RATE-BASED FAILURE DETECTION FOR CRITICAL-INFRASTRUCTURE SENSOR NETWORKS

By

BRETT EMERY TRABUN JOHNSON

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

Master of Computer Science

WASHINGTON STATE UNIVERSITY
School of Electrical Engineering and Computer Science

MAY 2014

To the Faculty of Washington State University:

The members of the Committee appointed to examine the dissertation of BRETT EMERY TRABUN JOHNSON find it satisfactory and recommend that it be accepted.

David Bakken, Ph.D., Chair

Carl Hauser, Ph.D.

Larry Holder, Ph.D.

Thoshitha Gamage, Ph.D.

ACKNOWLEDGEMENT

First of all I would like to thank Thoshitha Gamage for his constant advice, guidance, and willingness to go through weekly meetings and phone calls that were at moments difficult, but in the end pushed this thesis to completion. Another thank you goes to my chair and advisor Dave Bakken for his advice and inspiration for this work. I would also like to thank Carl Hauser and Larry Holder for taking the time to be a part of my committee and reviewing my thesis. Thank all of you for the help and feedback throughout these three years.

And I must especially thank my amazing wife Kayla for her support and help during the years. This accomplishment would not have been possible without her.

This research was funded in part by Department of Energy Award Number > DE-OE0000097 (TCIPG).

Disclaimer: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

RATE-BASED FAILURE DETECTION FOR CRITICAL-INFRASTRUCTURE
SENSOR NETWORKS

Abstract

by Brett Emery Trabun Johnson M.S.
Washington State University
MAY 2014

Chair: David Bakken, Ph.D.

Reliable fault tolerant communication is a vital component in advanced power grids (smart grids) that need to properly and correctly function amidst malfunctioning equipment, natural disasters, and cyber attacks. GridStat is a managed publish subscribe middleware communication infrastructure specifically designed to meet the high availability, low latency reliable communication requirements of these critical infrastructures. In order to maintain its services in face of malfunctions and attacks on the IT infrastructure it is built on, Gridstat's capabilities needs to be augmented using an adaptation service that detects and reacts to these events. Traditional work in failure detection has shown by that using heartbeat messages between components, it is possible to effectively identify components that crash and cease to operate. However, this technique will fail to identify Quality of Service (QoS) failures. By exploiting GridStat's rate-based semantics and complete knowledge of sensor flows at all network nodes, a rate-based failure detector was developed. These are more robust and capable of identifying these QoS failures. Additionally, these strategies have been compared with heartbeat failure detectors by using the DETER testbed, and have show that rate-based failure detectors are a potential solution for identifying and locating QoS failures in a managed publish subscribe system.

Contents

Page

1. INTRODUCTION 1
   1.1. MOTIVATION 1
   1.2. SCOPE OF THESIS 2
      1.2.1. Contributions 4
   1.3. ORGANIZATION OF THESIS 4
2. BACKGROUND AND RELATED WORK 5
   2.1. GRIDSTAT 5
      2.1.1. Management Plane 6
      2.1.2. Data Plane 8
   2.2. FAILURE MODELS 10
      2.2.1. Types 11
         2.2.1.1. Crash Failures 11
         2.2.1.2. Rate Failures 12
         2.2.1.3. Latency Failures 13
         2.2.1.4. Byzantine Failures 13
      2.2.2. Components 14
      2.2.3. Location 14
         2.2.3.1. Border Failures 15
         2.2.3.2. Edge Failures 15
         2.2.3.3. Internal Failures 15

2.3. EXPONENTIAL WEIGHTED MOVING AVERAGES 16
2.4. FAILURE DETECTION 17
2.4.1. Heartbeat Failure Detectors 18
2.4.2. Failure Detection Services at Scale 21
2.4.3. QoS Failure Detection 22
2.4.3.1. Risk Modelling Failure Detection 22
2.4.3.2. Exponential Weighted Moving Averages As A Failure Monitor 23
3. FAILURE DETECTION ARCHITECTURE 26
3.1. GENERAL ARCHITECTURE 26
3.2. DESIGN DECISIONS 28
3.2.1. Location of Failure Detection Service 28
3.2.2. Location of Rate-Based monitoring 29
3.2.3. Heartbeat coordination by Failure Detection service . . 30
3.2.4. Broker HeartBeat monitoring of FEs in the data-plane . 30
4. FAILURE MONITOR DESIGN 31
4.1. HEARTBEAT FAILURE DETECTORS 31
4.2. RATE-BASED FAILURE DETECTORS 35
5. SUSPECT GRAPHS DESIGN 40
5.1. EXAMPLES OF EVENTS GENERATED BY FAILURES 40
5.1.1. Crash Failure 41
5.1.2. Rate Failure 42
5.2. ARCHITECTURE OF SUSPECT GRAPH 45
5.2.1. Relationship Graph Construction 47

5.2.2. Handling New Events 49
5.2.3. Weights for Events 51
5.2.4. Fault Tolerance 52
5.2.5. Multiple Failure Scenarios 53
6. ANALYSIS AND EVALUATION 55
6.1. DETER TESTBED 55
6.2. RATE DETECTION PRECISION TESTS 56
6.2.1. Testbed Set-up 56
6.2.2. Procedure 58
6.2.3. Expected Results 59
6.2.4. Results 60
6.3. PERFORMANCE TESTS 63
6.3.1. Testbed Set-up 63
6.3.2. Procedure 64
6.3.3. Expected Results 65
6.3.4. Results 65
6.4. SCALE TESTS 66
6.4.1. Testbed Set-up 68
6.4.2. Procedure 69
6.4.3. Expected Results 69
6.4.4. Results 70
7. CONCLUSIONS AND FUTURE WORK 72
7.1. CONCLUDING REMARKS 72
7.2. FUTURE WORK 73

7.2.1. Improvements to RBFD 73

7.2.2. Improvements to Suspect Graphs 75

7.2.3. Adaptation Service 77

1. APPENDIX: USE CASES 78

Bibliography **95**

List of Tables

6.1 Subscription set-ups for rate tests 58

6.2 Subscription set-ups for performance tests 63

6.3 Timeouts used for performance tests 64

List of Figures

2.1 Gridstat Architecture 6
2.2 Management Plane Architecture 7
2.3 GridStat Data Forwarding Features 9
2.4 Push and Pull Heartbeats 20
2.5 Bipartite graph model of failure localization as done in SCORE 22
2.6 Sample use of Exponential Weighted Moving Averages. 25
3.1 Single domain architecture with Failure Detection 27
4.1 Heartbeat monitor example in GridStat 32
4.2 Heartbeat fault tolerance 33
4.3 Rate-Based failure detector design 36
4.4 Rate-Based monitor combination of Failure Information 39
5.1 Suspect Graphs crash failure example 41
5.2 Suspect Graphs rate failure example 43
5.3 Left: an example of a high rate failure not being forwarded. Right: an example of a low rate failure being propagated to the next FE. 44
5.4 Left: an example of a rate failure from a subscriber edge FE. Right: an example of a low rate failure on a publisher edge FE. 45
5.5 Suspect Graphs data structures 46
5.6 A Visualization of what a heartbeat monitor at a FE would be monitoring and what relationship graph would be made by the set-up. 47

5.7 A Visualization of what a rate-based monitor at a FE would be monitoring and what relationship graph would be made by the set-up. 48
5.8 Logic for handling a new failure detection event from a monitor. 50
5.9 Example of rate failure near a subscriber. 52
5.10 Multiple failures on the same flow 53
6.1 DETER testbed set-up for 8 FEs 57
6.2 Accuracy for Precision Tests 60
6.3 Accuracy for Precision Tests 62
6.4 Mean time to diagnose a heartbeat failure. 66
6.5 Mean time to diagnose a rate failure. 67
6.6 Left: DETER testbed set-up for 12 FEs. Right: DETER testbed set-up for 16 FEs. 68
6.7 Mean time to diagnose and detect a rate failure. 70

1. INTRODUCTION

1.1. MOTIVATION

Communication for the electric power grid is critical in order to maintain the system in a stable state [1]. For power grid operators to make informed decisions on how to maintain a stable power grid, it is essential that they have all relevant status information available as timely and accurately as possible. With demand for power growing every year, the power grid inches closer to operational limits where brief lapses in communication can turn an unlucky combination of faults into a blackout.

GridStat is a middleware overlay network designed to operate in the electrical power grid in order to provide reliable communication. Operating in a publish subscribe communication model, GridStat can provide the capability for measurement units (publisher) in the power grid to send measurement data to desired destinations (subscriber) without the need for additional logic to be aware of the destinations themselves. This provides the flexibility to change the number of publishers and subscribers in the system at runtime.

However, communication systems are susceptible to failure as well. The critical communication systems can fail to forward power system data with the correct quality of service (QoS). Failures in QoS can include too long of a delay from source to destination of data (timing failure), or sending too much or too little information per unit of time (rate failure). Some examples of the more extreme requirements for

critical communication include rates of data up to 720 Hz, or other flows of latencies within five to twenty milliseconds [1]. Failures in the communication system can present incomplete or, potentially worse, incorrect information about the state of the power system leading operators or automated systems to make misinformed decisions while maintaining the power grid.

Mechanisms used in networks to determine when failures occur are known as failure detectors. Heartbeat failure detectors are a well known and commonly used failure detector that are capable of detecting crash failures in a network. Crash failures occur when a node in the network ceases to operate in a detectable way. However, nodes can fail to satisfy their QoS, as mentioned earlier, and heartbeat failure detectors do nothing to detect these types of failures. QoS failures are more difficult to detect, due in part to the fact that they depend more on the QoS requirements of the information being forwarded. Thus far, no other published work has been found that has shown to be capable of detecting rate failures where information flows in the network vary from what is specified that can scale to meet the high availability and size of power grid communication networks.

1.2. SCOPE OF THESIS

The design of a failure detection service for GridStat that will locate both crash and QoS failures in a way that is both accurate and scales to the size required by power control networks. The failure detection service leverages the strong management GridStat has over the communication network to monitor the QoS of flows. This is done by using heartbeat failure detectors to detect crashes along with rate-based failure detectors outlined in this work in order to detect QoS failures. The failure

information gathered by these monitors help determine the location of the problem by using a technique also outlined in this work known as suspect graphs.

Rate-based failure detection is essentially the monitoring of packet arrival times at each routing component in the network to determine if a flow of packets are arriving too fast or too slow to satisfy their specified rate. Due to variations in packet arrival times, known as jitter, simply measuring the interval between packets is not sufficient to detect a rate failure. It is more beneficial to measure the average interval between packets using methods to filter out the noise from outlier packets. Additionally, measuring and checking the interval of every packet will not scale to thousands of packets per second. Moving averages are a filter that take a subset of data, typically a window moving forward in a time series, and determine the average in order to smooth out short term fluctuations and bring out long term changes in the data. Exponential weighted moving averages (EWMA) are a type of moving average that generates a filtered average in a way that decreases the weighting for previous samples exponentially. EWMAs has been applied to round trip time estimation in TCP [2] and monitoring control charts in manufacturing [3]. In this work we apply EWMA monitoring to information flows in GridStat.

Suspect Graphs are a technique that relates failure detection information from multiple different failure detectors to determine the location of a failure. In essence, the concept works by overlapping the potential failed components between multiple monitor events from heartbeats or rate-based failure detectors to highlight the most likely component failures that could cause the current set of observed failures to trigger. By correlating all of the events together, it is possible to come to a conclusion of what failed link or router, referenced from here on as a component, has caused the

failure.

Both the concepts of rate-based failure detectors and suspect graphs have been evaluated in the DETER testbed [4] to determine how well they perform.

1.2.1. Contributions.

The contributions of this work are as follows:

- Design and implementation of in-network per-flow rate-based failure detectors (RBFD), useable in rate-based middleware-level forwarding engines.
- Use of EWMA for a rate estimation mechanism, which thus far has only been used in a control setting of manufacturing on much slower time scales (order of 1 minute [3])
- Integration of rate-based failure detectors with traditional heartbeat messages by using failure localization with Suspect Graphs.
- Experimental evaluation of above in terms of accuracy, completeness, and performance (detection time), and scalability in the DETER testbed.

1.3. ORGANIZATION OF THESIS

Chapter 2 outlines background necessary to understand the domain along with related work that addresses the the problem of QoS failure detection, and describe the design and features of GridStat that are relevant to this thesis. An overall architecture of the failure detection service in this work is presented in chapter 3. The design of the lower level HeartBeat and Rate-Based failure detectors are outlined in chapter 4. Suspect graphs are described in chapter 5. Analysis and Evaluation of this work is done in Chapter 6. Finally, conclusions and future work are outlined in Chapter 7.

2. BACKGROUND AND RELATED WORK

To best understand the contributions in this thesis, it is important to first go over background information. This background includes the middleware overlay network GridStat, the failure models and definitions used in this work, established work on failure detection, and the data filtering technique known as exponential weighted moving averages.

2.1. GRIDSTAT

GridStat is a rate-based status decimation network that is designed to meet the low latency and high availability requirements of the power grid. Though it was designed for the power grid, GridStat can also be applicable in other domains where data is rate-based. This includes other infrastructure networks such as oil and gas. GridStat itself is based on a publish-subscribe paradigm that allows status information to be communicated from one publisher to many subscribers in an asynchronous distributed environment. A Publisher has no need to be aware of the location, number, or requirements of the subscribers it is sending to. Rather, the publisher only needs to send its information into the network. This middleware approach allows application developers using GridStat to focus more on their own application rather than on heterogeneous networking technology, operating systems, and programming languages that exist in the network. The contribution that GridStat brings to publish-subscribe design is management of Quality of Service (QoS) for each of the subscriptions that a subscriber has to a publisher. The attributes of the QoS that a subscriber

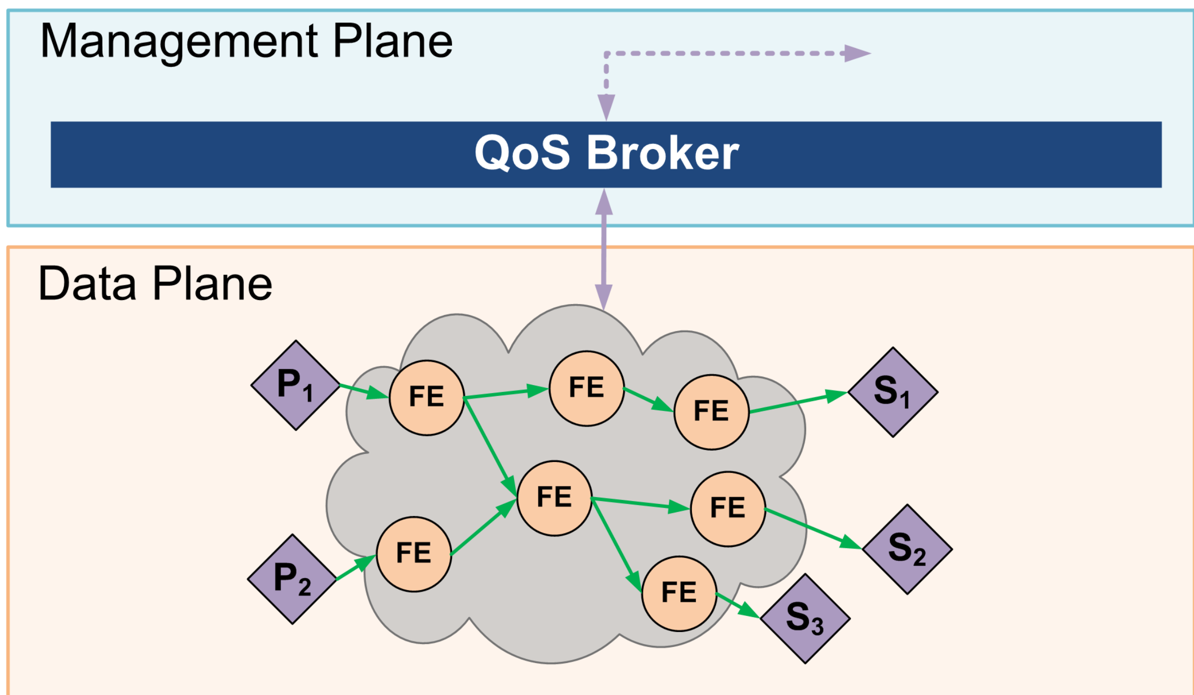

Figure 2.1. Gridstat Architecture

can specify consist of *redundancy*, *rate*, and *latency*. To meet the power grid requirements, GridStat is designed with scalability, flexibility, timeliness, and reliability.

