ADAPTIVE GRIDSTAT INFORMATION FLOW MECHANISMS AND MANAGEMENT FOR POWER GRID CONTINGENCIES

By

STIAN FEDJE ABELSEN

A thesis submitted in partial fulfillment of
the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

WASHINGTON STATE UNIVERSITY
School of Electrical Engineering and Computer Science

AUGUST 2007

ACKNOWLEDGEMENT

I would like to thank my advisor Dave Bakken for his guidance and advice throughout my studies here at Washington State University. I would also like to thank the entire GridStat team for valuable input and comments on my research, and to my committee members Carl Hauser and Min Sik Kim. A special thanks goes to Carl Hauser for his active role in my research and his willingness to help during the most frustrating times. A well-deserved thanks goes out to my friends in Pullman and Norway, who have made my stay much more enjoyable and socially acceptable.

Finally, I would like to thank all the organizations that have funded me throughout my studies. In particular, I have received a stipend from The Norwegian State Educational Loan Fund and tuition reduction from Washington State University. In addition, my research has been supported in part by grants CNS 05-24695 (CT-CS: Trustworthy Cyber Infrastructure for the Power Grid(TCIP)) and CCR-0326006 from the US National Science Foundation.

PUBLICATIONS

Stian F. Abelsen, Erlend S. Viddal, David Bakken and Carl Hauser, Adaptive Information Flow Mechanisms and Management for Power Grid Contingencies, in DSN ’08: Proceedings of the International Conference on Dependable Systems and Networks (DSN’08). To be submitted in Fall 2007.

ADAPTIVE GRIDSTAT INFORMATION FLOW MECHANISMS AND MANAGEMENT FOR POWER GRID CONTINGENCIES

Abstract

by Stian Fedje Abelsen, M.S.
Washington State University
August 2007

Chair: David E. Bakken

GridStat is designed to address the need for a flexible and robust communication system in the electrical power grid, and provides a specialization of the publisher-subscriber paradigm. GridStat middleware enables reliable delivery of data to any point through a network of forwarding engines called status routers, manages network resources and provides QoS (Quality of Service) for data streams. Furthermore, GridStat hides the details of lower-level network capabilities from application developers in order to enable the communication system to be deployed across different network technologies, operating systems, programming languages and device types. GridStat is divided into two planes; the management plane and the data plane. The management plane consists of a hierarchy of QoS brokers which collectively manage resources (status router network) and subscriptions in the data plane. The mode change mechanism is a feature introduced by GridStat, and allows quick adaptation of subscription flows.

A mode contains the necessary forwarding rules for a set of subscriptions and allows the status router network to quickly switch between bundles of subscriptions; an action called a *mode change*. The process of establishing individual subscriptions is a resource-intensive operation in which the deallocation and allocation of subscription bundles in run-time is expensive and may result in unsatisfactory subscription delays. GridStat enables subscription bundles to be allocated and pre-loaded into routing tables where *operating modes* control which routing tables the status router network will utilize. Previous GridStat versions had limited mode change capabilities. Mode switches were limited to a single administrative domain (GridStat *cloud*) and the mode namespace was shared between QoS brokers in the management plane. Additionally, the QoS brokers did not support the appropriate mechanisms to define and use modes in their respective administrative domains. The global and hierarchical mode change mechanisms and management implementation of this thesis supports these mechanisms and introduces the notion of global and hierarchical modes.

This work enables the electrical power grid industry to quickly adapt information flow streams to power grid contingencies, and provides control center operators the means to better understand and quickly prepare responses to such threats.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Dedication

To my family,

for their continual support.

CHAPTER ONE

INTRODUCTION

The electrical power grid is highly dependent on data monitoring and control capabilities in order to better understand and manage power transmissions over a highly complex network of transmission lines and substations. SCADA (Supervisory Control and Data Access) has in the last 40 years served as the electrical power grid's communication system and incorporates the requirements and network technologies back to when it was developed. The requirements for communication in the electrical power grid are changing. Growing concerns about terrorist attacks, changes in the power flow structure after the deregulation in 1996, new uses of technologies (IntelliGrid [2]) and an increased overall load to capacity ratio of the transportation line system demand a more flexible and adaptive communication network. The SCADA communication system features a centralized star-topology, point to point communication, lack of multicast, severe bandwidth constraints and proprietary protocols which are not sufficient to meet the requirements of today's grid. [4] and [1] discuss the limitations of SCADA in more detail.

GridStat is designed to address the need for a flexible and robust communication system in the electrical power grid, and provides a specialization of the publisher-subscriber paradigm. GridStat middleware manages network resources, enables reliable delivery of data to any point and provides QoS (Quality of Service) for data streams. GridStat hides the details of lower-level network capabilities from application developers in order to enable the communication system to be deployed across different network technologies, operating systems, programming languages and device types. GridStat is divided into two planes; the *management plane* and the *data plane*. The management plane consists of a hierarchy of QoS brokers which collectively manage resources and subscriptions in the data plane. The data plane is a virtual message bus and lets publishers provide data to the network and enables subscribers to establish subscriptions to status data through a status router network. The use of QoS, on a per-subscription basis, allows subscribers to specify multiple redundant delivery paths (spatial redundancy), subscription interval and delay. Furthermore, GridStat provides status data delivery to multiple recipients at different rates through the multicast property and the ability to control and switch routing tables in the status router network in run-time through the use of modes.

A mode contains the necessary forwarding rules for a set of subscriptions and allows the status router network to quickly switch between bundles of subscriptions; an action called a *mode change*. The process of establishing individual subscriptions is a resource-intensive operation in which the deallocation and allocation of subscription bundles at run-time is expensive and may result in unsatisfactory subscription delays. GridStat enables subscription bundles to be allocated and pre-loaded into the status routers' routing tables where *operating modes* control which routing tables the status router network will utilize. Depending on the GridStat deployment, status routers can utilize several routing tables corresponding to the operating modes, while inactive routing tables lie dormant.

Previous GridStat versions had limited mode change capabilities. Mode switches were limited to a single administrative domain (GridStat *cloud*) and the mode namespace was shared between QoS brokers in the management plane. Additionally, the QoS brokers did not support the appropriate mechanisms to define and use modes in a hierarchical or global context. The global and hierarchical mode change mechanism and management implementation of this thesis supports these mechanisms and introduces the notion of global and hierarchical modes. Global modes are defined and used by the top-level QoS broker in the management hierarchy and affect the entire status router network, whereas hierarchical modes are managed by interior QoS brokers and affect specific regions of the status router network, subject to the administrative domains of the interior QoS brokers that manage them.

The mode change mechanism will help utility companies (control centers), regional control centers, ISOs and nation-wide monitoring centers in pre-contingency planning for communication needs and to switch subscription bundles when contingencies do occur in the electrical power grid. Furthermore, modes enable data load shedding in the communications infrastructure in a similar manner as the electrical power grid utilizes power load shedding. For example, subscribers could specify two QoS sets; desired QoS and least desirable QoS, and switch between them when the network is congested.

The research contributions of this thesis are:

- Global and hierarchical modes: QoS brokers define and use modes to adapt communication in their respective administrative domains.
- Multiple simultaneously active routing tables in the data plane and the ability to switch between routing tables at run-time.
- The design and implementation of two mode change algorithms with different tradeoffs.
- An experimental evaluation which compares the mode change algorithms in terms of performance, resource usage and variance (time) in the presence of various temporal network conditions.

The remainder of this thesis is organized as follows: The GridStat framework is presented in Chapter 2. An explanation of modes, mode change operations and related mechanisms are presented in Chapter 3. The design of the hierarchical mode change algorithm is described in Chapter 4. The design of the flooding mode change algorithm is presented in Chapter 5. Experimental results from both mode change algorithms are presented in Chapter 6, followed by related work in Chapter 7, and conclusions and future work in Chapter 8.

CHAPTER TWO

STATUS DISSEMINATION AND GRIDSTAT

This chapter gives an overview of the GridStat architecture, a detailed description of the various GridStat applications and interaction models. More detailed information about GridStat and other baseline mechanisms can be found in [3] and [1].

2.1 GridStat Architecture

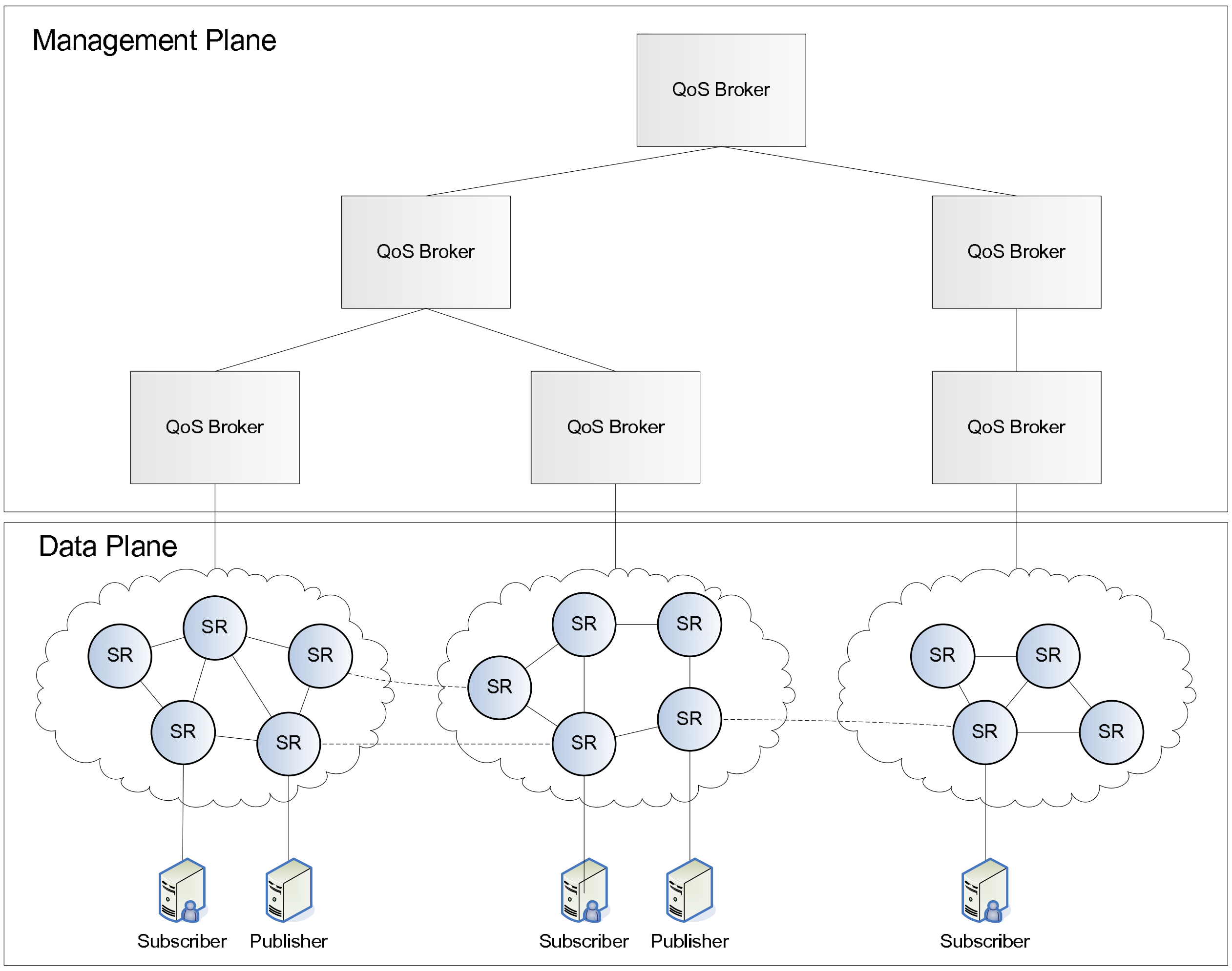


Figure 2.1: Status dissemination middleware.

GridStat is a publisher-subscriber framework that targets application domains where the majority of data is made available at periodic time intervals. GridStat is mainly designed to serve as a flexible and robust communication system in the electrical power grid, but has other applicable domains where the publisher-subscriber paradigm can be used. Figure 2.1 shows a small scale GridStat deployment subdivided in a management plane and a data plane. The management plane consists of *QoS broker* modules that collectively control and manage resources in the data plane. The data plane is populated by *status routers*, *publishers* and *subscribers,* where publishers provide data and subscribers can subscribe to data. The management hierarchy handles subscription requests and establishes paths from the publisher to the subscriber through a sequence of status routers.

2.1.1 Management Plane

The lowest level of the management plane consists of *leaf QoS brokers*. A leaf QoS broker manages and provides services to a set of status routers, publishers and subscribers. The leaf QoS broker manages a flat collection of status routers, called a *cloud*, where the leaf QoS broker has complete control over all available resources and the corresponding resource usage. The resources include event channels, status routers, publishers and subscribers. Event channels serve as communication links between status routers, in which leaf QoS brokers must control and make sure no allocated subscriptions exceed an event channel's bandwidth constraints. Additionally, the leaf QoS brokers must ensure that routing tables and computational resources are not overloaded in the status routers. The main responsibility of a leaf QoS broker is to control the allocation or deallocation of subscription paths between publisher and subscriber pairs in its cloud, and to ensure that the allocated path satisfies the QoS requirements specified by the subscriber.

Interior QoS brokers denote all non-leaf QoS brokers in the managements hierarchy. Interior QoS brokers manage multiple clouds and offer services to lower-level QoS brokers, and whose main responsibility is to allocate and deallocate inter-cloud subscriptions.

There is currently no limit on how many management levels the management plane can consist of. In practice, even at a nation-wide deployment in the electrical power grid or other critical infrastructures, there would likely be no more than 6-10 levels. An interior QoS broker has an abstract view on the details and population of the individual clouds, while lower-level QoS brokers, and especially leaf QoS brokers, maintain and control more state information on the details and available resources in the clouds. Interior QoS brokers require more computational resources while lower-level QoS brokers are mostly responsible for processing simple subscription allocation and deallocation requests that affect a small scope of the data plane, and therefore require less computational resources.

A subscription request initiated by a subscriber is first delivered to its leaf QoS broker, and then forwarded up the management hierarchy until it reaches the interior QoS broker which has both the publisher and subscriber in its hierarchical scope; the *owner* of the subscription. The owner executes its routing algorithm that first verifies whether there exists a path from the publisher to the subscriber. The clouds and inter-cloud event channels gathered from the routing algorithm are delegated to its children QoS brokers, which collectively process the next recursive step of the allocation algorithm. The distributed recursive algorithm continues until it reaches all the leaf QoS brokers, which inform the involved status routers to populate their routing tables with the necessary forwarding data for the inter-cloud path to be established. A local subscription request, with the publisher and subscriber in the same cloud, is delivered to the leaf QoS broker which handles the subscription request alone. From the perspective of a leaf QoS broker, a subscription request might come from a subscriber in its cloud or from its parent QoS broker. In the latter case, an inter-cloud subscription path is to be established, e.g., a subscription path that involves multiple clouds.

2.1.2 Data Plane

The data plane is a term used to describe a virtual message bus where subscription data flows between publishers and subscribers. The virtual message bus consists of status routers and event channels, whose main purpose is to forward status events from publishers to the subscriber applications that requested the data. A status router is in effect a router with additional functionality to provide forwarding of status events when subscribed to and at the right rate (rate filtering). The management plane controls the content of the routing tables in the status routers, and leaf QoS brokers inform status routers to add, remove or modify the content corresponding to a subscription allocation or deallocation request. In order for a status router to communicate with the management plane, it has a connection to the leaf QoS broker that controls it.

Publishers and subscribers interact with the management plane through the virtual message bus. A publisher or subscriber registers with an edge status router which serves as a proxy to the management plane and as an entry point to the virtual message bus. The status router establishes an event channel to its publisher or subscriber, which enables the publisher to provide data to the data plane or a subscriber to retrieve information off of the data plane to.

There are three types of status routers in GridStat. The first type is the *edge status router*, which serves as a connection point and proxy for publishers and subscribers. Additionally, it forwards events like any other status router. The second type is the *internal status router*, which simply forwards events and is transparent to publishers and subscribers. The third status router type is the *border status router* which serves as a cloud's entry and exit point. Interior QoS brokers use the inter-cloud event channel capacity to accommodate and allocate inter-cloud subscription paths, where border status routers have an important role in the overall subscription allocation algorithm. Leaf QoS brokers treat border and internal status routers equally in terms of allocating resources within a cloud, and are therefore transparent to the roles of the border status router in an inter-cloud subscription allocation.

The management plane manages and controls the resources provided by all status routers, event channels and active subscriptions in the data plane. Since resources are monitored and controlled, the latency from the publisher and the subscriber can be bound. When registering a subscription, subscribers associate a set of QoS parameters with the subscription request, and among these are a subscription interval, a latency request and redundancy. The management hierarchy attempts to find a path between the publisher and subscriber that is bound by the latency request parameter. If no such path exists, the subscription request is rejected. Additionally, if the subscription redundancy parameter is set, the management hierarchy attempts to find more than one path between the publisher and subscriber. Subscription traffic is therefore bound by a specified delay and is able to flow on several disjoint paths towards the subscriber, providing timeliness and reliability QoS to subscriber applications. The necessary algorithms to establish disjoint paths in the data plane is still under investigation and is not the focus of this thesis. More information can be found in [5].

2.1.3 Publisher Application and API

A publisher registers with GridStat through its edge status router, which functions as a proxy for communication with the management hierarchy. A publisher must first register the status variable and the publication interval with its leaf QoS broker through the edge status router. If the leaf QoS broker accepts the publication request, the publisher is permitted to publish status events at the given publication interval.

Publishers provide status information from *status variables* that change over time, and publish snapshots (*events*) of these at specific time intervals to their publisher API. Events that are sent out on the virtual message bus are called *status events*. The publisher API provides a push interface to the overlying publisher application. Whenever the overlying application uses the publisher API to push an event, the API blocks the call until the event is forwarded to the edge status router. The API supports many of the native Java types as well as user-defined type containers, and marshalls the event and produces a status event, which is sent out on the underlying communication layer. The API returns an error back to the overlying application if some communication error was detected.

2.1.4 Subscriber Application and API

A subscriber registers with GridStat through its edge status router, which functions as a proxy for communication with the management hierarchy. Subscribers can subscribe to *status variables* that have been registered in the management hierarchy by creating a subscription request with the desired status variable name, publisher name and a set of QoS parameters. The QoS parameters include the number of disjoint paths between the publisher and subscriber, a latency request and a subscription interval. The management hierarchy checks if the publisher and status variable exist and sets up one or more paths between the publisher and the subscriber that satisfy the given QoS parameters. Once the paths have been established the subscriber is able to access the subscribed status events through a specific *interaction model*. The subscriber API provides both a pull and push interaction model the overlying subscriber application can use to retrieve subscribed status events. Additionally, a QoS push interaction model can be used by the subscriber application to be notified whenever the timeliness requirements for any of its subscriptions are violated.

- **Push (Every subscribed status event is delivered to the subscriber application):** The subscriber application passes a notification object to the subscriber API which is used to notify the overlying application when a subscribed status event has been received and is ready to be accessed.

- **Pull (The subscriber application retrieves the last status event when needed):** The subscriber application passes a locally instantiated cache object of the variable type it subscribes to to the subscriber API. The read-only cache object will always contain the last seen status event. The subscriber application can access the object whenever it requires access to the subscribed variable.

- **QoS Push (signals QoS violations):** The QoS push interaction model can be used in combination with either the push or pull interaction model. The subscriber application passes a notification object to the subscriber API which is used to notify the overlying application if the requested QoS for the subscription is violated.

CHAPTER THREE

MODE CHANGE MECHANISMS AND MANAGEMENT

This chapter provides the foundation for global and hierarchical modes in order to fully understand the hierarchical and flooding mode change algorithms. More specifically, this chapter discusses mode terminology, mode definitions and the propagation and use of modes in the management hierarchy and in the data plane. Further, an overview of the RPC mechanism and the support for multiple active routing tables in the status router network is presented.

The research outlined in the remainder of this thesis is part of GridStat v4, which is a continuation of GridStat v3[3].

3.1 Assumptions and Limitations

The following assumptions have been made for global and hierarchical mode change mechanisms and management in GridStat:

- **Replicated QoS brokers** - Simplifies the recovery mechanism (Section 3.6) in which it does not need to circumvent failed QoS brokers. An overview regarding future work related to QoS broker replication is provided in Section 8.2.6.

- **No Byzantine failures** - Status routers and QoS brokers can trust the correct behavior of the mode change coordinator and other mediators. More details on Byzantine failures is provided in Section 8.2.10.

- **Synchronized clocks across all GridStat entities** - The flooding mode change algorithm (Chapter 5) relies on synchronized clocks in order to prevent subscription flows to temporarily break. More details on clock synchronization is provided in Section 8.2.9.

- **GridStat initialization** - The current implementation assumes that GridStat is fully initialized and ready to execute mode change operations.

3.2 Overview and Mode Terminology

Explaining a mode requires some insight into how a mode is defined and used in both the management plane and the data plane. A *mode definition* consists of an ID, a name and a set of data plane subscriptions, and is owned by a single QoS broker in the management hierarchy.

A mode definition can either be *static* or *dynamic*. When dynamic mode definitions are used, subscribers specify what modes their subscriptions will operate in, and QoS brokers can reject subscriptions if a subscription can potentially exhaust resources in the data plane. If a subscription request is valid, it is added to all the mode definitions it was set to operate in. Alternatively, a static mode definition contains a set of subscriptions that will be utilized when the mode is active. Subscription requests made in run-time can simply be rejected if they do not belong to the subscriptions listed in the mode definition(s). A static mode definition scheme is likely the most suitable scheme for the power grid industry, where a strict control environment managing subscriptions through authentication is needed. The current implementation of GridStat only supports dynamic mode definitions, but could be extended to support the requirements for a static mode definition scheme. In that case, the management hierarchy would need to pre-load information on subscription requests, e.g., subscriber credentials, publication value and QoS parameters. Additionally, the management hierarchy must support the appropriate mechanisms to allow subscription requests to be added, modified or even removed in run-time, for example through XML. Finally, live subscription requests (from subscribers) are matched against the pre-loaded subscription requests in the management hierarchy, and possibly rejected if the subscriber is unable to provide the required credentials.

Modes defined and owned by a QoS broker constitute a *mode set*, and exactly one of the modes in a mode set is active at any time. This means that every QoS broker always operates in one mode, or in a *default mode* if no modes are defined. A QoS broker that operates in a mode implies that all subscriptions contained in the subscription set of that mode should be active in the data plane.

For example, envision a leaf QoS broker that controls a set of status routers, a publisher and a subscriber. The leaf QoS broker has defined three modes, Green, Yellow and Red, and is currently operating in the Green mode. The publisher is configured to publish some data to the network, and the subscriber wishes to subscribe to that set of data in mode Green and Yellow. In the Red mode, the published data is not forwarded through the data plane since the subscriber is not interested in the information. More specifically, if the leaf QoS broker was to change its operating mode from Green to Red, the subscriber would not see any data from the publisher as it otherwise would have in Green and Yellow.

Status routers use modes to route status events that belong to the currently active set of operating modes. For example, a publisher is registered to publish a status event once every second. The publisher makes this status event available to the network every second, and if no subscribers are interested, the status event is discarded. Otherwise, the status event is forwarded along a path of status routers towards the subscriber application. A status router will only forward a status event if it belongs to a subscription that is to be utilized in at least one of the modes the status routers are currently operating in. Since every QoS broker in the management hierarchy always operates in a mode, all status routers operate in as many modes as there are levels in the management hierarchy above them. For instance, with x levels in the management hierarchy, a status router has x QoS broker ancestors (*ancestor scope*), and will therefore always operate in x modes simultaneously. This implication on the status router network enables coarse resource provisioning between mode sets, e.g. a top level QoS broker controls 40% of the available resources in its hierarchical scope while the QoS brokers beneath it must further provision the remaining resources between them.

Status routers have been extended to support and manage multiple routing tables (see Section 3.4). Each status router maintains a separate routing table for every mode defined in its ancestor scope. For example, in a GridStat configuration with a management hierarchy consisting of three levels, every status router will always operate in three modes, and therefore use three separate routing tables for routing. However, status routers might have tens or even hundreds of routing tables pre-loaded, either in memory or on disk, but only the routing tables that correspond to the active set of modes are used.

Subscribers use operating modes to determine what information is to be delivered to the user application. A subscriber retrieves a set of operating modes from its edge status router when it connects to the network or when a mode change is in progress. The subscriber delivers information to the user application only if the information belongs to one of the operating modes. There are exceptions, such as status alerts, which are delivered to the user application irrespective of operating modes. Publishers, on the other hand, do not have any notion of modes. The reason for this is that publishers only offer status events at certain time intervals irrespective of modes to the status router network. If there are no subscriptions to a published status event in the current set of operating modes, the edge status router will discard it.

3.2.1 Mode Structures

The `ModeInfo` structure is used in all communication where some information about modes is required. Subscribers use the `ModeInfo` structure to define in which modes their subscriptions should be utilized. All QoS brokers use the `ModeInfo` structure in mode change operations and leaf QoS brokers inform status routers in which modes they should operate in when they (re)connect to GridStat. The following list describes the purpose of each variable in the `ModeInfo` structure:

- **modes:** The sequence of mode identifiers is mainly used by leaf QoS broker to inform recently connected status routers which modes to operate in. The sequence is otherwise used wherever there is a need to deliver sets of operating modes between GridStat entities or within the entities themselves.

- **value:** The `value` variable is used to denote the new mode to switch to, and is used by mode change operations. For example, `value` is the mode identifier of B, if a mode change operation is initiated to change from mode A to B.

- **oldValue:** The `oldValue` variable is used to denote the current mode to switch from, and is used by mode change operations. For example, `oldValue` is the mode identifier of A, if a mode change operation is initiated to change from mode A to B.
- **QBName:** `QBName` is the name of the coordinator in a mode change operation.
- **timestamp:** The `timestamp` variable is the timestamp (in ms) at which status routers will change mode, and is used by the flooding mode change algorithm.
- **modeAction:** The `modeAction` variable denotes which phase of the hierarchical mode change algorithm the `ModeInfo` structure represents.
- **level:** The `level` variable denotes the location of the coordinator in the management hierarchy, and is used by both mode change algorithms.
- **index:** The `index` variable is used internally in the routing tables to enable easy access to the subscription intervals in different modes, and is primarily used when subscriptions are allocated.
- **changeID:** The `changeID` variable is a unique identifier within the hierarchical scope of the coordinator and is associated with each mode change operation. The variable enables quick access to stored `ModeInfo` structures through `Hashtable` and other derivations.
- **created:** The `created` variable is the timestamp of the initiation of a mode change operation, and is used by all QoS brokers and status routers to filter out redundant copies or previously executed mode change operations.

3.2.2 Propagation of Modes

All QoS brokers read information on their role in the management hierarchy from configuration files. Currently, QoS brokers read the mode definitions defined in their respective configuration files and store the modes in a `ModeContainer`. The `ModeContainer` is responsible for maintaining the mode set of a QoS broker and contains an array of `Mode` instances. Each `Mode` represents a single mode and holds the name and integer representation (mode identifier) of that mode.

The leaf QoS broker requires additional state information on the modes that are defined in its ancestor scope. The leaf QoS broker requests all the modes that are defined in its ancestor scope by contacting its parent QoS broker, which recursively repeats the process until the request reaches the root QoS broker. When the request returns, interior QoS brokers add their respective mode identifiers and operating mode. The leaf QoS broker stores the information returned from the request in an array of `ModeContainers`, one for each ancestor QoS broker. The ancestor mode set is used to inform status routers about all defined modes and operating modes they will operate in when they connect, or fail and reconnect, to GridStat. The leaf QoS broker is responsible for updating the ancestor mode set during mode change operations to reflect the operating modes of its parent QoS brokers. In this way, the leaf QoS broker is able to offer a consistent set of operating modes to status routers that might fail and reconnect to GridStat.

The subscriber receives all modes that are defined in its ancestor scope from and after it has registered with its edge status router. The set of modes allows the subscriber application to select in what modes a subscription will operate in and are sent with subscription requests to the management hierarchy. Additionally, the subscriber is informed about upcoming mode change operations from its edge status router and always knows which modes are currently active in its ancestor scope.

