

This type of algorithm is used in systems like Siena [2] and REBECA [6]. In these systems, subscription changes are propagated along a single path from a leaf to the root in the spanning tree. The system does not assume a subscription is established and start delivery for it until all brokers on the path from the subscribing leaf node to the root have confirmed (explicitly or implicitly) receiving the subscription. The system guarantees the S-Λ invariant in the absence of message losses (inter-broker links are assumed reliable) and when there is no information loss at a broker due to failure. Given these restrictions on failures, the algorithm can optimize many aspects of the generic algorithms.

We reuse the broker topology in Figure 2 but modify it such that broker b_2 does not have a redundant peer b'_2 or the system ignores the route through b'_2. Assume $S(e_2)(b_2)$ and $\overline{S}(e_2)(b_2)$ are broker b_2's presumed subscription state for broker b_1 before a subscription change SC_2. Broker b_2 then updates these values and computes a subscription change SC_1 to send to b_1. Because there is only a single path from a leaf node to the root where the publisher hosting broker resides, and there is no information loss due to failures, we have

$S(e_2) \equiv S(e_2)(b_2) - S_1$, $\overline{S}(e_2) \equiv \overline{S}(e_2)(b_2) - S_2$,

where $S(e_2)$, $\overline{S}(e_2)$ are the state of broker b_1 just before applying SC_1, and S_1 and S_2 are subscriptions whose conjunctions are covered by $S(e_2) - \overline{S}(e_2)$. That is, $S(e_2)$ and $\overline{S}(e_2)$ are subsets of $S(e_2)(b_2)$ and $\overline{S}(e_2)(b_2)$ and b_1 maintains all root conjunctions in b_2's $\Lambda(e_2)(b_2)$. Thus, the constraints are always satisfied and do not need to be explicitly sent.

The PHB sets the $S(m)$ of a data message m as "the active subscriptions the PHB maintains". This $S(m)$ can be modified at a broker b if b withholds from propagating a subscription s because s was covered. In this case, b can add s into $S(m)$. Since the system does not assume a subscription is established until it is confirmed (explicitly or implicitly) by all brokers on the path from the leaf node to the root, the sufficiency test against $S(m)$ is always satisfied. Therefore brokers can route a data message purely based on the matching results.

6.2 Virtual Time Vector-based Redundant-path Spanning Tree Algorithm

In [14], we described a set of subscription propagation algorithms using virtual time vectors in a spanning tree with redundant paths. In such algorithms, a SHB maintains a clock in the form of an integer counter. The algorithms also provide liveness and failure recovery mechanisms. However, this is orthogonal to our focus here on safety of subscription propagation.

As described in [14], an SHB advances its virtual time clock every once in a while a number of subscriptions and/or unsubscriptions are received. Each conjunction of a subscription is assigned a virtual start time that is the minimum of all its covering conjunctions' (if any) start times and the SHB's current clock time. The entire subscription s is assigned a virtual start time vt that is the maximum of all its conjunctions'.

We assume subscription s is also assigned a virtual end time $\overline{vt}$ using the SHB's current clock when unsubscription is requested. Subscriptions that are not yet unsubscribed have an undefined virtual end time. If the SHB advances its clock, indicating subscription changes need to be propagated, the SHB initiates an incremental subscription change message to its upstream brokers. The incremental change contains the set of conjunctions that are added or removed as well as the current virtual clock time and a constraint vector.

Every broker b maintains a virtual time vector, with one element for each downstream SHB. The default element value is 0. This virtual time vector is updated when b processes the subscription information.

Assume broker b with virtual time vector $b.vv = [(shb_1, v_1), \cdots, (shb_z, v_z)]$, where the SHBs in this vector are in the downstream of b. If we denote subscriptions entered/unsubscribed at shb_j as $shb_j.\mathbb{S}$ and $shb_j.\overline{\mathbb{S}}$, $S(e_i)$ of broker b is defined as $\{s \mid (s \in shb_j.\mathbb{S}) \wedge (s.vt \leq v_j)\}$, and $\overline{S}(e_i)$ as $\{s \mid (s \in shb_j.\overline{\mathbb{S}}) \wedge (s.\overline{vt} \leq v_j)\}$ for each shb_j ($j \in [1..z]$) in the downstream of an outgoing edge e_i. That is, b maintains information for *active/unsubscribed* subscriptions at shb_j with virtual start times earlier than or equal to time v_j and virtual end time later than v_j.