The architecture of GridStat is illustrated in Figure 2.1. GridStat is divided into a *management plane* that consists of *QoS brokers*. These QoS Brokers are responsible for creating and removing subscriptions from the *data plane* that consists of a graph of *forwarding engines*. Forwarding Engines are responsible for forwarding the periodic publications through the network. *Publishers* interact with GridStat by pushing data into the data plane where it is forwarded to any number of *subscribers*. GridStat employs various techniques to achieve high availability while completing this task, including redundant paths, multicast, and bandwidth control.

2.1.1. Management Plane.

The management of GridStat is controlled by a hierarchy of *QoS brokers* that are responsible for managing the resources of the data plane. A generalized example is shown in Figure 2.1.1. The QoS Brokers are responsible for allocating resources for a subscription, removing subscriptions, and the registration of publishers, publications, and subscribers. The lowest level QoS brokers in the hierarchy are known as *leaf QoS brokers* and all other brokers are known as *parent QoS brokers*. The leaf QoS brokers each manage a *domain* that consists of a collection of *forwarding engines*, publishers, and subscribers. Parent QoS Brokers maintain a collection of other leaf and parent QoS brokers, and are responsible for managing the data flows between domains. There is currently no limit on the depth of the hierarchy.

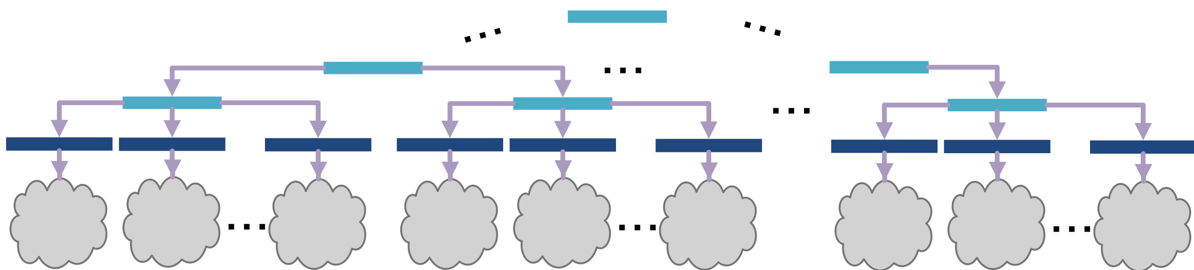

Figure 2.2. Management Plane Architecture

Leaf QoS brokers are the first to receive a request for a subscription. If the Leaf QoS Broker contains both the publisher and the subscriber in its own domain, then it will handle allocating a new subscription without communication with any other QoS broker. Should the publisher not exist in the leaf QoS broker's domain, then it will contact its parent QoS broker. The parent QoS Broker will check if it contains both publisher and subscriber in its child QoS brokers, and will continue to contact

its parent QoS broker if it does not. Once a parent QoS broker is found that resides over the domains of both publisher and subscriber, the parent QoS broker allocates the subscription path by recursively requesting child QoS brokers to allocate paths through their domains in order to connect the publisher to the subscriber. If a QoS broker receives a request for a subscription that it cannot fulfil, the QoS broker will reject the subscription request with the value of what QoS attribute could not be satisfied. After which, a subscriber may request a similar subscription with lower requirements.

2.1.2. Data Plane. The data plane in GridStat moves the data for each subscription from a publisher to subscribers. The plane is divided into domains that are each controlled by a leaf QoS broker. The elements that make up the data plane are known as *forwarding engines* (FEs). The three types of FEs can be classified as *edge*, *internal*, or *border* depending on what they are linked to. Edge FEs are the nodes that have a connection directly to publishers or subscribers in the network and are responsible for forwarding management requests from the subscriber or publisher to the leaf QoS broker along with forwarding data packets through the network. Publishers may communicate with the management plane through the edge FE to announce a publication and connect to the network. Subscribers use the edge FE to contact the management plane to request new subscriptions and connect to the network. Border FEs have links that connect to other domains. Subscriptions travelling over these inter-domain links are managed by a common parent QoS broker of both domains. FEs that are not an edge or border FE are known as internal FEs. Should an FE receive a packet that does not belong to a subscription that it is currently forwarding, the packet will be immediately dropped. Currently, the data plane uses UDP to send

data packets through the network. However, the design of GridStat is not specific to UDP and could be implemented on IP, Ethernet, or any other datagram switching protocol.

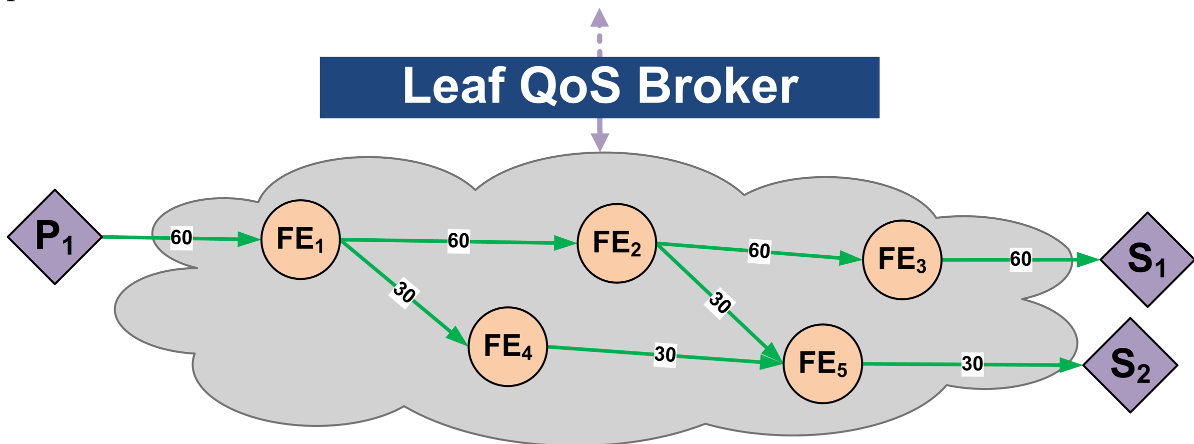

Figure 2.3. GridStat Data Forwarding Features

FEs use multicast, redundancy, and rate decimation to achieve the scalability and availability that GridStat provides. Multicast is implemented in order to prevent any link from forwarding a given packet more than once. If a FE receives a packet that needs to be forwarded down two output links, the FE will send a copy of the packet out both links. An example of this can be seen in Figure 2.1.2. At FE_1, each packet that arrives needs to be sent to S_1 and S_2. Instead of having P_1 send two copies of the packet to FE_1, FE_1 has the capability to copy a single packet and send it down both links.

Redundancy is an option that subscribers may request in order to have a subscription sent down multiple disjoint paths [5]. This is done primarily in order to still have an available stream when one of the paths is broken by failures. Additionally,

redundant paths cause subscriptions to have a slightly lower average latency. An example of redundant paths can be seen in Figure 2.1.2 for the subscription between P_1 and S_2. Because S_2 requires two redundant paths, the data flow sends packets down the two redundant paths $FE_1 -> FE_2 -> FE_5$ and $FE_1 -> FE_4 -> FE_5$ through the use of multicast. Note that in the current implementation of GridStat, publishers and subscribers may only connect to one edge FE and therefore 'true' redundancy is not achieved.

Rate decimation at the FEs is used to satisfy the QoS requirements of a subscriber while not overtaxing the network. The rate that a subscriber may request a publication can be equal to or less than the available publication rate. FEs will only send the rate of data requested by the subscriber through output links in order to prevent wasting network bandwidth or resources. This situation occurs when a FE needs to multicast an incoming packet down two different output links. The rate the FE sends down each of those output links will only be the required rate of subscribers further down that path. In Figure 2.1.2, this is seen in FE_1. Note that S_1 requires the publication at 60 times per second, and S_2 requires only half of that at 30. FE_1 simply forwards traffic to S_1 at 60 per second and S_2 at 30 per second. This is a unique feature of GridStat. In traditional IP multicast there is no choice available but to send the packet down both links at the maximum required rate, in this case 60 per second.

2.2. FAILURE MODELS

Failure detectors typically focus on crash failures in common networks such as the internet, where packet forwarding is done dynamically and by best effort. However, routing in the GridStat architecture is static, which allows strong management

of the network to maintain knowledge of the subscription flows, paths, QoS, and other attributes. This is effective managing the QoS of the flows, but GridStat must still be capable of acting when an IT component in the network fails. Typical failure detection work is effective when detecting and locating crash failures in the network, but tightly managed networks like GridStat have the opportunity for detecting QoS failures as well, which is not explored in most failure detection work. This work is targeted at detecting and locating rate failures that occur within the IT infrastructure responsible for sending subscription flow data. While doing this, the failure detection mechanisms must be designed and implemented in a way that does not reduce the scalability of GridStat, which would reduce its ability to be applied to the power grid.

2.2.1. Types.

The failure detectors presented in this thesis are designed to protect against the malfunctions that can affect the availability of GridStat's data plane. This section presents the specific types of failures that are being detected and located by the failure detection service. The failure detectors are designed to detect crash and rate failures.

2.2.1.1. Crash Failures.

A node in the data plane is considered *crash failed* if it has completely halted. These types of failures are unique in the fact that they can be typically detected by a lack of response from the component after the failure has occurred. These failures have the most dramatic effect on the system, in that they stop information that was being transmitted by them from flowing. Causes of a crash failure include power

failures, destruction of the physical hardware either by accident or malicious intent, or software bugs and hardware failures that cause the component to stop operation.

Redundancy failures are a product of crash failures. Recall that subscribers are allowed to request more than one redundant flow of a given subscription in GridStat. When a crash failure happens on one of the flows that a subscriber is supposed to receive, the subscriber will stop receiving duplicate packets. This occurrence is a redundancy failure in GridStat and is why redundant paths exist. If at least one of the paths remains operating correctly then the subscription can still get to its destination.

2.2.1.2. Rate Failures.

Rate failures happen when the periodic information from the flows arrive at receivers with spacing between packet arrival times being higher or lower than desired by the receiver. Rate failures in the data plane consist of a forwarding engine (FE) or a publisher sending subscriptions of data at a higher or lower rate than specified by the subscription. A type of failure like this can indicate a starvation of resources if the rate is too low, and indicate a malfunction or denial of service when the rate is too high. Malfunctions that can cause rate failures in the network include congestion in the network, denial of service attacks, or other software or hardware bugs.

Due to the rate decimation feature of GridStat, rate failures can have two types of behaviours. High rate failures are actually prevented from spreading into the FE network, due to FEs built in functionality to only send at the rate desired by the subscriptions. This means that, should a single FE be targeted for a denial of service attack, it will still only forward at the subscription rate. This does however still leave a vulnerability to attacks where false data is chosen to be forwarded instead of the true subscription traffic. Because of inherent difficulties in authenticating every

packet that reaches an FE, this sort of attack is difficult to prevent. Work has been done to evaluate techniques for authenticating packet flows in GridStat [6]. Low rate failures will exhibit a different pattern, due to the FE having no choice but to send at a lower rate in a drought of information. In this case, the low rate failure will be propagated down the subscription flow to the subscriber. Fortunately, this type of failure will not reach the subscriber if other redundant flows are still operating correctly.

2.2.1.3. Latency Failures.

Another failure type, called a *latency failure*, occurs when a piece of information that is produced by the publisher has been in transit for a period of time so long that it is delivered when a latency higher than promised. This type of failure is capable of happening when the resources of the network become strained, either by running close to capacity or during a denial of service attack. This type of failure was not targeted for this work, due to latency failures only truly being detectable at the subscriber end of the flow. An extension of the rate-based failure detector in this work could be used to monitor for latency failures at the subscriber in future work.

2.2.1.4. Byzantine Failures.

Another failure classification that is not covered by the failure detectors presented in this work is Byzantine failures. These types of failures are inspired from the Byzantine General's Problem where, in essence, any arbitrary failure can happen to messages sent and received by a process [7]. A failure detector that can diagnose and locate general Byzantine failures is outside the scope of this work.

Byzantine failures are extremely difficult to detect and diagnose due to their unpredictable nature. They also cannot be found using black-box failure detection

techniques [8]. This is because of a vulnerability to mute failures, where a black box failure detector may be getting information that the component is not failed, while the actual algorithm has completely halted. FEs that are still sending out heartbeats (a failure detector to be described in a later section), but no longer forwarding any traffic, are an example of this.

2.2.2. Components.

Many components in a network can fail. This also applies to the data plane in GridStat which is responsible for delivering the data from publishers to subscribers in the network. For the work in this thesis, the components that are being monitored for failures are the forwarding engines (FEs) and network links between them. FEs can have any number of neighboring FEs, which can allow for several different monitors to directly observe for failures. Links, on the other hand, will always connect one FE and either a publisher, subscriber, or another FE. This only gives two directly connected points a chance to monitor the link directly. From this point onward in the thesis, component or components will refer to links and FEs.

2.2.3. Location.

Recall that GridStat manages the communication infrastructure by dividing it into domains. By considering of how components lie in these domains, it is reasonable to describe the possible locations of communication components that can fail. The three main categories of failed components are border, edge, and internal.

2.2.3.1. Border Failures.

Failures are categorized as *border failures* if the suspected component has a link, or is a link to, a bordering domain. These failures are tricky, because in the architecture of GridStat, domains do not have knowledge of the topology of neighbouring domains. These failures require communication between domains in order to locate a given failure.

2.2.3.2. Edge Failures.

Edge failures are failures of a component that have a link to, or are a link to, a publisher or subscriber. These components are unique because in addition to forwarding data from source to destination, they also serve as a line of communication between the publisher or subscriber to the data plane (or brokers). Failures at edge locations are difficult to find because they can also prevent the failure detection logic at the subscriber from communicating with the failure detection service. Additionally, a current limitation of GridStat, is that only one FE or link can be connected to a Publisher or Subscriber. This makes the edge FE or link a single point of failure and difficult to tolerate failures. This limitation is planned to be removed, and when that happens edge failures will not be as potentially severe.

2.2.3.3. Internal Failures.

Finally if a failure is not in a location that can be categorized as a border or edge failure, then it is an *internal* domain failure. These failures happen to components that are mainly responsible for forwarding data and are linked to other intra-domain

components. The management of the domain will always have sufficient enough information about the neighbours of the failed component that it can determine the location of a failure without communicating with other domains.

2.3. EXPONENTIAL WEIGHTED MOVING AVERAGES

Moving averages filter out the noise of short term movements in information in order to add more confidence that the data is changing in the long term. In the setting of measuring attributes of periodic data, moving averages are used on a set of data in a window moving forward with time. If the series of data is changing over time, it will be observable with moving averages. By observing changes in the average between windows, it is possible to make predictions on the long term changes in the data.