3.3 Mode Change Operations

A QoS broker can through modes quickly switch routing tables in the data plane. This enables the management plane to decide which status events are allowed to be forwarded through the data plane and seen by subscriber applications. Figure 3.1 shows how operating modes are used in both

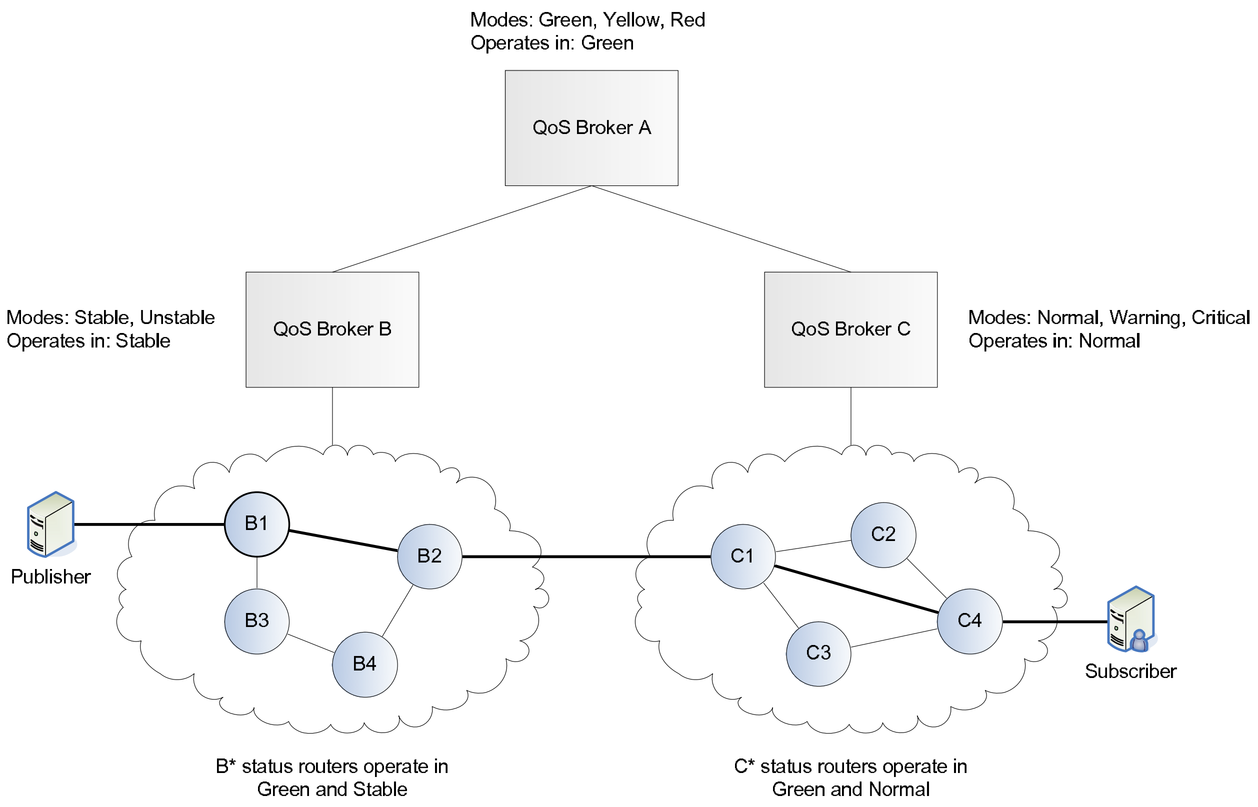


Figure 3.1: Publisher-subscriber interaction in GridStat.

the management plane and in the data plane. Each QoS broker has its own mode set and operates in a mode. All status routers in cloud B operate in Green and Stable as the QoS brokers in their ancestor scope, QoS broker A and B, operate in Green and Stable, respectively. The inter-cloud subscription can operate in all of QoS broker A's three modes Green, Yellow and Red since those modes control and manage routing tables in both clouds. Note that the inter-cloud subscription is unaffected by whichever modes leaf QoS broker B and C operate in. If the subscription is configured to operate in mode Yellow only, status router B1 does not forward status events to B2 as it is currently operating in mode Green's routing table which contains no forwarding information for the subscription.

A QoS broker can only change between modes that are defined in its mode set and acts as a *coordinator* in a mode change operation. A mode change operation that is *initiated* by a coordinator affects all the status routers and QoS brokers in its hierarchical scope. The main purpose of a mode change operation is to inform all status routers in the hierarchical scope of the coordinator to switch to the routing table associated with the operation. The number of status routers involved in a mode change operation varies with the population of status routers in the affected clouds and at what level in the management hierarchy the mode change operation was initiated from. Local mode change operations (within a cloud) might involve tens or hundreds of status routers, while a mode change operation initiated at an interior QoS broker can potentially involve several thousand status routers. One of the major challenges is to ensure that all the status routers involved in a mode change operation receive and switch to the corresponding routing table.

A mode change operation that switches the routing tables in all of the involved status routers *at the expected time* is called a *consistent* mode change operation. Otherwise the operation is called an *inconsistent* mode change operation, and additional *recovery mechanisms* (see Section 3.6) must be utilized in order to restore the operating modes on the status routers that are considered inconsistent. An inconsistent mode change operation will most likely result in some subscribers not being able to see subscribed data, as they otherwise would have after a consistent mode change operation. However, since subscription traffic is rate-based, the loss of some status events during a mode change operation is tolerable as the next status update value for a particular subscription is due to arrive within a short time period.

The following list describes the primary factors that might lead to an inconsistent mode change operation:

- **Link failure and QoS broker failure in the management plane:** In order for a mode change operation to reach the involved status routers, the operation has to be forwarded through the management hierarchy down towards the leaf QoS brokers. If a link failure or QoS broker failure prevents the operation to be forwarded to a QoS broker child, the child and all the QoS brokers and status routers in its hierarchical scope will not see the operation.

- **Link failure and status router failure in the data plane:** The leaf QoS brokers are responsible for contacting all the status routers in their hierarchical scope. When a status router is considered offline due to a link failure or a status router failure, the leaf QoS broker will not be able to forward the mode change operation to that status router. However, a link failure might not necessarily mean that a status router is offline.

- **Packet loss:** Packet loss might prevent a mode change operation to reach its desired destination. The management plane controls and manages the overall resources available in GridStat, and rejects subscription requests that exceed resources in the status router network. Thus, the management plane protects GridStat from resource overload.

- **Delayed mode change operations:** Some status routers might receive a mode change operation later than expected due to one or more factors described above. In that case, the mode change operation is deemed inconsistent until all status routers execute the mode change.

Two mode change algorithms have been implemented in GridStat v4. *The hierarchical mode change algorithm* uses the management hierarchy to disseminate mode change operations and gather acknowledgements from status routers and QoS brokers. The hierarchical mode change algorithm enables all subscriptions registered to operate in the coordinator's *current* and *new* mode to flow during the entire mode change. Thus, subscribers with subscriptions registered in both the current and new mode continue to receive status events during hierarchical mode change operations that switches between those modes. The hierarchical mode change algorithm is discussed in more detail in Chapter 4.

The *flooding mode change algorithm* disseminates mode change operations directly out on the data plane by using the limited flooding mechanism in GridStat. When a status router receives the operation it forwards the operation on all outgoing event channels, except the event channel from which it received the operation. As status routers may receive multiple copies of the same operation, redundant copies are discarded. Status routers are informed to change from the current mode to the new mode at some future timestamp. The flooding mode change algorithm is discussed in more detail in Chapter 5.

The two mode change algorithms provide different tradeoffs. The hierarchical mode change algorithm is a resource intensive algorithm which is split into five message phases in order to enable *transferred* subscriptions present in both modes in a mode change, e.g., from Green to Yellow, to flow. A message phase means the coordinator has to initiate and and propagate a mode change phase (message) down to all status routers in its hierarchical scope, and the next phase cannot be initiated until the previous phase has completed. The flooding mode change algorithm, on the other hand, is a best-effort algorithm, and disseminates mode change operations directly out on the data plane where status routers are informed to switch at a predetermined future time. The flooding mode change algorithm is efficient, in terms of resource usage and performance, but does not guarantee any subscriptions to flow during the mode change operation. The flooding mode change algorithm relies on the status routers' ability to switch modes at the exact same time and therefore requires all status routers to be time synchronized.

3.4 Multiple Routing Tables

A status router resides in a cloud that is collectively administered by a *branch* of QoS brokers in the management hierarchy. Since a QoS broker always operates in a mode, a status router operates in as many modes as there are QoS brokers in its management hierarchy branch. That is, if the management hierarchy has a depth of three levels, each status router is administered by three QoS brokers and will always operate in three modes, and therefore use three separate routing tables for forwarding status events to the next downstream status router. Status event forwarding is performed by a sequence of simple steps:

- The status router reads the header of the incoming status event and learns its ID and publisher timestamp.
- The status router looks up the ID in a status event hash table and retrives a structure commonly referred to as a mode holder.
- The mode holder contains sets of forwarding data required to forward the status event to the next downstream status router(s).
- As the status router can operate in several modes, it cycles through the forwarding data sets per operating mode and processes all the forwarding data sets.
- A forwarding set contains an event channel and the forwarding rate for the particular status event. If the status event does not conform to the forwarding rate, the status router filters it and continues to process the next forwarding set, if any.
- A status event which passes the rate filtering test is sent to the next downstream status router over the event channel given in the forwarding set.
- The status router repeats the process for all the forwarding sets that are located in all the corresponding mode holders the status router is currently operating in.

The global and hierarchical mode change mechanism has added some complexity to the overall forwarding algorithm employed by the status routers. Compared to the previous version of GridStat, status routers have to cycle through several mode holder structures instead of only one, and therefore adds some complexity to the status routers and forwarding algorithm, both in terms of resource usage and forwarding delay. Since status routers support multiple routing tables there must be certain mechanisms in place to ensure that the multicast property of GridStat is preserved, and is best illustrated by an example. For example, a status router operates in modes A and B, and is responsible for fowarding status events that belong to two subscriptions that subscribe to the same published value. Subscription 1 subscribes in mode A, subscription 2 in mode B and the status router must send the published status event to the same next downstream status router. The status router processes the mode holder for mode A and forwards the status event downstream, and continues to process the mode holder for mode B. Without any multicast preservation methods, the status router would forward the same status event to the same next downstream status router, which would clearly violate the multicast property in GridStat. This is overcome by associating an incoming status event with a *sent-map* which holds information on which outgoing event channels the status event has been forwarded.

3.5 The RPC Mechanism

The RPC mechanism is a newly added feature to GridStat v4 that allows connection-oriented communication to be conducted on top of GridStat's publisher-subscriber architecture [7]. The RPC mechanism in GridStat facilitates two-way communication between GridStat entities whereas the standard publisher-subscriber paradigm does only support one-way communication between publishers and subscribers.

In order to utilize the RPC mechanism, the two endpoints must be connected to an edge status router and embed a publisher and a subscriber to be able to send and receive information off of the GridStat network. As the RPC mechanism is built on top of the publisher-subscriber architecture in GridStat, an RPC connection consists of two subscriptions: client-server and server-client. In order to establish a connection, one of the two participants must act as the client. First, the client attempts to force a subscription on the server. That is, the client requests the management hierarchy to let the server subscribe to the client. A connection is initiated by the client which creates a subscription request containing the service type, information about the forced subscription and QoS requirements. More specifically, the subscription request contains the number of redundant paths, maximum subscription interval, maximum publication interval, operating modes, connection latency, temporal redundancy, the name of the client (publisher), the name of the server (subscriber) and the RPC status variable the server (subscriber) will subscribe to. The connection request is sent to the client's edge status router which forwards the request to the management hierarchy. The management hierarchy checks whether there exists a path between the client and server and that the path meets the specified QoS requirements. If so, the management hierarchy establishes the path(s) in the data plane and informs the client that the connection has been established. This means the server subscribes to the given RPC status variable and the client is therefore able to communicate with the server by publishing status events. The client proceeds to publish a connection request to the server along the recently established path and asks it to force a subscription on the client. The server, if it accepts the connection, creates a connection request with the QoS requirements passed to it by the client and forwards it to the management hierarchy. The management hierarchy, similarly to the first subscription, forces a subscription on the client and informs the server when the operation is completed. Since the client now subscribes to an RPC status variable published by the server, the bi-directional connection is fully established when the server publishes a connection accept to the client. Both the client and the server store the RPC connection in their state for future use.

The RPC mechanism offers several advantages compared to CORBA as it is built on top of the publisher-subscriber paradigm in GridStat:

- RPC connections can utilize the spatial redundancy property in GridStat.
- RPC connections can easier utilize a temporal redundancy scheme.
- A flexible timeout-management scheme can be employed.
- Pre- and post-conditions on the client and server provide sufficient control mechanisms to ensure that certain properties are in place prior to or after the RPC call has been executed at the receiving end. For instance, a pre-condition on a substation actuator may employ a security check to verify that a line has been de-energized prior to switching a circuit breaker.

The management hierarchy has been extended to benefit from the advantages offered by the RPC mechanism. QoS brokers and status routers have been configured to utilize the RPC mechanism and embed both a publisher instance and a subscriber instance as an alternative to CORBA. Control traffic is still conducted over CORBA, while communication related to mode change operations are implemented to utilize the RPC mechanism, and is thereby able to benefit from GridStat's QoS guarantees.

3.5.1 Initialization of The QoS Broker

The QoS broker attempts to connect both its publisher instance and subscriber instance to the same edge status router in the data plane. The chosen edge status router is configurable in the QoS broker startup scripts. Since the edge status router may be in a startup phase, the QoS broker is forced to repeatedly attempt to connect to it. Once connected, the QoS broker attempts to establish an RPC connection to its parent QoS broker and employs the connection protocol explained in Section 3.5. The root QoS broker has no parent and remains idle until it receives any incoming RPC requests from its direct QoS broker children. The management hierarchy might fail to establish RPC connections due to several reasons:

- All QoS brokers that are involved in establishing the RPC connection might not have started or they are not yet ready to handle any subscription requests.
- The server, either the QoS broker parent or its subscriber, might not have started or are not yet ready to handle any subscription requests.
- During the startup phase of GridStat, the data plane can be scarcily populated with status routers and might not satisfy the demanded QoS requirements for the RPC connection or there might simply exist no path.

The management hierarchy returns an error message if it is unable to force a subscription on the server and the QoS broker (client) must resend the subscription request at a later time. When the subscription has been established, the QoS broker continues to send a connection request to the server (parent QoS broker) which results in a fully functioning two-way communication channel between the two QoS brokers if the server successfully manages to force a subscription back on the client. Both QoS brokers finalize the connection phase by storing the RPC connection object in their internal state for later use.

3.5.2 Initialization of The Status Router

A status router embeds both a publisher instance and a subscriber instance in order to utilize the RPC mechanism and attempts to create an RPC connection to its leaf QoS broker. The status router connects the publisher instance and the subscriber instance to itself, thus eliminating any repeated connection attempts which the QoS brokers face. Once the publisher and subscriber are connected, a subscription request is created and sent to the management hierarchy in a similar manner to that of the QoS brokers. The leaf QoS broker and the status router finalize the connection phase by storing the RPC connection object in their internal state for later use.

3.5.3 Spatial and Temporal Redundancy

Conducting communication over the RPC mechanism provides more flexible control of message delivery. Since RPC connections utilize data plane resources one can associate QoS requirements with any RPC connection as is possible with standard subscriptions. Temporal and spatial redundancy allow RPCs to span several paths across the data plane and increases the probability of message delivery. Subscription latency demands enable the client to take additional measures when a delivery confirmation is not received when expected. In addition, the RPC mechanism will benefit from further development of technologies in the data plane as it completely utilizes the publisher-subscriber paradigm and mechanisms in GridStat. An ongoing project, for instance, investigates how to secure data plane communications through encryption, from which the RPC mechanism will directly benefit through its reuse of GridStat technologies (see Section 8.2.8).

Communication related to mode change operations utilize some of the overall properties provided by the RPC mechanism:

- Mode change communication over RPC within the management hierarchy or from the management hierarchy to the data plane can utilize spatial redundancy if and when needed through the use of modes.
- RPC delivery confirmations provide the means for the client to resend a mode change message, or schedule one for a later time, when a delivery confirmation is not received within the expected time window.

An RPC connection can be configured to resend the call when a delivery confirmation is not received within the expected time window. More specifically, the RPC mechanism resends the call after a preconfigured timeout and employs the temporal redundancy scheme as many times as the connection setup states. All QoS brokers add an additional layer on top of the temporal redundancy scheme provided by the RPC mechanism, and thereby have the ability to queue outgoing mode change messages and schedule them for sending by using the RPC mechanism at a later time. For example, an RPC connection can be configured to resend a mode change message five times in a row with a preconfigured timeout in between when the message is unable to reach the destinated server (detected through missing delivery confirmations). If the message has not been successfully sent after five retries, the QoS broker queues the message and schedules it for a later delivery attempt. The send scheduler forwards the message to the appropriate RPC connection where the message repeats the temporal redundancy scheme. If the RPC connection still experiences network failures or server failure, the send scheduler is responsible for delivering the mode change message when possible through the same steps outlined above. The QoS broker's send scheduler and the RPC mechanism combined provide a flexible tool to make sure mode change messages and mode change acknowledgements *eventually* get delivered to their respective destinations.

3.6 Recovery Mechanisms and Acknowledgement Aggregation

In order to tolerate some degree of network failures and to eventually ensure consistent mode change operations, a recovery mechanism was implemented to assist the hierarchical and flooding mode change algorithms. The recovery mechanism is triggered by a QoS broker that detects missing mode change acknowledgements caused by failed QoS brokers, failed status routers or link failures, and attempts to resolve these situations when possible.

An important part of a mode change operation is to gather acknowledgements from the participants in the operation. Status routers which receive a mode change operation respond with a mode change acknowledgement up to their leaf QoS broker. When a leaf QoS broker receives the first acknowledgement for a particular mode change operation it immediately starts an *aggregation round* to gather acknowledgements from its status routers. The leaf QoS broker stores the name of the status router and the mode change identifier, and starts a timer which times out to stop the aggregation round. The timeout value is subject to a pre-configured setting in the startup scripts. Additional acknowledgements are registered in a similar manner. The leaf QoS broker stops the aggregation round when all expected acknowledgements have been received; otherwise, it times out. The leaf QoS broker finalizes the aggregation round and prepares a response to the coordinator that initiated the mode change operation after the aggregation round has completed successfully. If the leaf QoS broker is the coordinator, it updates the operating modes table in its state and marks the mode change operation as complete. Otherwise, if the coordinator is located at a higher level in the management hierarchy, the leaf QoS broker updates its ancestor modes table and prepares an acknowledgement and sends it up to its parent QoS broker for further processing.

Interior QoS brokers go through the exact same sequence of steps as the leaf QoS brokers and start aggregation rounds when the first acknowledgement from one of its direct children QoS brokers are received. The process continues until the coordinator of the mode change operation has finalized the aggregation round.

An aggregation round that is terminated by the timer implies that one or more status routers, or QoS broker children, did not respond with an acknowledgement. The QoS broker initiates the recovery mechanism which attempts to contact the GridStat entities in question. The reasons why some GridStat entities neglect to respond with an acknowledgement are (child is a QoS broker or status router):

- The child is unable to send an acknowledgement up to its parent due to network failures, e.g. link failures or lossy links.
- The child itself failed.
- The child did not receive the mode change operation, or any acknowledgements if a QoS broker.

The QoS broker derives which children did not respond with an acknowledgement and attempts to contact them. If an interior QoS broker, it first contacts the direct QoS broker children in question and forces them to initiate the recovery mechanism on their own if necessary. The QoS broker children check their state to determine if the aggregation round for the mode change operation in question has started or is complete:

- If the QoS broker has no knowledge of the mode change operation in question it initiates the recovery mechanism which includes all its children (QoS brokers or status routers).
- If the QoS broker has an ongoing aggregation round for the mode change operation in question it simply discards the recovery message and initiates the recovery mechanism on its own if necessary. When the hierarchical mode change algorithm and the non-blocking scheme (Section 3.6.1) are used, the queried QoS broker sends an acknowledgent to its parent QoS broker. If the blocking scheme is employed, the queried QoS broker sends an acknowledgement to its parent QoS broker first when it has received all awaiting acknowledgements.

- The QoS broker has completed the aggregation round for the mode change operation and might be unable to respond with an acknowledgement to the parent QoS broker due to network failures. This scenario is probable, but less likely than the ones above, as the recovery message from the parent QoS broker managed to get delivered.

The leaf QoS broker employs the same scheme as interior QoS brokers but will instead attempt to contact the status routers that did not respond with an acknowledgement. The recovery mechanism affects the data plane as follows:

- The status router first checks if the mode change operation is pending (waiting to be activated) or has already been registered (activated).
- If the mode change operation is not registered, the status router executes the operation and responds with an acknowledgement.
- If the mode change operation is registered, the status router responds with an acknowledgement.

The recovery mechanism is initiated when network failures, QoS broker failures or status router failures prevent all participants from delivering and aggregating acknowledgements up towards the coordinator in the management hierarchy. Therefore, querying participants for their acknowledgements and status may be a time-consuming process, and not necessarily as straightforward as outlined above. For instance, a QoS broker may be unable to send a recovery message to one of its children because a link failure prevents the message to be delivered. The same child might have completed its aggregation round, but is unable to send the acknowledgement to its parent because of the same link failure. A similar scenario can be envisioned when a QoS broker or status router fails. In such circumstances, the sender queues recovery messages in an outgoing pool and delivers them when the QoS broker or status router rejoins the GridStat network (see Section 3.5.3).

3.6.1 Blocking and Non-Blocking Aggregation

The hierarchical mode change algorithm can employ a blocking or non-blocking scheme. A blocking scheme prevents the next mode change phase to commence until all status routers have executed the previous mode change phase and delivered acknowledgements to the management hierarchy. When the coordinator receives confirmation that all status routers and QoS brokers have seen and processed the mode change phase it initiates the next phase. Alternatively, a non-blocking scheme allows the coordinator to initiate the next mode change phase even though the previous mode change phase has not completed. That is, if QoS broker failures, status router failures, link failures or temporal network anomalies (e.g., link loss) prevent some status routers in executing the mode change phase, the management hierarchy initiates the recovery mechanism and in parallel sends acknowledgements up towards the coordinator. The coordinator assumes that lower-level QoS brokers restore the modes at deemed inconsistent status routers when possible through the recovery mechanism when it receives acknowledgements from its direct QoS broker children.

CHAPTER FOUR

DESIGN OF THE HIERARCHICAL MODE CHANGE ALGORITHM

The hierarchical mode change algorithm is an extension of the mode change algorithm in GridStat v3 and is modified to be utilized in a management hierarchy of any size and from any location. Currently, the application layer in the management plane is responsible for initiating a mode change operation by using the QoS broker API. It is envisioned that in the future this process will be policy driven, initiated as the implementation already supports or initiated from an authenticated application in the data plane. A mode change operation is initiated by invocating the changeMode method on either a QoSBroker or LeafQoSBroker instance. The hierarchical mode change algorithm is divided into five distinct phases which enable *transferred* subscriptions present in both modes of a mode change operation, e.g., from Green to Yellow, to flow. Furthermore, the five phases eliminate any status router overload scenarios during a hierarchical mode change operation. The following list shows how the hierarchical algorithm affects the status routers in a mode change from *Green* to *Yellow*.

1. **The inform phase** - Edge status routers inform their subscribers about the upcoming mode change. This phase is a standalone phase in order to ensure that all subscribers have been informed about potential QoS violations prior to switching routing tables.

2. **The prepare phase** - Edge status routers switch to the temporary routing table *Green* ∩ *Yellow*. This phase ensures that subscription traffic that belongs in both modes (*Green* and *Yellow*) is forwarded through the status router network. Subscription traffic that belong to either *Green* or *Yellow* is dropped at the edge status routers. This step in the hierarchical mode change algorithm eliminates any status router overload scenarios (incoming queues and outgoing queues) as subscriptions are only removed.

3. **The internal change phase** - Internal status routers switch to *Yellow*'s routing table. Since

all edge status routers operate in a temporary routing table and only forward a smaller set of subscriptions (in mode *Green* and *Yellow*), the internal status routers can safely switch to mode *Yellow* without overloading any status routers downstream.

4. **The edge change phase** - Edge status routers switch from the temporary routing table *Green* ∩ *Yellow* to *Yellow*'s routing table. Since internal status routers operate in *Yellow* and expect to receive subscription traffic for mode *Yellow*, it is safe for edge status routers to finally switch.

5. **The commit phase** - Edge status routers inform their subscribers about the completed mode change. This phase is a standalone phase in order to ensure that all modification to routing tables in the status router network is complete and that subscribers will receive status events conforming to the desired QoS.

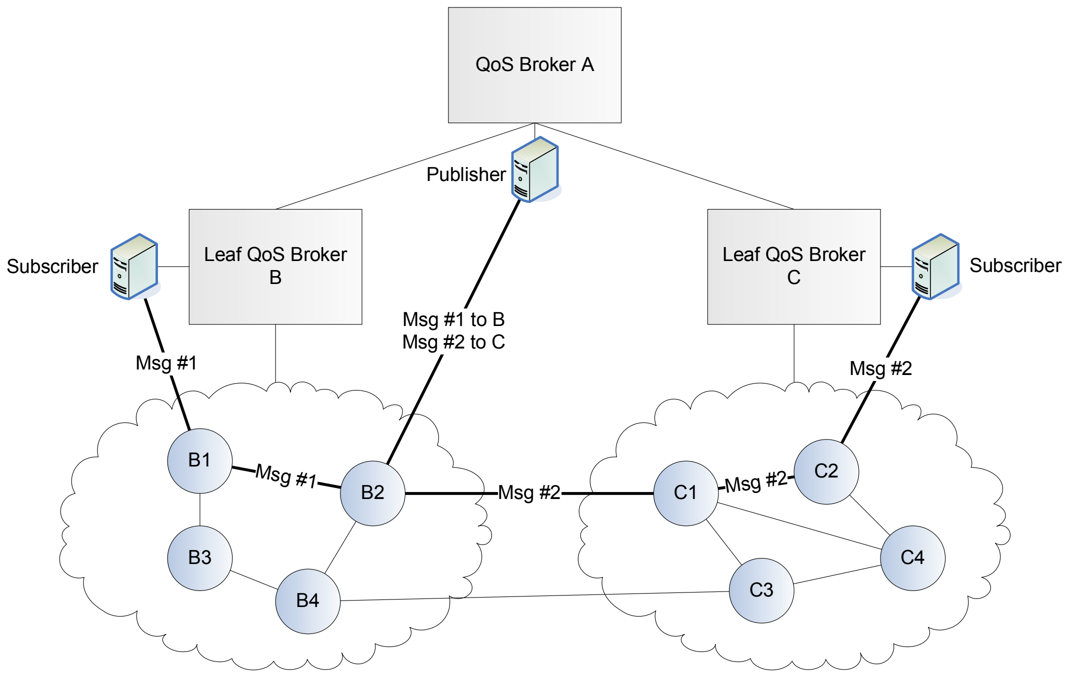


Figure 4.1: The hierarchical algorithm - hierarchical dissemination

Common for all the phases in the hierarchical mode change algorithm is the propagation of the operation down the management hierarchy towards the data plane. The coordinator of the mode change operation sends the operation to all its children QoS brokers, and they repeat the process until the operation reaches the leaf QoS brokers. Figure 4.1 shows how the root QoS broker uses its RPC connections to send a mode change operation (phase) down to its children leaf QoS brokers B and C. The leaf QoS brokers forward the operation to all the status routers in their administrative domains.

Figure 4.2 depicts the leaf QoS brokers forwarding the mode change operation (phase) to the status routers in their respective control domains. Both leaf QoS brokers forward four copies (four status routers) of the mode change operation to their status routers. The operation is executed by each status router which responds with an acknowledgement. The leaf QoS brokers gather acknowledgements and send a single acknowledgement up to their parent QoS broker.

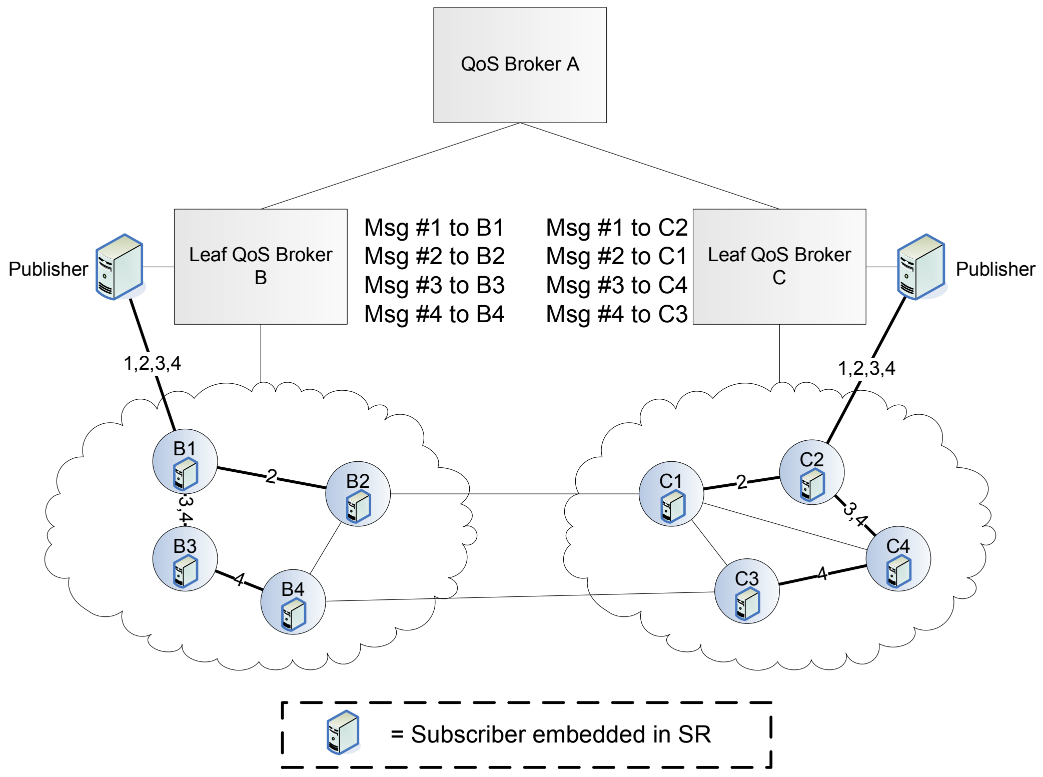


Figure 4.2: The hierarchical algorithm - cloud dissemination

The following detailed explanation of the individual mode change phases assumes a blocking scheme (see Section 3.6.1) in which the management hierarchy blocks the execution of the next phase until all participants have executed the current mode change phase.