A data message m in this algorithm also carries a virtual time vector $m.vv = [(shb_1, v_1), \cdots, (shb_z, v_z)]$. The $S(m)$ set of a data message m is defined as the subscriptions entered at shb_j with virtual start time no later than v_j, i.e., $\{s \mid\ (s \in shb_j.\mathbb{S}) \wedge (s.vt \leq v_j)\}$.

We examine how this algorithm embodies the constraints and sufficiency test.

Constraints In this virtual time vector based algorithm, constraints are embodied by requiring that two brokers on each end of an edge e have the same vector values for SHBs that are in the downstream of e. We describe below that this vector value equivalency implies the satisfaction of all constraints computed at a broker (e.g., b_2 in Figure 2) when b_2 applies the incremental change it receives for computing new values sent to an upstream broker (e.g., b_1).

First b_2 and b_1 having the same vector values for SHBs in the downstream of e_2 means that

$$S(e_2) = S(e_2)(b_2) = S(e_4) \cup S(e_5) \tag{6.4}$$
$$\overline{S}(e_2) = \overline{S}(e_2)(b_2) = \overline{S}(e_4) \cup \overline{S}(e_5) \tag{6.5}$$

We have $S(e_2) - \overline{S}(e_2) = S(e_2)(b_2) - \overline{S}(e_2)(b_2)$, constraint II is satisfied.

Type I, III and IV require in broker b_1 the existence of some subscriptions that also exist in broker b_2. They are always satisfiable if b_1 and b_2 have the same set of subscriptions.

Sufficiency Test The virtual time vector based algorithm embodies the sufficiency test by requiring the brokers have vector elements that are greater than or equal to the corresponding elements in the message's vector for SHBs in the downstream of e.

Consider broker b_2 in Figure 2 and a data message m received through its incoming edge e_2. We prove below that b_2 is sufficient for m on an outgoing edge (e.g., e_4) if its vector elements are sufficiently large for the relevant SHBs. We show that $S(m, e_4) - \overline{S}(e_4) \subseteq S(e_4) - \overline{S}(e_4)$. This is satisfied if for every $s \in S(m, e_4) - \overline{S}(e_4)$, $s \in S(e_4) - \overline{S}(e_4)$.

For every $s \in S(m, e_4) - \overline{S}(e_4)$, $s \in S(m, e_4)$ and $s \notin \overline{S}(e_4)$. From the aforementioned mapping of $S(m, e_4)$, we have $s.vt \leq m.vv(s.shb)$, where $s.shb$ denotes the SHB at which s subscribes. Because the algorithm requires sufficiency test of $b_2.vv(s.shb) \geq m.vv(s.shb)$, we have $s.vt \leq b_2.vv(s.shb)$. Hence, from the mapping of $S(e_4)$, we have $s \in S(e_4)$. Because $s \notin \overline{S}(e_4)$, $s \in S(e_4) - \overline{S}(e_4)$. The sufficiency test is satisfied.

7 Related Work

In this section we survey previous works on subscription propagation and content-based routing in publish/subscribe systems.

Siena [2] and XNet [3] use a topology that has redundancy, with multiple routes between servers. However, the subscriptions are only propagated along a single selected *best route* in a spanning tree. This makes it slow to recover from a spanning tree link failure by dynamically switching to another route.

Elvin [10] is architectured as a single server or a cluster of servers in a LAN. In contrast, our work is applicable to wide-area topologies.

Snoeren et. al [11] propose an approach for improving reliability and low latency by sending simultaneously over redundant links in mesh-based overlay networks. The protocol does content-based routing and provides high level of availability. However, there does not seem to be any guarantee of reliable delivery when subscriptions are dynamically added and removed.

REBECA [6, 7] uses a network constructed as a tree of brokers. Their subscription aggregation techniques, such as filter merging, are applicable to our work. The system has a self-stabilization component that uses time based leases to validate routing entries in brokers. This is a viable technique for best-effort delivery, but does not support reliable delivery since it is possible for a broker to filter a message that is relevant for a downstream subscriber.