The most basic form of moving averages is the Simple Moving Average (SMA). A SMA finds the arithmetic mean of the set of data given and gives even weight to each of the data points in the set. Exponential Weighted Moving Averages (EWMA) expands on SMA by placing stronger weight to the most recent data points. The calculation for the average, given the next data point, is determined by the following equation:

$$Average_i = \alpha * Sample_i + (1 - \alpha) Average_{i-1} \tag{2.1}$$

The value α is a smoothing factor between 0 and 1, where higher values give much more impact to the recent data. The exponential element of the technique is

given by the fact that each data point has exponentially less weight with each calculation of a new value. An initial value is usually supplied to shorten the learning period of EWMA in order to keep it close to the true average of the data at the start of monitoring. In most applications, when the true average is unknown, the initial value is determined by using SMA.

2.4. FAILURE DETECTION

No distributed system designer can expect that his or her system will run for an extended period of time without any failures. Eventually, a software bug or hardware failure will cause a component of the system to exhibit behaviours outside of specification. Fault tolerant systems are designed to continue, through failures, and a key component to doing this is by using a failure detector. A failure detector is a mechanism in the system responsible for identifying when a process has ceased performing according to specification.

Failure detectors themselves are not necessarily perfectly accurate and most are classified as unreliable failure detectors. These types of failure detectors mark processes in the system as either *suspected* or *unsuspected*, depending on whether they have found evidence indicating a process has failed. At first thought, unreliable failure detectors may not seem very useful, but Chandra and Toueg in 1996 showed that by using unreliable failure detectors, a distributed system can successfully reach consensus, a problem that was before unsolvable [7]. Failure detectors are further classified by their accuracy and completeness. *Accuracy* of a failure detector is defined by its ability to identify properly working components of the system as not

suspected. Accuracy can be defined as strong, weak, eventually strong, and eventually weak. A *strong failure detector* must never suspect a process as being failed before it has actually failed. In other words, a strong accuracy failure detector will have no false positives. Weak accuracy failure detectors must have at least one correct running process that is never falsely accused as being suspected. A failure detector is *eventually strong* or *eventually weak* when after a specific point in time, they have the properties of strong or weak accuracy forever. In practice, this can be shortened from forever to a sufficiently long enough time to perform a goal. The *completeness* for a failure detector defines how effective that failure detector is at identifying all of the incorrect processes and can be defined as strong or weak. In shorter terms, correctness restricts the number of false negatives, or crashed processes that are not identified. If a failure detector for every monitoring process will always correctly identify every incorrect process, then it is considered to be strongly complete. A failure detector with weak completeness will have at least one correct process capable of identifying every incorrect process that is being monitored [7].

2.4.1. Heartbeat Failure Detectors.

Chandra and Toueg introduced the concept of what is commonly called a heartbeat failure detector [7]. Heartbeat failure detectors have received much attention from the research community because of their capability to be very effective with little overhead.

Heartbeat failure detectors operate by having monitoring nodes in the network observe monitored nodes. These monitored nodes broadcast messages in the background at a continuous rate to be received by the monitors. Similar to a biological heartbeat, if a monitor node p does not receive a heartbeat message from node q after

a time-out δ, then p will add q to a list of nodes that are suspected to crash. In a basic heartbeat system, every node is a monitoring node that monitors all of its immediate neighbours. Increasing the heartbeat rate will give a smaller average time between a failure happening and its detection, but will also add more monitoring overhead to the network. The heartbeat rate affects the quantity of false positive and false negative suspects. Given too large of a heartrate, nodes will be suspected more often, and with too small of a heartrate, failed nodes can go unnoticed for a significant amount of time. These are clear trade-offs that designers of systems must decide upon.

An implementation of heartbeat could even exist without time-outs. To make the failure detection work, instead of suspecting nodes, a count of the number of heartbeats received from each node being monitored is sent. Supposing that crashed processes do not restart, a monitor receiving this list of counts will know a process has crashed when the heartbeat count is the same repeatedly [9].

Additionally, heartbeat failure detectors can be operated in push or pull styles of operation [10]. Push failure detectors that monitored nodes continuously send 'I am alive' messages to monitors at the heartbeat rate. In the pull method, the monitor nodes prompt for the liveness with 'are you alive?' messages that the monitored nodes reply to with 'I am alive'. The push style of heartbeats are typically used due to the smaller required bandwidth, but pull heartbeats are useful when monitored nodes have no concept of time to know when to send heartbeats.

Research into heartbeats has focused on optimizing and reducing the performance impact on the system. Some effective techniques include piggybacking, using

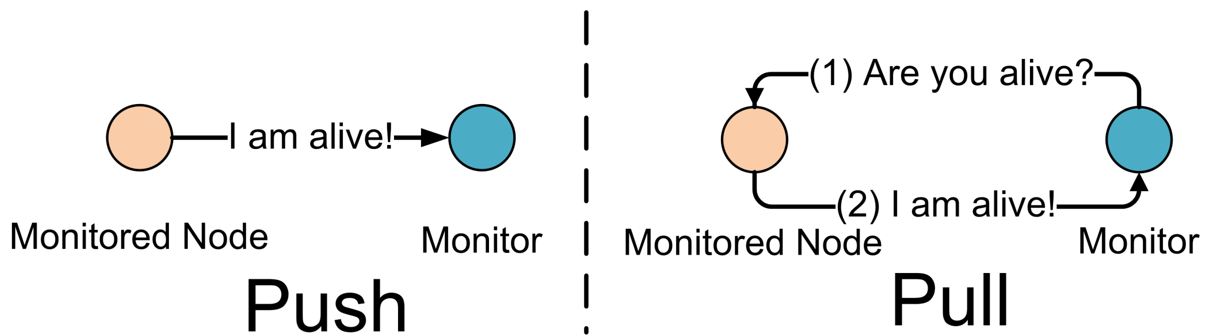

Figure 2.4. Push and Pull Heartbeats

an adaptable heartrate [11, 12], and optimizing the heartrate for bandwidth or latency [13]. Piggybacking heartbeat messages is a simple technique for using the application's own communications to act as heartbeats when possible [11,14]. Should a monitor node p receive an application message from q, it can treat the message as a heartbeat and reset the time-out period δ. This allows q to reset its sending time-out whenever it sends an application message and avoid sending extra information.

Another technique to optimize the performance of a heartbeat failure detector is to use an adaptable heartrate [11, 12]. The concept of the adaptable heartrate is to change the heartbeat rate depending on the current stress on the system. For instance, if the system is experiencing heavy traffic, then the heartbeat rate can be reduced to use less bandwidth and thus reduce the overhead on the system. Assuming the stressed network, *ipso facto*, is sending more packets than usual, combining this technique with piggybacking reduces the heartbeat overhead on the system while still achieving a relatively small "heartbeat" rate.

So and Sirer in [13] formalize an optimization problem for the heartbeat rate, taking into account the expected lifetimes of nodes. They formulate a latency-minimizing

failure detector and a bandwidth-minimizing failure detector. The latency-minimizing method minimizes the latency to failure detection given a constraint on bandwidth, and the bandwidth-minimizing failure detector minimizes bandwidth given a latency constraint. The lifetime of nodes are taken into account by providing short lived nodes with a larger heartrate due to the increased likelihood of failing. Heartbeat failure detectors are a well explored method of providing detection of crashed components of a system at a small cost, which makes them a failure detector of choice for most networks.

2.4.2. Failure Detection Services at Scale.

Several different types of failure detection services have been proposed in research. Once the network they are managing starts to grow in size, the scalability of the failure detection service starts to become a major design point. For most services, they can be categorized by scaling through a hierarchy, or gossip architecture.

Hierarchy schemes benefit from being fairly simple to create and understand. In these schemes, failure events are sent to a centralized location, thus allowing sending multiple events wrapped in a single message to save bandwidth. An example of a hierarchical structure has been made using CORBA, utilizing failure detectors addressed as first-class objects [10].

Architectures built on gossip style have been applied to failure detection services [15]. Communicating by gossip is similar to flooding the network with an update. If a node n wishes to communicate an update s to the network, it would randomly select another node to communicate with and send the update s. The receving node, m, would then select one of its neighbors randomly to share the update with, while n selected another neighbor to tell. Given enough time, most of the network

would become aware of the update s. Gossip architectures have unique benefits that make them applicable for failure detection, the strongest of which is that there is no need to be as strongly aware of the underlying topology as it changes.

2.4.3. QoS Failure Detection.

2.4.3.1. Risk Modelling Failure Detection.

Another method of diagnosing a failures location that is relevant to the work in this thesis, is based on risk modelling of the network. Examples of these types of failure detectors are CLUSTER MAX-COVERAGE [16], MAX-COVERAGE [17], Shrink [18], and SCORE [19]. All of the previously mentioned failure detectors work off of the same basic premise. Given some knowledge of the topology of the network, it is possible to pre-calculate what the state of the system will look like for a given failure. When a failure does occur in the system, the observed state of the system is compared against the pre-calculated scenarios to determine the most likely candidates for failure. This may not be feasible in a generic internet, however, in GridStat's target domain, the topology of the data plane changes very slowly and is completely known and tightly managed.

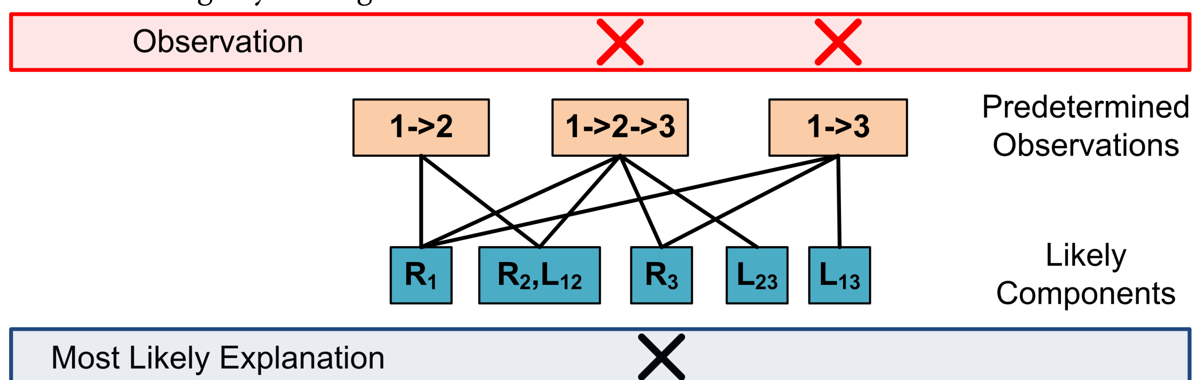

Figure 2.5. Bipartite graph model of failure localization as done in SCORE

The problem is formally defined as follows. Consider a given a set of components $C = c_1, c_2, ..., c_n$ and a set of potential failure groups $F = f_1, f_2, ..., f_m$. Each failure group is a collection of components that share an observable property o_j that will ultimately fail when one of the components in the risk group fails. Note that a component can exist in more than one failure group. When a failure occurs in the system, a set of observed failures $O = f_1, f_2, ..., f_k$ is given. The problem is to find the most likely set of components that will cover the set of observations , O. These components are the most likely candidates for failure, and are thus marked as suspected. In Figure 2.4.3.1, observations O_1 and O_2 are observed as failures. Given the bipartite graph in the figure, it is mostly likely that the failure of component C_2 is responsible for the given observations.

2.4.3.2. Exponential Weighted Moving Averages As A Failure Monitor.

EWMAs have been applied in several applications, most notably, in *Round Trip Time and Variance Estimation* in TCP [2]. In a more failure detection oriented application, EWMA has also been used in manufacturing control charts. In manufacturing process control, EWMA is used to detect small shifts in the mean of a process in order to indicate an out-of-control scenario [3]. The main idea of EWMA Control Charts, is that there are two bounds known as the Upper and Lower Control Bound. The bounds are determined by the following formula:

$$Bound = \bar{x} \pm 3 * \hat{\sigma}\sqrt{\frac{\alpha}{2-\alpha}} \tag{2.2}$$

Here, $\bar{x}$ is the desired average value and $\hat{\sigma}$ is the current standard deviation

in the data. When the EWMA moves above or below the Upper or Lower Control Bounds, then the out-of-control scenario has occurred. To caculate $\hat{\sigma}$ a method proposed by the RFC for TCP round trip time estimation is used [2]. The method is very similar to calculating the EWMA value seen in 2.1, for this equation the difference between the estimated average and the sample is used to estimate the variance between packets. The following equation below, is used to calculate $\hat{\sigma}$.

$$\hat{\sigma}_i = \beta * |\bar{x}_i - Average_i| + (1 - \beta) * \hat{\sigma}_{i-1} \tag{2.3}$$

An example of this moninoring technique can be seen in Figure 2.6 where the scenario is in control for the first 30 values and then moves out of control for the next 30, before returning to normal. The black bars show when the true failure begins and ends.

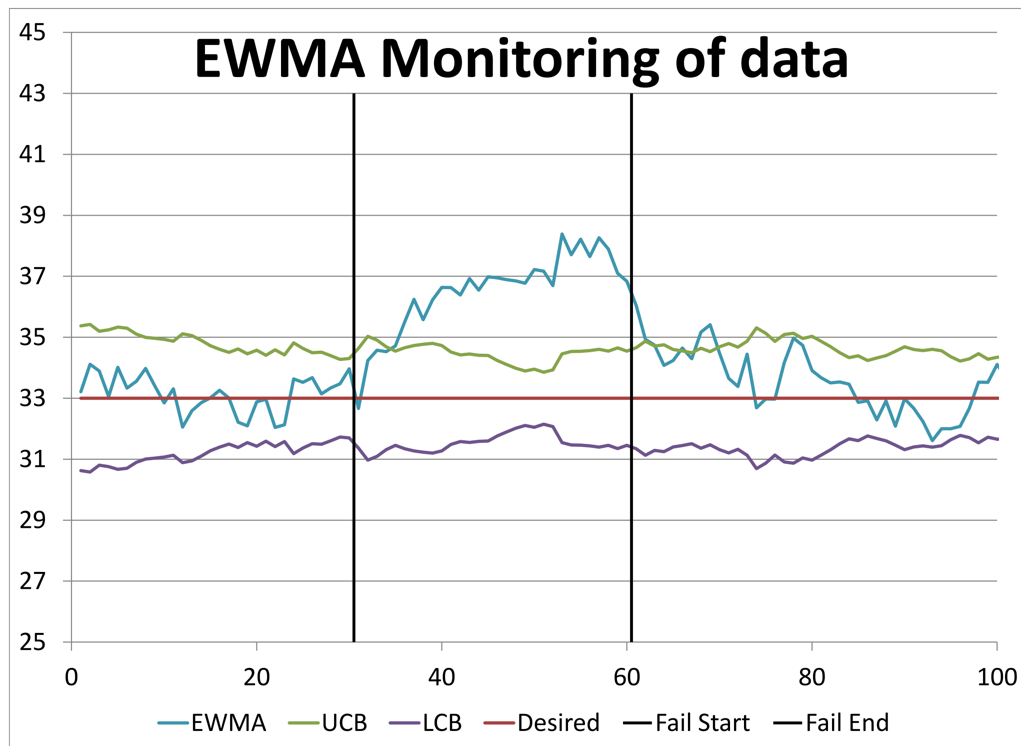

Figure 2.6. Sample use of Exponential Weighted Moving Averages. The vertical axis is the millisecond interval measurement between packets, and the horizontal axis is the index of the sample number. In this figure the desired interval is 33ms with a standard deviation of 2ms. At index 30 the rate changes to 37ms in a rate failure scenario and returns to normal operation at index 60.

3. FAILURE DETECTION ARCHITECTURE

In this work, two main modules for failure detection were added to the architecture of GridStat. First is the *failure detection service*, which functions as the main receiver of failure events from monitors, and uses the information to determine where the source of the failure is likely located. The other module added is the *rate-based failure detector*, which is a failure module in the data-plane meant to monitor the rates of subscription flows. This chapter outlines the basic architecture of GridStat with rate failure detection, and elaborates on decisions that determined the design of the architecture in this work.