4.1 Phase 1 - The Inform Phase

The first phase of the hierarchical mode change algorithm is the inform phase. The coordinator propagates the mode change operation down the hierarchy and towards all the *edge status routers* in its hierarchical scope. The edge status routers inform all their subscribers about the upcoming mode change. The inform message makes subscribers aware that subscription terminations are due to the upcoming mode change operation, and will not deliver QoS violation callbacks to their overlying application. Figure 4.3 shows edge status routers contacting their subscribers about the

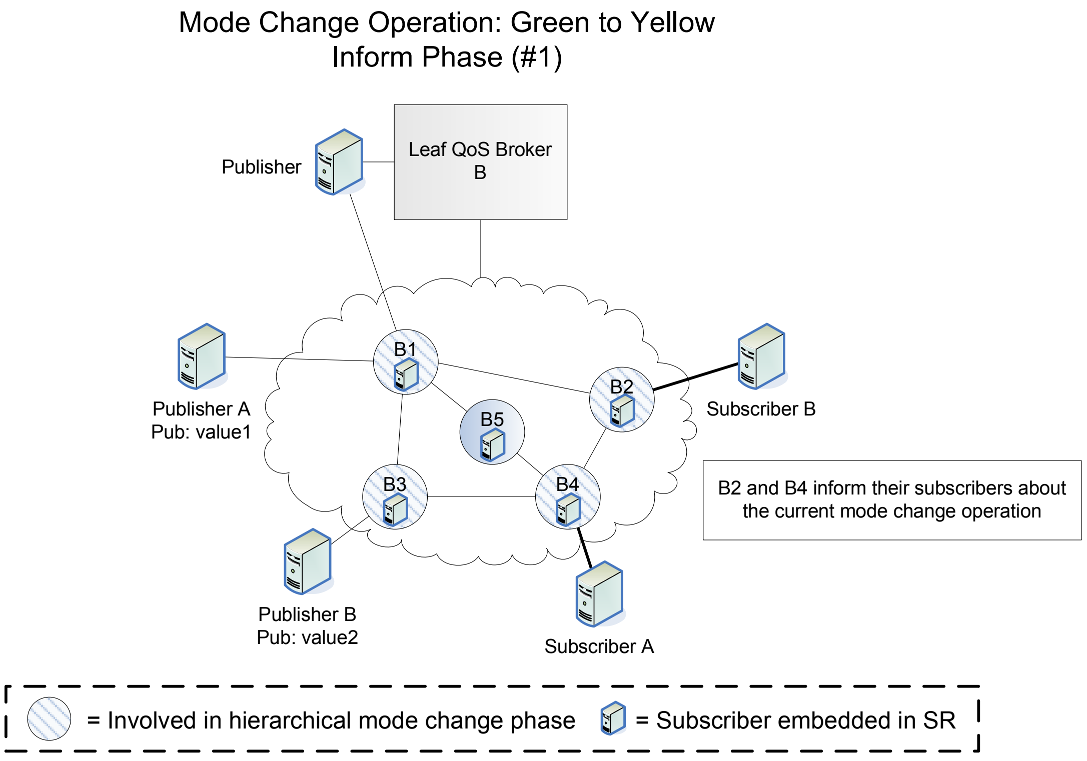

Mode Change Operation: Green to Yellow
Inform Phase (#1)
Publisher
Leaf QoS Broker B
B1
B2
B3
B4
B5
Publisher A Pub: value1
Publisher B Pub: value2
Subscriber A
Subscriber B
B2 and B4 inform their subscribers about the current mode change operation
= Involved in hierarchical mode change phase
= Subscriber embedded in SR

Figure 4.3: The hierarchical algorithm - inform phase

upcoming mode change operation. When all subscribers have been informed, the edge status router sends an acknowledgement up to the management hierarchy.

4.2 Phase 2 - The Prepare Phase

The second phase of the hierarchical mode change algorithm is the prepare phase. The coordinator propagates the mode change operation down the hierarchy and towards all the *edge status routers* in its hierarchical scope. The edge status routers create a temporary routing table with all the forwarding information for subscriptions registered in both the old and the new mode (see Section 4.6). For example, the prepare phase of a mode change from A to B creates a temporary routing table that contains those subscriptions registered in both A and B. The subscriptions contained in the temporary routing table are forwarded according to the highest subscription interval (lowest rate) selected from the two modes. As subscriptions only can be removed, no resources will be exhausted during this phase of the algorithm. Figure 4.4 shows the operating modes of the status

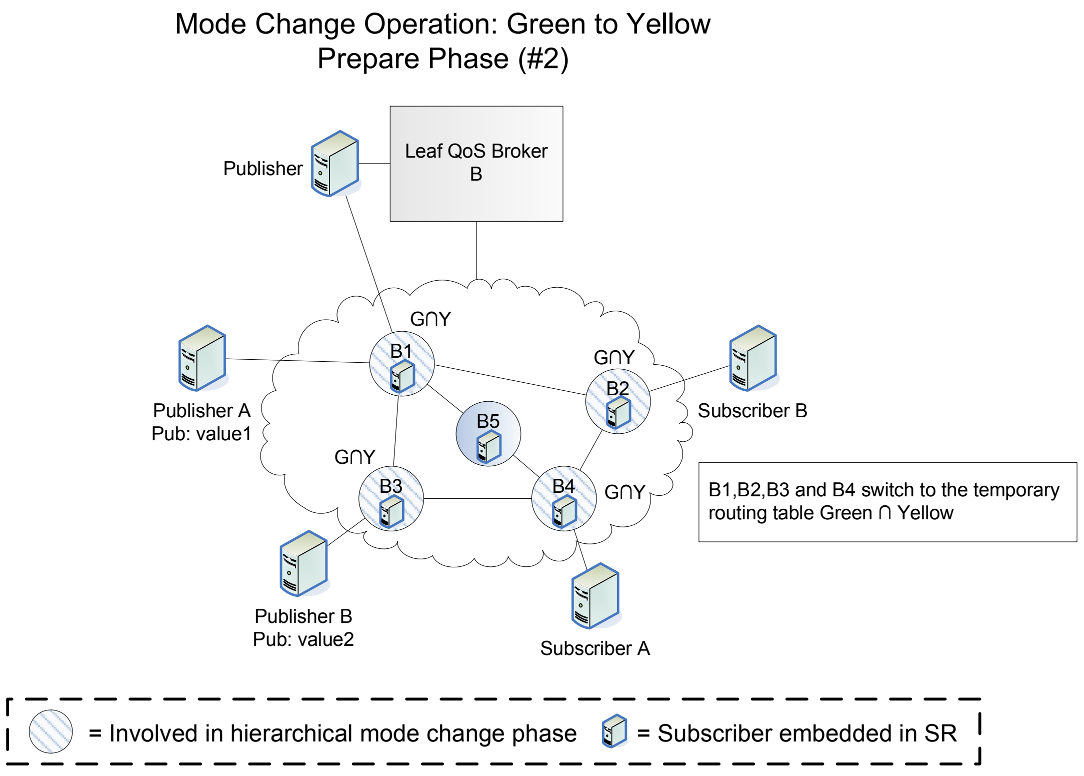
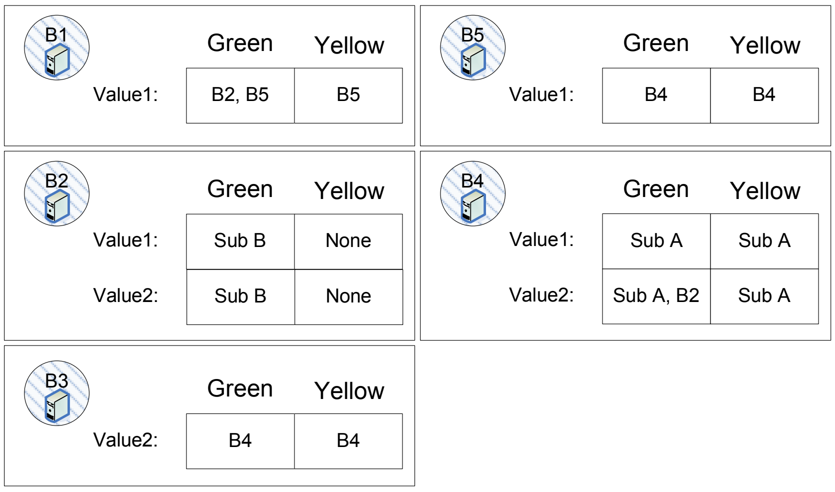

Mode Change Operation: Green to Yellow
Prepare Phase (#2)
Publisher
Leaf QoS Broker B
G∩Y
B1
G∩Y
B2
B5
Subscriber B
Publisher A
Pub: value1
G∩Y
B3
B4
G∩Y
B1,B2,B3 and B4 switch to the temporary routing table Green ∩ Yellow
Publisher B
Pub: value2
Subscriber A
= Involved in hierarchical mode change phase
= Subscriber embedded in SR

Figure 4.4: The hierarchical algorithm - prepare phase

routers when the prepare phase is executed. When the temporary routing tables have been successfully created, the edge status routers respond with an acknowledgement up to the management hierarchy.

Assume the following subscriptions are in place prior to executing the prepare phase (see Figure 4.4): Subscriber A subscribes to value1 and value2 in modes Green and Yellow, and subscriber B subscribes to value1 and value2 in mode Green. The routing tables at all status routers, B1 through B5, are presented in Figure 4.5.



B1	Green	Yellow
Value1:	B2, B5	B5

B5	Green	Yellow
Value1:	B4	B4

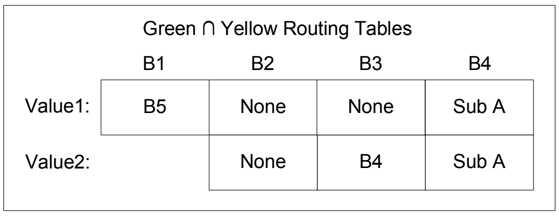
B2	Green	Yellow
Value1:	Sub B	None
Value2:	Sub B	None

B4	Green	Yellow
Value1:	Sub A	Sub A
Value2:	Sub A, B2	Sub A

B3	Green	Yellow
Value2:	B4	B4

Figure 4.5: The hierarchical algorithm - routing tables

The temporary routing tables Green ∩ Yellow at all edge status routers, B1 through B4, are presented in Figure 4.6. When all edge status routers operate in Green ∩ Yellow, B1 forwards value1 to B5 only as subscriber B does not subscribe to value1 in mode Yellow. B5 still operates in mode Green, and forwards value1 to B4 according to its routing table (see Figure 4.5) and B4 delivers value1 to subscriber A. The second publication, value2, is only subscribed to by subscriber A, and is forwarded along the path from B3 to B4.


Green ∩ Yellow Routing Tables

	B1	B2	B3	B4
Value1:	B5	None	None	Sub A
Value2:		None	B4	Sub A

Figure 4.6: The hierarchical algorithm - temporary routing tables

4.3 Phase 3 - Internal Change Phase

The third phase of the hierarchical mode change algorithm is the internal change phase. The coordinator propagates the mode change operation down the hierarchy and towards all the *internal status routers* in its hierarchical scope. All internal status routers will at this point in the entire mode change operation switch to the routing table of the new mode. The internal status routers will receive status events subject to the highest subscription interval (lowest rate) as edge status routers are currently operating in the temporary routing table. Subscribers will observe one of the following two scenarios:

- If the subscription interval in the new mode is higher than that of the old mode, the subscriber receives status events belonging to this stream with the subscription interval of the new mode.
- If the subscription interval in the new mode is lower than that of the old mode, the subscriber receives status events belonging to this stream with the subscription interval of the old mode.

No resources can be exhausted during this phase as the largest subscription interval from either the old or the new mode is used. Furthermore, involved subscribers will not issue QoS violation callbacks to the overlying applications as they have been informed about the mode change. Figure 4.7 shows the routing table compositions in the data plane after the internal change phase is complete. When the internal status routers have switched to the new mode, the status routers send acknowledgements up to the management hierarchy.

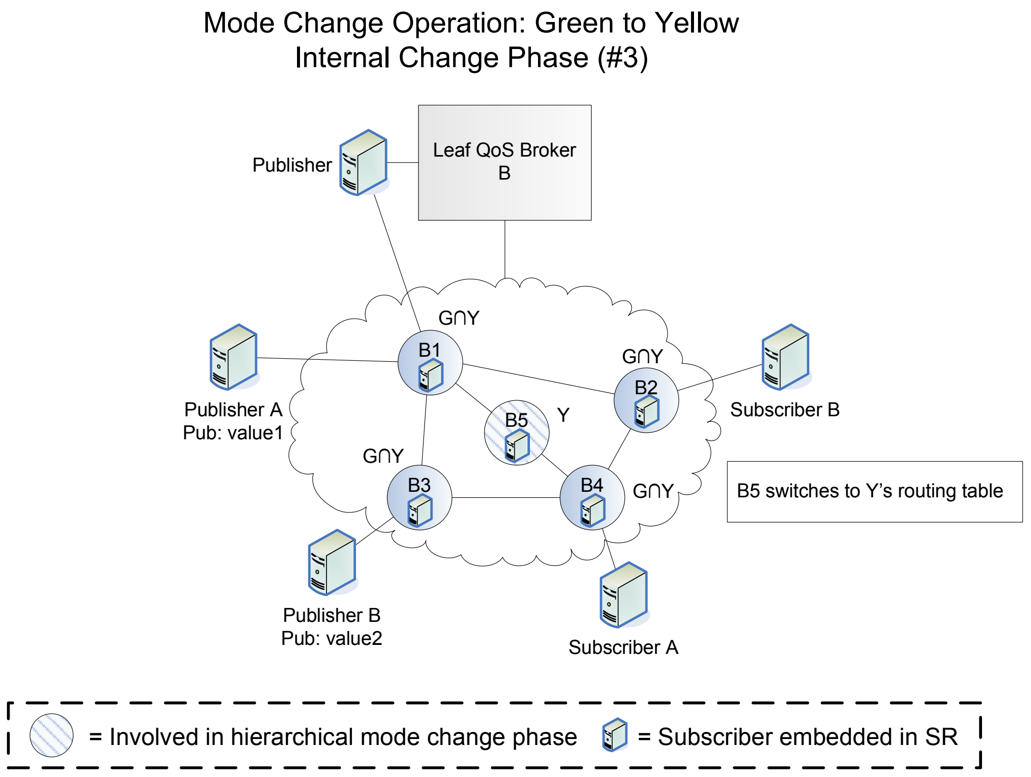

Mode Change Operation: Green to Yellow
Internal Change Phase (#3)
Publisher
Leaf QoS Broker B
G∩Y
B1
G∩Y
B2
Subscriber B
Publisher A
Pub: value1
B5
Y
G∩Y
B3
B4
G∩Y
B5 switches to Y's routing table
Publisher B
Pub: value2
Subscriber A
= Involved in hierarchical mode change phase
= Subscriber embedded in SR

Figure 4.7: The hierarchical algorithm - internal change phase

4.4 Phase 4 - Edge Change Phase

The fourth phase of the hierarchical mode change algorithm is the edge change phase. The coordinator propagates the mode change operation down the hierarchy and towards all the *edge status routers* in its hierarchical scope. The edge status routers switch from the temporary routing table created in the prepare phase to the routing table of the new mode. Hence, edge status routers forward status events conforming to the subscription interval of the new mode. Figure 4.8 shows that all status routers operate in the new mode after the edge change phase is complete. The edge status routers finalize the fourth phase of the hierarchical algorithm by sending acknowledgements up to the management hierarchy.

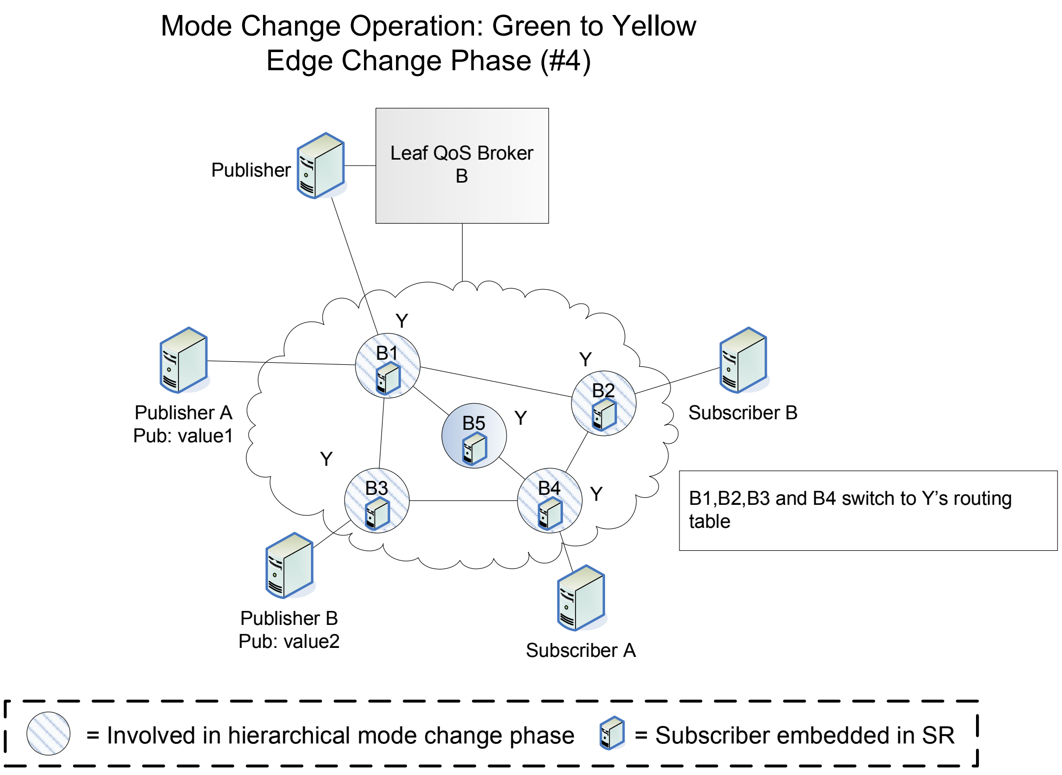

Mode Change Operation: Green to Yellow
Edge Change Phase (#4)
Publisher
Leaf QoS Broker B
Y
B1
Y
B2
Subscriber B
Publisher A
Pub: value1
B5
Y
Y
B3
B4
Y
B1,B2,B3 and B4 switch to Y's routing table
Publisher B
Pub: value2
Subscriber A
= Involved in hierarchical mode change phase
= Subscriber embedded in SR

Figure 4.8: The hierarchical algorithm - edge change phase

4.5 Phase 5 - The Commit Phase

The fifth phase of the hierarchical mode change algorithm is the commit phase. The coordinator propagated the mode change operation down the hierarchy and towards all the *edge status routers* in its hierarchical scope. The edge status routers inform their subscribers about the completed mode change. Subscribers will at this time expect to receive subscribed status events at the subscription interval of the newly activated mode if the overall mode change algorithm was consistent. Figure 4.9 shows edge status routers contacting their subscribers to inform about the completed mode change operation. The edge status routers finalize the phase by sending acknowledgements up to the management hierarchy.

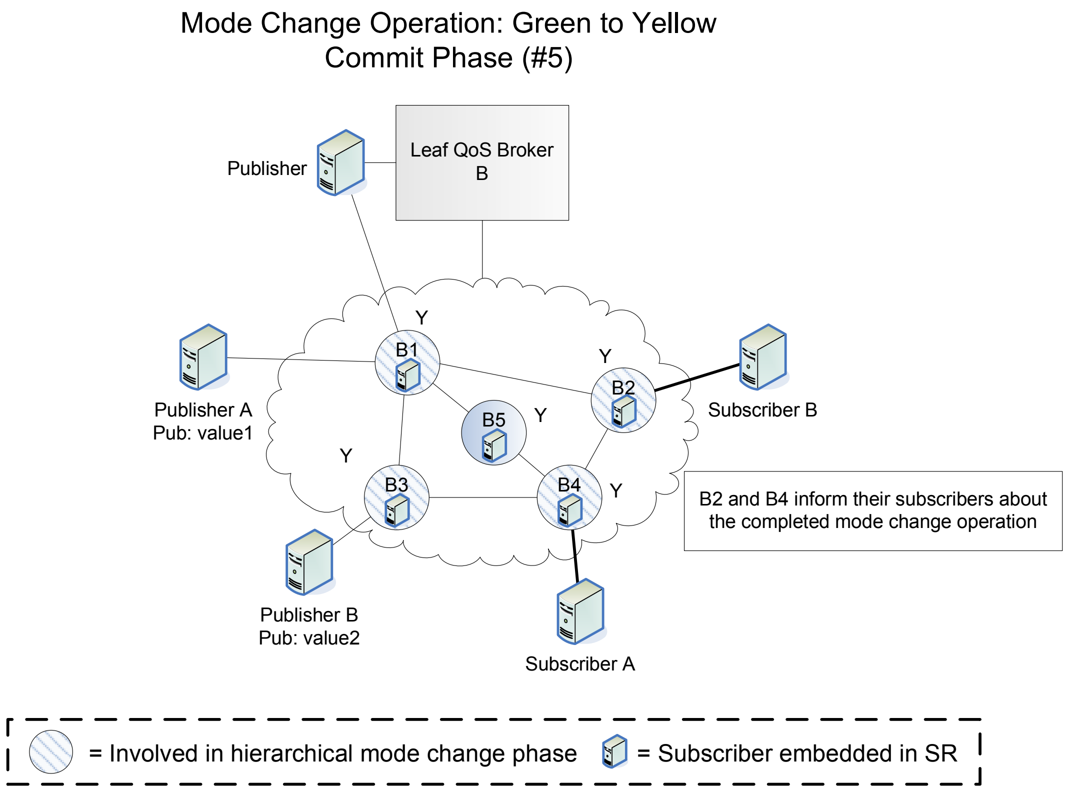

Mode Change Operation: Green to Yellow
Commit Phase (#5)
Publisher
Leaf QoS Broker
B
Y
B1
Y
B2
Subscriber B
Publisher A
Pub: value1
B5
Y
Y
B3
B4
Y
B2 and B4 inform their subscribers about the completed mode change operation
Publisher B
Pub: value2
Subscriber A
= Involved in hierarchical mode change phase
= Subscriber embedded in SR

Figure 4.9: The hierarchical algorithm - commit phase

4.6 Construction of Temporary Routing Tables

A crucial part of any hierarchical mode change operation is to create a temporary routing table that consists of forwarding information for subscriptions present in the modes a mode change operation switches from and to. For example, a mode change operation switching from Green to Yellow creates a temporary routing table Green $\cap$ Yellow. Every edge status router that participates in a hierarchical mode change operation has to create this routing table when informed to by the management hierarchy. A temporary routing table is created by retrieving the mode holders for the two modes involved in a hierarchical mode change operation and adding the forwarding information that is present in both modes to a new temporary mode holder. Note that a subscription configured to operate in two or more modes utilize the same path through the data plane in both modes, but may be forwarded at different rates. The status router selects the highest subscription interval (lowest rate) from the two modes for a particular subscription and uses it when the status router operates in the temporary mode. The temporary routing table is created during the prepare phase of the hierarchical mode change algorithm and causes no resource exhaustion as subscriptions are only removed.

4.7 Rate Filtering

This section investigates how the hierarchical mode change phases, prepare, internal change and edge phase affect a subscription that is to operate in modes Green and Yellow. Figure 4.10 shows that the edge status routers, B1 and B4, operate in mode Green ∩ Yellow while B5 still operates in mode Green. The subscriber subscribes to the publisher's only publication value1 every 100 ms

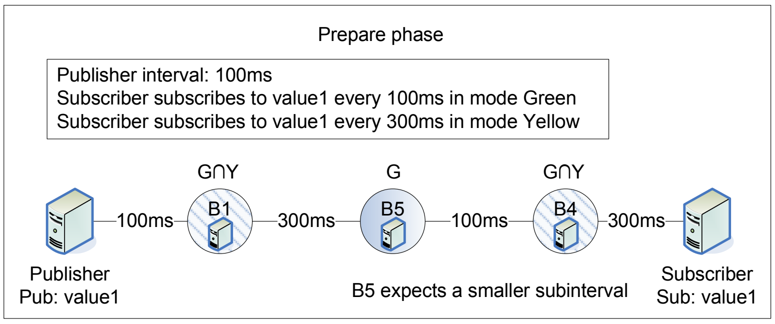

Prepare phase
Publisher interval: 100ms
Subscriber subscribes to value1 every 100ms in mode Green
Subscriber subscribes to value1 every 300ms in mode Yellow
G∩Y
G
G∩Y
B1
B5
B4
100ms
300ms
100ms
300ms
Publisher
Pub: value1
B5 expects a smaller subinterval
Subscriber
Sub: value1

Figure 4.10: The hierarchical algorithm - subscription rates after the prepare phase

in mode Green, and every 300 ms in mode Yellow. The publisher publishes status events (value1) every 100 ms. Since B1 operates in Green ∩ Yellow it will forward status events from the publisher according to the highest subscription interval (lowest rate) from Green or Yellow, and will therefore forward 1 out of 3 received status events to B5. B5, on the other hand, expects to receive status events from B1 at a higher rate (100 ms), but will simply forward status events to B4 when it receives them. That is, at an interval of 300 ms. B4 forwards every status event to the subscriber as it receives them at the expected subscription interval and is not required to perform any rate filtering. If the subscription intervals were reversed the subscriber would still receive status events at the same interval as edge status routers pick the highest subscription interval (lowest rate) from the two modes involved in the mode change operation. The only difference is that B5 receives status events at the expected subscription interval (300 ms).

Figure 4.11 illustrates how the next phase, the internal change phase, affects the subscription. The internal status router B5 switches from Green to Yellow and forwards status events that belong

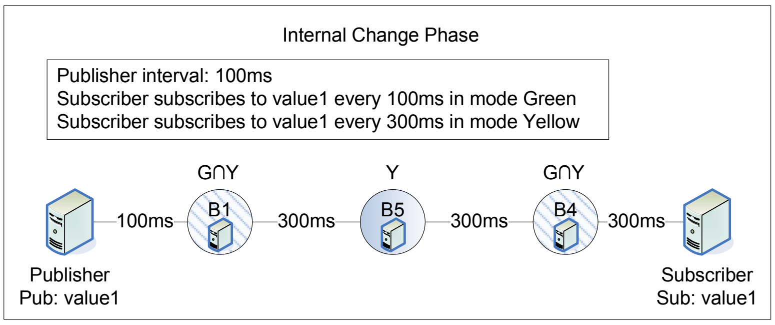

Internal Change Phase
Publisher interval: 100ms
Subscriber subscribes to value1 every 100ms in mode Green
Subscriber subscribes to value1 every 300ms in mode Yellow
G∩Y
Y
G∩Y
100ms
B1
300ms
B5
300ms
B4
300ms
Publisher
Pub: value1
Subscriber
Sub: value1

Figure 4.11: The hierarchical algorithm - subscription rates after the internal change phase

to the single subscription downstream every 300 ms. As B5 receives those status events every 300 ms it is not forced to perform any rate filtering.

Figure 4.12 shows the operating modes and subscription intervals used after B1 and B4 change from Green ∩ Yellow to Yellow. Although none of the rates alter for this example, the edge status

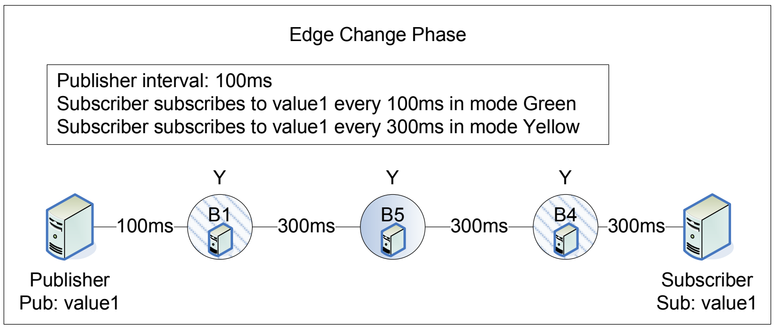

Edge Change Phase
Publisher interval: 100ms
Subscriber subscribes to value1 every 100ms in mode Green
Subscriber subscribes to value1 every 300ms in mode Yellow
Y
Y
Y
100ms
B1
300ms
B5
300ms
B4
300ms
Publisher
Pub: value1
Subscriber
Sub: value1

Figure 4.12: The hierarchical algorithm - subscription rates after the edge change phase

routers forward status events conforming to the subscription interval in mode Yellow.

The rationale for selecting the highest subscription interval (lowest rate) in the prepare phase is not to overload any internal status routers with traffic during the edge change phase. Overload scenarios at internal status routers in this section refer to incoming status router queues. Figure

4.13 and 4.14 show what happens if edge status routers pick the subscription interval from mode Green in a hierarchical mode change operation switching from Green to Yellow.