JEDI [5] also uses a network of dispatching servers interconnected into a tree topology. Their recent work [8, 4] deals with how to minimize the re-propagation of subscription information when the tree is changed by adding and deleting links. This is complementary to the work described in this paper, since we do not consider tree changes. However, it is not clear how their work can be extended to networks with redundant paths without requiring agreement between the multiple paths. In addition, the algorithm described in [4] does not consider the case of server failure while the algorithm is running.

Tapestry [13] provides fault tolerant routing by dynamically switching traffic onto precomputed alternate routes. Messages in Tapestry can be duplicated and multicast ”around” network congestion and failure hotspots with rapid reconvergence to drop duplicates. However, it does not support content-based routing.

Scribe [9] is a large-scale and fully decentralized event notification system built on top of Pastry - a peer-to-peer object location and routing substrate overlaid on the Internet. However, SCRIBE does not support content-based routing or wildcard topic subscriptions, as the creation and subscription of a topic are explicitly associated with a rendezvous point.

8 Conclusions

We have presented a model and the correctness criteria for subscription propagation and content-based routing. Based on the model, we developed a generic algorithm that can deal with redundant routing paths and support reliable delivery. The generic algorithm is highly available and asynchronous in that it does not mandate agreement of subscription states between redundant paths. We also presented an aggregation scheme that is based on covering relationships between conjunctions of subscription filters. Several examples are provided on interpreting previously known algorithms as encodings and optimizations of the generic algorithm under different circumstances.

References

[1] S. Bhola, R. Strom, S. Bagchi, Y. Zhao, and J. Auerbach. Exactly-once delivery in a content-based publish-subscribe system. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN'2002)*, pages 7–16, 2002.

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

[3] R. Chand and P. Felber. A scalable protocol for content-based routing in overlay networks. In *Proceedings of the IEEE International Symposium on Network Computing and Applications (NCA'03), Cambridge, MA*, April 2003.

[4] G. Cugola, D. Frey, A. L. Murphy, and G. P. Picco. Minimizing the reconfiguration overhead in content-based publish-subscribe. In *To appear in Proceedings of ACM Symposium on Applied Computing (SAC04)*, 2004.

[5] G. Cugola, E. D. Nitto, and A. Fuggetta. The jedi event-based infrastructure and its application to the development of the opss wfms. *IEEE Transactions on Software Engineering*, 27(9):827–850, September 2001.

[6] G. Mühl. *Large-Scale Content-Based Publish/Subscribe Systems*. PhD thesis, Darmstadt University of Technology, September 2002.

[7] G. Mühl, L. Fiege, and A. P. Buchmann. Filter similarities in content-based publish/subscribe systems. In *Proceedings of International Conference on Architecture of Computing Systems (ARCS'02)*, 2002.

[8] G. P. Picco, G. Cugola, and A. L. Murphy. Efficient content-based event dispatching in the presence of topological reconfiguration. In *Proceedings of ICDCS*, pages 234–243, 2003.

[9] A. Rowstron, A. Kermarrec, M. Castro, and P. Druschel. Scribe: The design of a large-scale event notification infrastructure. In *Proceedings of 3rd International Workshop on Networked Group Communication (NGC 2001), UCL, London, UK*, November 2001.

[10] B. Segall, D. Arnold, J. Boot, M. Henderson, and T. Phelps. Content based routing with elvin4. In *Proceedings of AUUG2K, Canberra, Australia*, April 2000.

[11] A. Snoeren, K. Conley, and D. Gifford. Mesh-based content routing using xml. In *Proceedings of the 18th ACM Symposium on Operating System Principles*, 2001.

[12] P. Triantafillou and A. Economides. Subscription summarization: A new paradigm for efficient publish/subscribe systems. In *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*, 2004.

[13] B. Zhao, L. Huang, A. Joseph, and J. Kubiatowicz. Exploiting routing redundancy using a wide-area overlay. Technical Report UCB/CSD-02-1215, University of California, Berkeley, 2002.