Durring the work on this thesis, several use cases of failures were made to aid in the design of the Failure Detection Architecture. These use cases can be found in Appendix 1.

3.1. GENERAL ARCHITECTURE

The failure detection work is divided into two realms of operation; the monitors in the data plane watching individual flows for faults, and the logic in the service that digests all events from the monitors into a diagnosis of the failure. Figure 3.1 highlights a comparison between GridStat with and without the failure detection elements.

In Figure 3.1, there are a couple new modules added to the GridStat architecture. On the right-hand side of Figure 3.1, the new green modules are shown. In

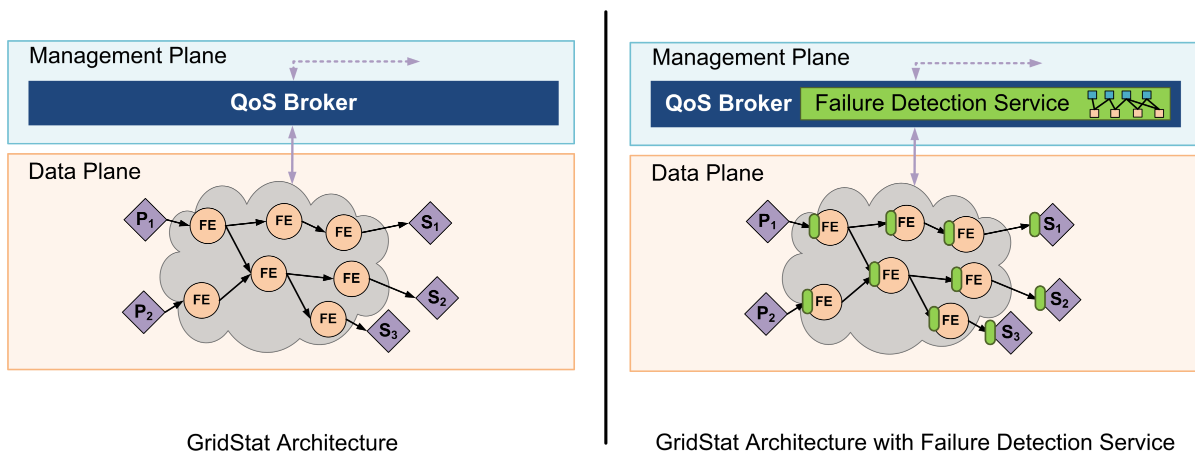

Figure 3.1. Left: Single domain architecture. Right: Single domain architecture with Failure Detection.

the management-plane, the failure detection service has been added, and in the data-plane, modules for rate-based failure detectors have been added to the FEs and subscribers.

Additional work was done to enhance the heartbeat failure detection mechanisms. Functionality in the failure detection service now coordinates heartbeat events in order to determine if the heartbeat failure involves a FE, or simply a link to one.

The failure detection service is a module in the management of GridStat that is responsible for monitoring the status of components in the data-plane. It is also the main point that all failure events are sent to when they are generated by the monitors in the data-plane. This information is needed for GridStat QoS Brokers to communicate status information, and in the future it will be essential to communicate with an adaptation service that will actively try to repair failures. The failure detection service requires the capability to communicate with all of the monitors, access the status of each of the components, and have knowledge of the current set-up of flows.

A bipartite relationship graph of the relationships between monitors and components in the data-plane is managed by the failure detection service. This graph is used to quickly determine what a new failure event from a monitor is related to. To create the graph, the failure detection service requires knowledge of the set-up of the data-plane, including the current subscription flows.

The rate-based failure detector is an additional module attached to the FEs. By watching the incoming rate of subscriptions, the monitor can report when QoS is no longer being satisfied en-route to the subscriber. Knowledge of the desired subscription rate for each flow and the arrival times of data packets is required by the rate-based monitor to operate.

3.2. DESIGN DECISIONS

Several key choices in design focused this work into the architecture that it is. In this section, decisions for failure detection module placement and decisions for current functionality to extend are discussed.

3.2.1. Location of Failure Detection Service.

The failure detection service was implemented within the QoS Broker. This is mainly due to the brokers having the same requirements to manage the FEs. First of all, the Broker has knowledge of the subscriptions in the domain that the failure detection service requires in order to locate failures and specify desired QoS attributes, which will be discussed in the following chapters. Additionally, it is assumed that a leaf QoS broker has redundant communication paths to the data-plane, allowing a failure detection service at the broker to have a channel to receive failure events from monitors even if one has crashed. This means that there is no need to worry that a

failure in one FE will affect the failure detection communication for another FE.

Another location the failure detection service could have resided at would have been as a separate process in the management plane. The advantage of this set-up would be that failure monitors could continue to operate even if the broker had failed. Unfortunately, there are at least two limitations of this set-up. The first, is that implementing another process for the failure detection service would require communication of subscription flow set-ups and status of components between the broker and failure detection service. This would increase the complexity of the system and open up possibilities for failures. In addition, the extra time required to communicate status information to the broker would be costly. Another downside, is that if the QoS Broker has failed already, the status information provided by the failure detection service would become less useful because nothing can be done by the management service without the broker.

While this location of the failure detection service described works well for failures located in one domain, it will also diagnose failures that affect subscriptions across multiple QoS Broker domains. Failures that effect multiple domains were not a focus of this work, however it will be pointed out later how multiple domain involvement can be added to the framework.

3.2.2. Location of Rate-Based monitoring.

The implementation of rate-based monitors is done within the FE process, similar to the heartbeat implementation. This again is because the requirements to observe incoming packets and desired rates is needed by the FE as well. Additionally, if the FE process had crashed it would be caught by heartbeat monitors, and the information provided by a rate-based monitor on the crashed node would not be useful.

Another location that the rate-based monitor could exist, is in a separate piece of hardware on incoming network interfaces of the FE. One strong advantage of a hardware based monitor is that it would be able to very quickly measure the rate of incoming packets while only adding a small delay to arrival times, instead of taking up CPU time like the implementation within the FE process. A hardware rate-based monitor may be explored in future work, but is not covered in this thesis.

3.2.3. Heartbeat coordination by Failure Detection service.

Heartbeat monitors existing in GridStat already had the capability to report link crash events to the broker. Before this experiment, the heartbeats would only transfer missing events to the broker that would just update the link status. In this work, the failure detection capabilities were extended in order to coordinate heartbeat status messages. Using this the failure detection service determines whether the heartbeat events indicate that a link is crashed or a FE has crashed.

3.2.4. Broker HeartBeat monitoring of FEs in the data-plane.

Another improvement to heartbeats was the addition of broker heartbeat monitoring of FEs. This enhancement was done to ensure that every FE, with the exception of edge FEs, has three heartbeat monitors detecting if the FE has crashed. Additionally, this gives the FE the capability to determine if the communication with the management broker has failed.

4. FAILURE MONITOR DESIGN

This chapter explores the design of the failure detectors used to monitor the data-plane. The types of monitors explored in this research are heartbeat failure detectors (HBFD) and rate-based failure detectors. The focus of this chapter is to examine the operation of the monitors in the data-plane of GridStat, explain how the monitor determines if the system is operating abnormally, and consider what is sent to the failure detection service when an event occurs. The work done by the Failure Detection Service to diagnose the location of failure based on these events will be discussed in a later chapter.

4.1. HEARTBEAT FAILURE DETECTORS

Recall that, heartbeat failure detectors operate by having nodes send periodic "I am alive" messages to their neighbours. These neighbours will become suspicious of the node failing, if after a time out period, no new messages from the node are received. The implementation of heartbeats in GridStat follows this same design in the data-plane in order to maintain awareness at the FEs of which neighbours are still running. In addition to FEs monitoring each other locally, the FEs also are monitored by the failure detection service, which is monitoring all of the FEs in the domain. Heartbeat failure detection is used in GridStat in order to detect crash failures of links and FEs quickly and efficiently without relying on any subscription flow data.

An example of this is shown in Figure 4.1, which shows the set of potential monitors for a single FE that is forwarding data in GridStat. These monitors are: the

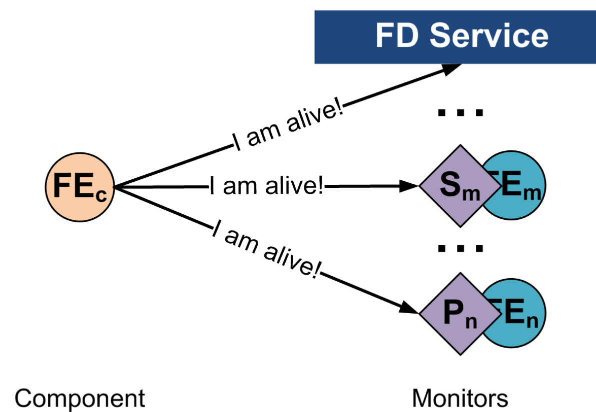

Figure 4.1. Heartbeat monitor example in GridStat.

Failure Detection service, a FE or Subscriber data is being sent to, and a publisher or FE where data is being received from.

Note, it is also possible for heartbeats to operate in a pull method where monitors instead prompt "are you alive?" from the components. However, the push method of heartbeats was chosen for GridStat because it uses half of the bandwidth in comparison. Additionally, push heartbeats have less risk of dropping heartbeats, because there is only one chance to drop a heartbeat message instead of two.

The length of time outs for the heartbeat messages in GridStat is a configurable value. A desirable heartbeat for GridStat depends on the trade-off of resource usage and detection time. For instance, if the time out is short, then more resources will be used to handle the heartbeat events. In the evaluation of the work presented in this thesis, several different heartbeat time outs were used to observe this relationship.

The time out is an important factor for accuracy of heartbeats. Too short a time out will generate more false positive events, and too long a time out will leave crashed components undetected. Beyond the short term accuracy, heartbeats have

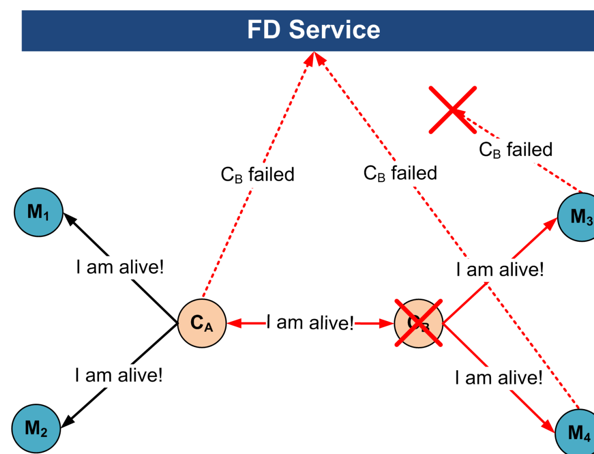

Figure 4.2. Example of fault tolerance of heartbeats. Given three monitors of a component, a single monitor can fail to send an event to the service without causing a misdiagnosis to occur.

been shown to be able to detect all crashed components and not suspect any properly functioning components given a long enough period of time[7].

One very helpful property of heartbeat failure detection is that it is tolerant of failures. For example, it is possible that a heartbeat failure event sent to the failure detection service could be dropped, or not detected at all. Since several heartbeat monitors are watching every component, occasionally dropping an event will have a low probability of introducing inaccuracies.

An example of this can be seen in Figure 4.2. The component C_B being monitored in this figure is shown having suffered a crash failure. In this failure, there are three other FE monitors observing C_B, the monitors are C_A, M_3 and M_4. All three of the monitors identify the crash failure and send events to the failure detection service, but unfortunately the message from M_3 was lost. Because at least two of the monitors managed to get event messages to the failure detection service, it still had

enough information to determine that C_B had failed.

Observe that the fewest monitors an FE can have while connected to a flow is three; one for the FE being where data is being received, one for the FE where data is being sent, and the QoS broker monitor. In these situations one of the heartbeat notification messages can be dropped and the failure detector will still successfully identify the failed internal, or border, FE as failed. If FEs are even more connected than this minimal amount, then heartbeats become even more loss tolerant.

The exception is an edge FE, where it is possible to only have fewer monitors due to receiving the data packet from or sending them to a publisher or subscriber. In this situation, the heartbeat failure detector is not loss tolerant. In the worst case, the edge FE is only connected to a publisher and subscriber. There is no good remedy for this issue, because heartbeat detection from a publisher or subscriber cannot be relied upon like FE heartbeats. This is due to publishers and subscribers having no information about the location of the QoS Broker. In order to send a message to the broker, it would need to route through the FE, which if crashed, would drop all heartbeat monitoring messages, thus rendering it useless for a publisher or subscriber. This could be reconsidered when GridStat allows publishers and subscribers to connect to more than one edge FE, thus having the ability to forward status infromation from a publisher or subscriber.

This simple failure detector is an effective solution for locating crash failures. It has the advantageous properties of being loss tolerant, capable of detecting multiple failures, and able to detect crash failures even when no flow is being forwarded through the FE or link.

4.2. RATE-BASED FAILURE DETECTORS

Heartbeat failure detection is very effective at locating and detecting crash failures, but it has no capability to detect when the QoS of flows is violated by rate or latency failures. To detect these issues, a failure detection method called *rate-based failure detection* was developed. In this section, it will be explained how rate-based failure detectors monitor the flows of subscriptions in GridStat.

The method of monitoring rate in this work is influenced by the method of using EWMA to monitor manufacturing processes[3]. In this case the EWMA is used to monitor shifts in the mean of individual measurements between packets. Using EWMA, it is possible to take this packet by packet rate sample and smooth it out to determine the long term average. By using the EWMA control chart techniques, the FE can send a failure suspicion notification to the failure detection service when the EWMA rate average for a flow crosses the upper or lower control bounds.

Figure 4.3 shows the design of the rate-based failure detector for a FE. On the left is the packet forwarding module that serves the FEs' main functionality of forwarding packets. On each new packet a FE must forward it to the desired destination in as little time as possible. To monitor the rate without interfering with this process, a simple sample of the rate between packets is taken on each packet arrival. This sample is stored in a temporary location by flow that is continuously overwritten.

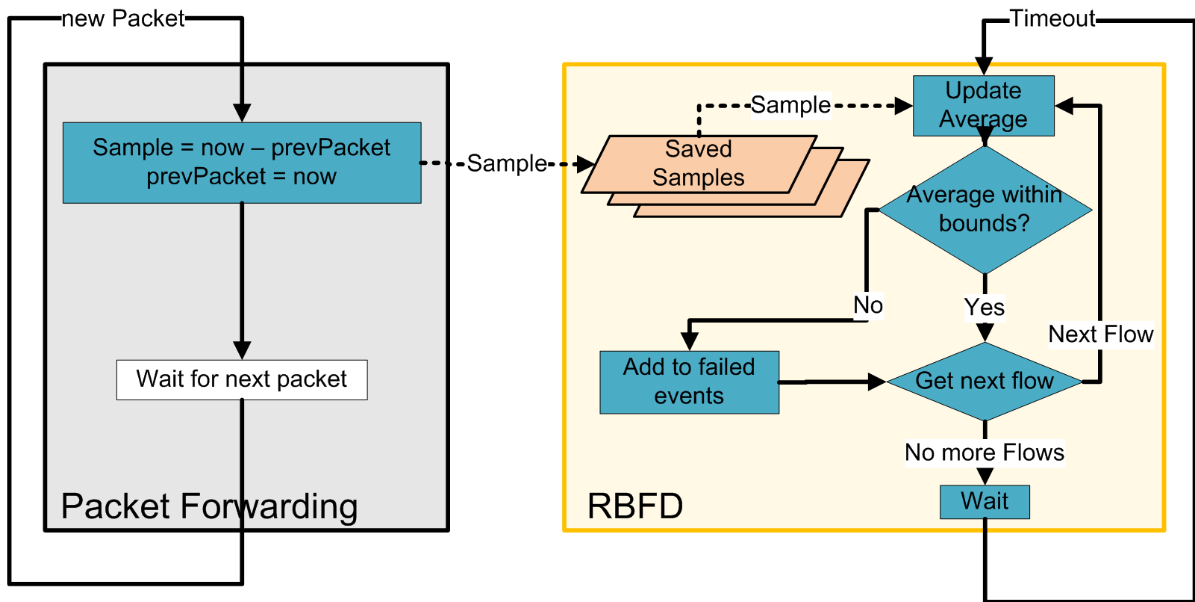

Figure 4.3. Rate-Based failure detector design.