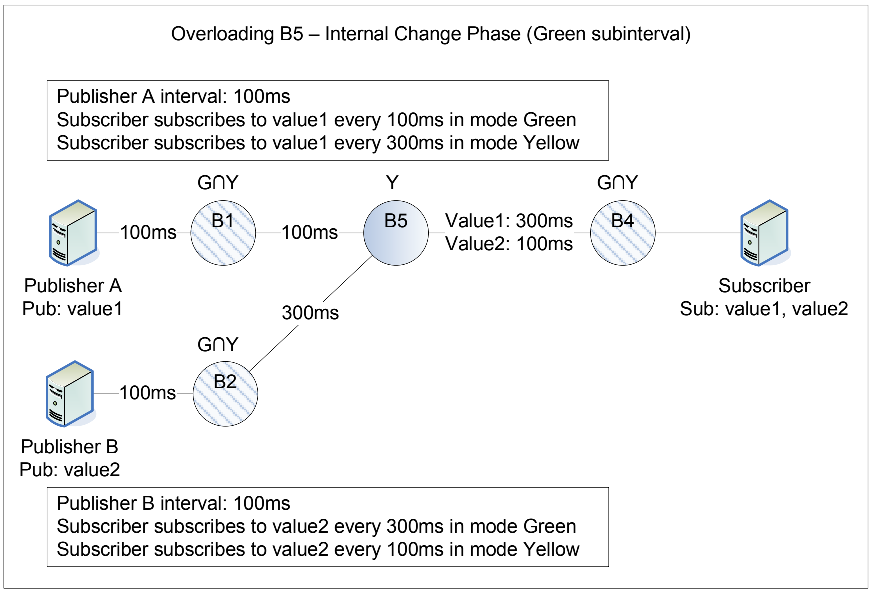

Overloading B5 – Internal Change Phase (Green subinterval)
Publisher A interval: 100ms
Subscriber subscribes to value1 every 100ms in mode Green
Subscriber subscribes to value1 every 300ms in mode Yellow
G∩Y
Y
G∩Y
100ms
B1
100ms
B5
Value1: 300ms
Value2: 100ms
B4
Publisher A
Pub: value1
300ms
Subscriber
Sub: value1, value2
G∩Y
100ms
B2
Publisher B
Pub: value2
Publisher B interval: 100ms
Subscriber subscribes to value2 every 300ms in mode Green
Subscriber subscribes to value2 every 100ms in mode Yellow

Figure 4.13: Status router resource overload - internal phase

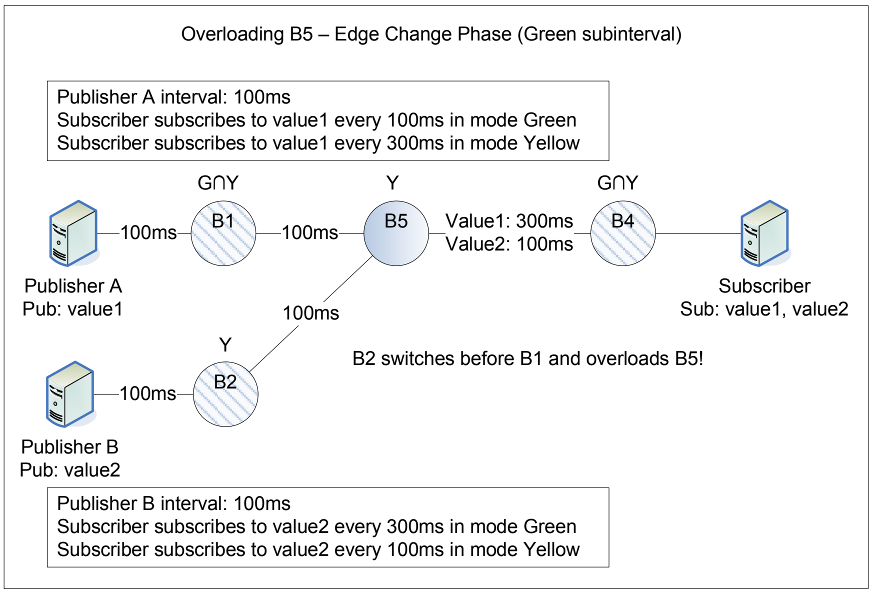

Overloading B5 – Edge Change Phase (Green subinterval)
Publisher A interval: 100ms
Subscriber subscribes to value1 every 100ms in mode Green
Subscriber subscribes to value1 every 300ms in mode Yellow
G∩Y
Y
G∩Y
100ms
B1
100ms
B5
Value1: 300ms
Value2: 100ms
B4
Publisher A
Pub: value1
100ms
Subscriber
Sub: value1, value2
Y
B2 switches before B1 and overloads B5!
100ms
B2
Publisher B
Pub: value2
Publisher B interval: 100ms
Subscriber subscribes to value2 every 300ms in mode Green
Subscriber subscribes to value2 every 100ms in mode Yellow

Figure 4.14: Status router resource overload - edge change phase

The subscriber subscribes to value1 at publisher A every 100 ms in mode Green and 300 ms in mode Yellow. The secondary subscription is between the subscriber and publisher B with subscription intervals 300 ms and 100 ms, in mode Green and Yellow, respectively. An assumption is that B5 becomes overloaded with two incoming streams at 100 ms. Figure 4.13 shows that B1 and B2 forward status events to B5 with rates 100 ms and 300 ms.

Figure 4.14 assumes that B2 switches from Green $\cap$ Yellow to Yellow prior to B1 and causes B5 to become overloaded as the subscriber subscribes to value2 every 100 ms in mode Yellow. At this point, B5 remains overloaded until B1 changes to mode Yellow. If the edge status routers picked the highest subscription interval in the prepare phase, B5 would not become overloaded when B2 switches mode prior to B1 in the edge change phase. The reason is that B1 and B2 would forward status events from publisher A and B at a subscription interval of 300 ms, and when B2 switches to mode Yellow prior to B1 in the edge change phase it will not cause any problems for B5.

4.8 Pseudo-Code for The Hierarchical Mode Change Algorithm

The following pseudo-code and explanations show how a mode change phase is propagated from the coordinator and down through the management hierarchy towards a set of status routers. Furthermore, the pseudo-code for each individual mode change phase is presented at the status router level.

4.8.1 Coordinator Code

The following code outline shows the simple steps a coordinator (QoS broker) has to follow in order to switch modes by using the hierarchical mode change algorithm. It first defines a ModeInfo container (See Section 3.2.1) with the required parameters and sends it to its direct QoS broker children. Then, it awaits acknowledgements from its direct QoS broker children prior to initiating the next mode change phase.

```
for Each mode change phase do
    Define a ModeInfo container with the required parameters
    Register the mode change phase as pending
    for Each QoS broker child do
        Send the mode change phase over the established RPC connection
    end for
    Await acknowledgements from each direct QoS broker child
    Mark the mode change phase as complete
end for
```

4.8.2 Interior QoS Broker Mediator Code

An interior QoS broker in the hierarchical scope of the coordinator simply forwards the mode change phase to its direct QoS broker children.

```
for Each QoS broker child do
    Send the mode change phase over the established RPC connection
end for
```

4.8.3 Leaf QoS Broker Mediator Code

In a similar manner to an interior QoS broker, a leaf QoS broker forwards the mode change phase to a set of status routers in its administrative cloud.

```
Retrieve all internal status routers from state
Retrieve all edge status routers from state
if Inform phase then
    Send the mode change phase to all edge status routers
end if
```

```
if Prepare phase then
    Send the mode change phase to all edge status routers
end if
if Internal change phase then
    Send the mode change phase to all internal status routers
end if
if Edge change phase then
    Send the mode change phase to all edge status routers
end if
if Commit phase then
    Send the mode change phase to all edge status routers
end if
```

4.8.4 Inform Phase Code

The following code outline shows the process of informing subscribers about the upcoming mode change operation from an edge status router. If the mode change phase has already been executed, the edge status router responds with an acknowledgement to its leaf QoS broker. Otherwise, it retrieves the list of all connected subscribers and informs them about the upcoming mode change operation. Finally, it responds with an acknowledgement to its leaf QoS broker.

```
if Mode change phase has previously been executed then
    Send an acknowledgement up to the management hierarchy
else
    Register the mode change phase
    for Each subscriber do
        Inform the subscriber about the upcoming mode change operation
    end for
    Send an acknowledgement up to the management hierarchy
end if
```

4.8.5 Prepare Phase Code

The following code outline shows the process of creating the temporary routing table Green ∩ Yellow in a mode change from Green to Yellow at an edge status router. First, if the mode change phase has previously been executed, the edge status router responds with an acknowledgement to its leaf QoS broker. Otherwise, it locks the routing table(s) and creates a temporary routing table (Green ∩ Yellow). The edge status router adds forwarding rules for the subscriptions that are registered to operate in both modes to the temporary routing table and picks the highest subscription interval. Finally, it switches from Green to Green ∩ Yellow, unlocks the routing table(s) and sends an acknowledgement to its leaf QoS broker.

```
if Mode change phase has previously been executed then
    Send an acknowledgement up to the management hierarchy
else
    Register the mode change phase
    Lock the routing table
    Create an empty temporary routing table
    for Each subscription in Green do
        if Subscription in Yellow then
            Add the forwarding rules for the subscription to the temporary routing table
            Pick the highest subscription interval from Green or Yellow
        end if
    end for
```

```
        Change from Green’s routing table to the temporary routing table
        Unlock the routing table
        Send an acknowledgement up to the management hierarchy
    **end if**
```

4.8.6 Internal Change Phase Code

The following code outline shows the process of switching from Green to Yellow in an internal status router. First, if the mode change phase has previously been executed, the edge status router responds with an acknowledgement to its leaf QoS broker. Otherwise, it lock the routing table(s) and switches to the routing table for mode Yellow. Finally, it unlocks the routing table(s) and sends an acknowledgement to its leaf QoS broker.

```
    **if** Mode change phase has previously been executed **then**
        Send an acknowledgement up to the management hierarchy
    **else**
        Register the mode change phase
        Lock the routing table
        Change to Yellow’s routing table
        Unlock the routing table
        Send an acknowledgement up to the management hierarchy
    **end if**
```

4.8.7 Edge Change Phase Code

The following code outline shows the process of switching from the temporary routing table Green ∩ Yellow to Yellow’s routing table. First, if the mode change phase has previously been executed, the edge status router responds with an acknowledgement to its leaf QoS broker. Otherwise, it locks the routing table(s) and switches from the temporary routing table created in the prepare phase to the routing table of mode Yellow. Finally, it unlocks the routing table(s) and sends an acknowledgement to its leaf QoS broker.

```
if Mode change phase has previously been executed then
    Send an acknowledgement up to the management hierarchy
else
    Register the mode change phase
    Lock the routing table
    Change from the temporary routing table to Yellow's routing table
    Unlock the routing table
    Send an acknowledgement up to the management hierarchy
end if
```

4.8.8 Commit Phase Code

The following code outline shows the process of informing subscribers about the completed mode change operation from an edge status router. If the mode change phase has already been executed, the edge status router responds with an acknowledgement to its leaf QoS broker. Otherwise, it retrieves the list of all connected subscribers and informs them about the completed mode change operation. Finally, it responds with an acknowledgement to its leaf QoS broker.

```
if Mode change phase has previously been executed then
    Send an acknowledgement up to the management hierarchy
else
    Register the mode change phase
    for Each subscriber do
        Inform the subscriber about the completed mode change operation
    end for
    Send an acknowledgement up to the management hierarchy
end if
```

4.8.9 Leaf QoS Broker Acknowledgement Aggregation Code

The following code outline shows the actions of a leaf QoS broker after all status routers have acknowledged the pending mode change phase. If the coordinator resides higher up in the management hierarchy, the leaf QoS broker sends an acknowledgement up to its parent QoS broker. Otherwise, it continues with the next mode change phase.

```
Retrieve the list of status routers from state
if Mode change phase is the commit phase then
    for Each status router in state do
        Update the modes table to reflect the operating modes of this status router
    end for
    Update the list of what modes all status router should operate in
end if
if The coordinator is higher up in the management hierarchy then
    Send acknowledgement to parent QoS broker
else
    if Commit phase then
        Update operating modes
    else
        Initiate next phase
    end if
end if
```

```
Mark the mode change phase as complete
```

4.8.10 *Interior QoS Broker Acknowledgement Aggregation Code*

The following code outline shows the actions of an interior QoS broker after all direct QoS broker children have acknowledged the pending mode change phase. If the coordinator resides higher up in the management hierarchy, the QoS broker sends an acknowledgement up to its parent QoS broker. Otherwise, it continues with the next mode change phase.

```
if The coordinator is higher up in the management hierarchy then
  Send acknowledgement to parent QoS broker
else
  if Commit phase then
    Update operating modes
  else
    Initiate next phase
  end if
end if
Mark the mode change phase as complete
```

4.9 Failure Scenarios

The hierarchical mode change algorithm is able to consistently change between modes and enables *transferred* subscriptions to flow under the assumption that all status routers execute the mode change phases sequentially (blocking scheme) and in the correct timeframe. By using the non-blocking scheme (see Section 3.6.1) in hierarchical mode change operations, QoS broker failures, status router failures and network failures may interfere with hierarchical mode change operations and lead to broken subscription flows until the failure is resolved.

The hierarchical algorithm is divided into five distinct phases in order to preserve *transferred* subscriptions during mode change operations and to avoid any overload scenarios. This means transferred subscriptions are guaranteed to flow but the subscription interval may vary according to the modes the participating status routers operate in. The following failure scenarios assume the use of a non-blocking scheme in hierarchical mode change operations and overload scenarios refer to incoming status router queues. A non-responsive status router refers to a status router which is able to forward status events but unable to communicate with the management hierarchy.

A status router which does not participate in any of the hierarchical mode change phases becomes inconsistent when the mode change operation is complete, e.g., from Green to Yellow. During the mode change operation, however, the non-responsive status router is able to forward status events that belong to *transferred* subscriptions, but possibly at the wrong rate. After the commit phase, the status router operates in the wrong mode and will either not be able to forward any of the status events that belong to the new mode or will forward status events at the wrong rate (transferred subscriptions).

If the non-responsive status router has failed, it is not able to forward any status events and therefore breaks subscriptions flows that pass through it. The operating modes of a failed status router will be restored by the leaf QoS broker whenever it reconnects to the GridStat network. If a status router reconnects to GridStat and resumes its responsibilities during a hierarchical mode change operation, it will first be restored to the mode the mode change operation is switching from, and then updated with the mode change phases that have been executed. Status routers log incoming mode change operations (and phases) and ensure a correct ordering of execution.

Additional causes for a non-responsive status router may be network partitions, link failures or heavy link loss that affect communication between the management hierarchy and the data plane. A network partition in the data plane, in nature, leads to an inconsistent hierarchical mode change operation and forces the management hierarchy to restore the operating modes in the data plane whenever communication to the partition is re-established. Link failures are tolerable, to some extent, under the assumption that RPC connections between status routers and leaf QoS brokers utilize an appropriate number of redundant paths. A leaf QoS broker may be unable to contact one of its status routers if a set of failed links prevent any of the redundant paths to forward mode change operations to the status router. In such circumstances, the leaf QoS broker can initiate the recovery mechanism (Section 3.6) which will queue pending mode change operations (phases) and continually retry the call at specific intervals. Thus, the recovery mechanism is able to deliver queued mode change operations when the assumed failed status router restores contact with the management hierarchy.

An alternative scheme, not yet implemented in GridStat, would be for the leaf QoS broker to flood the mode change operation to its cloud by using the flooding mechanism instead of individually contacting each status router. The flooding mechanism can utilize the maximum amount of redundancy available in the cloud, and thereby increase the probability of delivering the mode change operation to the assumed failed status router. However, there is still no guarantee that the flooding mechanism is able to deliver the message to a non-responsive status router. The presence of heavy link loss is closely related to link failures, but here the recovery mechanism has a chance to deliver the mode change operation (phase) through the temporal reundancy property in the RPC mechanism over the lossy link itself, or through a redundant path, if any.

The hierarchical mode change algorithm eliminates overload issues during mode change operations when all status routers participate by selecting the highest subscription interval (lowest rate) in the prepare phase. However, there exists overload scenarios when one or more status routers do not participate in a hierarchical mode change operation. Figure 4.15 shows three status routers responsible for forwarding status events from publisher A and publisher B towards the subscriber. Edge status routers B1 and B2 operate in modes Green, Green $\cap$ Yellow and Yellow during a mode change operation switching from Green to Yellow. Assume that B5 is overloaded when both B1 and B2 forwards status events at a rate of 100 ms, which can only occur when B1 operates in

Green and B2 in Yellow. More specifically, if B1 does not participate in the mode change operation, but still forwards events, B5 becomes overloaded when the operation is complete (B2 operates in Yellow).

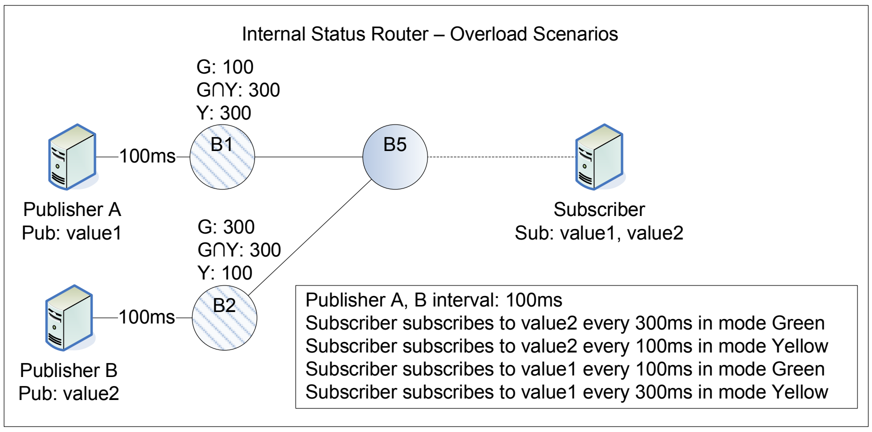

Internal Status Router – Overload Scenarios
G: 100
G∩Y: 300
Y: 300
100ms
B1
B5
Publisher A
Pub: value1
Subscriber
Sub: value1, value2
G: 300
G∩Y: 300
Y: 100
100ms
B2
Publisher B
Pub: value2
Publisher A, B interval: 100ms
Subscriber subscribes to value2 every 300ms in mode Green
Subscriber subscribes to value2 every 100ms in mode Yellow
Subscriber subscribes to value1 every 100ms in mode Green
Subscriber subscribes to value1 every 300ms in mode Yellow

Figure 4.15: Status router overload - upstream inconsistent router

This issue can be solved by one of the following modifications to GridStat:

- A resource management scheme collectively employed by the management hierarchy can prevent overload scenarios, as depicted in Figure 4.15, by closely monitoring how subscriptions affect hierarchical mode change operations. If a subscription request will cause a status router to become overloaded in a mode change operation, e.g., switching from A to B, the request is rejected.

- Leaf QoS brokers can block aggregation rounds until all status routers have acknowledged a mode change phase. QoS brokers employ a similar blocking scheme, but awaits acknowledgements from its direct QoS broker children. This scheme implies that a hierarchical mode change phase has to be delivered, executed and acknowledged by every status router and QoS broker participant, and effectively solves any overload issues in the data plane. The disadvantage is that hierarchical mode change operations block and increase the overall execution time in the presence of network or GridStat failures. See Section 3.6.1 for more details.

CHAPTER FIVE

DESIGN OF THE FLOODING MODE CHANGE ALGORITHM

The flooding mode change algorithm is an alternative to the hierarchical mode change algorithm and offers better statistical delivery guarantees to the data plane. The flooding mode change algorithm delivers mode change operations directly to the status routers through the limited flooding mechanism in GridStat. In order to utilize the limited flooding mechanism, the QoS brokers embed a publisher instance that connects to some edge status router in the QoS broker's hierarchical scope. The QoS broker can publish mode change operations through the publisher instance, where status routers forward the operation to all their status router neighbors, except the one they received the operation from. The flooding mechanism will eventually stop when all status routers have been informed. The limited flooding mechanism benefits from the amount of redundant paths in the data plane and is thus more resilient to network failures than the hierarchical mode change algorithm. Whereas the hierarchical mode change algorithm attempts to preserve the subscriptions registered in the two involved modes, the flooding mode change algorithm switches directly to the new mode. Figure 5.1 shows a flooding mode change initiated by QoS broker A which floods the mode change operation directly out on the data plane through its embedded publisher instance, and is able to deliver the operation to all participants within five message rounds. The diagram assumes an equal link delay, and the event channel labels refer to the message round in which the operation is flooded.

The flooding mode change algorithm assumes that all status routers in the hierarchical scope of the coordinator have synchronized clocks. Synchronized clocks across the data plane in GridStat can be achieved through GPS synchronization methods, and is discussed more in Section 8.2.9.

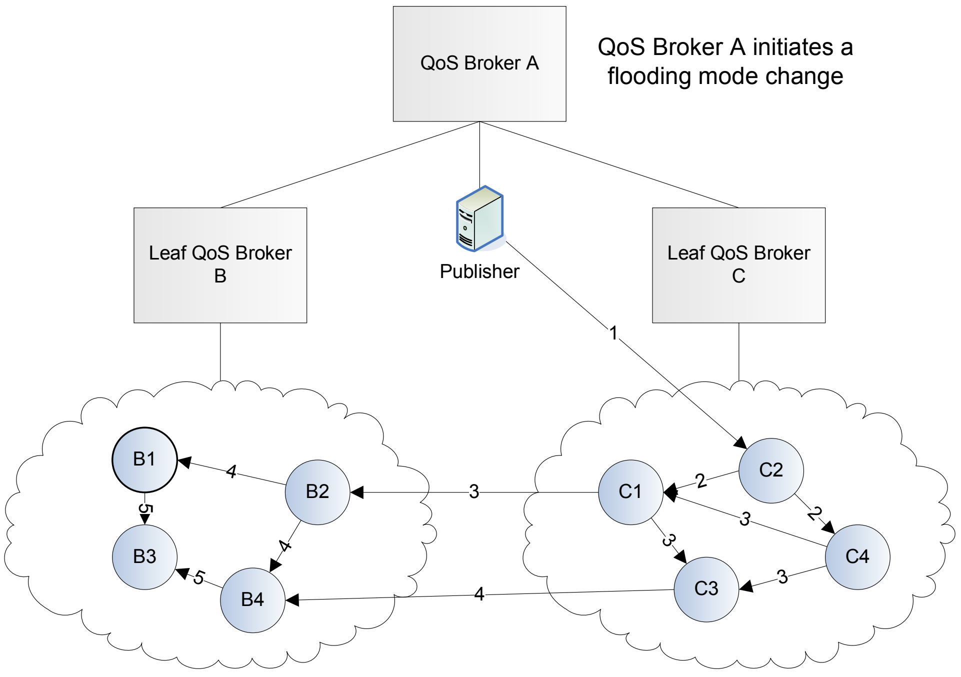


Figure 5.1: The flooding mechanism

5.1 Preparation and Use of a Flooding Variable

A QoS broker continually attempts to connect its publisher instance to an edge status router in the data plane during the startup phase. Once the publisher is connected, the QoS broker registers a publication dedicated for disseminating flooded mode change operations. The publication request is sent from the publisher to its edge status router, and then forwarded to the leaf QoS broker that holds the publisher in its hierarchical scope. The leaf QoS broker validates the publisher and the publication variable, adds the publication information to its state and replies with a variable ID. The QoS broker client registers the recent publication as a flooded publication by contacting all the status routers in its hierarchical scope. The status routers mark the routing entry for the particular publication as a flooding variable, and thereby forward any incoming status events belonging to the publication to all their status router neighbors.

A QoS broker prepares a flooded mode change operation by initializing a container for the information necessary for the status routers to execute the operation (see Section 3.2.1). The flooded mode change operation utilizes a timestamp variable to inform status routers at what time to execute the operation. The QoS broker determines the future execution timestamp based on its own clock, and therefore illustrates the necessity of synchronized clocks across all GridStat entities. The QoS broker serializes (Java) and delivers the mode change operation to its embedded publisher instance which publishes the mode change event to its edge status router. The edge status router checks the embedded variable identifier in all incoming status events and determine whether the status event is of a standard subscribed-to variable or a flooded variable. In the latter case, the status router looks up the variable identifier in its routing table and determines at what *level* the flooded status event is to be forwarded. The use of *level* in the flooding mechanism refers to the *flooding domain* which corresponds to the hierarchical scope of the QoS broker that initiated the flooding mode change operation. For example, the flooding domain of a specific QoS broker in the management hierarchy corresponds to all the status routers that are contained in the hierarchical scope of that QoS broker, and the *level* setting is set to the location of the QoS broker in the management hierarchy. Status routers will most likely receive redundant copies of a flooded variable from their neighbors, but will quickly discard them not to over-utilize any network resources. In addition, a status router will not forward a flooded status event to status routers it has previously received the same status event from, thereby preserving network resources. More information on the flooding mechanism can be found in [3].

5.2 Activation of Flooding Mode Change Operations

When a status router receives a flooded mode change operation it first registers the operation in its state and deserializes the event stream in order to retrieve the mode change structure. The status router checks if the mode change operation is registered (previously executed) or currently pending. If so, the mode change operation is discarded as it is currently pending to be activated at the desired timestamp or has already been activated. Prior to activating the timer, the status router informs all its subscribers about the upcoming mode change operation. More specifically, subscribers are informed to be aware of potential QoS violations that may occur until the mode change operation is committed. The next step of the flooding mode change algorithm is for the status router to check the timestamp variable in the mode change structure and create a timer that is scheduled to time out at that timestamp. If the timestamp has already passed the timer triggers immediately; otherwise, the timer times out when the mode change operation is scheduled to be activated. The status router ends with sending an acknowledgement up to its leaf QoS broker to confirm that it received the operation. During the timeout phase of a flooding mode change operation the status router continues to forward any incoming status events and process additional flooding mode change operations, if any. At the time a timer event is triggered the status router exclusively locks the routing table and performs the mode switch. That is, the status router changes from the previously activated mode to the new mode according to the information passed in the mode change structure that belongs to the particular mode change operation. The mode switch effectively means that the status router will forward subscription traffic for the new mode, while the *deactivated* mode prohibits any of its registered subscription traffic to be forwarded. It is important to mention that a subscription belonging to both the old and the new mode is continually forwarded, whereas subscription traffic that belongs solely to the old mode is dropped at the publisher's edge manager. When the mode switch is complete the status router unlocks the routing table and informs its subscribers that the particular mode change operation is completed.

5.3 Pseudo-Code for The Flooding Mode Change Algorithm

The following pseudo-code and explanations show how a mode change operation is delivered to a set of status routers in the data plane. Furthermore, the pseudo-code shows in detail how a status router uses and executes mode change operations.

5.3.1 Coordinator Code

The following code outline shows the process of flooding a mode change operation directly out on to the data plane. The coordinator defines a ModeInfo container with the required parameters and registers the mode change operation as pending. Then, it serializes the ModeInfo container and publishes the byte stream directly out on to the data plane. Finally, it marks the mode change operation as complete when it receives mode change acknowledgements from its direct QoS broker children.

```
Define a ModeInfo container with the required parameters
Register the mode change as pending
Serialize the ModeInfo container into a byte stream
Publish the byte stream
if All acknowledgements from direct QoS broker children are received then
    Mark the mode change as complete
end if
```

5.3.2 Status Router Processing Code

The following code outline shows the process of receiving a flooded mode change operation and storing it for later activation at the status router level. First, the status router forwards the flooded message on all outgoing links except the link it initially received the operation on. Then, it decodes the message and checks if the mode change operation has previously been executed or is currently pending. If so, it sends an acknowledgement to its leaf QoS broker. Otherwise, it registers the mode change operation, marks it as pending and creates a timer which will trigger when the mode change is to occur. Finally, the status router sends an acknowledgement to its leaf QoS broker.

```
Forward the flooded message on all outgoing links except the receiving link
Decode byte stream and retrieve the ModeInfo container
if The mode change operation has previously been executed or is pending then
    Send an acknowledgement to the management hierarchy
```

```
else
    Register the mode change operation
    Mark the mode change operation as pending
    Create a Timer that will trigger when the mode is to be switched
    Send an acknowledgement to the management hierarchy
end if
```

5.3.3 Status Router Activation Code

The following code outline shows the process of executing a mode change when the timer triggers (in a mode change from Green to Yellow). If the status router is an edge status router, it locks the routing table(s) and switches to the routing table of mode Yellow. Then, it unlocks the routing table(s) and informs all connected subscribers about the completed mode change operation. If the status router is an internal status router, it only switches to the routing table of mode Yellow. Finally, the status router marks the mode change operation as completed and stops the timer.

```
if Status router type is an edge status router then
    Lock the routing table(s)
    Switch to the routing table of mode Yellow
    Unlock the routing table(s)
    Retrieve the list of connected subscribers
    for Each connected subscriber do
        Inform subscriber about the completed mode change operation
    end for
end if
if Status router type is an internal status router then
    Lock the routing table(s)
    Switch to the routing table of mode Yellow
    Unlock the routing table(s)
**end if**
Mark the mode change operation as completed
Stop the Timer
```

5.3.4 Leaf QoS Broker Acknowledgement Aggregation Code

The following code outline shows the actions of a leaf QoS broker after all status routers have acknowledged the pending mode change operation. If the coordinator resides higher up in the management hierarchy, the leaf QoS broker sends an acknowledgement up to its parent QoS broker. Otherwise, it marks the mode change operation as complete.

```
Retrieve the list of status routers from state
for Each status router in state do
    Update the modes table to reflect the operating modes of this status router
end for
Update the list of what modes all status router should operate in
if The coordinator is higher up in the management hierarchy then
    Send acknowledgement to parent QoS broker
else
    Update operating modes
end if
Mark the mode change phase as complete
```

5.3.5 Interior QoS Broker Acknowledgement Aggregation Code

The following code outline shows the actions of a QoS broker after direct QoS broker children have acknowledged the pending mode change operation. If the coordinator resides higher up in the management hierarchy, the QoS broker sends an acknowledgement up to its parent QoS broker. Otherwise, it marks the mode change operation as complete.

```
if The coordinator is higher up in the management hierarchy then
    Send acknowledgement to parent QoS broker
else
    Update operating modes
end if
Mark the mode change phase as complete
```

5.4 Failure Scenarios

This section investigates the limitations of the flooding mode change algorithm and how to better tolerate or even overcome them in future versions of GridStat. In comparison with the hierarchical mode change algorithm, the flooding mode change algorithm utilizes the flooding mechanism in GridStat that efficiently disseminates a message to a group of status routers in the data plane. As status routers forward the flooded message on all outgoing links, except the one they received the operation from, the flooding mechanism is efficient in term of *message rounds* and is able to tolerate link failures and link loss to some extent. Overload scenarios for the flooding mode change algorithm is discussed in more detail in Section 5.4.1.