[14] Y. Zhao, D. Sturman, and S. Bhola. Subscription propogation in highly-available publish/subscribe middleware. In *ACM/IFIP/USENIX 5th International Middleware Conference (Middleware 2004) (To Appear)*.

Appendix A Informal Proof of Theorem 1

Theorem 1. *Reliable delivery of a subscription s can be guaranteed if the following conditions are satisfied:*
1. (Correctness of subscription propagation) Every broker in the pub/sub system maintains S-Λ invariant.
2. (Correctness of content-based routing) Routing brokers use sufficiency-directed content-based routing.
3. If there is a sufficiently long period of time for which the system runs without failure, newly published messages (or silences) start to arrive at the SHB of s with $S(m)$ such that $s \in S(m)$ unless the system fails again. Delivery starting and ending points of s can be chosen in the aforementioned way.
4. The system guarantees eventual monotonicity of $S(m)$ for messages in [starting point, ending point] and the SHB only accepts the transmission of messages with monotonic $S(m)$.

Proof. Condition 1 guarantees that if a message matches a subscription in $S(e) - \overline{S}(e)$ of a broker, then the broker will not filter out this message because it maintains conjunctions that covers the conjunctions in $S(e) - \overline{S}(e)$.

Condition 2 guarantees that if s has not been unsubscribed and a message m has $S(m, e)$ such that $s \in S(m, e)$, then m will not be filtered out by a broker for an outgoing edge e if $m \in M(conj(s))$. This is because sufficiency-directed content-based routing will only filter out a message if the sufficiency test passes, that is, $S(m, e) - \overline{S}(e) \subset S(e) - \overline{S}(e)$. Because $\Lambda(e) \sqsupseteq_c S(e) - \overline{S}(e)$, we have $\Lambda(e) \sqsupseteq_c S(m, e) - \overline{S}(e)$. Because $s \in S(m, e)$, then $\Lambda(e) \sqsupseteq_c conj(s)$. By definition of $\sqsupseteq_c$, $M(conj(s)) \subseteq M(\Lambda(e))$. Hence, if $m \in M(conj(s))$, $m \in M(\Lambda(e))$, that is, m will not be filtered out.

Condition 3 guarantees that if the systems run long enough without failures, eventually some data message will arrive with $S(m)$ such that $s \in S(m)$. This way we can pick a delivery starting point.

Condition 4 guarantees that all messages (or their corresponding silences if filtered) after the delivery starting point of s arrives at the SHB of s with $s \in S(m)$. Combined with Condition 2, those data messages matching s will not be filtered out because they have the right $S(m)$. Thus, the system guarantees a gapless sequence of messages for s after its delivery starting point.

□

Appendix B Proof of Theorem 2

For simplicity of description, we make the following assumptions

1. Every subscription has exactly one conjunction. The general case can be proved by dividing a multi-conjunction subscription into several subscriptions, each has exactly one conjunction of the original subscription;

2. The broker network is a redundant binary tree where each non-leaf node has exactly two children.

We use Figure 4 to illustrate. We consider a path $b_1 \to b_2 \to \cdots \to b_n \to b_{n+1} \to \cdots$, starting from leaf node broker b_1. We denote the other child of b_i as b'_{i-1}, and the edge from b_i to its children b_{i-1} and b'_{i-1} as e_{i-1} and e'_{i-1}. For simplicity, let us denote the clients connected to leaf node broker b_1 as b_0 and b'_0, and edges as e_0 and e'_0.

We examine the properties of broker b_{n+1} in the case that a subscription change is propagated from b_0.

Lemma 1. *For subscription s and its conjunction λ, if $s \notin \Delta\overline{S}(e_n)$, then "$-\lambda$" $\notin \Delta\Lambda(e_n)$.*

This lemma is obvious because

1. At leaf broker b_0, if and only if $s \in \Delta\overline{S}(e_0)$, will "$-\lambda$" $\in \Delta\Lambda(e_0)$.

2. No new "$-\lambda$" is generated during any of the operations at any upstream broker.

Lemma 2. *For $\lambda_1, \lambda_2 \in \Lambda(e_n)(b_n)$, if $\lambda_1 \succeq_c \lambda_2$, and $\lambda_1 \in \Lambda(e_{n-1})$, then $\lambda_2 \in \Lambda(e'_{n-1})$. Otherwise, if $\lambda_1 \in \Lambda(e'_{n-1})$, then $\lambda_2 \in \Lambda(e_{n-1})$. Similarly, their subscriptions belong to different downstream.*