This means, that for each flow in the FE, there is a latest sample value that can be used by the exponentially weighted moving average (EWMA) monitoring to estimate the current rate. The other side of Figure 4.3 shows the logic taken by the monitor on a time out separate from the data forwarding work done by the FE.

This sampling is done in a separate thread, because sampling every packet has the potential to quickly reduce the scalability of FEs. For each extra unit of time the EWMA calculation takes for each packet, then the delivery time of the packet would be slowed by that amount for each hop. Given a large number of hops, this effect could make it impossible to achieve a latency that is required for the more extreme power grid requirements. Instead, rate-based failure detectors work by calculating the EWMA at a separate rate that is tunable to a desired overhead. This truly allows

the user to pick how much overhead is desired to allow for rate-based failure detection. Choosing the rate for the EWMA updates has similar trade-offs to a heartbeat rate. Shorter time outs will create more overhead for the FE, and higher time outs will increase the time it takes to detect a failure after it has happened.

Referring back to Figure 4.3, observe that for every time out, the rate-based monitor will iterate through all of the flows being monitored at that FE. For each flow, the temporary sample value is used to update a moving average that is being tracked for the flow, and a deviation value being tracked for the flow. Recall that, the new average for EWMA is determined by the equation 2.1, and the new deviation is found with the equation 2.3.

Deviation must be based off of the estimated average and not the desired rate for the subscription in order to measure estimated deviation of rate between packets and not the deviation between packet rate and the desired rate. Once this is done, a check is made to see if the new average has crossed out of the acceptable bounds for failure. If it has, then an event is generated and stored to be sent to the failure detection service.

After iterating through all of the flows, any events that have been generated are changes in status that need to be reported to the failure detection service. If a report is needed, all of the flow status information for that FE are gathered into a message and sent to the failure detection service. Once all of this work is done, the rate-monitoring waits until the next time out to do another check.

The bounds used to determine if the EWMA is in a failure state is determined by the formula 2.2. In the formula, $\hat{x}$ is the desired rate for the flow.

Note that, the FEs and subscriber must already know what the desired rate is for

each flow, and can use that as the desired average to base the EWMA bounds around. Fortunately, this is already present in order to implement GridStat's capability to do rate down-sampling, where every FE must know what rate to send data through each output link. From this, it can be easily inferred that we also know what the incoming rate should be at FEs. Subscribers trivially know the rate due to having specified it to create the subscription.

The choices for the values of EWMA parameters α and β are a trade-off to consider for rate-based detection. Due to the equations used to determine average and deviation, the values must be between 0 and 1. Typically when choosing α, a lower value will create fewer false positives by making the average less sensitive to new sample. However, this will also add more delay to determine if a failure exists. For EWMA charts in industrial process control, α is typically 0.3 [3]. The value for β has a slightly different trade-off. A higher beta value will cause the deviation to be sensitive, thus causing the bounds to grow large when variance in the data is high. This is beneficial for spontaneous jumps in deviation and rate that resolves itself in time. However growing the bounds too much makes the failure detector vulnerable to not detecting a failure at all if the deviation during the failure is high.

The status information for all of the flows at an FE are sent in a failure event in order to give the failure detection service enough information to rule out possible failure locations, and to prevent message loss from confusing the service. Observe the example in Figure 4.4. In this example, two separate failure scenarios are shown. In the top scenario, only one of these flows has failed. Using this information, the failure detection service can eliminate C_A as a suspect. The other scenario shows that both flows will fail if C_A had failed. The failure status of both flows is sent in this

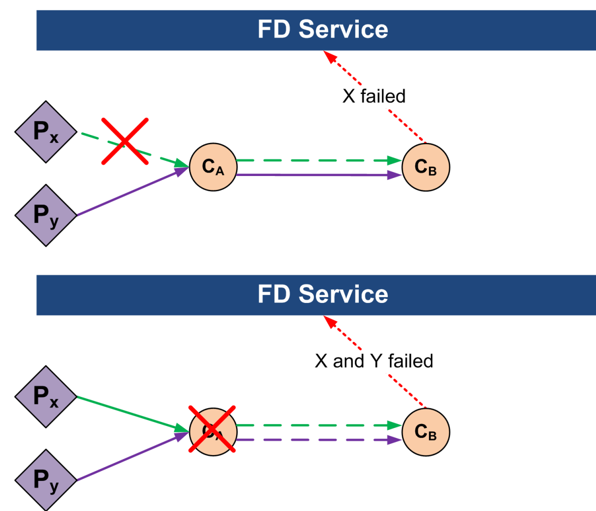

Figure 4.4. Rate-Based monitor combination of Failure Information. In the above example, only publication x failed and that is sent to the failure detection service. In the bottom, since both Y and X appear failed, C_b will only send one failure notification that it observes both X and Y to have failed.

case to help the failure detection service diagnose the failure. Additionally, if the two failures were sent in separate messages, they would run a higher risk of dropping one of the messages.

By adding rate detection logic to the FEs, the rate-based failure detection has a much finer granularity than what previously could only be achieved by discovering QoS failures at the subscriber. These rate-based failure detectors also have the potential to detect rate failures before the failure has reached the subscriber, given a large enough end-to-end latency. In the design of rate-based failure detectors, it is important to also be mindful of their impact on the scalability of FEs. These failure detectors must be capable of monitoring for failures while maintaining the their capability to deliver power system control data with milliseconds of latency.

5. SUSPECT GRAPHS DESIGN

To correctly diagnose where failures are happening, the failure detection service operates in the management plane of GridStat. This service is responsible for receiving all of the failure events from the HBFD and RBFD monitors. Once a failure has been found by a monitor, a failure event will be sent to the service to diagnose the root cause of the problem. The technique developed in this work to diagnose failures, is known as suspect graphs. This technique of suspect graphs is used to efficiently coordinate the information arriving from the different types of failure monitors, while still allowing the monitors to detect their unique events in the data plane itself. In essence, suspect graphs are a bipartite graph of links between a set of monitors and a set of components being monitored. When a new failure event arrives, the graph is used to quickly identify components that are related to the event and determine if enough evidence to indicate a failure has occurred.

This chapter will outline examples of failures in GridStat to demonstrate design requirements for suspect graphs, and describe the details of how suspect graphs are used to locate failures in the data-plane.

5.1. EXAMPLES OF EVENTS GENERATED BY FAILURES

First, consider the following conceptual examples of how suspect graphs find responsible components given failure events. This will provide an understanding of what suspect graphs are trying to achieve.

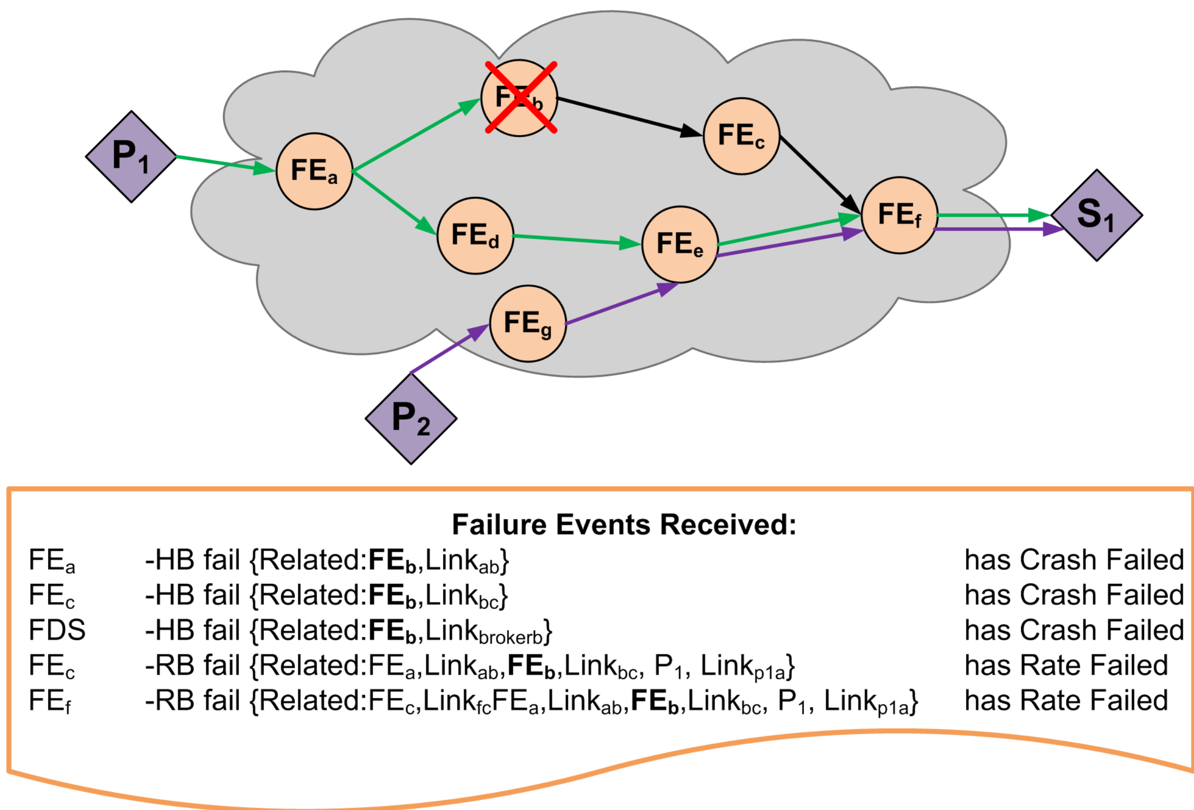

Figure 5.1. Suspect Graphs crash failure example

5.1.1. Crash Failure.

Given a data plane setup shown in Figure 5.1, we will examine what happens when FE_b crashes.

Note that FE_b is being monitored by several different components. Heartbeat monitoring is being done by FE_a, FE_c, and the *failure detection service*. On top of that, the subscriptions through FE_b mean that FE_c, FE_f, and SUB_1 are monitoring the rate of subscriptions FE_b touches. This means that, optimally, all of the events in the bottom of Figure 5.1 can arrive at the failure detection service in any random order.

In Figure 5.1, notice how the events that arrive at the failure detection service

all have related components to each of the events. Exactly how they are determined to be related is covered in a later section. It can be shown at a high level that the likely suspect for these events is a common component related to all of the events, which in this case, is FE_b. The failure detection service should determine that FE_b is the most likely component for failure and mark it as suspect.

5.1.2. Rate Failure.

In the previous crash failure example, it is possible to see how heartbeats are most effective at quickly identifying a small number of suspicious components. Now consider cases where the component has not crash failed. Looking at the previous set-up in Figure 5.2, assume instead of FE_b crash failing that FE_d starts sending packets too far apart and causes a rate failure. The set of possible events to arrive at the failure detection service are in the bottom of Figure 5.2.

In this case, there is not a final graph of just one suspect in the end, because FE_d, FE_a, PUB_1, and the links between them are all potential suspects in all of the rate failure events. This is a possibility to happen in rate-based failure detection because monitors can only truly monitor individual flows from the receiving end. Thus, it is difficult to conclude anything more specific than deducing that the failure is closer to the publisher than the monitor.

In the case of multiple potential suspects, the best strategy for the failure detection service is to mark all of them as suspect. This is done in order to ensure that the failed component is at least marked suspect once, because it is better to mark a correctly working component as suspect than leaving a failed component as a non-suspect. A false negative indicates a failing component that has not been identified.

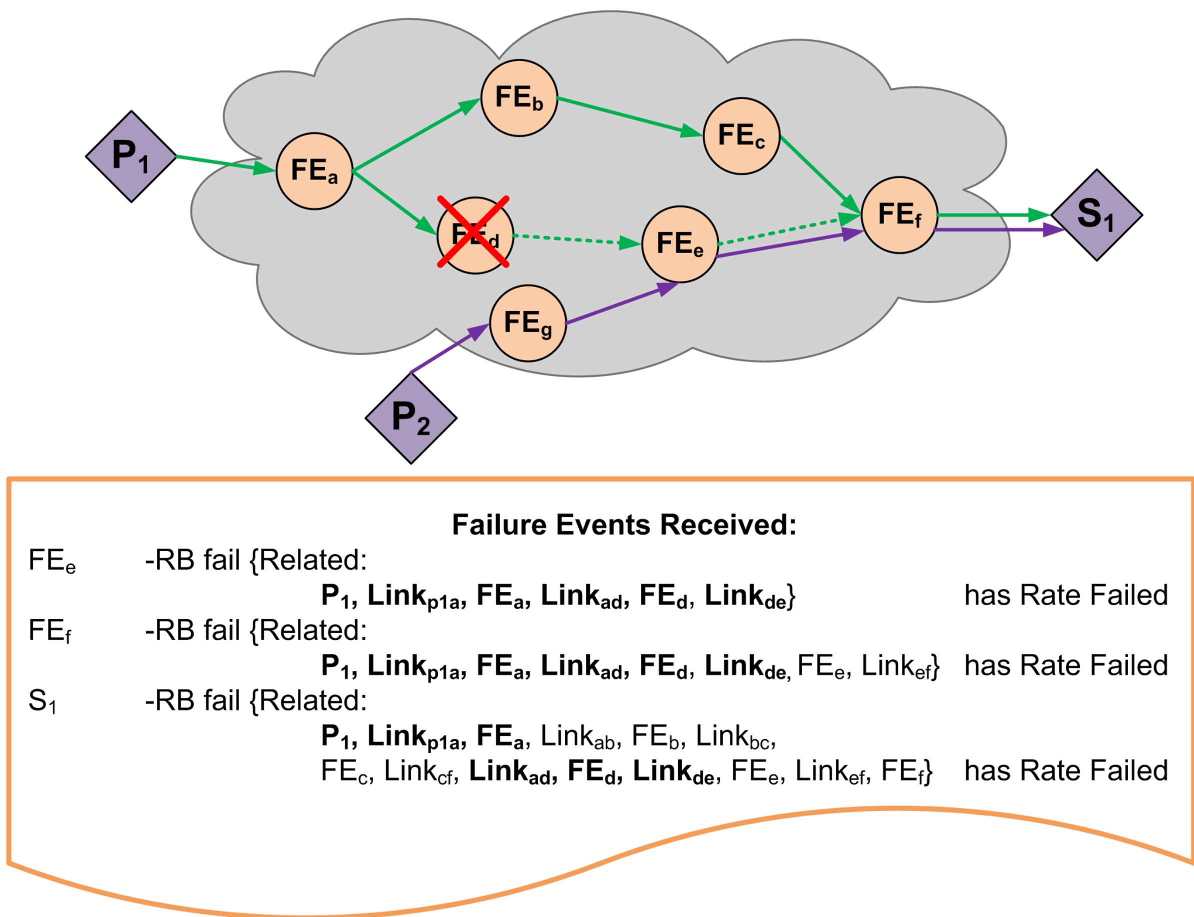

Figure 5.2. Suspect Graphs rate failure example

This can cause undetectable problems for the system, which is why it is a better trade-off to cause more false positives.