Tolerance of link failures and link loss in flooding mode change operations that involve several clouds highly depend on the inter-connectivity between clouds. The flooding mechanism benefits from the amount of redundancy within a cloud, and by flooding, a message can reach a specific status router through several disjoint paths. However, the case might not be that simple with flooding mode change operations that span more than one cloud. A network topology with few inter-cloud communication links might increase the risk of inconsistent flooding mode change operations as flooding mode change messages between clouds may not benefit from the same redundancy as within a single cloud. This scenario stresses the necessity of a well-connected network topology, both within and between clouds, as well as a high degree of redundancy in the data plane for a deployment of GridStat.

A flooding mode change operation that is disseminated across several clouds and is unable to reach one or more clouds leaves the operating modes in the data plane inconsistent unless additional recovery mechanisms are in place. The status routers in the clouds that originally received the mode change operation send acknowledgements to their leaf QoS broker and up towards the coordinator. The clouds which have not received the flooding mode change operation remain idle. The QoS brokers that manage the clouds that originally received the mode change operation aggregate acknowledgements and forward a response up the management hierarchy towards the coordinator. An aggregation round that is initiated at the QoS broker which contains one or more of the clouds that did not receive the mode change operation in its hierarchical scope will discover an inconsistent mode change operation when the aggregation round times out. The QoS broker proceeds to initiate the recovery mechanism (Section 3.6), which hopefully, will be able to contact the inconsistent status routers and QoS brokers before the status routers have been informed to switch. Figure 5.2 depicts an inconsistent cloud that is recovered by the recovery mechanism. Inconsistent clouds are cleanly handled by the recovery mechanism but might restore the modes at

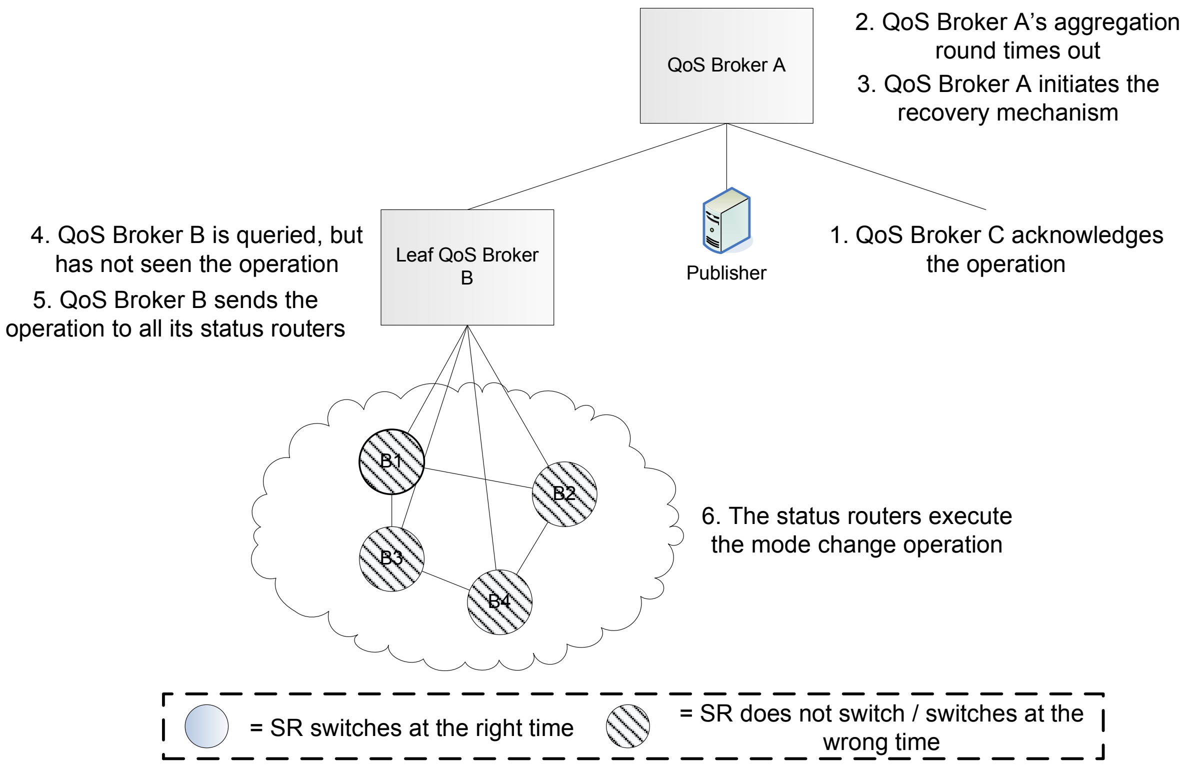

QoS Broker A
2. QoS Broker A's aggregation round times out
3. QoS Broker A initiates the recovery mechanism
4. QoS Broker B is queried, but has not seen the operation
5. QoS Broker B sends the operation to all its status routers
Leaf QoS Broker B
Publisher
1. QoS Broker C acknowledges the operation
B1
B2
B3
B4
6. The status routers execute the mode change operation
= SR switches at the right time
= SR does not switch / switches at the wrong time

Figure 5.2: The recovery mechanism in flooding mode change operations

the inconsistent status routers later than the destined mode switch time. In that case, subscriptions that go through the inconsistent status routers, and which belongs to the new mode, are cut-off until the recovery mechanism restores the status routers. Otherwise, the previously inconsistent status routers continue as planned to switch to the new mode at the destined timestamp. Figure

5.3 shows an inconsistent status router which breaks the subscription that passes through it, and the subscription remains broken until the status router executes the mode change operation or is restored by the recovery mechanism. A flooding mode change operation is associated with a future

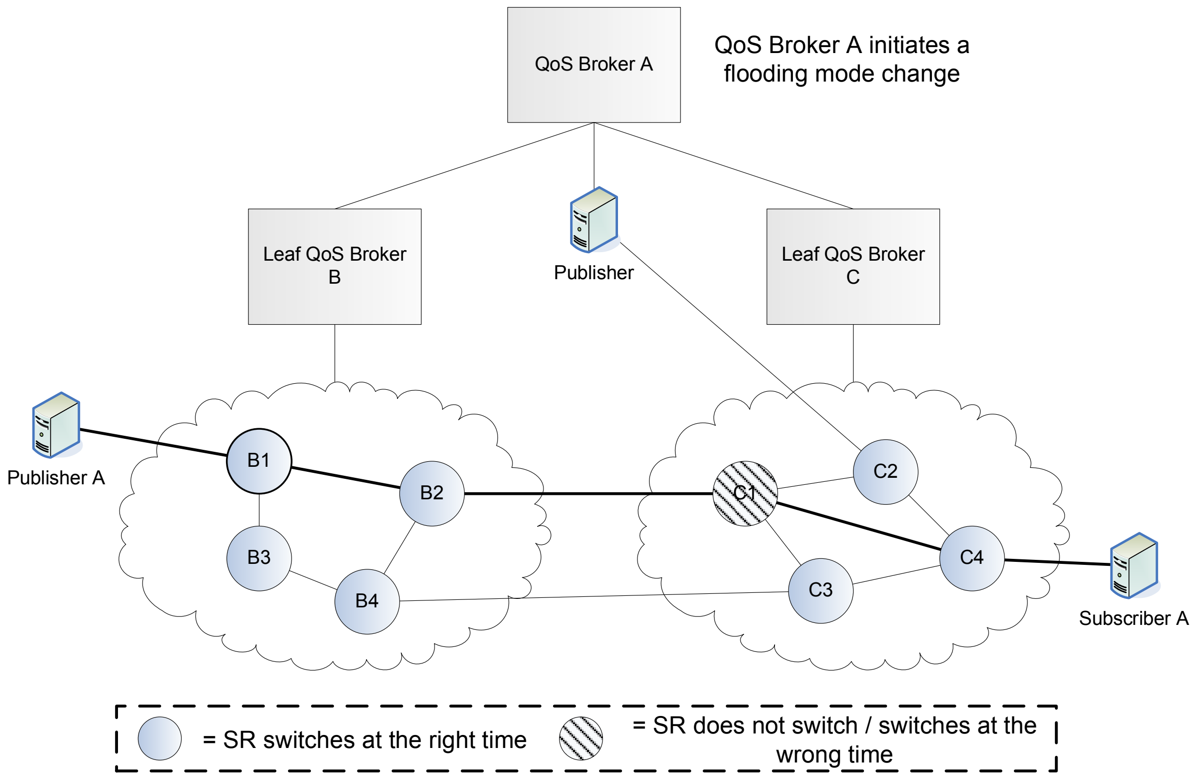

QoS Broker A
QoS Broker A initiates a flooding mode change
Leaf QoS Broker B
Publisher
Leaf QoS Broker C
Publisher A
B1
B2
B3
B4
C1
C2
C3
C4
Subscriber A
= SR switches at the right time
= SR does not switch / switches at the wrong time

Figure 5.3: Inconsistent status routers in flooding mode change operation

mode switch timestamp and, if chosen carefully, leaves room for the recovery mechanism to run. Since the flooding mode change algorithm utilizes the flooding mechanism and benefits from the amount of redundancy in the data plane, the recovery mechanism has a relatively low chance to restore any inconsistent status routers. On the other hand, the recovery mechanism facilitates services to continously query inconsistent status routers and restore their operating modes whenever they re-establish contact with the management hierarchy. The associated mode switch timestamp can be tuned to quickly perform flooding mode changes with a short idle period or, alternatively, incorporate a long idle period to make flooding mode changes more resilient to temporal network anomalies through the use of the recovery mechanism. Depending on the situation, the ability to tune the mode switch timestamp in runtime can be useful under various network conditions.

A shortcoming of the flooding mechanism is when a flooding mode change operation is lost at the first link to the data plane, e.g., between the coordinator's publisher and its edge status router. There are no mechanisms in place to acknowledge a successful flooding, and the publisher is therefore not able to convey that the flooding mechanism failed to the coordinating QoS broker. In such a circumstance, none of the status routers will receive the flooding mode change operation and QoS brokers will not aggregate any acknowledgements up towards the coordinator. In short, the flooding mode change operation is not executed and GridStat has no knowledge of it, except from the coordinator. This shortcoming can be overcome by one or more of the following solutions:

- The coordinator can time out and initiate the recovery mechanism if it has not received any acknowledgements from its direct QoS broker children (or status routers).
- The coordinator can flood the mode change operation out on the data plane through several edge status routers. This requires some modification to GridStat but can potentially make the flooding mechanism more efficient. Although this is not a complete solution it will make the flooding mechanism more resilient to link failures and link loss on the first link to the data plane.
- The coordinator can receive an acknowledgement from the first edge status router that receives the mode change message, and enables the publisher to employ a temporal redundancy scheme.

5.4.1 Status Router Overload Scenarios

The flooding mode change algorithm is a best-effort algorithm and offers a high statistical delivery guarantee to all status routers participating in a mode change operation. However, the flooding mode change algorithm does not prevent overload scenarios. There are two ways status routers can become overloaded. Incoming buffer queue overload scenarios can occur when one or more status routers, long-term or short-term, are left operating in an inconsistent mode (Figure 4.15 in Section 4.9). Outgoing buffer queues can become overloaded when a status router switches directly between two modes, e.g., Green to Yellow.

An example of overloading an outgoing buffer queue (status router) is illustrated in Figure 5.4. The diagram shows a status router's forwarding rate when operating in mode Green and Yellow and

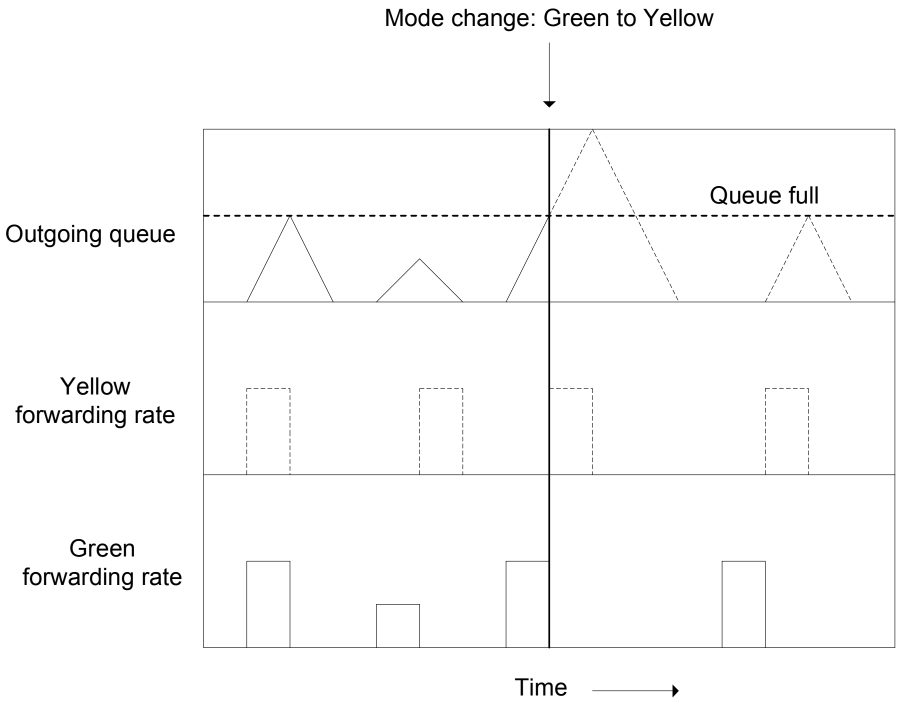


Figure 5.4: Status router - outgoing buffer queue overload.

the contents of its outgoing buffer queue (assume one outgoing link). The status router operates in mode Green and is told to switch to mode Yellow. At that time, the outgoing queue is currently full as the status router processes a burst of status events that belong to mode Green. With a full outgoing buffer queue, the status router switches to Yellow and begins to process a burst of status events which belongs to that mode. This causes the status router to drop many status events as the outgoing buffer queue is full. More specifically, many of the status events that belong to the new operating mode is lost due to an overloaded outgoing buffer queue.

The flooding mode change algorithm is affected by the above overload scenario as it switches directly between two modes. This issue, however, can be overcome by flushing every outgoing buffer queue after the status router has switched modes. In that case, the outgoing buffer queues are empty and status events belonging to the current mode are forwarded to the next downstream status router. Unless the status router flushes its outgoing buffer queues, the queued status events that belong to the previous active mode would have been dropped at the next downstream status router, and thus wasting network resources.

The flooding mode change algorithm is more liable to overload scenarios than the hierarchical mode change algorithm as it switches directly between modes at a predetermined timestamp. The hierarchical mode change algorithm, on the other hand, prevents overload of outgoing buffer queues by creating the temporary routing table (Section 4.6). However, the hierarchical mode change algorithm is just as liable to incoming buffer queue overload as the flooding mode change algorithm when using the non-blocking scheme.

CHAPTER SIX

EXPERIMENTAL RESULTS

This chapter evaluates the current global and hierarchical mode change mechanisms and management implementation in GridStat. The same set of experiments are conducted by both mode change algorithms, and the following list denotes points of interest:

- How well does the hierarchical mode change algorithm scale? How do the hierarchical scope and the width of the management hierarchy affect the results?
- How well does the flooding mode change algorithm scale? How do the hierarchical scope and the width of the flooding domain affect the results?
- How well can both mode change algorithms tolerate link loss? How do various link loss settings affect the results?
- How do various link latency settings affect the results?

6.1 Experiment Setup

The experiments were conducted on a 16-node cluster at the Electrical Engineering and Computer Science department at Washington State University. The hardware and software specifications are described in detail below:

Hardware and Software Specifications:

- 14 Intel Dual Xeon 3.06 GHz, 1 GB of RAM and 1 Gb network interface.
  - Redhat 9 (2.4.20-8smp kernel)
- 1 Intel Pentium III (Coppermine) 1 GHz, 512 MB of RAM and 100Mb network interface.

    - Ubuntu 6.10 (Edgy) Linux Distribution (2.6.17.10 kernel).

- Java Standard Edition 5.0 (build 1.5.0_11-b03).

The cluster nodes were used to run all the GridStat entities necessary to conduct the various experiments. The Ubuntu system ran a link emulator which was used to emulate link latency and link loss on a per-link basis (data plane links) in GridStat.

Java Virtual Machine Arguments:

- -jar: The Java applications are wrapped inside jar files.

- -Xms128m -Xmx128m: Statically set the heap size of the java applications to prevent unnecessary garbage collector runs.

GridStat Settings

Figure 6.1 shows the GridStat experimental setup with 7 QoS brokers and 20 status routers. A cloud consists of five status routers: three edge status routers and two internal status routers. Additional settings are listed below:

- QoS brokers communicate with other QoS brokers, and leaf QoS brokers with status routers through dedicated RPC connections that have been established prior to activating any mode change operations.

- RPC connections between leaf QoS brokers and status routers are configured to utilize two redundant paths.

- Inter-QoS broker RPC connections are configured to utilize one path only.

- Whenever a GridStat entity does not receive an RPC acknowledgement after *some* timeout, it retries the RPC call. The RPC retry timeout value is subject to an experiment setup (Table 6.1).

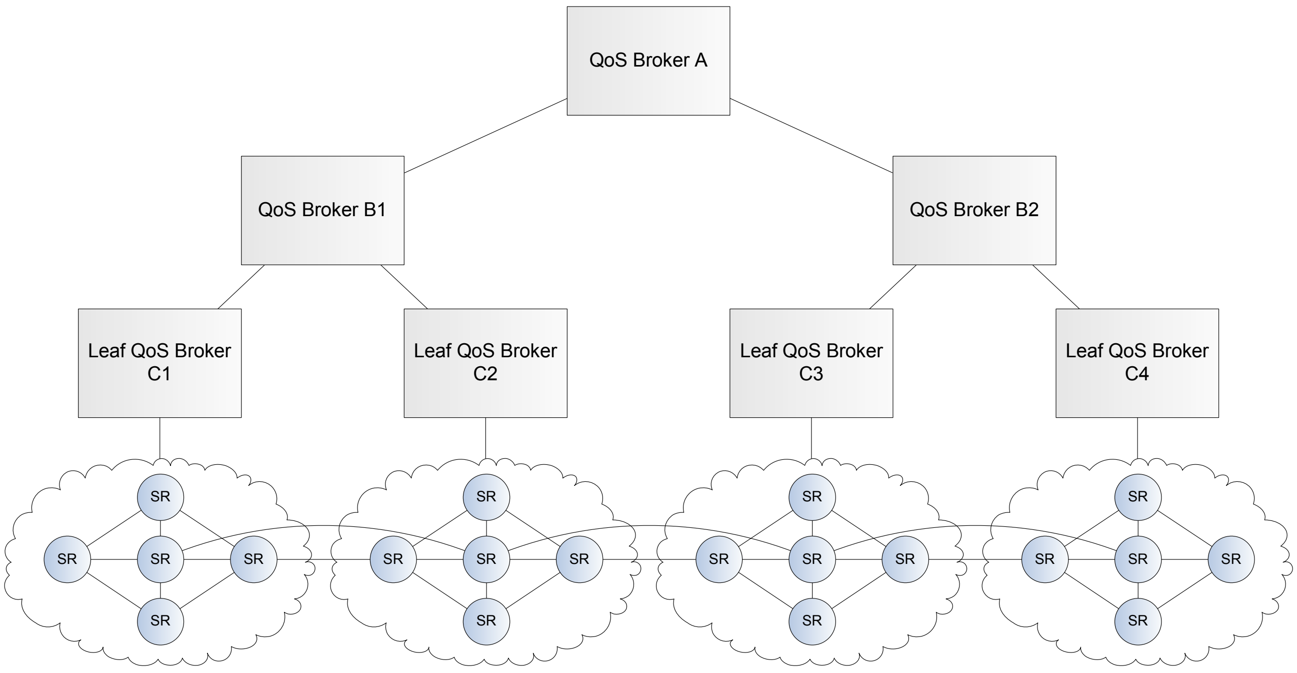


Figure 6.1: GridStat experimental setup

- All status routers are launched as edge status routers in order to utilize RPC connections with their leaf QoS broker. A consequence is that the edge status routers *configured* to act as internal status routers are included in the inform- and commit phase of the hierarchical mode change algorithm.

Link Emulator Settings

- A link is associated with a latency, a probability of packet loss and a burstiness setting. When a packet loss triggers, the link will consecutively lose as many packets as the burstiness setting suggests. If the burstiness setting is variable-sized, e.g. 3-5, the link will at a minimum lose 3 consecutive packets, but no more than 5 (the actual number is subject to a uniform distribution).

- The probability of triggering a packet loss is adjusted to the desired packet loss probability setting divided by the mean burstiness setting. More specifically, with a packet loss probability setting of 8% and a burstiness setting of 3-5, the trigger probability is $8\%/4 = 2\%$.

- A link will not trigger a new packet loss when in the middle of an ongoing loss sequence.
- Edge links, publisher to edge status router or subscriber to status router, do not lose packets.

Experiments were conducted over a wide range of link parameters:

- The same set of experiments were conducted with 0 ms, 1 ms, 2 ms, 4 ms and 8 ms link latencies. These link latencies were chosen to reflect realistic values for a GridStat deployment in a critical infrastructure.
- The same set of experiments were conducted on a GridStat deployment with 0%, 1%, 2%, 4% and 8% link loss. These link loss settings were chosen to show that the implementation works under a range of network conditions.

6.2 RPC Retry Timeout Overview

In order to tolerate link loss the RPC retry timeout is a crucial setting to quickly adapt to packet loss over RPC connections. When a distributed call is sent over an RPC connection the client awaits an RPC acknowledgement. If an RPC acknowledgement has not been received after the RPC connection times out, the client will reattempt to send the RPC call to the server. If the RPC acknowledgement itself got lost, the server will discard redundant messages. The selected RPC retry timeouts outlined in Table 6.1 are based on the highest observed RPC round-trip times in the conducted experiments. More specifically, several experiments, one per link latency and for each level in the management hierarchy, were conducted in order to find the highest RPC round-trip time for the particular experiment setup. The listed RPC retry timeout values will be used in mode change operations initiated at one of the three possible levels in the management hierarchy at the various link latency experiments.

Link Latency	Top Level	Second Level	Leaf Level
0 ms	10	10	10
1 ms	15	15	15
2 ms	30	25	20
4 ms	60	40	30
8 ms	120	70	55

Table 6.1: Experiment RPC retry timeouts.

6.3 Hierarchical Mode Change Experiments

This section includes the experimental results from hierarchical mode change operations conducted at all the three levels in the management hierarchy. Mode change operations are initiated at either QoS broker A, B1 or C1 in Figure 6.1. Table 6.2 lists all the various network conditions, link latency, link loss and burstiness setting and the respective mode change times for hierarchical mode change operations initiated at the top level (see Appendix A for results from second and leaf level). The times presented in the experimental results throughout this section highly depend on

Experiment	Loss %	Min. burst	Max. burst	0 ms link lat.	1 ms link lat.	2 ms link lat.	4 ms link lat.	8 ms link lat.
1	0%	0	0	162.66	279.16	402.69	671.01	1237.31
2	1%	1	1	168.48	309.41	447.49	762.43	1361.01
3	1%	1	2	171.64	316.34	454.93	772.63	1400.48
4	1%	1	4	180.18	327.95	474.30	815.79	1486.06
5	1%	3	5	182.51	344.88	468.94	791.05	1520.83
6	2%	1	1	178.59	332.95	483.59	826.37	1508.32
7	2%	1	2	188.50	355.85	508.06	861.13	1616.64
8	2%	1	4	205.35	381.12	556.37	927.13	1759.23
9	2%	3	5	221.86	432.62	537.45	948.64	1917.66
10	4%	1	1	213.71	417.41	576.95	1008.27	1916.58
11	4%	1	2	240.12	439.00	607.74	1059.65	1992.49
12	4%	1	4	252.57	454.35	652.78	1160.58	2154.97
13	4%	3	5	271.38	494.65	715.08	1179.67	2297.58
14	8%	1	1	361.18	577.15	841.95	1448.47	2659.82
15	8%	1	2	333.97	585.02	857.15	1551.41	2897.88
16	8%	1	4	382.76	673.30	920.42	1627.72	3018.97
17	8%	3	5	439.17	755.58	1035.97	1737.45	3245.15

Table 6.2: Hierarchical mode change experiments initiated by the top-level QoS broker.

several factors:

- The RPC retry timeouts affect the overall mode change times in the presence of link loss. That is, higher link latency settings increase the RPC retry timeout values being used.

- One-path RPC connections employed by interior QoS brokers are liable to link loss (especially bursty loss), whereas the established two-path RPC connections used for communication between the leaf-level QoS brokers and the data plane tolerate link loss to some extent.

- As RPC connections are established on top of subscriptions, mode change times depend on the length of subscription paths. That is, the number of event channels a mode change message has to traverse in order to reach its destination. The *longest* RPC connection (subscription path) are: 5 event channels from top level QoS broker to middle level QoS broker, 4 event channels from middle level QoS broker to leaf level QoS broker and 3 event channels from leaf level QoS broker to any status router. For example, the estimated round-trip time to deliver a mode change operation to any status router from the top level QoS broker with an 8 ms link latency setting is: $((8ms*5)+(8ms*4)+(8ms*3))*2 = 192ms$. QoS broker (per level) and status router processing times and link emulator overhead come in addition.

6.3.1 *Top-level Experiment*

Figure 6.2 depicts the average time per experiment (100 operations) for mode change operations activated from the top level in the management hierarchy. As expected, the mode change times increase when the overall probability of packet loss (per link) increases. When increasing the link latency, the increase in mode change completion time is more notable, which correlates to higher RPC connection traversal latencies and RPC retry timeout values.

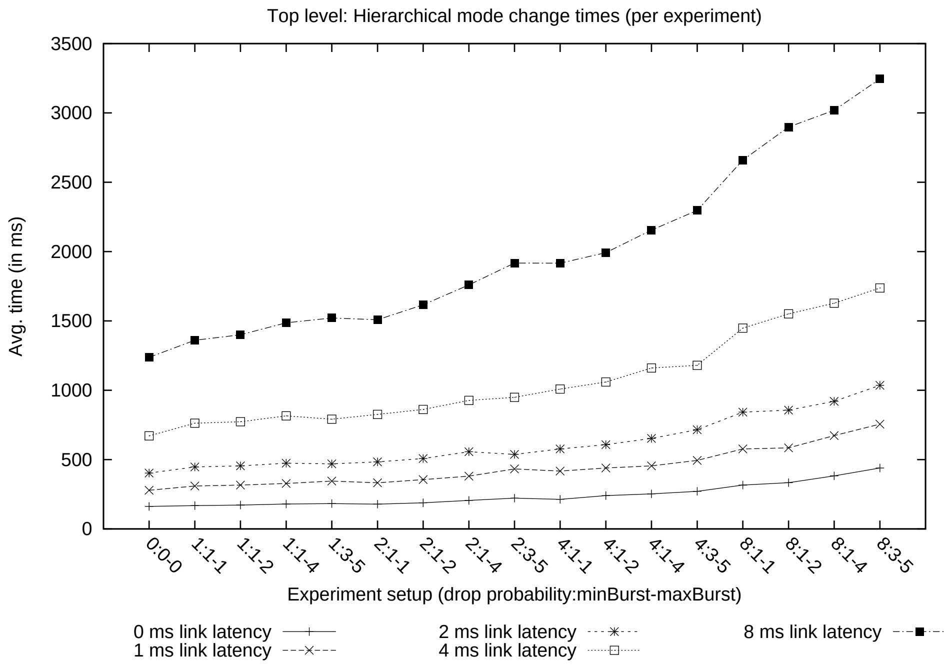


Figure 6.2: Hierarchical mode change operations at the top level in the management hierarchy.

6.3.2 *Second-level Experiment*

Figure 6.3 depicts the average time per experiment (100 operations) for mode change operations activated from the second level in the management hierarchy. The experiments conducted at the second level in the management hierarchy share the same trends as the experiments conducted at the top level. The experiments conducted with the same link loss and latency settings, but with variable burstiness settings, are not subject to the same increase in overall mode change times as is the case in Figure 6.2. The reason for this behavior is that the number of utilized one-path RPC connections have decreased, which makes mode change operations conducted at the second level in the management hierarchy less vulnerable to link loss. The two one-path RPC connections are between the coordinator of the mode change operations and its two leaf QoS broker children, while communication between the leaf QoS broker and status routers go over two redundant paths.

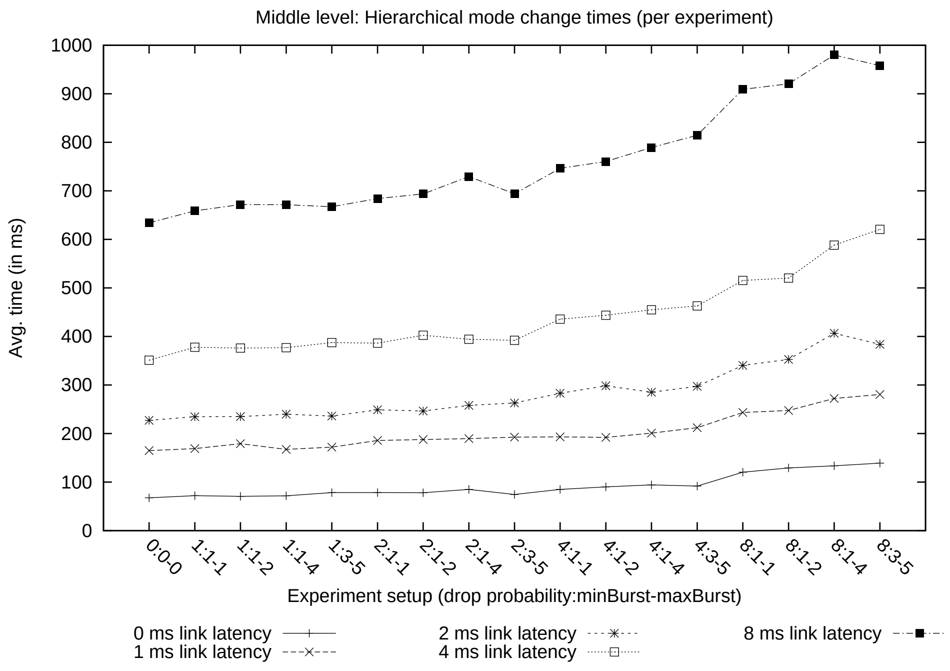


Figure 6.3: Hierarchical mode change operations at the second level in the management hierarchy.