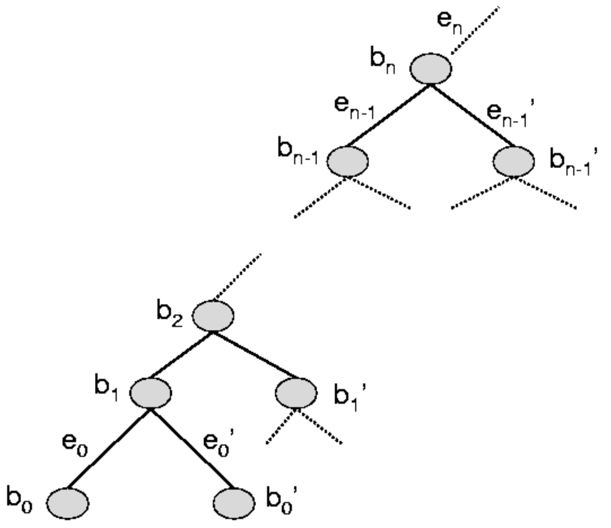

Figure 4:

This lemma is obvious because the minimization operation will make sure conjunctions from same downstream will not have any covering relationship.

Theorem 2. *For any broker b and an outgoing edge e in a redundant spanning tree, if its $S(e)$, $\overline{S}(e)$ and $\Lambda(e)$ sets initially satisfy the S-Λ invariant and are only modified by the operations and constraint type I, II and III described in Section 4.2, the broker maintains S-Λ invariant.*

It is sufficient to prove $\forall s \in S(e_n) \cup \Delta S(e_n) - \overline{S}(e_n) - \Delta\overline{S}(e_n)$, $conj(s) = \lambda$, there exists $\lambda_0 \in \Lambda(e_n) + \Delta\Lambda(e_n)$ and $\lambda_0 \succeq_c \lambda$.

Proof. We consider all possible cases of s.

Case 1. *$s \in \Delta S(e_n)$ and "$+\lambda$" $\in \Delta\Lambda(e_n)$.*

In two cases this will happen: "Adding a Root Conjunction" and "Adding a non-Root Conjunction". In both cases, λ is covered by either itself or an existing conjunction that will continue to exists in the new DAG of $\Lambda(e_n) + \Delta\Lambda(e_n)$.
The theorem holds in this case.

Case 2. *$s \in \Delta S(e_n)$ and "$+\lambda$" $\notin \Delta\Lambda(e_n)$.*

In this case, "$+\lambda$" is dropped at a broker $b_i (i \leq n)$. That is, "$+\lambda$" $\in \Delta\Lambda(e_{i-1})$ but "$+\lambda$" $\notin \Delta\Lambda(e_(i))$. This can only happen when λ is added as a non-root conjunction to the new DAG of $\Lambda(e_i)(b_i)$. As described in Section 4.2, a type I constraint is generated. Since a constraint is preserved all the way during propagation, broker b_{n+1} must also satisfy this constraint in order to apply the incremental changes. This constraint on b_{n+1} guarantees that there exists a subscription s' and its conjunction λ', such that $s' \in S(e_n) - \overline{S}(e_n)$ and $\lambda' \succeq_c \lambda$.

From descriptions in Section 4.2, λ' appears in b_i's DAG of $\Lambda(e_i)(b_i)$ even after b_i applies $\Delta\Lambda(e_{i-1})$, hence "$-\lambda'$" $\notin \Delta\Lambda(e_i)$. From Lemma 1, $s' \notin \Delta\overline{S}(e_i)$. Because $\Delta\overline{S}(e_i) = \Delta\overline{S}(e_n)$, $s' \notin \Delta\overline{S}(e_n)$. Thus $s' \in S(e_n) - \overline{S}(e_n) - \Delta\overline{S}(e_n)$. This is Case 3. Hence λ' is covered by a conjunction λ'' in $\Lambda(e_n) + \Delta\Lambda(e_n)$. Because $\lambda' \succeq_c \lambda$, $\lambda'' \succeq_c \lambda$. The theorem holds.