Conceptually, if a failure happens closer to the publisher, there are likely fewer false positives, and if the failure is closer to the subscriber there are likely more false positives. This is because failures closer to a subscriber are likely to have fewer FEs monitoring the rate of subscription flows, with a larger pool of potential causes of failure.

This issue of dealing with a potentially large pool of distant components for a monitor is a difficult design issue for suspect graphs to work around. Fortunately, there are a couple factors that alleviate the large number of false positives seen. The first, is that in environments with subscription flows running in multiple directions, it is possible to have monitors watching components on multiple output links. A FE near a subscriber in one flow could be close to a publisher in another. The other way the issue is alleviated, is through the design of suspect graphs by applying the weights added to components connected to a rate based event only to nodes that are within a configurable distance from the monitor itself. This distance from monitor concept is covered in a later section.

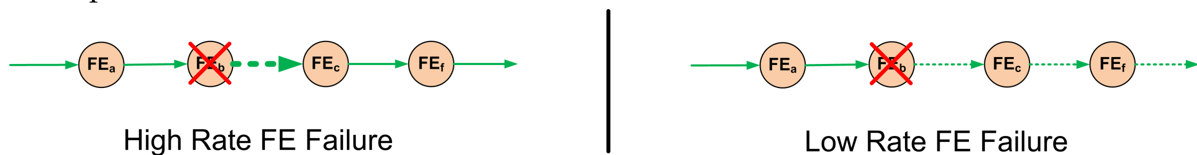

Figure 5.3. Left: an example of a high rate failure not being forwarded. Right: an example of a low rate failure being propagated to the next FE.

Another important observation of rate failures in GridStat, is that the spread of low rate failures and high rate failures are different. A high rate failure will actually stop at the first FE it encounters, while a low rate failure will continue down a flow until it reaches a point that redundant paths meet. High rate failures are stopped by Gridtat's rate dissemination feature. Recall that subscribers do not need to receive flows at the same rate as publications can go, and that FEs will only send as much data as is requested by the subscriber QoS. This means that even if a FE received a high rate flow, it will not send forward the high rate to the next FE. However, low rate failures are not avoided by this, and will be seen throughout a flow from the origin of the failure down to either the meeting of redundant paths or the subscriber.

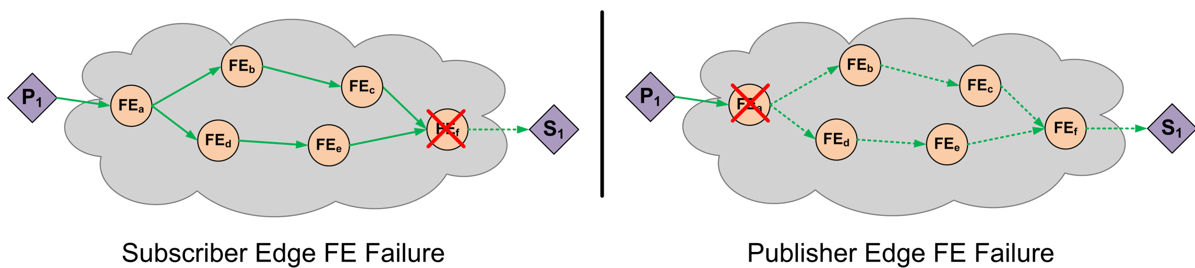

Figure 5.4. Left: an example of a rate failure from a subscriber edge FE. Right: an example of a low rate failure on a publisher edge FE.

An example where rate failures are very difficult to diagnose can be seen in Figure 5.4. These failures are particularly hard to solve due to the failure originating at an edge FE, either connected to a publisher or subscriber. In the subscriber case, this is difficult due to the subscriber being the only rate monitor for the FE. The failure detection service may receive an event from the subscriber, but the single event will have to suspect several FEs. The more apparent failure is a low rate failure on a publisher edge FE, in which a low rate failure will exist for all of the flows for that publisher. In both cases, the difficulty comes from the failure being at a location where there is no redundancy. Fortunately, this is a known issue for GridStat and in the future edge FEs will allow for redundancy by allowing publishers and subscribers to connect to multiple edge FEs.

5.2. ARCHITECTURE OF SUSPECT GRAPH

The suspect graph is essentially a large relationship graph for the domain, contained in the failure detection service, that maps monitors to components. Recall that the relationship graph is based on the risk modelling failure detectors scheme. The

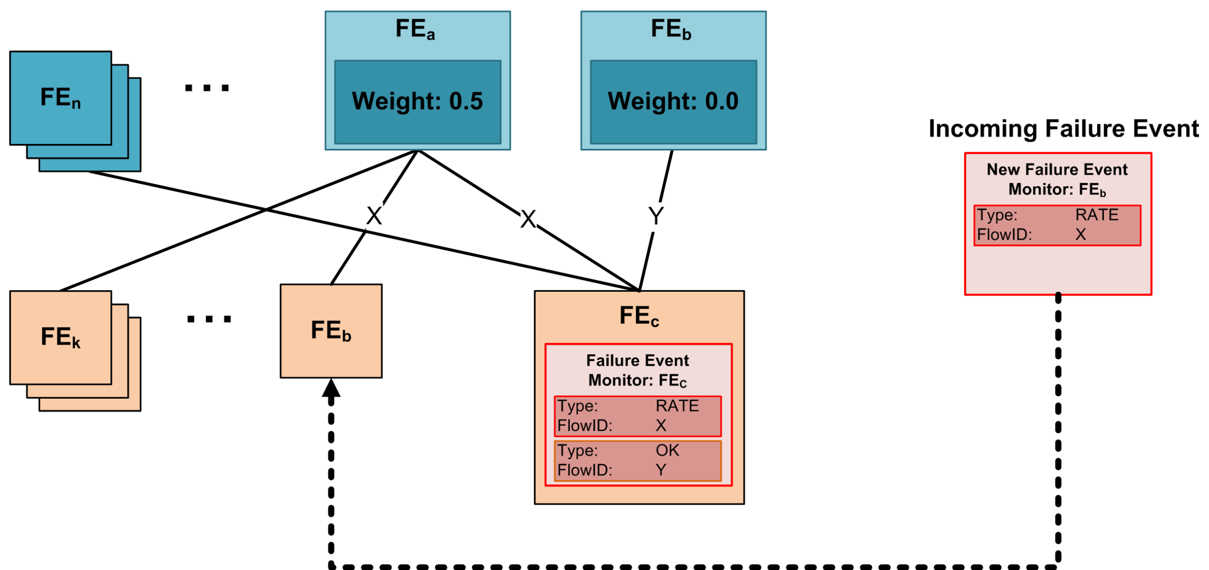

Figure 5.5. Suspect Graphs data structures

relationship graph for the failure detection service is a bipartite graph mapping the relationships between the monitors and the components that can fail. An example of the data structures can be seen in Figure 5.5, where the bottom row of nodes in the figure are the monitors and the top row are the components. The component nodes in the graph contain basic status information such as whether the component is suspect, and a value for how much suspicion has been cast on that node. Monitors in the graph act as keys to look up the components quickly using a failure event, and contain the last event received by that monitor.

Some additional information is embedded into the links of the relationship graph in order to aid with eliminating false positives. Recall that the failures reported from a FE for a rate-based event are aggregated. In a situation where a RBFD is receiving more than one subscription, if only one subscription is failing, then the

subscriber can report that one subscription is failing and one is working properly. This information is be used to eliminate any components in the failing subscription that are also shared with the working one.

5.2.1. Relationship Graph Construction.

The initial construction of the relationship graph takes place when GridStat starts. Later modifications to the graph happen when subscriptions are added or removed.

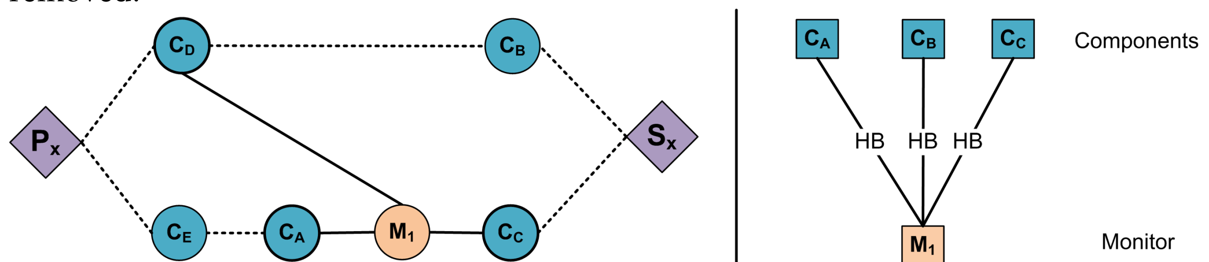

Figure 5.6. A Visualization of what a heartbeat monitor at a FE would be monitoring and what relationship graph would be made by the set-up.

When the GridStat broker starts, it is given a configuration of the FEs that are in its domain. Using this, it is possible to set up the heartbeat monitors in the relationship graph and the FE components in the graph. Since with heartbeats every FE is a monitor, a set of FE monitors and FE components are added to the graph. For each neighbour a FE has, a link is made between that FE as a monitor and the FE component it is linked to.

In Figure 5.6 is an example of how the neighbours of M_1 are added as components to the heartbeat monitor M_1. For the links of the relationship graph, a unique heartbeat flow is given to distinguish it from subscription flows. In the example the heartbeat flow is denoted by HB.

The relationships for rate-based failures are created at subscription time. This

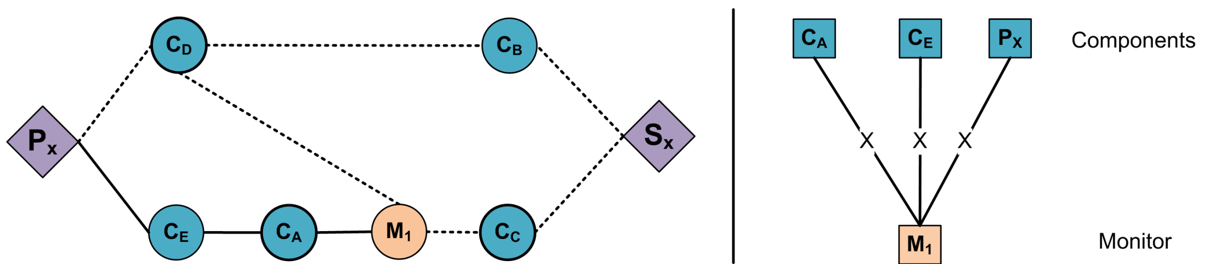

Figure 5.7. A Visualization of what a rate-based monitor at a FE would be monitoring and what relationship graph would be made by the set-up.

occurs when a subscriber requests a connection to a publication, creating a new subscription. While the static routing paths are being allocated, the FEs are looped through from closest to the publisher to next closest to the subscriber for each of the redundant paths. In this loop, it is possible to keep track of the upstream FEs and links that the current FE depends on. This information is kept in the graph by creating a connection between the component and monitor.

For example, when a new subscription is made between P_X and S_X in Figure 5.10, connections are made to show what components the FEs are monitoring the rate of. C_A, C_E and P_X are added as components to the rate monitor M_1 with the flow is denoted by X on the links.

In this loop is where the worst time complexity takes place. When thinking of failure detection alone, it is important to build the relationship graph by looping through each of the components for each of the paths giving a $O(N^2)$ time complexity. However, in order to establish routing paths, it is essential to step through all of these elements anyway, making it more logical to calculate at this time.

The relationship graph in this work only contains components that are FEs. In

future work to expand the failure detection, links as components could be added as well. Note that the only major change to allow links would be to add the links to the relationship graph when it is changed. This can be done trivially in both the initial set-up for heartbeats, and on new subscriptions.

5.2.2. Handling New Events.

The following is how newly arriving events are handled at the FD Service. A new event can either be a new failure, or a recovery event that is needed to remove the failure event. Upon the receiving a new event, the service will lookup the monitor that the event came from in the relationship graph. If the monitor has already reported a failure event previously, then that event will be removed from the weighting in the graph before the new event is added. The previous event from that monitor is then stored in the monitor node of the graph.

For both adding and removing a new event, the procedure is similar. First, the event is used to look up the monitor node in the relationship graph. The next step is to iterate through connected components to the monitor and add an appropriate suspect weight to the component. If the suspect weight of a component rises above a specified threshold, then the component is marked suspect by the failure detection service. Once finished iterating through components, the new event is stored with the monitor node for future updates from the same monitor.

As mentioned previously, when adding weight or removing weight from a component, a check is made to see if it crosses over a threshold limit for failures. If the threshold is crossed, then the component changes state from non suspect to suspect (or vice versa). The standard threshold is set to the value of 1.0 currently. The value of the threshold could be set to something else, but it is conceptually easier to think

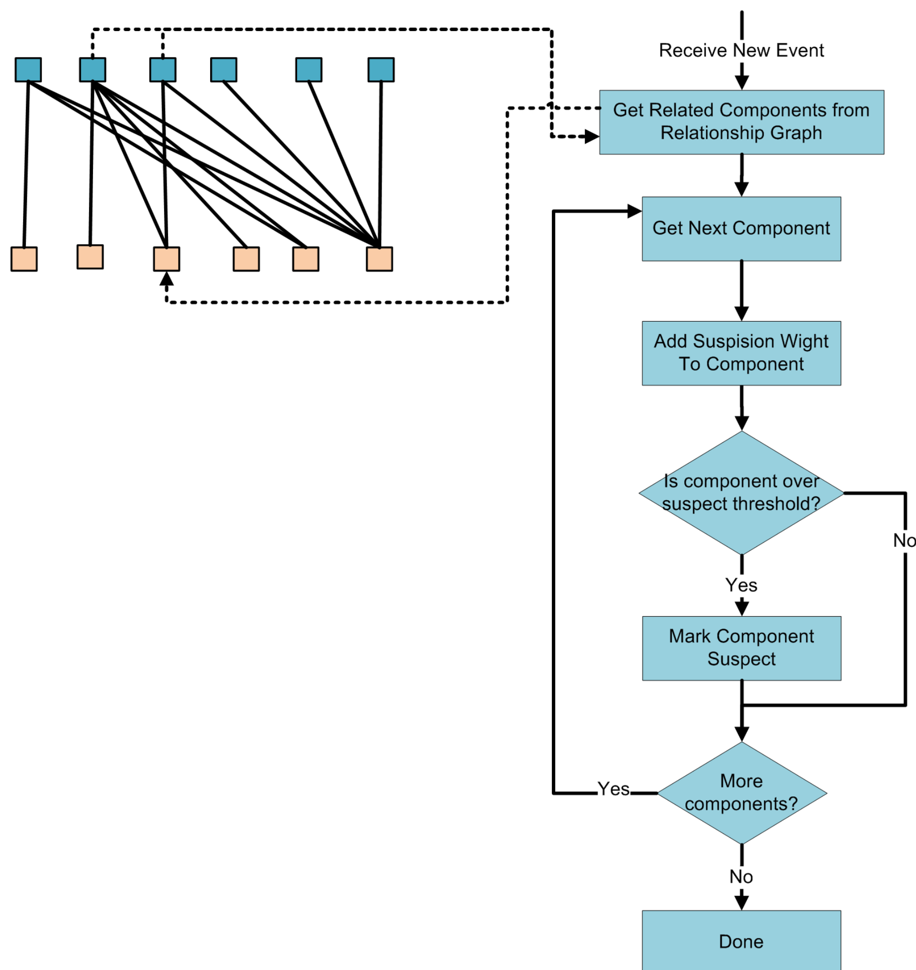

Figure 5.8. Logic for handling a new failure detection event from a monitor.

of the threshold as 1 and have the weight for events be between 0 and 1.