6.3.3 Leaf-level Experiment

Figure 6.4 depicts the average time per experiment (100 operations) for mode change operations activated from the leaf level in the management hierarchy. The experiments conducted at the various link latency settings show a relatively flat trend, which suggests that the two redundant paths utilized by the RPC connections between the leaf QoS broker and its status routers are able to withstand link loss up to 4% without any significant implication on the mode change times. An 8% link loss setting increases the average mode change times which clearly illustrates the impact of having only two redundant paths between the leaf QoS broker and its status routers.

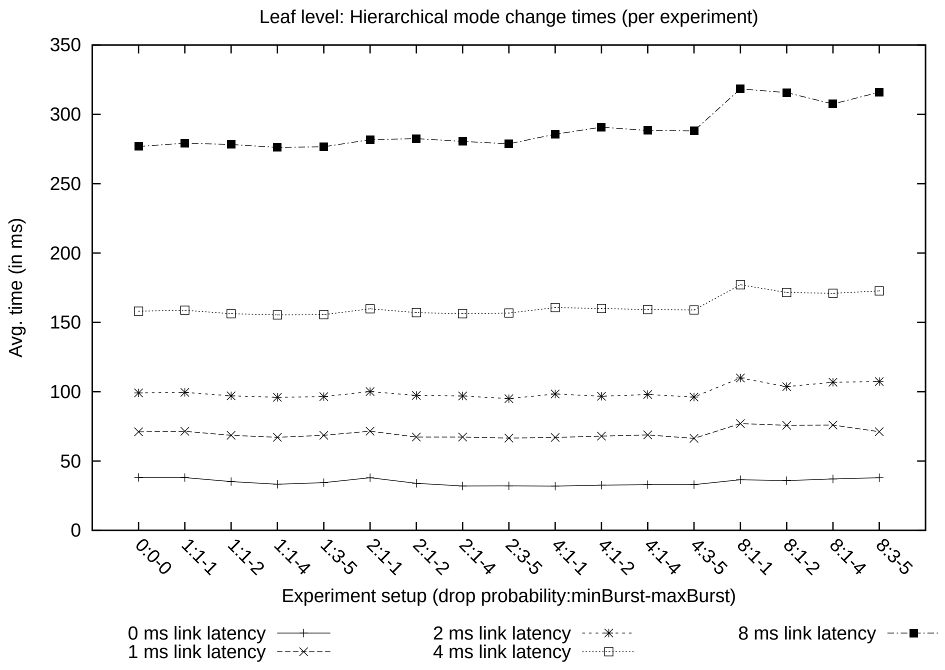


Figure 6.4: Hierarchical mode change operations at the leaf level in the management hierarchy.

6.4 Flooding Mode Change Operations

This section includes the experimental results from flooded mode change operations conducted at all the three levels in the management hierarchy. Figure 6.5 illustrates the experimental setup and shows which status routers the QoS brokers are connected to. Furthermore, mode change operations are initiated at either QoS broker A, B1 or C1 in Figure 6.5, and refer to top level, middle level and leaf level, respectively. All results from flooding mode change operations initiated at the top level are listed in Table 6.3 (see Appendix A for results from second level and leaf level). A total of 300 mode change operations are executed in sequential order per experiment setup. Once all the status routers have switched to the new mode and the coordinator of the operation has received acknowledgements from its descendants, the next operation follows.

Common for all flooded mode change operations is that the coordinator *floods* the mode change message directly out on the data plane without propagating the message down through the hierarchy towards the status routers. The status routers forward the flooded mode change message to all

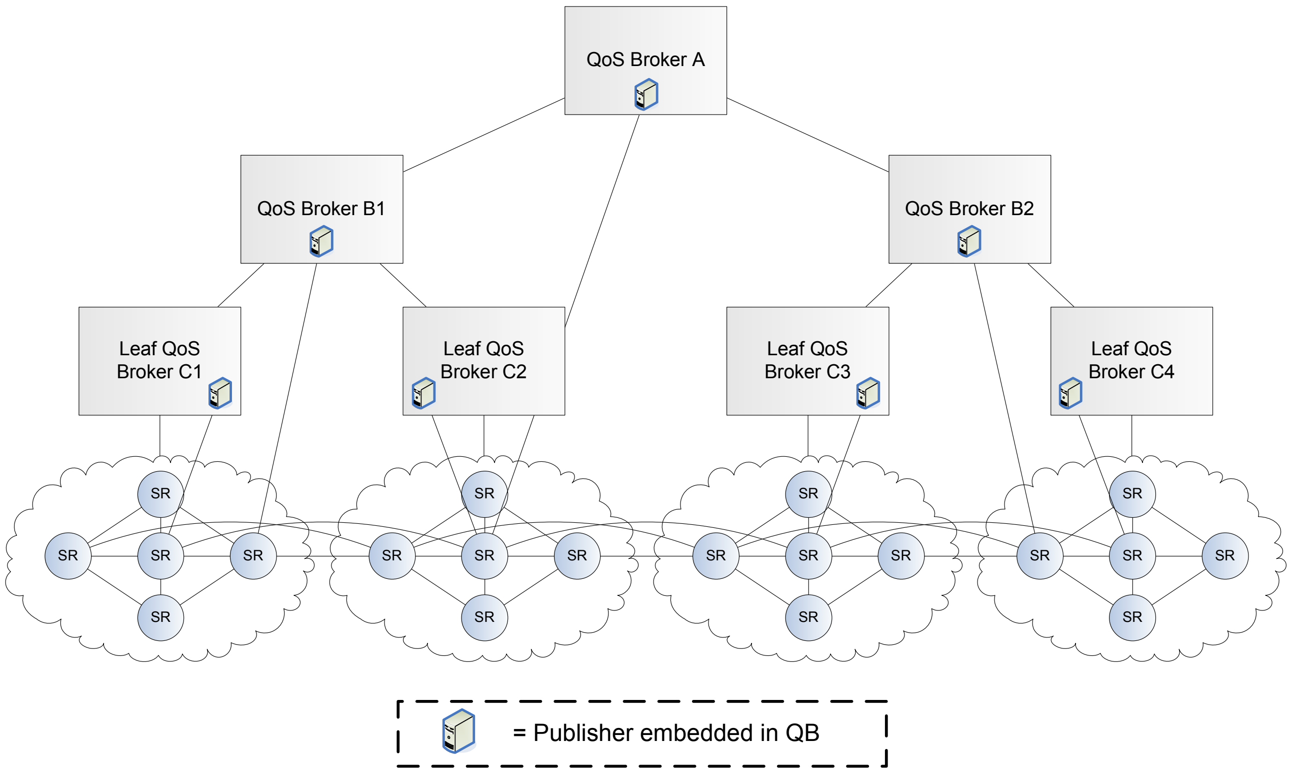


Figure 6.5: GridStat flooding connection points

Experiment	Loss %	Min. burst	Max. burst	0 ms link lat.	1 ms link lat.	2 ms link lat.	4 ms link lat.	8 ms link lat.
1	0%	0	0	11.35	18.57	21.50	28.00	44.27
2	1%	1	1	13.84	20.01	22.72	29.32	45.69
3	1%	1	2	13.72	19.73	22.19	29.34	45.87
4	1%	1	4	13.19	19.40	21.93	29.22	45.19
5	1%	3	5	13.07	19.21	22.03	28.46	45.17
6	2%	1	1	12.97	19.52	21.74	28.70	46.21
7	2%	1	2	13.52	19.29	21.73	28.08	45.16
8	2%	1	4	14.06	19.23	21.88	28.56	44.56
9	2%	3	5	13.04	19.30	22.28	28.58	45.06
10	4%	1	1	13.15	18.59	22.68	29.61	47.39
11	4%	1	2	13.20	18.79	21.31	29.00	46.09
12	4%	1	4	13.39	18.41	21.42	29.08	46.60
13	4%	3	5	13.71	19.04	21.43	29.17	46.08
14	8%	1	1	13.13	18.91	21.81	31.06	50.08
15	8%	1	2	13.27	19.11	22.38	30.58	48.98
16	8%	1	4	12.89	18.50	22.17	29.85	48.58
17	8%	3	5	13.62	18.55	21.20	29.94	47.80

Table 6.3: Flooding mode change experiments initiated by the top-level QoS broker.

their immediate neighbors except from the neighbor they received the mode change message from. Thus, the flooded mode change algorithm benefits from the amount of redundancy present in the data plane in order to reach all the target status routers for the respective mode change operation. The times presented in the following diagrams are the times when the *last* status router received the message, averaged over 300 mode change operations.

The number of message rounds required by the flooding mechanism to deliver the mode change operations to all status router participants, per level in the management hierarchy, are:

- QoS broker A needs four message rounds in order to deliver its mode change operations to all status router participants, according to the best flooding path.
- QoS broker B1 needs four message rounds.
- QoS broker C1 needs two message rounds.

6.4.1 Top-level Experiment

Figure 6.6 shows the average time per experiment (300 operations) for flooded mode change operations activated from the top level in the management hierarchy. As the mode change messages are

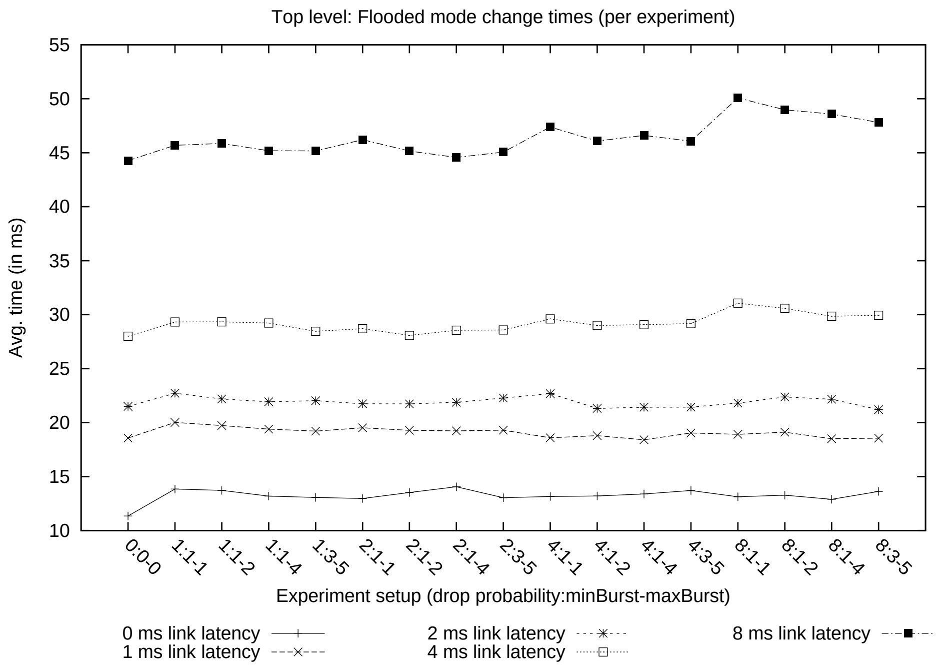


Figure 6.6: Flooded mode change operations at the top level in the management hierarchy.

disseminated directly on to the data plane, the five graphs, one per latency setting, illustrate how resilient the flooded mode change algorithm is against lossy links with various burstiness settings. With a link latency setting set to 8 ms, the flooded mode change reaches all the target status routers after approximately 45 ms, whereas the hierarchical algorithm requires 1200-3200 ms (Figure 6.2) depending on the link loss and burstiness setting. The experiments conducted with 8% link loss passes a threshold where the redundancy available in the data plane is not able to propagate the mode change message to the farthest away status router according to the best path, or close to the best path, in which the average mode change time increases.

6.4.2 Second-level Experiment

Figure 6.7 shows the average time per experiment (300 operations) for flooded mode change operations activated from the second level in the management hierarchy. A flooded mode change

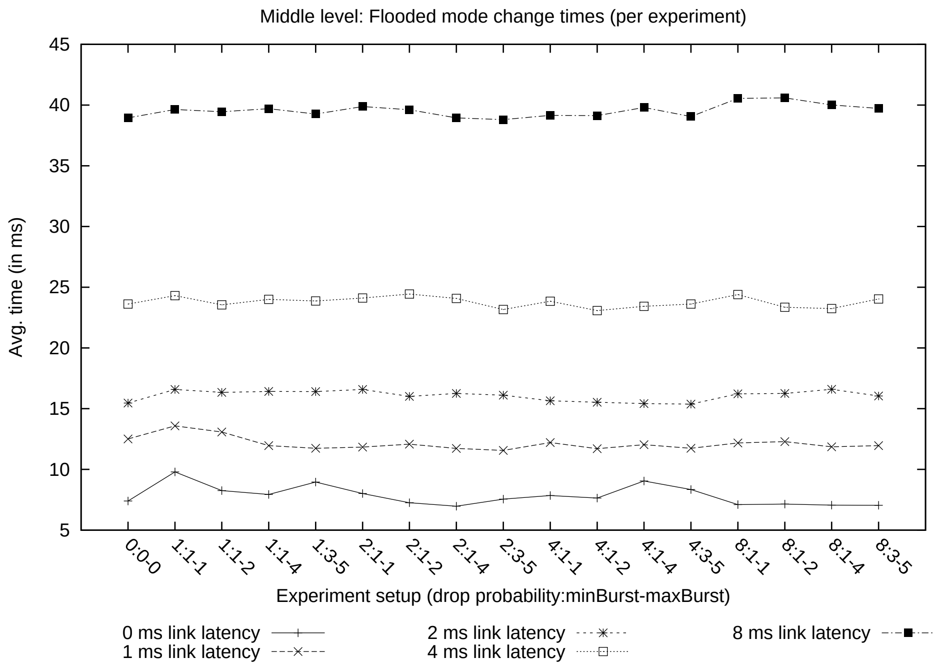


Figure 6.7: Flooded mode change operations at the second level in the management hierarchy.

disseminated from the second level in the management hierarchy involves 10 status routers, evenly distributed among two clouds. The trend is similar to the results in Figure 6.6 across all the conducted experiments, but shows slightly lower average mode change times. The lower mode change times are subject to fewer status routers involved in the experiments (less link emulator overhead) and an additional best-case flooding path. Interestingly, the results from this experiment show a better resilience against 8% link loss in comparison with flooding mode change operations initiated by the top level QoS broker (Figure 6.6). Figure 6.5 shows that the best-case flooding path from QoS broker A and QoS broker B1 require four message rounds in order to reach all status router participants. However, QoS broker B1 has an additional best-case flooding path that only requires four message rounds, whereas QoS broker A requires five message rounds when link loss prevents the best-case path from delivering the mode change message to all status router participants.

6.4.3 Leaf-level Experiment

Figure 6.8 shows the average time per experiment (300 operations) for flooded mode change operations activated from the leaf level in the management hierarchy. A flooded mode change dissem-

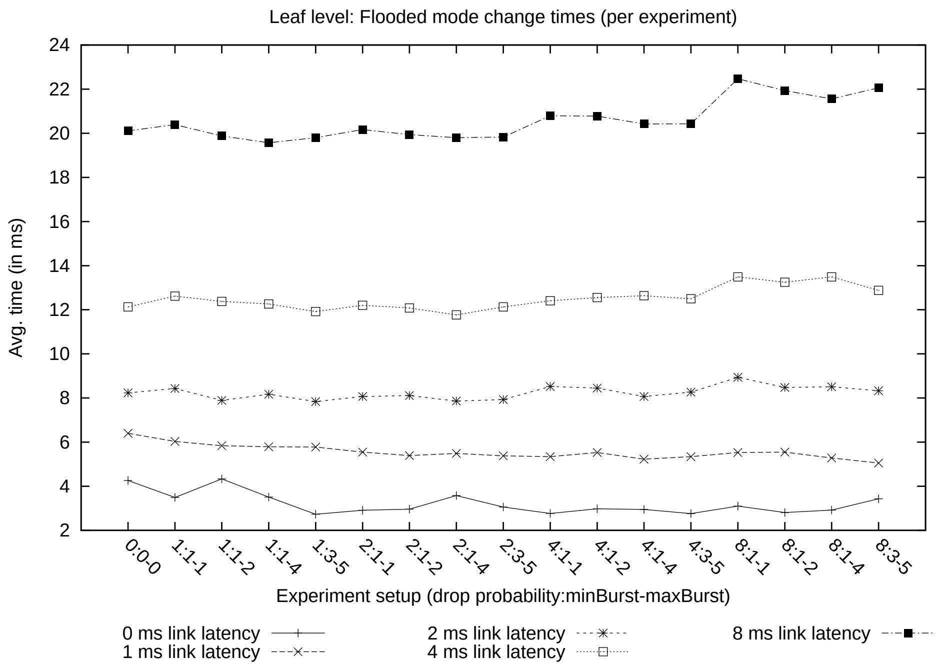


Figure 6.8: Flooded mode change operations at the leaf level in the management hierarchy.

inated from the leaf level in the management hierarchy involves 5 status routers in a single cloud controlled by the leaf QoS broker (coordinator). The experiments conducted with 8% link loss share the same trend as the experiments with the same network settings conducted at the second level in the management hierarchy, but are more efficient as the leaf QoS broker only requires two message rounds, through the flooding mechanism, to reach all status router participants. Leaf QoS broker C1 suffers from a single best-case flooding path, and requires three message rounds when link loss prevents the best-case path from delivering the mode change message to all status router participants. This explains the impact on the average mode change times under an 8% link loss setting in comparison with the experiments conducted by QoS broker B1 in Figure 6.7 which has two best-case flooding paths.

6.5 Hierarchical vs. Flooding Algorithm Comparison

The following diagrams illustrate how the hierarchical and flooding mode change algorithms scale when activating mode change operations at the three different levels in the management hierarchy. Common for all diagrams is a representation of all the experiments conducted at the link latency setting set to 0 ms, 1 ms and 8 ms for both the hierarchical and the flooding mode change algorithm.

6.5.1 Top-level Comparison

Figure 6.9 shows the average time per experiment, 100 operations for the hierarchical algorithm and 300 operations for the flooding algorithm, for mode change operations activated from the top level in the management hierarchy. The diagram clearly illustrates the impact of disseminating the mode change operations and collecting mode change acknowledgements through the management hierarchy for the hierarchical mode change algorithm. As inter-QoS broker RPC connections only utilize a single path, the effect of increased link loss settings directly corresponds to a higher overall mode change completion time caused by higher RPC retry delays. The flooding algorithm does not suffer from the same propagation delays and RPC retry delays as the hierarchical mode change algorithm since the operations are disseminated directly out on the data plane by using the flooding mechanism in GridStat. The flooding algorithm solely depends on the *width* of the *flooding domain*, the origin point of the flooding mechanism and the general level of redundancy

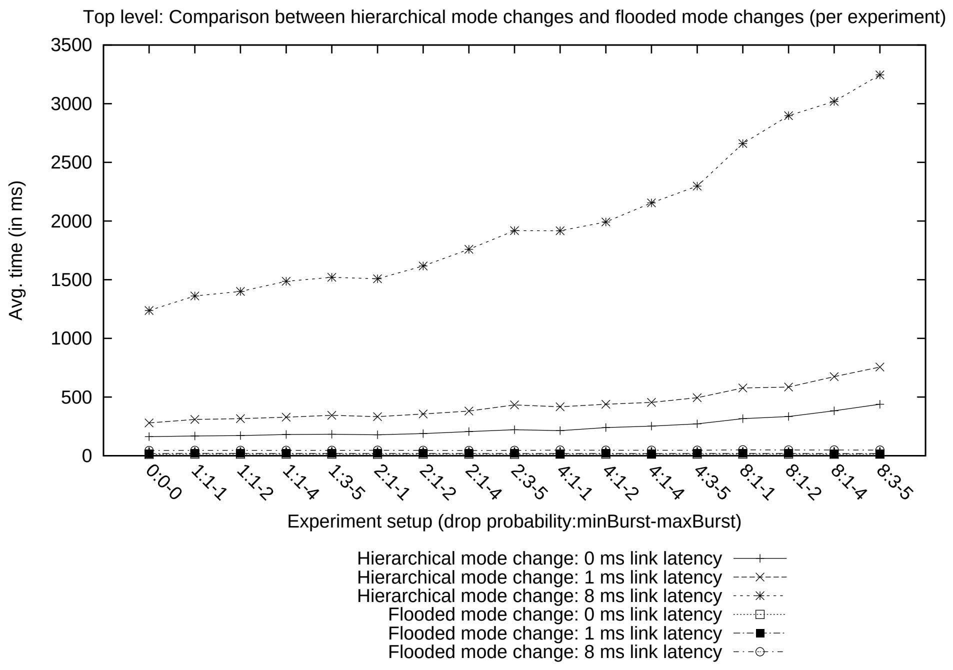


Figure 6.9: Hierarchical vs. flooding mode change operations conducted at the top level (0, 1 and 8 ms link latencies).

available in the data plane. The results from the flooding algorithm conducted with 8 ms link latency shows that the average mode change times are well below the hierarchical experiment 0:0-0 with a link latency setting set to 0 ms.

6.5.2 *Second-level Comparison*

Figure 6.10 shows the average time per experiment for mode change operations activated from the second level in the management hierarchy. The flooding algorithm still outperforms the hierarchical algorithm under all the different network conditions (note that the flooding experiments conducted with 8 ms link latency are significantly lower than the hierarchical experiments conducted with 0 ms link latency). The hierarchical results show a more flat outline than the experiments conducted at the top level (Figure 6.9) as the number of one-path RPC connections have decreased.

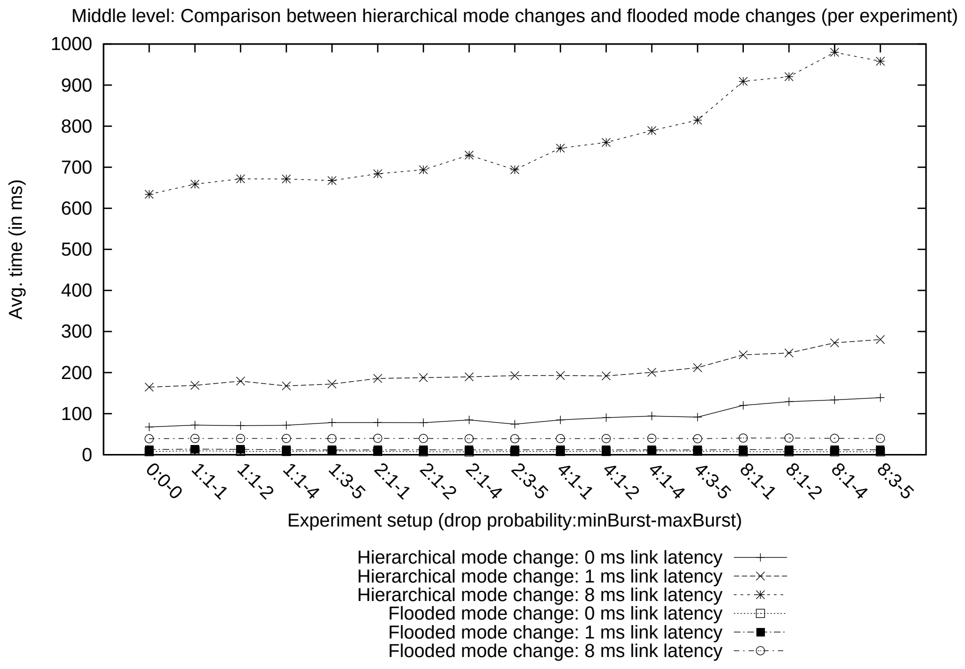


Figure 6.10: Hierarchical vs. flooding mode change operations conducted at the second level (0, 1 and 8 ms link latencies).

6.5.3 Leaf-level Comparison

Figure 6.11 shows the average time per experiment for mode change operations activated from the leaf level in the management hierarchy. Since the RPC connections utilize two redundant paths in the hierarchical algorithm, the significance of higher system loss is negligible, while at 8% link loss an increase in hierarchical mode change times is caused by the limitation of having only two redundant paths between the leaf QoS broker and a status router. The flooding algorithm still outperforms the hierarchical algorithm by almost an order of magnitude.

6.6 Link Traversals

Table 6.4 shows the number of links (event channels) traversed by both mode change algorithms. The number of traversals is averaged over all the experiments conducted at a specific level in the management hierarchy. For example, the hierarchical algorithm at the top level traverses a total of 2124 event channels, averaged over all experiments conducted at that level. The hierarchical mode

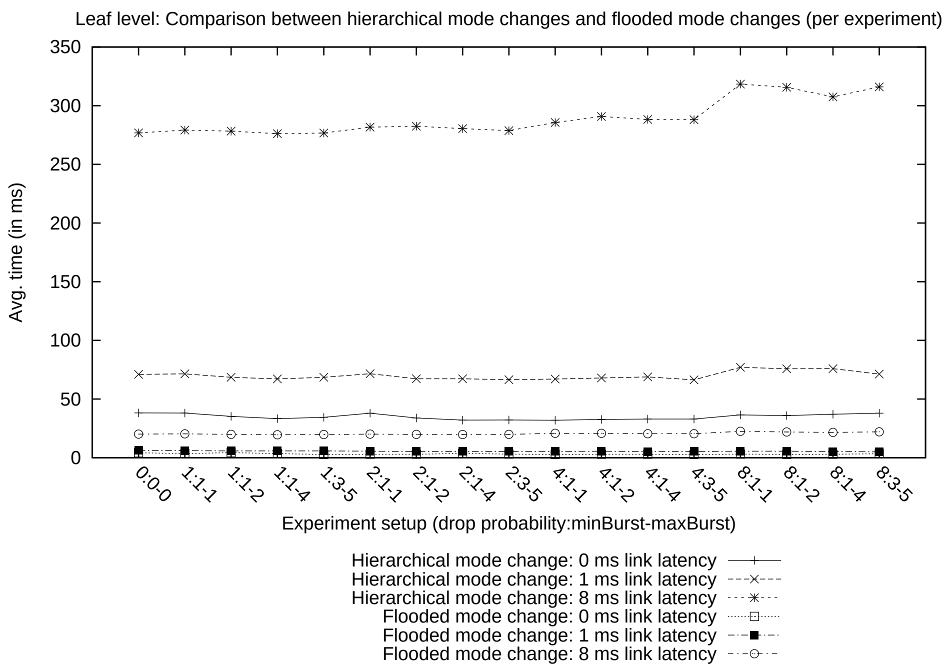


Figure 6.11: Hierarchical vs. flooding mode change operations conducted at the leaf level (0, 1 and 8 ms link latencies).

change algorithm requires more link traversals since the coordinator disseminates mode change operations through the management hierarchy and towards the data plane. The flooding mode change algorithm, on the other hand, *saves* a trip through the management hierarchy and uses it only for aggregating acknowledgements up towards the coordinator. In addition, the leaf QoS brokers utilize the spatial redundancy property in GridStat through its established RPC connections with the status routers it controls, which increases the number of link traversals for both algorithms.

Algorithm	**Top Level**	**Second Level**	**Leaf Level**
Hierarchical	2124	833	340
Flooding	367	165	84

Table 6.4: Link traversals

6.7 Scalability Results

The following diagrams compare the same experiments conducted at the three levels in the management hierarchy in order to see how the algorithms scale when increasing the hierarchical scope of the hierarchical algorithm or the flooding domain of the flooding algorithm.

6.7.1 The Hierarchical Algorithm

Figure 6.12 depicts experiments conducted at the three levels in the management hierarchy with 1 ms link latency, and Figure 6.13 shows the experiments conducted with 8 ms link latency. Figure 6.12 depicts an increase in time between the experiments conducted at the leaf and second level which is approximately the double of the mode change times achieved at the leaf level, and the same results are seen between the experiments conducted at the second and top level at lower link loss settings. Figure 6.12 also shows the impact of the one-path inter-QoS broker RPC connections (top level experiments utilize more one-path RPC connections than the experiments at the second level).

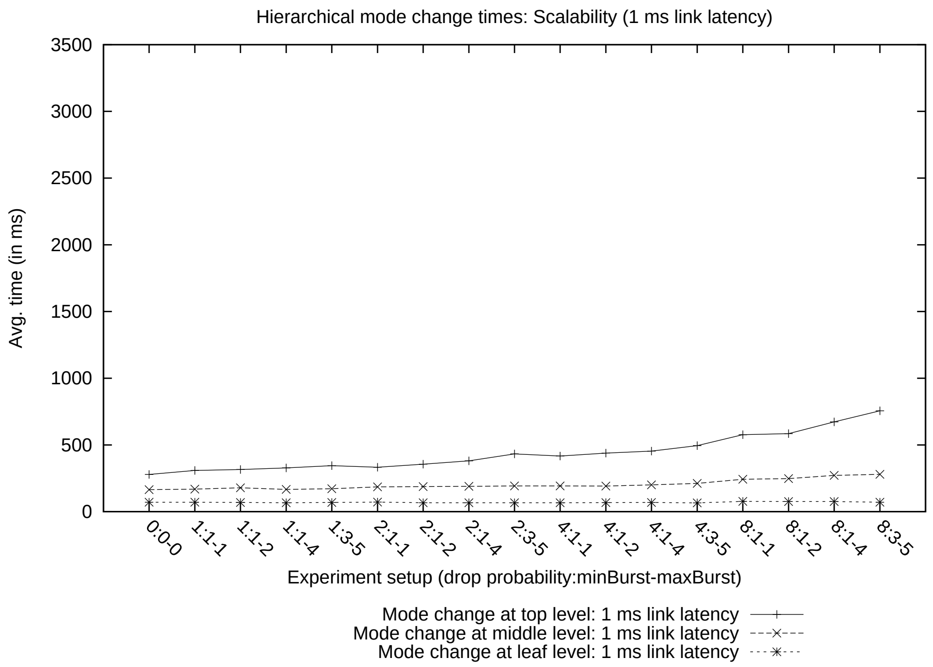


Figure 6.12: Scalability results using the hierarchical algorithm (1 ms link latency).