Case 3. *$s \notin \Delta S(e_n)$, hence $s \in S(e_n) - \overline{S}(e_n) - \Delta\overline{S}(e_n)$.*

We prove this case by induction and contradiction.

Induction base: At broker b_1, subscription s's conjunction λ will appear in the DAG of $\Lambda(e_0)$ because client (broker b_0) has only single subscription/conjunction and thus no consolidation occurs.

Induction assumption Suppose at broker $b_i (i \leq n)$, for all $s \in S(e_{i-1}) - \overline{S}(e_{i-1}) - \Delta\overline{S}(e_{i-1})$ and its conjunction λ, there exists a $\lambda_0 \in \Lambda(e_{i-1}) + \Delta\Lambda(e_{i-1})$ and $\lambda_0 \succeq_c \lambda$.

Induction rule: We prove that at broker b_{n+1}, this holds for any subscription $s \in S(e_n) - \overline{S}(e_n) - \Delta\overline{S}(e_n)$.

Suppose s's conjunction λ is not covered by any conjunction in $\Lambda(e_n) + \Delta\Lambda(e_n)$.

Because $s \in S(e_n) - \overline{S}(e_n) - \Delta\overline{S}(e_n)$, $s \notin \Delta\overline{S}(e_n)$. From Lemma 1, "$-\lambda$"$\notin \Delta\Lambda(e_n)$. The fact that λ is not covered by any conjunction in $\Lambda(e_n) + \Delta\Lambda(e_n)$, means that there was a conjunction $\lambda' \in \Lambda(e_n)$, $\lambda' \succeq_c \lambda$, and λ' was removed in the new DAG, that is, "$-\lambda'$" $\in \Delta\Lambda(e_n)$. This case is handled as described in "Removing a Root Conjunction" in Section 4.2. A type II constraint is generated

$S(e_n) - \overline{S}(e_n) \subseteq S(e_n)(b_n) - \overline{S}(e_n)(b_n)$.

Because $s \in S(e_n) - \overline{S}(e_n)$, $s \in S(e_n)(b_n) - \overline{S}(e_n)(b_n)$. From Equation 4.1 and 4.2,

$S(e_n)(b_n) = S(e_{n-1}) \cup \S(e'_{n-1})$, $\overline{S}(e_n)(b_n) = \overline{S}(e_{n-1}) \cup \overline{S}(e'_{n-1})$

thus $s \in S(e_{n-1}) \cup S(e'_{n-1}) - (\overline{S}(e_{n-1}) \cup \overline{S}(e'_{n-1}))$. Because one subscription can only enter at one leaf broker, we have either $s \in S(e_{n-1}) - \overline{S}(e_{n-1})$ or $s \in S(e'_{n-1}) - \overline{S}(e'_{n-1})$.

Because the theorem holds at broker b_n, λ is covered by a conjunction λ'' (this could be λ itself) in the DAG of $\Lambda(e_n)(b_n)$. Since there is a "$-\lambda'$" in $\Delta\Lambda(e_n)$, there must be a λ' in the DAG of $\Lambda(e_n)(b_n)$. As described in "Remove a Root Conjunction" in Section 4.2, either "$+\lambda''$" is added to $\Delta\Lambda(e_n)$ or a constraint of type III is added. This constraint requires broker b_{n+1} to have a subscription $s_1 \in S(e_n) - \overline{S}(e_n)$ such that s_1's conjunction $\lambda_1 \succeq_c \lambda''$. In addition $s_1 \neq s$. Then the theorem reduces to prove that s_1 is covered by a conjunction in the DAG of $\Lambda(e_n)$. Repeat this process, the theorem requires subscription s_2, s_3, ..., s_k,... to be covered by a conjunction in $\Lambda(e_n)$ or a "$+\lambda^k$" is added to $\Delta\Lambda(e_n)$. As each of this $s_{1..k}$ is distinct, and the number of subscriptions in the downstream of e_n is finite, this step must terminate at adding an additive conjunction to $\Delta\Lambda(e_n)$, and thus the theorem holds.