As far as computational complexity goes, handling new events is optimized to be the fastest step. Upon receiving a new event, the only major work to be done is iteration through components related to the monitor, and potentially marking new suspects in that loop. This makes the addition of new events scale based linearly on the number of components connected to a monitor. The number of subscriptions

and disjoint flows directly effect the number of components linked to a given monitor. With more subscriptions and flows, it is more likely that the relationship graph will be very connected, giving many components to iterate through for each event. Roughly estimating, we can see that each new event experiences $O(\hat{c})$ computational complexity, given on average $\hat{c}$ connections per monitor. In the worst case scenario, all of the components are connected to all monitors.

To alleviate the computational complexity even more, unique approaches could be explored in future work. For instance, if instead of monitoring every subscription flow in the network, a select subset of flows in the network could be chosen to monitor instead. The selection could cover most or all of the FEs in the network, thus covering all the components anyway. Additionally, the subscribers on the edges of the network could be monitoring all flows, and when failures begin to be detected, the FEs could dynamically start monitoring flows related to the current failing event. This approach can be explored in future research.

5.2.3. Weights for Events.

The weight applied to a component by a new event is determined by the type of that event. For example, if the new event is a heartbeat failure failure, then the weight is 1/2. This is simply because once there are two heartbeat failures for the same component, it becomes clear that it has likely failed. Therefore, the goal is for any two heartbeat events to reach the failure threshold.

For rate-based events, the weight is a complex issue. Ideally, the goal is for rate-based events to have similar weight as heartbeats. Then, if two rate monitors claim a node is failed, it can be determined that it is likely the QoS failing.

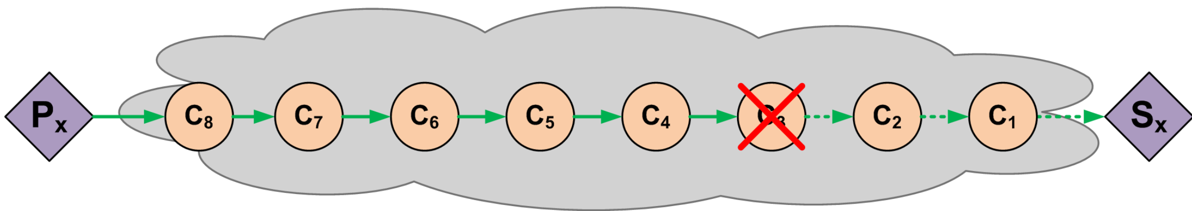

Figure 5.9. Example of rate failure near a subscriber.

However, take into consideration monitors close to the subscriber. If two failures happen close to the subscriber, the majority of the FEs in the system will be marked as suspect. Figure 5.9 is an example of this scenario. This is unacceptable, thus the weight for FE failures is less the further the component is from the monitor. To alleviate this, the weight from a rate-based event must be limited in a way that prevents all the components earlier in the path from being marked as suspect. This can be done by limiting the distance the components are from the monitor that can be marked. This distance is a configurable option, but should be explored in further work for what an ideal distance is. In this work the distance used was four hops away from the monitor.

5.2.4. Fault Tolerance.

Just like the data being forwarded by GridStat, there is no true guarantee that failure detection will operate correctly all the time. In this case, the failure that could harm the failure detection service is dropping or delaying failure monitor events on their way to the service. Because of this risk, suspect graphs were designed to tolerate some loss when it comes to failure event messages.

When a loss of an event happens, the suspect graphs will still be able to operate normally, assuming that there are enough monitors for a component still functioning

correctly. For heartbeats, this was shown to be true in most cases because of FEs having two neighbours and the broker acting as three separate heartbeat monitors. For rate-based failure detection, this is very dependent on the distance monitors are away from the componentsthey are connect to in the relationship graph, and how the weight for rate-based events is determined. In any case, the suspect graph is considered to be loss tolerant if it will not incur a false negative with missing events.

5.2.5. Multiple Failure Scenarios.

Multiple component failures can be categorized with two main distinctions: either independent or dependent, depending on whether they are both on the same flow. Independent failures occur when a set of random failures exist in the data plane but on completely unrelated flows, which is the best case scenario for all the failure detectors as they logically just operate as if they are all single failures anyway. Failures are considered dependent when they occur in locations where flows are related, which makes it very tricky for QoS failure detectors to identify.

When it comes to detecting multiple failures, heartbeat failure detectors actually handle detecting them well. This is because they do not depend on their neighbours forwarding any data. Instead, they only continue to announce they are alive. However, they only identify crash scenarios.

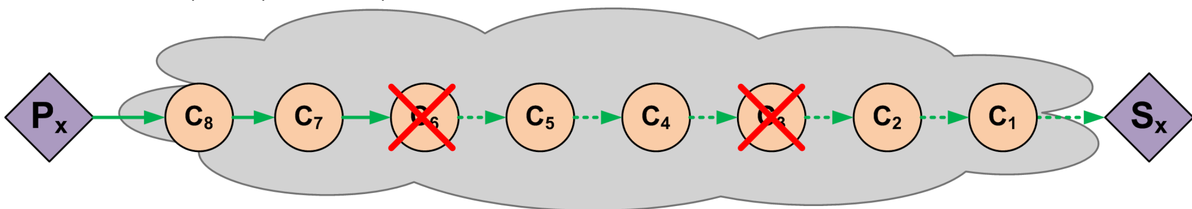

Figure 5.10. Multiple failures on the same flow

Rate-based failures do have a dependency on all of the FEs that sent the flow

to the monitor. This makes related failures tricky, because two failures can occur on the same subscription flow. Observe in Figure 5.10 how two failures on the same flow have the same failure event behaviour as if there was only one. Note that, if there is more than one flow going through the 'hidden' FE, then there could be more monitors to accurately find the failures. If two failures are completely independent when it comes to flows, then rate-based monitors work the same as they did for single failure events.

6. ANALYSIS AND EVALUATION

Multiple configurations of GridStat were run on a DETER testbed in order to observe and prove that this failure detection service detects and diagnoses crash and QoS failures in a way that is scalable and accurate.

To evaluate the failure detection service, specific attributes of the failure detection service must be measured. These attributes include the precision and accuracy of the rate-based failure detection, the capability to diagnose failures quickly, and the capability to scale to sizes useful to power IT networks. This is necessary to show that the service can meet the demands of a large communication infrastructure while remaining as accurate as possible.

6.1. DETER TESTBED

DETER is a cyber-defense initiative in the US. The DETER testbed is a testbed framework build on top of Emulab. The testbed is provided to support security research efforts.

This testbed was chosen for testing because it is isolated from outside world influences, has hooks to manipulate link bandwidths and FE nodes, and is configured into arrangements that are required for testing. Additionally, the DeterLab infrastructure is designed for experimentation on a large scale [4].

6.2. RATE DETECTION PRECISION TESTS

The factors that are critical to any failure detection framework are accuracy and completeness. This service would be useless without a reason to be confident in its results. The failure detection service is run through a battery of tests to observe how reliable the failure detection is. The tests count how many malfunctioning components are missed entirely (false negative) and how many correctly working components are marked as suspect (false positive). The design of the failure detection service with suspect graphs should present a zero count of false negatives. False positives should also be minimized, but not at the expense of adding a any false negatives. With these values, it is possible to observe how precise the failure detection service is. Additionally, it provides a strong argument that the failure detection service is effective at what it does, and therefore useful in practice.

6.2.1. Testbed Set-up.

The goal of this experiment is to examine how the failure detection performs on a simple domain with rate failures. An image of the set-up used can be seen in Figure 6.1. Observe that there are 8 FEs in this set-up. FE_{f1}, FE_{f2}, FE_{f3}, and FE_{f4} are all serving as internal FEs to the domain. FE_{p1}, FE_{p2}, FE_{s1}, and FE_{s2} are all serving as edge FEs that are directly connect to a publisher or subscriber. Note that the edge FEs do represent a single point of failure similar to the publishers and subscribers because of a current limitation of one edge FE per publisher or subscriber in GridStat. A planned change for GridStat will allow support for multiple edge FEs

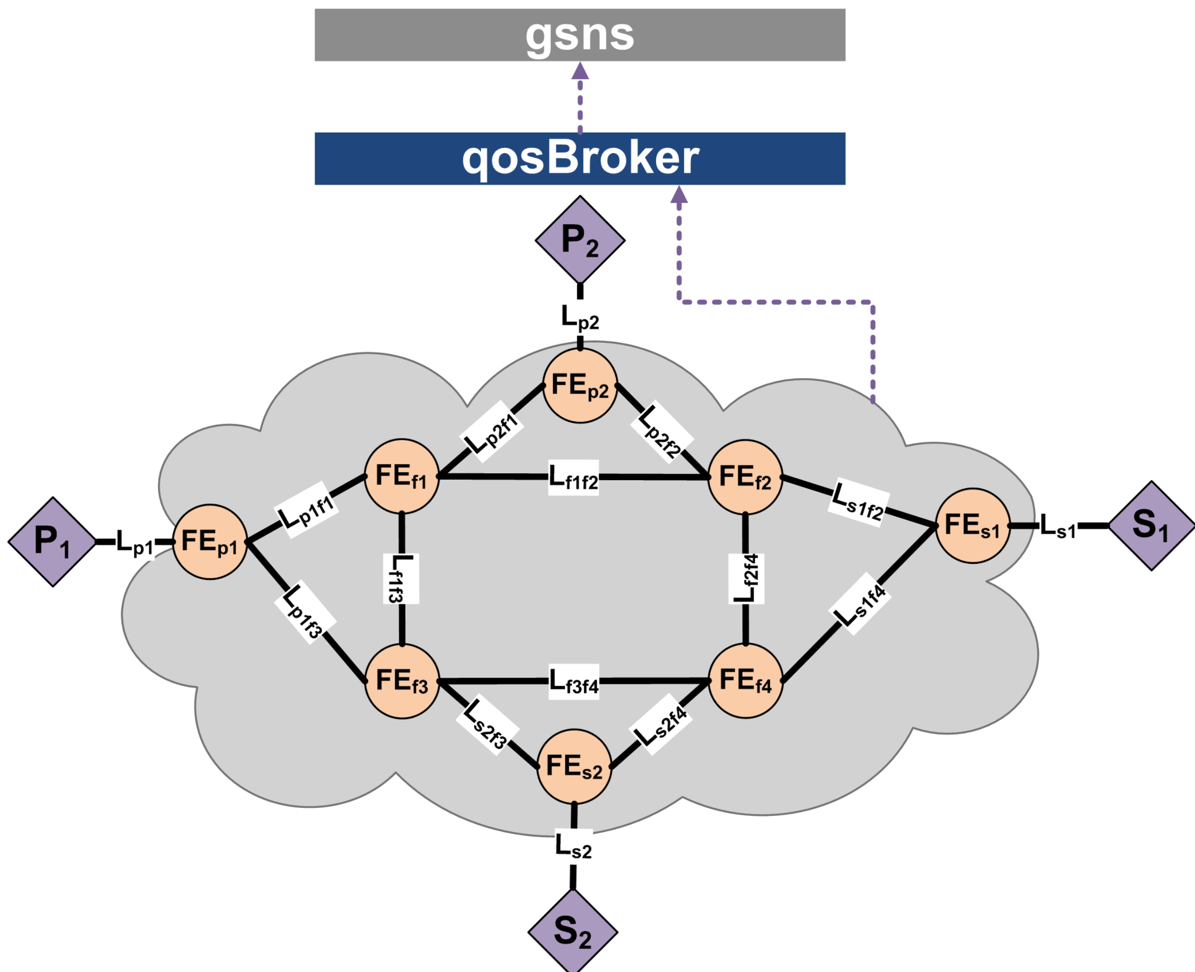

Figure 6.1. DETER testbed set-up for 8 FEs

for a publisher or subscriber. In this testbed set-up, it is possible to have two disjoint paths starting from either of the publishers' edge FE to either of the subscribers' edge FE.

For the two experiments, the subscription set-ups are shown in Table 6.1. A rate of 80 milliseconds was chosen for the rate in order to give some room for a faster rate to be inserted, and to observe how the failure detection service reacts. Also, the time

Table 6.1. Subscription set-ups for rate tests.

Flow Set 1			
Publisher	Subscriber	Rate (Hz)	Disjoint Flows
P_1	S_1	12.5	2
P_2	S_2	12.5	2

between heartbeats and updating the rate-based failure detector is 400 milliseconds.

6.2.2. Procedure. The goal of the rate testing is to see how well the rate-based failure detection operates in difficult to detect situations. In order to do this, these tests set up GridStat in the previously described arrangement (Figure 6.1) and failed each of the FEs in the system independently. To prevent one failed FE from impacting the test of another failed FE, the entire testbed is restarted for each FE failure. In these tests, a rate failure is created by using a backdoor that was created for this work. In this backdoor, the rate that a FE forwards its subscriptions can be changed to a different rate. It should be noted that this backdoor does not exist in the production deployment of GridStat. The rate failures tested were drifts of the rate 0, 1, 2, 4, 8, and 16 milliseconds away from the true rate desired. These tests were run three separate times and results presented here represent an average of those runs.

All of the logging produced by GridStat was stored for each individual test, but an aggregate of all the results was processed in order to gain perspective on how the test set-up performed as a whole. Note that the test automation was run on the same machine as the failure detection service in order to eliminate issues of clock skew in the logs.

Overall, the goal is to measure the accuracy of the failure detection service in

this experiment. As raw data, each experiment measures every instance of a rate-based monitor becoming suspicious of a failure. In post-processing after the experiment, all of the events generated by monitors and the failure detection service are compared to the ground truth of whether that component was failed by the test or not. True Positive, False Positive, True Negative, and False Negative counts will be generated by this post-processing. These counts are taken to calculate accuracy with the following equation:

$$\frac{true\,positive + true\,negative}{Total\,Tests} \tag{6.1}$$

6.2.3. Expected Results.

It is expected that these tests will show that the rate failures are difficult to detect at a point close to the desired rate. At this point the rate-based failure detectors will likely churn detect events, giving many false positive and false negatives.

As the failure rate gets further from the desired rate, the accuracy of the detection and diagnosis is expected to increase to a point. Even in an obvious failure scenario, there will still be some false positives in the diagnosis of the rate failure. This is due to previously outlined limitations in the rate-based failure detection, where it is difficult to pinpoint a failure when there are not enough monitors to gather information from. For example, these situations can occur when the failure is near the subscriber. The benefit of this intended trade-off is that there should be no false negatives in the results.

Additionally, for the 0 rate change run, it is expected to have the fewest false positive detetions. This is due to the system operating in its specified state, and thus no failures should be triggered. This is an important point to have, because the system will be very difficult to trust with constant false positives even in a correct system.