Figure 6.13 share the same trend as Figure 6.12, but the impact of the number of one-path inter-QoS broker RPC connections is more clear.

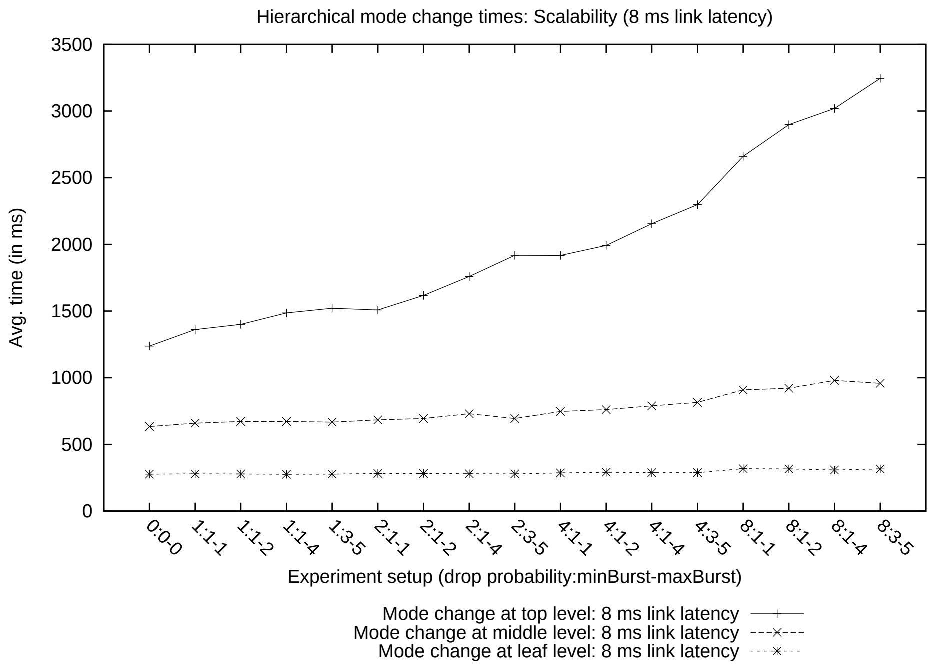


Figure 6.13: Scalability results using the hierarchical algorithm (8 ms link latency).

At first, one might expect that the linear increase is caused by the number of status routers involved in the hierarchical mode change operations: 5 at the leaf level, 10 at the middle level and 20 at the top level. This is not necessarily true. The hierarchical mode change algorithm depends on the number of levels in the management hierarchy, the *length* of the RPC connections between the QoS brokers and the corresponding delays. The experimental layout for the hierarchical mode change algorithm in Figure 6.1 shows a binary management tree, where the length of the RPC connections are: 5 links between A and B1, 4 links between B1 and C1 and 3 links between C1 and each status router, and suggests that the paths become longer higher up in the management hierarchy. However, the length of RPC connections depend on the topology in the data plane and at which status router the QoS broker's publisher and subscriber are connected to. With a trinary tree where each QoS broker has three direct children QoS brokers, the situation might be different depending on how the status router network is divided between the leaf QoS brokers. It is important to mention that the status router network remains the same; the number of status routers and the underlying network topology. The difference lies with a wider management hierarchy and slightly less populated GridStat clouds. This suggests that a trinary tree has fewer levels than a binary tree and that the length of RPC connections between QoS brokers are longer. In that case, a binary tree and a trinary tree should yield approximately the same results.

6.7.2 The Flooding Algorithm

Figure 6.14 depicts experiments conducted at the three levels in the management hierarchy with 1 ms link latency, and Figure 6.15 shows the experiments conducted with 8 ms link latency. The diagrams represent the average time at which all status routers have received the mode change operation, and give a notion of how much time should be allocated for flooding a mode change operation to all status router participants. For example, Figure 6.15 (8 ms link latency) shows that flooding mode change operations initiated by the top level QoS broker require approximately 50 ms to deliver the operation to all status router participants, even under stringent network conditions. This means that, in the average case, status routers can safely switch modes at any time after that. However, some time variance must be taken into account, as shown in Section 6.9.2.

The experiments conducted at the three levels with 1 ms link latency (figure 6.14) show a minimal increase in mode change times that is caused by larger flooding domains when the coordinator resides higher up in the management hierarchy, and with a larger flooding domain increases the number of message rounds for the mode change operation to reach all status router participants. Another factor is the starting point of the flooding mechanism, e.g., flooding from the *middle* of the flooding domain is more efficient than initiating a flood from an edge in the flooding domain. An example of this is shown in figure 6.15 which depicts a large gap between the experiments conducted at the leaf and second level. Logically, one would argue that the experiments conducted at the second level should be closer to the results achieved by mode change operations conducted at

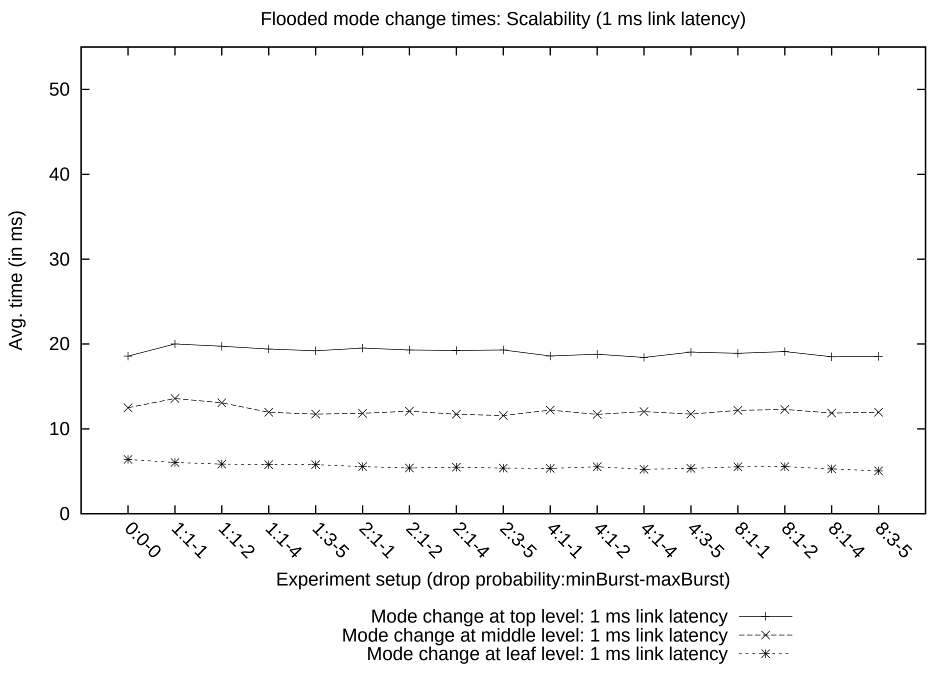


Figure 6.14: Scalability results using the flooding algorithm (1 ms link latency).

the leaf level. However, flooding mode change operations activated from the second level initiate the flooding mechanism from an edge in the flooding domain, while the leaf QoS broker initiates the flooding mechanism from the center status router in its single administrative cloud. This, in effect, means that the leaf level requires two message rounds to disseminate the operation to all the status router participants (one from the leaf QoS broker to the cloud), while the second level requires four message rounds. An optimization would be to let the second level initiate the flooding mechanism from the center node in one of its two administrative clouds, which would reduce the number of message rounds with one. Note that the actual topology in the involved clouds in a flooding mode change operation and the degree of redundancy between them have to be taken into consideration when determining which status router should serve as the point of origin for the operation.

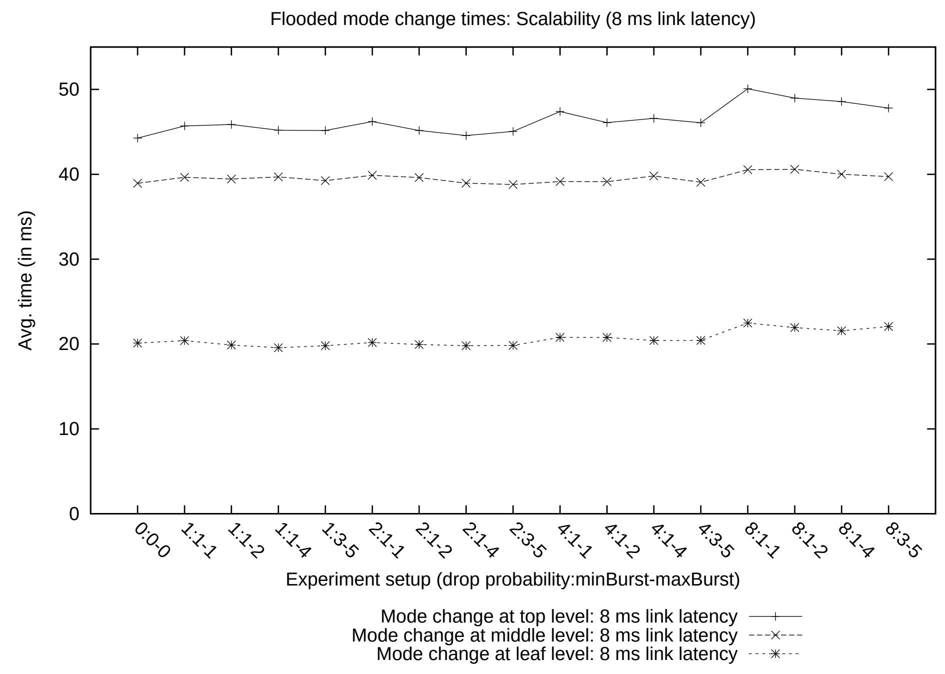


Figure 6.15: Scalability results using the flooding algorithm (8 ms link latency).

6.8 Flood Inconsistencies

The flooding algorithm has to some degree a limitation depending on the number of redundant links (event channels) connecting clouds. For example, if two clouds are only connected through one single link, there is a relatively high probability in the presence of link loss that one of the clouds will not receive a mode change operation disseminated out on the data plane by using the flooding mechanism. This scenario can potentially leave the data plane inconsistent as one cloud has no knowledge of any recent mode change operation. GridStat handles such scenarios through the management hierarchy (see Section 3.6) where QoS brokers eventually, after some timeout, attempt to correct the situation in the data plane by using the hierarchical mode change algorithm. The recovery mechanism will only contact the status routers the management hierarchy has not received any acknowledgements from and informs them to change to the new mode. It is important to mention that the flooding mechanism is not affected much by bursty loss (triggered by the link emulator) as a flooded message is only sent once per link, but bursty loss might affect the next flooding mode change operation.

Table 6.5 shows the number of times such scenarios outlined above occur under various network conditions by using the flooding algorithm where clouds are inter-connected by three event channels. All experiments are conducted at the top level in the management hierarchy and each experiment represents 300 flooding mode changes (1 instance means 1/300 mode change operations).

Experiment	Loss %	Min. burst	Max. burst	0 ms link lat.	1 ms link lat.	2 ms link lat.	4 ms link lat.	8 ms link lat.
1	0%	0	0	0	0	0	0	0
2	1%	1	1	0	0	0	0	0
3	1%	1	2	0	0	0	0	0
4	1%	1	4	0	0	0	0	0
5	1%	3	5	0	0	0	0	0
6	2%	1	1	0	0	0	0	0
7	2%	1	2	0	0	0	0	0
8	2%	1	4	1	0	0	0	0
9	2%	3	5	0	0	0	0	0
10	4%	1	1	0	1	0	1	0
11	4%	1	2	0	0	1	0	0
12	4%	1	4	1	0	0	0	0
13	4%	3	5	1	0	0	0	0
14	8%	1	1	5	4	3	0	1
15	8%	1	2	5	0	2	3	0
16	8%	1	4	4	0	0	1	2
17	8%	3	5	0	1	0	0	0

Table 6.5: Flood inconsistency occurrences.

6.9 Mean and Standard Deviations

Figure 6.16, 6.17, 6.18 and 6.19 show the mean and standard deviation for experiments conducted with 0 ms and 8 ms link latency at the top level by using both the hierarchical and the flooding algorithm. The experiments conducted with link latency settings 1 ms, 2 ms and 4 ms are listed in Appendix A.

6.9.1 The Hierarchical Algorithm

Figure 6.16 depicts the mean and standard deviation results with 0 ms link latency from hierarchical mode change operations initiated by the top level QoS broker. Higher link loss probabilities do not

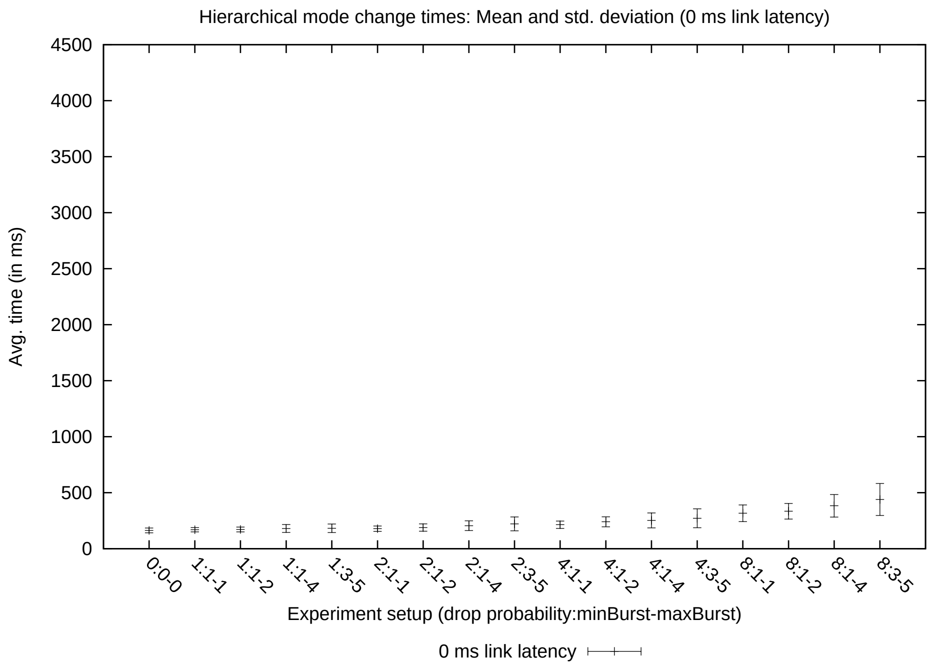


Figure 6.16: Mean and standard deviation using the hierarchical algorithm (0 ms link latency).

affect the results significantly as QoS brokers can quickly respond to link loss through the temporal redundancy property inherent in the RPC mechanism. RPC connections utilized by the top level QoS broker await RPC acknowledgements for 10 ms (Table 6.1) after a mode change message has been sent, and explains the deviations seen throughout all the experiments.

Figure 6.17 depicts the mean and standard deviation results with 8 ms link latency from hierarchical mode change operations initiated by the top level QoS broker. In comparison with Figure

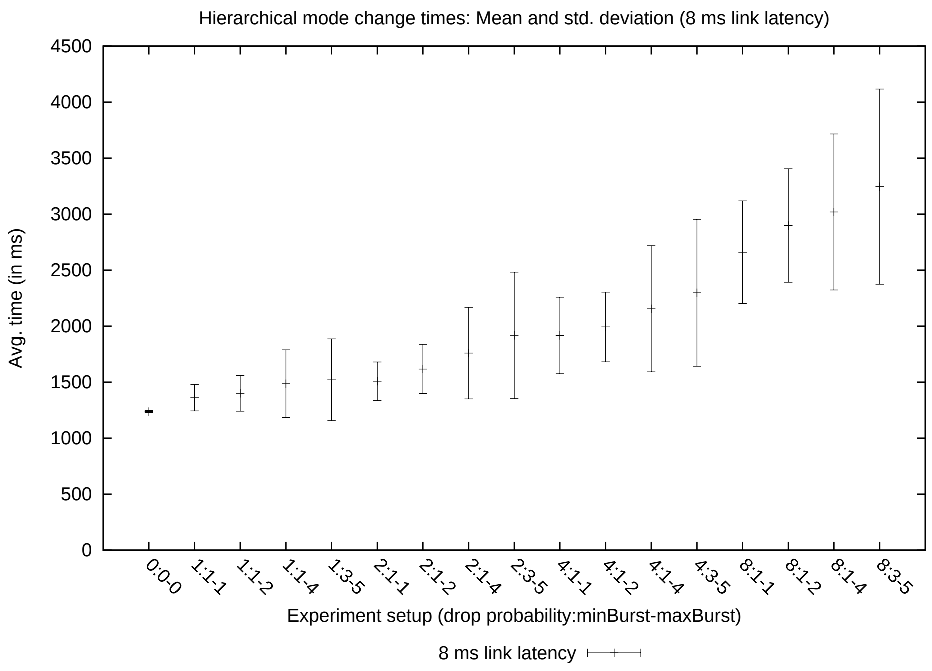


Figure 6.17: Mean and standard deviation using the hierarchical algorithm (8 ms link latency).

6.16, a higher link latency and RPC retry timeout (120 ms) clearly causes larger deviations. For example, four sequential losses with an 8 ms link latency and 120 ms RPC retry timeout result in an added delay of 480 ms to a hierarchical mode change operation, but only 40 ms with 0 ms link latency and 10 ms RPC retry timeout.

6.9.2 The Flooding Algorithm

Figure 6.18 shows the mean and standard deviation results with 0 ms link latency from flooding mode change operations initiated by the top level QoS broker. The flooding domain contains 20 status routers evenly distributed in four clouds, where clouds are inter-connected by three event channels. The standard deviations, for all experiments, are within the same range and do not increase when the overall link loss setting increases. Link loss during flooding mode change operations may cause a higher number of necessary message rounds for all participants to receive the

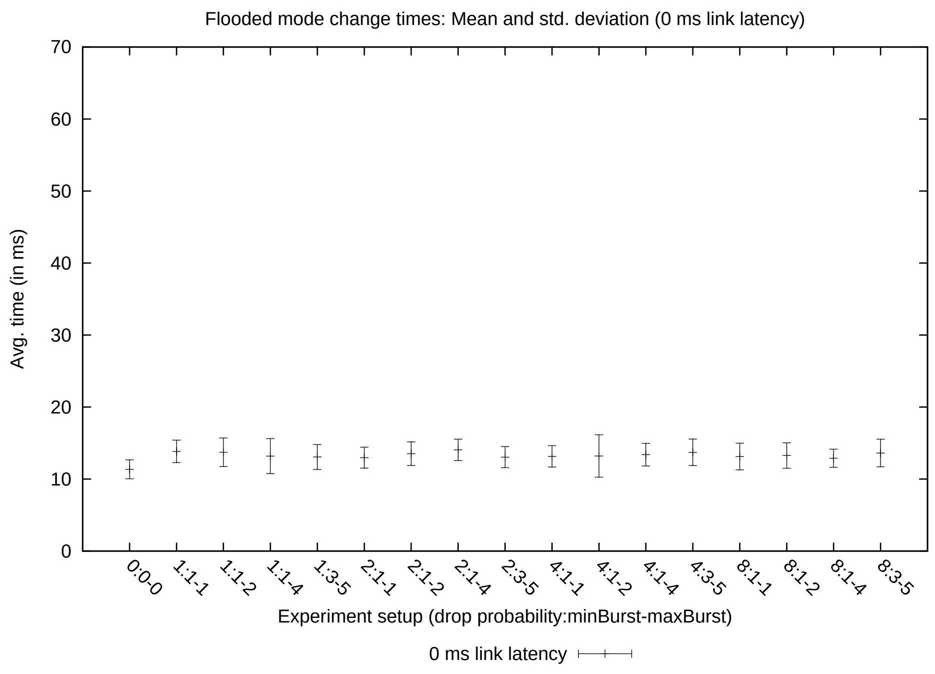


Figure 6.18: Mean and standard deviation using the flooding algorithm (0 ms link latency).

operation, and the standard deviation is caused by link traversal times and link emulator overhead.

Figure 6.19 shows the mean and standard deviation results with 8 ms link latency from flooding mode change operations initiated by the top level QoS broker. The flooding domain contains 20 status routers evenly distributed in four clouds, where clouds are inter-connected by three event channels. The standard deviations increase with a link latency of 8 ms as the number of necessary message rounds may increase when the data plane is experiencing heavy link loss. For example, one additional message round increases the flooding mode change time by 8 ms in comparison to 0 ms (plus link emulator overhead) in Figure 6.18. Furthermore, the diagram depicts the average time at which all status router participants have received the mode change operation. For example, with 8% link loss and a burstiness setting set to 1-1, all status router participants receive the mode change operation after 50 ms (average). However, as seen from the time variance, the coordinator should provide at least 55 ms for the flooding mechanism to run prior to switching modes. That is, status routers, for this particular example, can safely switch modes 55 ms after the coordinator

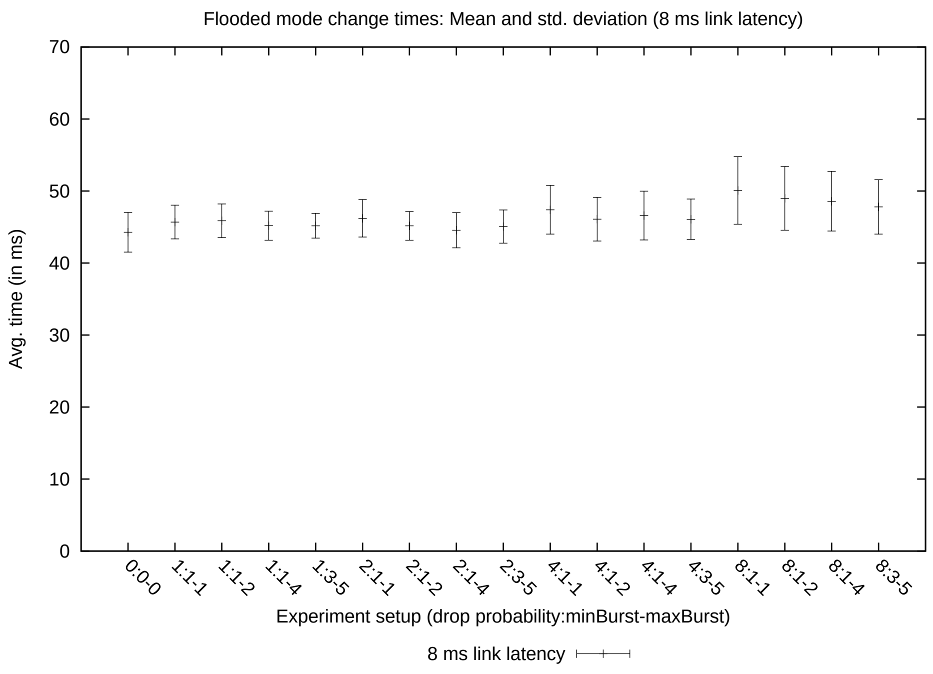


Figure 6.19: Mean and standard deviation using the flooding algorithm (8 ms link latency).

initiated the mode change operation.

CHAPTER SEVEN

RELATED WORK

Given that both GridStat and the global and hierarchical mode change mechanism are novel, there is not much related work that closely resembles the contributions of this thesis. Previous work has been done in network routing using multiple routing tables, but this work has adressed multiple simultaneously active routing tables for differentiated QoS routing[6][8]. More specifically, two routing tables are used; one for QoS traffic and another for best-effort traffic. This allows for differentiated routing strategies for the two traffic classes. For example, QoS traffic emphasizing lower drop rates could be forwarded along less loaded paths reducing the probability of drops due to congestion at the expense of longer paths and thus higher delay. The mechanisms proposed in this thesis share the property of previous work in the area of supporting multiple routing tables, but is novel in that a status router is collectively managed by a set of QoS brokers where each QoS broker controls a distinct set of routing tables, and has the ability to switch between them in run-time.

CHAPTER EIGHT

CONCLUSIONS AND FUTURE WORK

8.1 Conclusions

The implementation of global and hierarchical mode change mechanisms and management enables GridStat to quickly switch between bundles of subscriptions in run-time. GridStat pre-loads and populates routing tables in the status router network according to subscription modes where mode change operations effectively switches between routing tables. The alternative approach to switch between bundles of subscriptions would be for GridStat to replace subscription bundles through allocation and de-allocation methods. The allocation and de-allocation of subscriptions are expensive operations and will delay data streams when they are needed the most, and is undesirable for contingency handling in the electrical power grid. The power grid industry can benefit from this mechanism in GridStat by identifying and creating distinct mode holders for the information that is needed during various critical situations in the electrical power grid. Furthermore, GridStat's modes implementation enables load shedding for data streams and corresponds to how load shedding enables transmission adjustments in the electrical power grid.

This thesis has presented two mode change algorithms that offer different tradeoffs with respect to consistency, resource usage and efficiency. The hierarchical mode change algorithm enables subscriptions present in both the old and new modes to flow during a mode change operation through five execution phases. In addition, the hierarchical mode change algorithm prevents bandwidth, status router computational resources and queues to become exhausted. The flooding mode change algorithm is an efficient best-effort algorithm that informs status routers to change modes through the flooding mechanism, and therefore provides a high statistical delivery guarantee. However, the flooding mode change algorithm is not able to prevent computational resources and queues in the status router network to become overloaded.

Both mode change algorithms forward a mode change operation towards a set of status routers, either through the management hierarchy or by using the flooding mechanism, and are therefore liable to network failures. The recovery mechanism is initiated when one or more status routers do not respond to a pending mode change operation. In that case, the recovery mechanism attempts to contact the non-responsive status routers with instructions to restore their operating modes when the network failures are resolved.

Further, this thesis has presented the use and integration of the RPC mechanism in GridStat. The RPC mechanism enables GridStat entities to communicate over allocated subscriptions and will therefore benefit from the QoS GridStat provides. The RPC mechanism was primarily designed for external applications, e.g., actuator control, but this thesis shows that it is also useful internal to GridStat.

The experimental evaluation shows the mode change completion times for both mode change algorithms under various network conditions. The experimental results show that the hierarchical mode change algorithm scales linearly when increasing the hierarchical scope of a mode change operation. However, the algorithm adds a significant delay to the overall mode change completion time in the presence of link loss. The reason for this behavior is that QoS brokers do not utilize redundant subscription paths in their established RPC connections. The hierarchical mode change algorithm is expected to perform much better during poor network conditions with redundant communication paths in the management hierarchy. The results show that the flooding mode change algorithm completes a mode change operation more than an order of magnitude faster than the corresponding experiment conducted with the hierarchical mode change operation. Furthermore, the flooding mode change algorithm is less prone to link loss and shows a minimal impact on the overall mode change completion times. In term of scalability, the flooding mode change algorithm scales linearly and clearly shows that the width of the flooding domain and the point at which QoS brokers initiate the flooding mechanism from increases the mode change completion times.

The experimental evaluation demonstrates that the global and hierarchical mode change mechanisms and management implementation in GridStat is a practical and feasible approach for contingency adaptation in the electrical power grid. The experiments show that both algorithms scale linearly in the management hierarchy, are resilient to link loss through a well-connected status router network, and together with the recovery mechanism and an assumed QoS broker replication scheme provide the necessary means to ensure consistent mode changes. The mode change algorithm that is best suited for the electrical power grid will require investigation into standard practices and in close coordination with representatives from the electrical power grid industry.

8.2 Future Work

8.2.1 Multiple Mode Sets Per QoS Broker

Currently, a QoS broker does only support the definition and use of a single mode set and will always operate in one mode from its mode set. An improvement would be for QoS brokers to define multiple mode sets, and therefore operate in several modes. For example, a utility company might be interested in monitoring a common set of important power grid variables which will never change independent of which contingencies or power grid failures the utility will face. By supporting multiple mode sets, a utility can define a *baseline* mode which it will always operate in, and use additional mode sets for contingency monitoring. Furthermore, the support for multiple mode sets allows modes to be categorized. Utility companies can, for instance, define several modes, belonging to the same mode set, for detailed region-wise power line monitoring which can be used in the presence of power line failures. In order to support multiple mode sets, the use and storage of mode definitions at QoS brokers and status routers must be altered, but the overall mode change mechanisms can be utilized as is.

8.2.2 Optimized Routing Table Construction

With the introduction of multiple active routing tables in the status routers is an associated overhead in the forwarding algorithm. Status routers have to iterate through all forwarding data sets per processed status event and obviously slows down the forwarding algorithm. This section investigates how this step can be eliminated from the forwarding algorithm through alternate constructions of the routing table.

Status routers can, during mode change operations, calculate a single mode holder instance that contains all the necessary forwarding information for status events that belong to the current set of operating modes. This means, that for each mode change operation, the status routers have to calculate and construct a new routing table. Furthermore, the status routers must be careful to construct the new routing table such that the multicast mechanism is preserved. If subscriptions are added in run-time the forwarding sets must be carefully placed in the already existing structure.