□

Appendix C Proof of Theorem 3

Theorem 3. *We consider broker b and its outgoing edge e in a network using operation and all types of constraints described in Section 4. For every conjunction $\lambda \in \Lambda(e)$, there exists a subscription s such that $\lambda \in conj(s)$ and $s \in S(e) - \overline{S}(e)$.*

Proof. We prove by induction.

1. **Base case**
   Initially, $S(e) \equiv \overline{S}(e) \equiv \Lambda(e) = \emptyset$, so this satisfies the theorem.

2. **Induction assumption**
   Suppose the theorem holds for broker b before processing an incremental update ($\Delta S(e)$, $\Delta\overline{S}(e)$, $\Delta\Lambda(e)$ and C) from its downstream.

3. **Induction rule**
   We prove for every conjunction $\lambda \in \Lambda(e) + \Delta\Lambda(e)$, there exists a subscription s, such that $\lambda \in conj(s)$ and $s \in S(e) \cup \Delta S(e) - \overline{S}(e) - \Delta\overline{S}(e)$.

   There are two types of events to consider:

   (a) "$+\lambda$"$\in \Delta\Lambda(e)$ - because this may result in (new) conjunctions that do not have a subscription in $S(e) - \overline{S}(e)$ sets. This can occur in two cases:

i. λ belongs to a new subscription s. In this case, $s \in \Delta S(e)$ and $s \notin \Delta\overline{S}(e)$.
Because $\Delta S(e)$ stays the same throughout the propagation process 4.2, it is exactly the same as the value that is initially computed at the SHB. Because the members in $S(e)$ and $\overline{S}(e)$ are originated at SHBs and each subscription only connects from one SHB, $\Delta S(e) \cap \overline{S}(e) = \emptyset$. We thus have $s \notin \overline{S}(e)$.
Therefore
$s \in S(e) \cup \Delta S(e) - \overline{S}(e) - \Delta\overline{S}(e)$
The theorem holds.

ii. λ is generated in the case of "Removing a Root Conjunction" described in Page 7. From constraint IV, we have $\exists s \in sub(\lambda)$ such that $s \in S(e) - \overline{S}(e)$. Because there is a "$+\lambda$" but not "$-\lambda$" in $\Delta\Lambda(e)$, $s \notin \Delta\overline{S}(e)$. Thus
$s \in S(e) \cup \Delta S(e) - \overline{S}(e) - \Delta\overline{S}(e)$
The theorem holds.

(b) $s \in \Delta\overline{S}(e)$ - because this may result in b not removing conjunctions whose subscription are removed. Initially, every $s \in \Delta\overline{S}(e)$ will have a "$-\lambda$" in $\Delta\Lambda$. Two cases could happen to this "$-\lambda$" when a downstream broker, e.g., b_1 computes the $\Delta\Lambda$ for subscription aggregation. We examine the activities happening in b_1 and prove that if "$-\lambda$" is removed by b_1 from the $\Delta\Lambda(e)$ it sends upstream, an upstream broker will not have λ in its $\Lambda(e)$ DAG either.

i. "$-\lambda$" stays in $\Delta\Lambda(e)$. This can only happen in the case of "Remove a Root Conjunction". The theorem holds in this case.

ii. "$-\lambda$" is removed. This can only happen in the case of "Removing a non-Root Conjunctions" described in Section 4. In this case, a type I constraint is generated.
Thus there exists s' and $\lambda' \in conj(s')$ such that $s' \in S(e) - \overline{S}(e)$ and $\lambda' \succeq_c \lambda$. Because $s'.\lambda'$ exists in downstream broker b_1's DAG, it is not being removed. Hence $s' \notin \Delta\overline{S}$. Thus we have
$s' \in S(e) \cup \Delta S(e) - \overline{S}(e) - \Delta\overline{S}$.
From theorem 2, either $s'.\lambda' \in \Lambda(e) + \Delta\Lambda(e)$, or $s'.\lambda'$ is covered by a conjunction in it. In both cases, because $s'.\lambda' \succeq_c \lambda$, λ will be removed by the minimization, and thus the theorem holds.

$\square$