Alternatively, instead of constructing routing tables on the fly during mode change operations, status routers can pre-calculate a single mode holder instance for all possible permutations of operating modes and have routing tables ready for use. This solution optimizes the current routing table implementation, but at the cost of higher resource usage. Status routers must store all routing table permutations and re-calculate them when subscriptions are added in run-time, which necessitates a large storage space for status routers that are envisioned to be run on light-weight hardware.

An optimization to the routing table permutations scheme would be for the management hierarchy to pre-calculate all possible routing table permutations or calculate a single routing table during mode change operations. This is a reasonable approach as QoS brokers will run on high-end machines. However, the management hierarchy is responsible for sending routing tables down to all status routers participating in a mode change operation, and may therefore increase the risk of inconsistent mode change operations in the presence of network failures.

8.2.3 Link Delay and Noise Measurements

The RPC mechanism uses a pre-configured retry timeout value when awaiting RPC acknowledgements and is common for all GridStat entities that utilize the RPC mechanism. When the RPC retry timeout value is lower than the connection's actual round-trip time, the sender might resend unnecessary RPC calls. On the other hand, a high RPC retry timeout value adds additional delay to the RPC calls in the presence of temporal network anomalies. Therefore, GridStat must provide a scheme for measuring subscription path delays and noise per established RPC connection. Such a measurement scheme can be incorporated into the RPC mechanism or provided as an additional layer on top of the RPC mechanism. A measurement scheme provided by the RPC mechanism has the sole benefit of being transparent to the overlying application, and the RPC mechanism itself is responsible for monitoring its established connections. Alternatively, the overlying application can provide the necessary means for implementing a measurement scheme on top of the RPC mechanism and allows the client to perform its desired measurements. Which method best suits GridStat is subject to the end-to-end argument and is a future design decision.

8.2.4 Alternative Flooding Mode Change Algorithm Design

The flooding mode change algorithm informs status routers to switch between modes at a predetermined future timestamp. An optimization to, or an alternative design to, the flooding mode change algorithm is for status routers to switch modes when needed to. A status event belongs to a subscription established to operate in one or more modes, and has an associated published timestamp and an estimated delivery deadline (subscriber). In this proposed design, a status router switches modes when a status event belonging to the new mode with a delivery deadline after the mode switch timestamp passes through it. The following observations are made in a flooding mode change operation switching from Green to Yellow:

- **Status events in Yellow**: Status events published prior to the mode switch timestamp but with a delivery deadline after the mode switch timestamp are forwarded through the status router network and delivered to the subscriber applications. Status routers on the path(s) switch from Green to Yellow when processing the status event.

- **Status events in Green**: Status events with a delivery deadline *close*, but prior, to the mode switch timestamp can not be guaranteed to reach the subscriber applications. The reason is that some status routers on the paths towards the subscribers may already have switched to mode Yellow, due to status events in Yellow with a delivery deadline after the mode switch timestamp. This introduces a notion of uncertainty regarding status events in Green that should be delivered prior to the mode switch timestamp.
- **Status events in Green and Yellow**: Status routers in the status router network can operate in either Green or Yellow during the mode change operation and forwards status events in Green and Yellow towards the subscriber applications, but possibly at the wrong rate.

The current implementation of the flooding mode change algorithm is used as is to deliver the mode switch timestamp to the status router network, where status routers switch modes when subscription traffic in the new mode with a delivery deadline after the mode switch timestamp passes through them. Subscription traffic in the new mode with delivery deadlines after the mode switch timestamp is guaranteed to be delivered to the subscription applications, whereas subscription traffic in the old mode with delivery deadlines just prior to the the mode switch timestamp might become dropped. More specifically, subscription traffic in the old mode with a delivery deadline prior to the mode switch timestamp might become dropped *after* the first status event in the new mode with a delivery deadline after the mode switch timestamp has caused some of the status routers to switch modes. In this time period, status routers operate in either the old or the new mode, and status routers can receive subscription traffic in both modes, which makes them liable to overload scenarios. This can be overcome by discarding status events in the old mode after the first status event in the new mode (with delivery deadline after the mode switch timestamp) has been sent, thus creating a status router behavior which closely resembles the prepare phase (see Section 4.2) in the hierarchical mode change algorithm. However, during this time, the status router network will drop status events in the old mode with a delivery deadline prior to the mode switch timestamp.

8.2.5 The Hierarchical Mode Change Algorithm and Flooding

The flooding mode change algorithm provides high statistical delivery guarantees through the flooding mechanism in GridStat. The current implementation of the hierarchical mode change algorithm, on the other hand, is not able to benefit from the flooding mechanism as it disseminates mode change operations (phases) through the management hierarchy by using the RPC mechanism. The RPC mechanism benefits from GridStat's QoS guarantees, most importantly spatial redundancy, but can not meet the same statistical delivery guarantees as the flooding mechanism. The flooding mechanism can be incorporated into the hierarchical mode change algorithm as follows:

- The leaf QoS brokers can, as an alternative to the RPC mechanism, flood the mode change operations out in its respective clouds.
- The coordinator of a mode change operation can flood mode change phases directly out on the status router network. Status routers will execute the mode change phase and send acknowledgements up to the management hierarchy, where QoS brokers collectively aggregate and send acknowledgements up towards the coordinator by using the established RPC connections.
- The recovery mechanism can utilize the flooding mechanism. For example, if a status router does not respond with an acknowledgement, the leaf QoS broker can flood the mode change phase to its cloud, and thereby be able to contact the assumed failed status router with a high statistical delivery guarantee, unless the status router is partitioned or offline.

8.2.6 QoS Broker Replication

QoS broker replication has served as a major assumption for both mode change algorithms, and thus no mechanisms have been implemented in order to circumvent failed QoS brokers. The recovery and RPC mechanism provide the means to continue mode change operations once QoS brokers resume operation through the assumed replication scheme. Active or passive replication serve as two suitable replication strategies for GridStat.

8.2.7 Resource Management

The introduction of global and hierarchical modes in GridStat stresses the necessity of a resource management scheme in GridStat. A resource management scheme is collectively employed by the management hierarchy to control and manage resources in the data plane and to make sure resources do not become overloaded at any time during operation. Global and hierarchical modes enable status routers to utilize several routing tables and increases the complexity of any resource management scheme. That is, the resource management scheme must ensure that no resources become overloaded with any set of operating modes.

8.2.8 Security

Security concerns have not been the focus of this thesis, but the global and hierarchical mode change mechanisms and management implementation might benefit from ongoing work on securing communication within the data plane and management plane. One project aims to secure data plane communication (subscriptions) through encryption and verification methods, and as mode change operations are disseminated through GridStat's RPC mechanism, the current global and hierarchical mode change implementation will directly leverage from the results of that project. Another issue is for status routers and QoS brokers to *trust* the contents of a mode change operation. For example, a misbehaving leaf QoS broker mediator might, by using the hierarchical algorithm, change the mode change operation contents and inform its status routers to change to a wrong mode. Therefore, appropriate mechanisms to detect or tolerate Byzantine failures, to some degree, need to be investigated for future versions of GridStat.

8.2.9 Clock Synchronization

The flooding mode change algorithm relies on synchronized clocks across all GridStat entities in order to inform all status router participants to switch modes at a predetermined timestamp. The Network Time Protocol (NTP) was used in order to synchronize (time) all the cluster nodes used in the experimental evaluation (Chapter 6). However, NTP is not suitable in a wide-area network which spans several network technologies. The correctness of the forwarding of status events is proportional to the level of clock synchronization between status routers. To ensure a high quality of forwarding, clock synchronization should be in the low milliseconds, or ideally within microseconds, as this is well below publishing rates commonly used in power grid applications. GPS synchronization products are available that provide accuracy of one microsecond to UTC.

8.2.10 Byzantine Failures

The hierarchical mode change algorithm and the flooding mode change algorithm rely on QoS brokers and status routers to behave correctly. Byzantine failures with respect to a QoS broker can affect the two mode change algorithms as follows:

- A QoS broker does not forward the mode change operation (phase) to its children QoS brokers.
- A QoS broker tampers with the mode change operations, but forwards them to all QoS broker children. This might cause all status routers in the hierarchical scope of the QoS brokers to execute the wrong mode change phase, or switch to the wrong mode.
- A QoS broker might not send an acknowledgement up to its parent QoS broker during aggregation rounds.
- A QoS broker might not initiate the recovery mechanism or execute a recovery request.

The above list presents Byzantine failure scenarios for QoS brokers. However, the same set of problems might be caused by an exploited or misbehaving status router, but in that case, a Byzantine failure will not cause much harm. In order to tolerate Byzantine failures, GridStat must employ an extensive amount of security features and failure detection mechanisms. Furthermore, status routers might benefit from voting mechanisms in order to detect exploited mode change operations.

APPENDIX

APPENDIX ONE

EXPERIMENTS

This appendix presents additional experimental results from the hierarchical and flooding mode change algorithms.

A.1 Hierarchical Algorithm Results

Table A.1 shows the average mode change times for all hierarchical mode change experiments conducted at the second level in the management hierarchy, and Table A.2 for the leaf level, respectively.

Experiment	Loss %	Min. burst	Max. burst	0 ms link lat.	1 ms link lat.	2 ms link lat.	4 ms link lat.	8 ms link lat.
1	0%	0	0	67.51	164.73	227.14	351.05	634.18
2	1%	1	1	72.08	168.97	234.67	377.80	658.78
3	1%	1	2	70.66	179.14	234.85	376.17	671.62
4	1%	1	4	71.63	167.42	239.94	376.92	671.54
5	1%	3	5	78.19	171.93	236.05	387.24	667.40
6	2%	1	1	78.40	185.80	248.78	386.16	684.18
7	2%	1	2	77.96	187.52	246.50	402.41	693.76
8	2%	1	4	84.89	189.58	258.03	394.22	729.52
9	2%	3	5	74.43	192.61	262.96	391.97	693.98
10	4%	1	1	84.79	193.00	282.87	435.85	746.50
11	4%	1	2	90.24	191.94	298.39	443.70	760.55
12	4%	1	4	94.29	200.69	285.09	455.06	789.31
13	4%	3	5	91.68	211.96	297.23	462.91	814.52
14	8%	1	1	120.28	243.32	340.45	515.32	909.08
15	8%	1	2	129.12	247.57	352.74	520.43	920.50
16	8%	1	4	133.42	272.37	406.57	588.06	979.92
17	8%	3	5	139.10	280.36	383.82	620.49	958.07

Table A.1: Hierarchical mode change experiments initiated by the second-level QoS broker.

Experiment	Loss %	Min. burst	Max. burst	0 ms link lat.	1 ms link lat.	2 ms link lat.	4 ms link lat.	8 ms link lat.
1	0%	0	0	38.14	71.02	99.18	158.12	276.86
2	1%	1	1	38.11	71.44	99.48	158.73	279.24
3	1%	1	2	35.18	68.52	97.06	156.26	278.25
4	1%	1	4	33.26	67.11	95.93	155.36	276.11
5	1%	3	5	34.46	68.57	96.46	155.62	276.65
6	2%	1	1	37.96	71.54	100.05	159.78	281.72
7	2%	1	2	33.91	67.32	97.26	157.00	282.49
8	2%	1	4	32.01	67.29	96.97	156.25	280.46
9	2%	3	5	32.15	66.52	95.02	156.70	278.78
10	4%	1	1	31.90	67.03	98.38	160.67	285.71
11	4%	1	2	32.58	67.97	96.75	160.00	290.79
12	4%	1	4	32.95	68.82	98.02	159.27	288.33
13	4%	3	5	32.99	66.35	96.14	158.97	288.05
14	8%	1	1	36.53	77.00	109.86	177.16	318.39
15	8%	1	2	35.86	75.81	103.66	171.53	315.67
16	8%	1	4	37.07	75.94	106.80	170.94	307.54
17	8%	3	5	37.97	71.20	107.29	172.70	315.99

Table A.2: Hierarchical mode change experiments initiated by the leaf-level QoS broker.

A.2 Flooding Algorithm Results

Table A.3 shows the average mode change times for all flooding mode change experiments conducted at the second level in the management hierarchy, and Table A.4 for the leaf level, respectively.

Experiment	Loss %	Min. burst	Max. burst	0 ms link lat.	1 ms link lat.	2 ms link lat.	4 ms link lat.	8 ms link lat.
1	0%	0	0	7.40	12.50	15.46	23.61	38.94
2	1%	1	1	9.79	13.58	16.58	24.31	39.64
3	1%	1	2	8.25	13.08	16.34	23.55	39.45
4	1%	1	4	7.93	11.96	16.42	24.00	39.70
5	1%	3	5	8.96	11.74	16.41	23.87	39.26
6	2%	1	1	8.01	11.83	16.58	24.11	39.88
7	2%	1	2	7.25	12.08	16.01	24.44	39.62
8	2%	1	4	6.97	11.73	16.25	24.08	38.95
9	2%	3	5	7.55	11.56	16.11	23.17	38.80
10	4%	1	1	7.85	12.21	15.65	23.85	39.15
11	4%	1	2	7.64	11.70	15.53	23.08	39.13
12	4%	1	4	9.04	12.03	15.42	23.43	39.80
13	4%	3	5	8.34	11.74	15.37	23.62	39.06
14	8%	1	1	7.10	12.18	16.22	24.40	40.55
15	8%	1	2	7.14	12.29	16.25	23.35	40.59
16	8%	1	4	7.05	11.86	16.59	23.24	40.01
17	8%	3	5	7.04	11.96	16.04	24.03	39.73

Table A.3: Flooding mode change experiments initiated by the second-level QoS broker.

Experiment	Loss %	Min. burst	Max. burst	0 ms link lat.	1 ms link lat.	2 ms link lat.	4 ms link lat.	8 ms link lat.
1	0%	0	0	4.26	6.40	8.23	12.13	20.11
2	1%	1	1	3.49	6.03	8.43	12.62	20.39
3	1%	1	2	4.33	5.84	7.89	12.38	19.88
4	1%	1	4	3.51	5.79	8.17	12.26	19.57
5	1%	3	5	2.73	5.78	7.84	11.92	19.80
6	2%	1	1	2.91	5.55	8.07	12.20	20.17
7	2%	1	2	2.96	5.39	8.11	12.08	19.94
8	2%	1	4	3.58	5.49	7.86	11.77	19.80
9	2%	3	5	3.06	5.38	7.93	12.13	19.83
10	4%	1	1	2.77	5.34	8.52	12.41	20.79
11	4%	1	2	2.98	5.53	8.45	12.55	20.78
12	4%	1	4	2.95	5.23	8.07	12.64	20.42
13	4%	3	5	2.76	5.34	8.27	12.50	20.43
14	8%	1	1	3.10	5.53	8.94	13.49	22.47
15	8%	1	2	2.81	5.55	8.48	13.25	21.93
16	8%	1	4	2.92	5.28	8.51	13.49	21.56
17	8%	3	5	3.43	5.05	8.33	12.88	22.06

Table A.4: Flooding mode change experiments initiated by the leaf-level QoS broker.

A.3 Mean and Standard Deviations

A.3.1 The Hierarchical Algorithm

Figures A.1, A.2 and A.3 show the means and standard deviations for hierarchical mode change experiments conducted at the top-level QoS broker with a link latency setting set to 1 ms, 2 ms and

4 ms repectively.

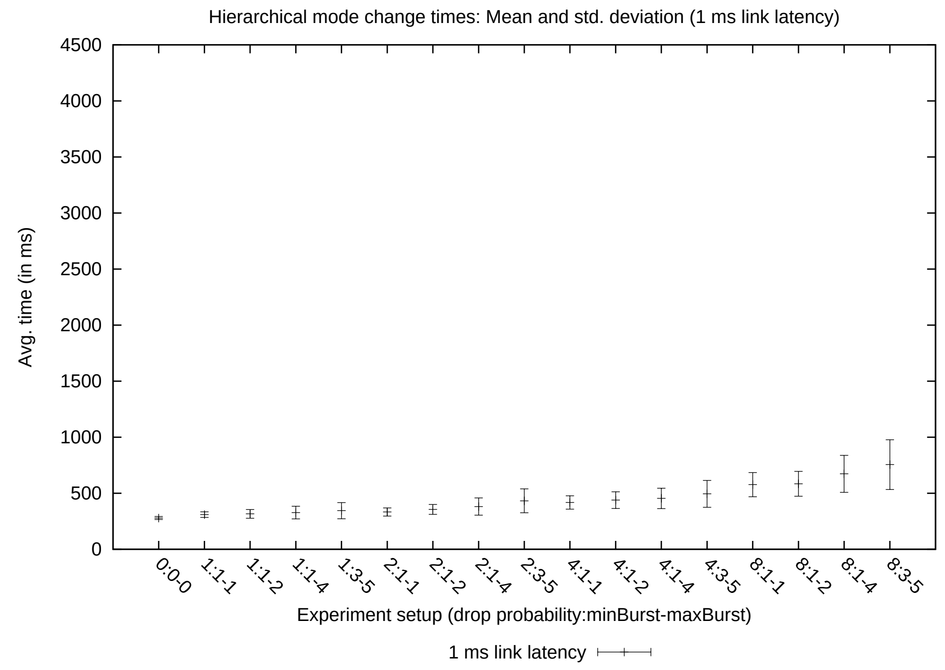


Figure A.1: Mean and standard deviation using the hierarchical algorithm (1 ms link latency).

A.3.2 The Flooding Algorithm

Figures A.4, A.5 and A.6 show the means and standard deviations for flooding mode change experiments conducted at the top-level QoS broker with a link latency setting set to 1 ms, 2 ms and 4 ms repectively.

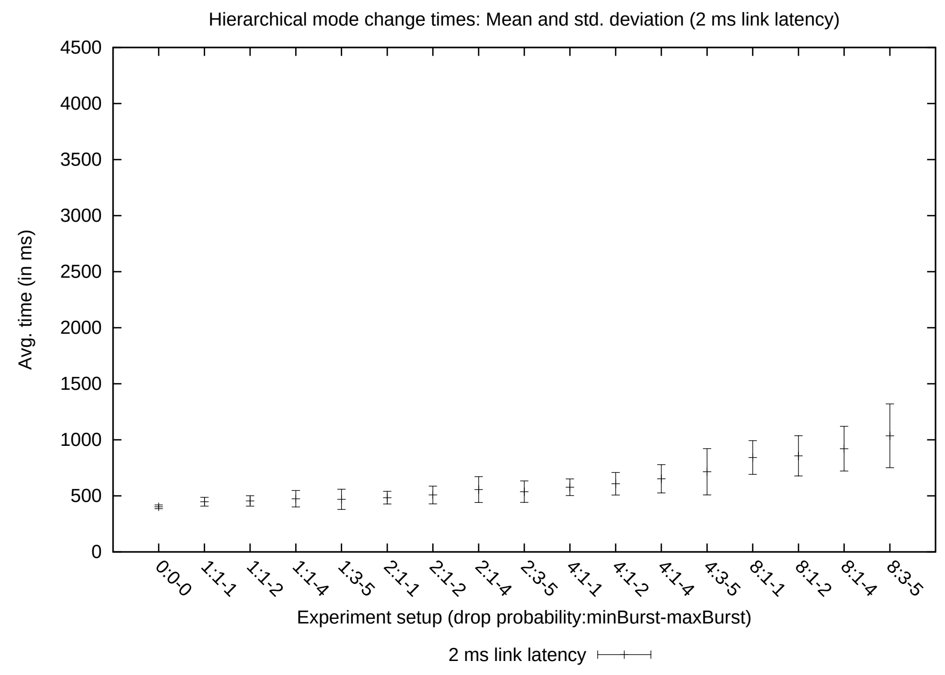


Figure A.2: Mean and standard deviation using the hierarchical algorithm (2 ms link latency).

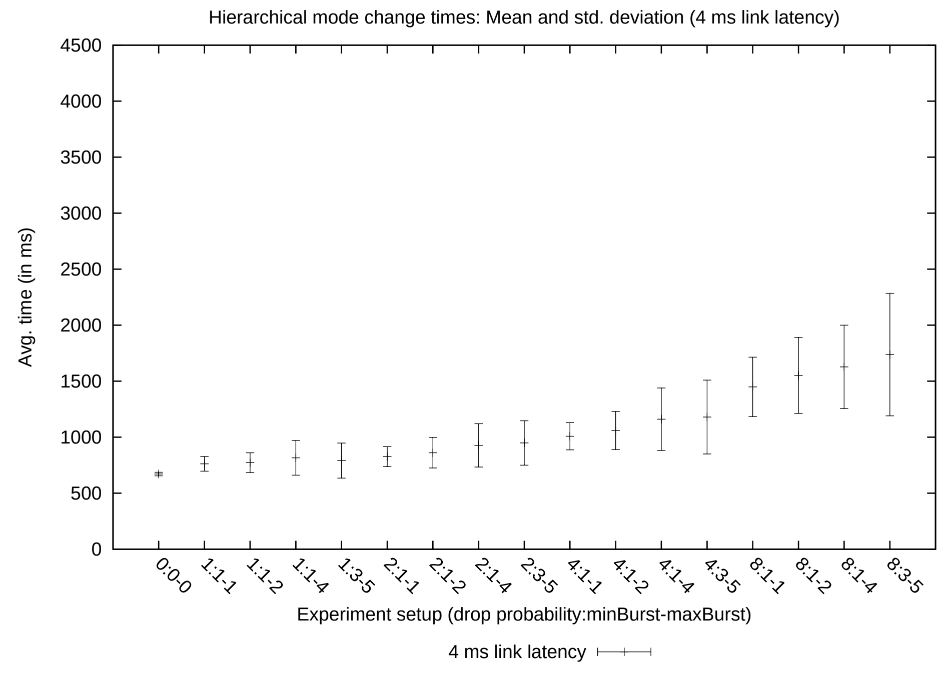


Figure A.3: Mean and standard deviation using the hierarchical algorithm (4 ms link latency).

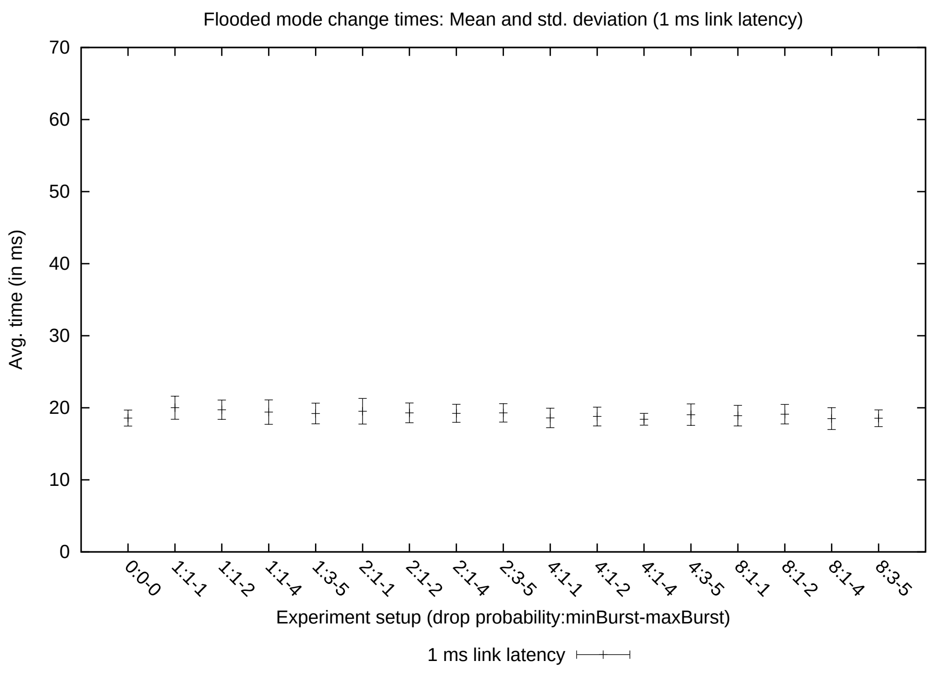


Figure A.4: Mean and standard deviation using the flooding algorithm (1 ms link latency).

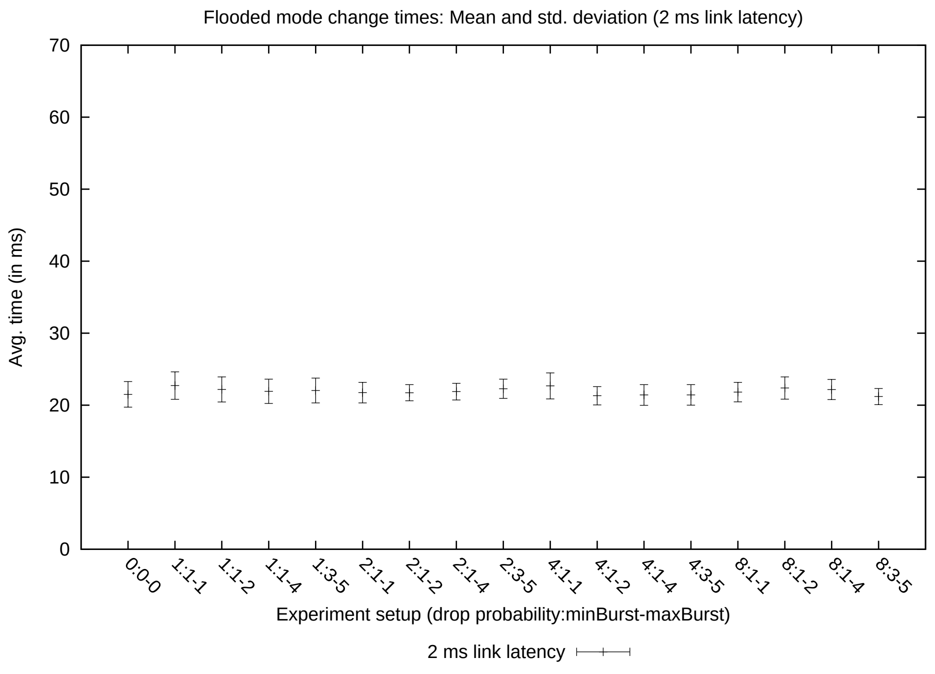


Figure A.5: Mean and standard deviation using the flooding algorithm (2 ms link latency).

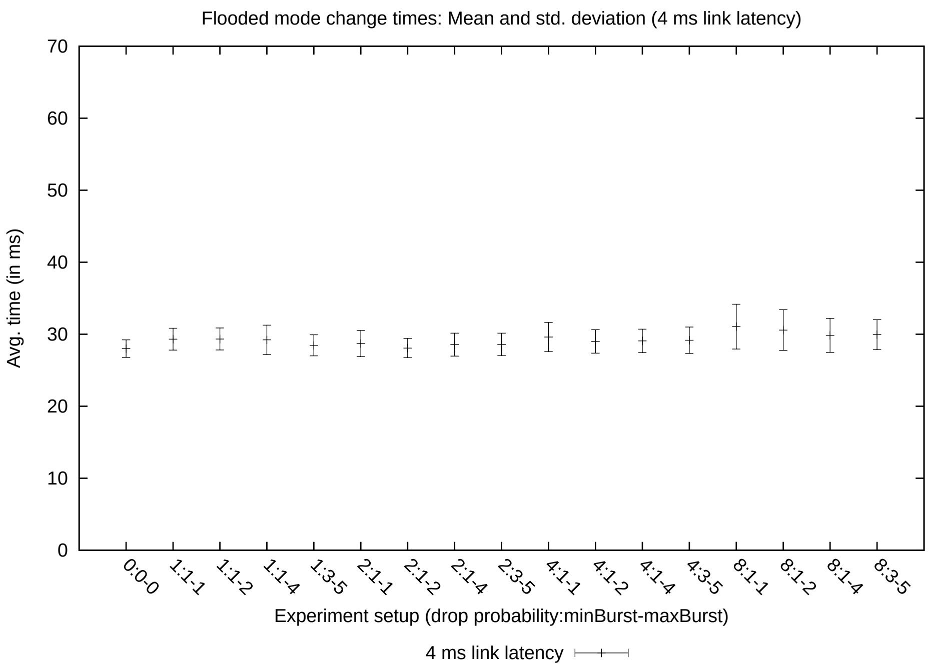


Figure A.6: Mean and standard deviation using the flooding algorithm (4 ms link latency).

BIBLIOGRAPHY

[1] D. E. Bakken, C. H. Hauser, H. Gjermundrød, and A. Bose. Towards more flexible and robust data delivery for monitoring and control of the electric power grid. Technical report, TR-GS-009, School of Electrical Engineering and Computer Science, Washington State University, May 2007.

[2] EPRI/CEIDS. The integrated energy and communication systems architecture, volms i-iv, July 2004.

[3] K. H. Gjermundrød. *Flexible QoS-managed status dissemination framework for the electrical power grid*. PhD thesis, Washington State University, 2006.

[4] C. Hauser, D. E. Bakken, and A. Bose. A failure to communicate: next generation communication requirements, technologies, and architecture for the electric power grid. *Power and Energy Magazine, IEEE*, 3:47–55, March-April 2005.

[5] V. S. Irava. *Low-cost delay-constrained multicast routing heuristic and their evaluation*. PhD thesis, Washington State University, 2006.

[6] H. Kochkar, T. Ikenaga, Y. Hori, and Y. Oie. Multi-class qos routing with multiple routing tables. In *Communications, Computers and Signal Processing, 2003. PACRIM. 2003 IEEE Pacific Rim Conference on, Vol.1, Iss., 28-30 Aug*, pages 388–391.

[7] E. S. Viddal. Ratatoskr: Wide-area actuator rpc over gridstat with timeliness, redundancy, and safety. Master's thesis, Washington State University, expected August 2007.

[8] Y. Wang, R. Kantola, and S. Liu. Adding multi-class routing into the diffserv architecture. In *Systems Communications, 2005. Proceedings, Vol., Iss., 14-17 Aug*.