6.2.4. Results.

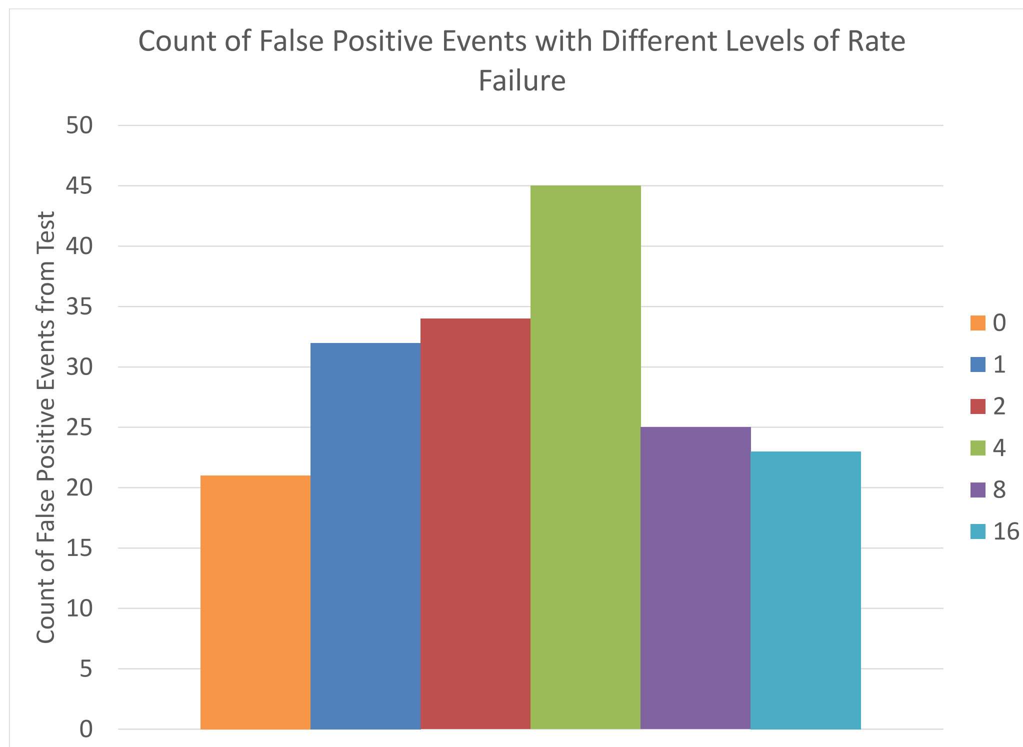

Figure 6.2. Accuracy for Precision Tests

As expected, the results in Figure 6.2 show that there is indeed a spike of false positives when the rate is close to the desired rate. The numbers in the figure represent the number of false positives over five separate runs of the test suite. In the

figure, notice the failure churn point is mostly set around 2 milliseconds. After this point, the false positives become much less frequent. The 0 rate tests were also as expected. The number of false positives that were triggered in the detection are just as low as the false positives for 16 milliseconds. This indicates that when the system is obviously failing or operating normally, the false positive failures are triggered from the same noise. For the false positives that do exist, it is possible that they can be eliminated with further tuning of the bounds and beta values used, which can be explored in future implementation of the failure detection.

Additionally in Figure 6.2, it can be seen that after getting past this churn point, there are still a significant number of false positive events seen in the 4, 8, and 16 millisecond rate events. The number of false positive events is likely to lower the accuracy, as can be seen in the next figure.

Figure 6.3 shows the accuracy values found for detecting and diagnosing a rate failure. Again, accuracy is calculated with the formula in Equation 6.1, otherwise best thought of as the percentage of correct assessments from the failure detection service. The left set of bars represent the accuracy for detection for rate failures at the FEs and the right set show the accuracy of the diagnosis provided by the failure detection service. An accuracy close to one is ideal, meaning that the failure detection is correct almost all of the time.

For the detection side of the results, this looks exactly as to be expected. Specifically, the accuracy suffers when the rate failure is only a few milliseconds away from desired. The detection gets more accurate the further the rate failure is away from the desired value.

Unfortunately, the accuracy for diagnosing a failure appears to be much lower.

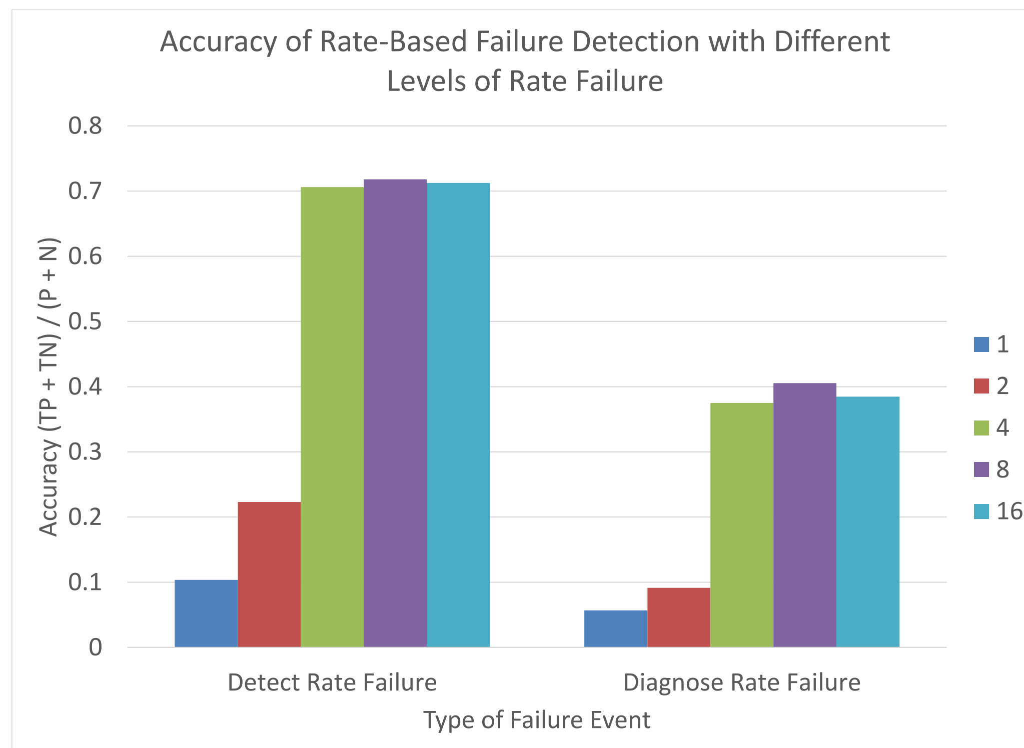

Figure 6.3. Accuracy for Precision Tests

This actually can be seen in the design weaknesses noted in the previous chapter. There is a specific case where the false positives spike in this set-up. When the failed FE is the edge FE for the test, after the rate failure starts, all of the FEs for that subscription will notice a rate failure. This causes the failure detection service to mark almost all of the FEs as rate failed, which makes sense because all of them are reporting failures. Unfortunately, that means for this edge case, one true positive will be caught with four false positives. Additionally, due to the design trade-offs that favor

false positives to false negatives, no false negative results were found in the diagnosis of failures in these tests.

6.3. PERFORMANCE TESTS

Another attribute of rate-based failure detection that must be addressed is how quickly it can diagnose a failure. This is especially important for this work because prompt failure notifications provide ample time to a future adaptation service that can address the problem. Using a set-up similar to the previously described tests, these tests are focused on observing how soon the failure detection service detects and diagnoses a failure after it has happened.

6.3.1. Testbed Set-up.

Just like the previous test suite, this arrangement for the testbed is the same as Figure 6.1. The difference with the previous tests here, is that instead of just one subscription set-up there are several. In Table 6.2, all the subscription rates used to test the failure detection service is outlined.

Table 6.2. Subscription set-ups for performance tests.

Flow Set 1			
Publisher	Subscriber	Rate (Hz)	Disjoint Flows
P_1	S_1	12.5	2
P_2	S_2	12.5	2

Flow Set 2			
Publisher	Subscriber	Rate (Hz)	Disjoint Flows
P_1	S_1	3.125	2
P_2	S_2	3.125	2

Flow Set 3			
Publisher	Subscriber	Rate (Hz)	Disjoint Flows
P_1	S_1	0.781	2
P_2	S_2	0.781	2

Additionally, the time outs between checks for heartbeat and rate-based detectors is varied in this test suite. The time outs used in the experiments are outlined in Table 6.3.

Table 6.3. Timeouts used for performance tests.

Time outs Tested (milliseconds)			
200	400	800	1600

6.3.2. Procedure.

The basic procedure for these test suites is the same as in the precision tests in the previous section. Using the testbed set-up (Figure 6.1), individually and independently fail every FE in the testbed and record all failure events. In addition to the rate failure, each FE is also crash failed in order to ensure heartbeats are also working properly. The rate failure in these tests is set to three times the desired rate. This is due to the point of these tests, which is seeing how soon a failure is detected, not how sensitive it is.

Using these measurements, it should be possible to calculate the time to failure detection and failure diagnosis. Time to failure detection can be found from taking the difference between the test scripts times for starting failures and the failure monitor's times of detection. Note there is a potential for clock skew for this measurement. To prevent this, DETER runs NTP to synchronize time between all machines in the active testbed. Time to failure diagnosis is found from taking a similar measurement. Diagnosis time is not affected by clock skew because the automation running the failures is being run on the same machine as the failure detection service.

6.3.3. Expected Results.

It is expected that heartbeat failure detection will have no dependency on the rate of the subscriptions. The major factor for heartbeat failure detection is the time outs used between heartbeats and the number of missed heartbeats to failure. The same property should apply to rate-based failure detection as well. For instance, the longer the time out, the longer expected time until a failure is detected.

Unlike heartbeats, the rate-based failure detectors will be directly dependent on the rate of the subscriptions. As the subscription rates get larger, the time to detect and diagnose a failure should also linearly increase.

6.3.4. Results.

Figure 6.4 is a simple diagram showing that heartbeats are operating just as expected. The black line on the figure is the time for three missed heartbeats to occur, which is the fastest possible time the heartbeats could detect a failure. All of the different subscription rates fall on nearly the same points, demonstrating that the heartbeat failure detection does not depend on the rate of the subscriptions. As the time outs for heartbeats increase, the mean time to diagnose a crash failure increased linearly with the time out.

The results that show the mean time to diagnose a rate failure, shown in Figure 6.5, are a bit more interesting. In this figure, it can be observed that as the subscription rate increased, so did the time to diagnose a failure. Fortunately, it appears that the relationship between subscription rate and diagnosis time is linear. The figure also shows that as the time out to check rates increased, the time to diagnose a failure slightly increased with it. However, this was not anywhere near the scale as with the

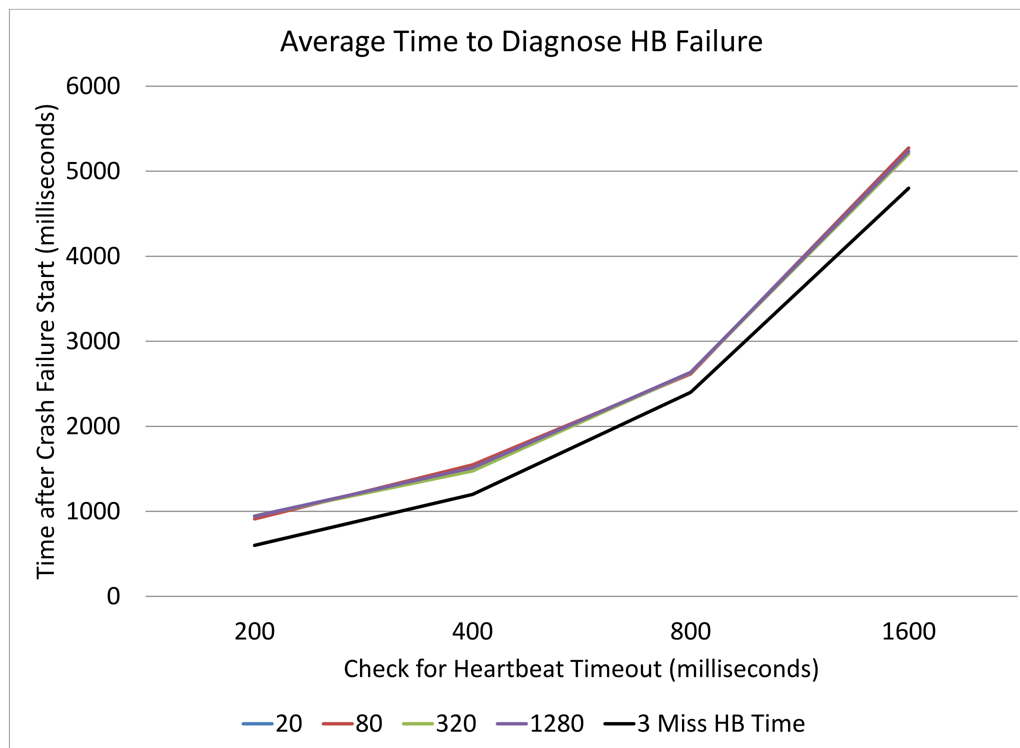

Figure 6.4. Mean time to diagnose a heartbeat failure.

subscription rate changing.

6.4. SCALE TESTS

Scale is a critical factor when it comes to the failure detection service. It must be shown that the failure detection service will both find and detect failures quickly in large infrastructures, and that the failure detection service itself will not limit the

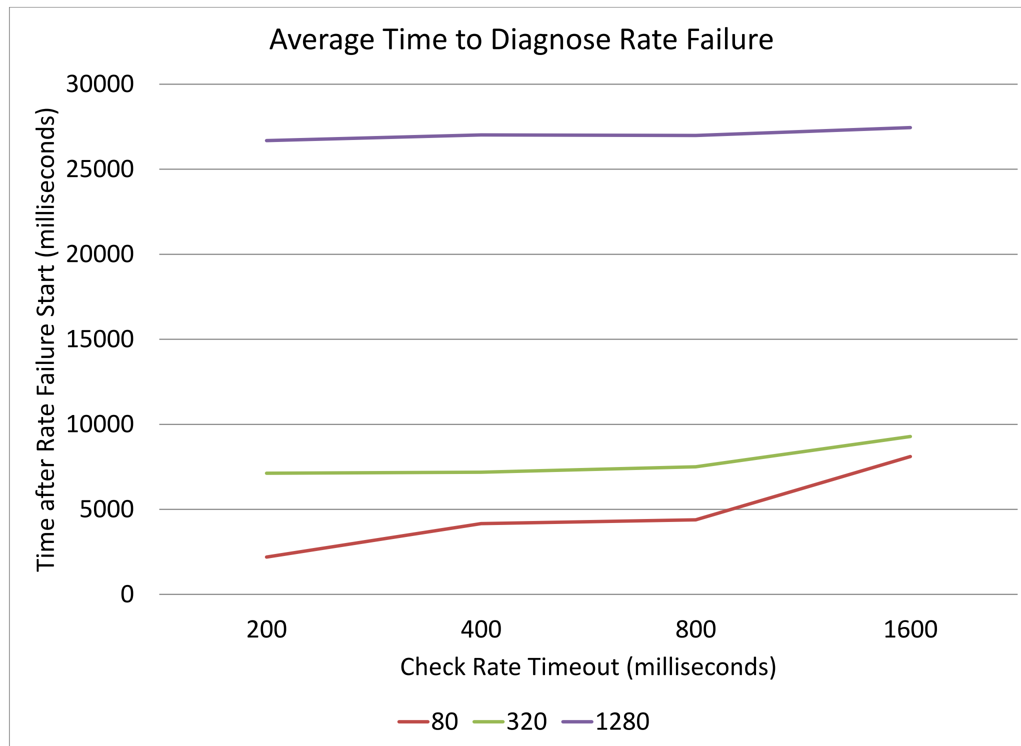

Figure 6.5. Mean time to diagnose a rate failure.

scalability of GridStat. This is because power systems are only going to grow in complexity in the future. This will require the failure detection system to be capable of meeting the scale of communication infrastructures that will be larger than current ones we see now. Failing to show that the system will work effectively for large scenarios makes it impossible to argue why failure detection should be a feature in GridStat.

In order to measure the capability of the failure detection service to operate at

a large scale, the tests will be used to measure the mean time to failure detection, mean time to failure diagnosis, and additional end to end latency of failure detection overhead. The mean time to failure detection and diagnosis for different testbed set-ups will allow the user to examine how the detection time changes as the system has more FEs to account for.

6.4.1. Testbed Set-up.

In this test suite, multiple testbed set-ups are used. The set-up from Figure 6.1 is once again used, along with two slightly larger set-ups. With only a handful flows and 8 FEs, this experiment will provide a strong baseline of what results look like and will be a good comparison with larger test cases to see how manipulating the number of FEs effects the results.

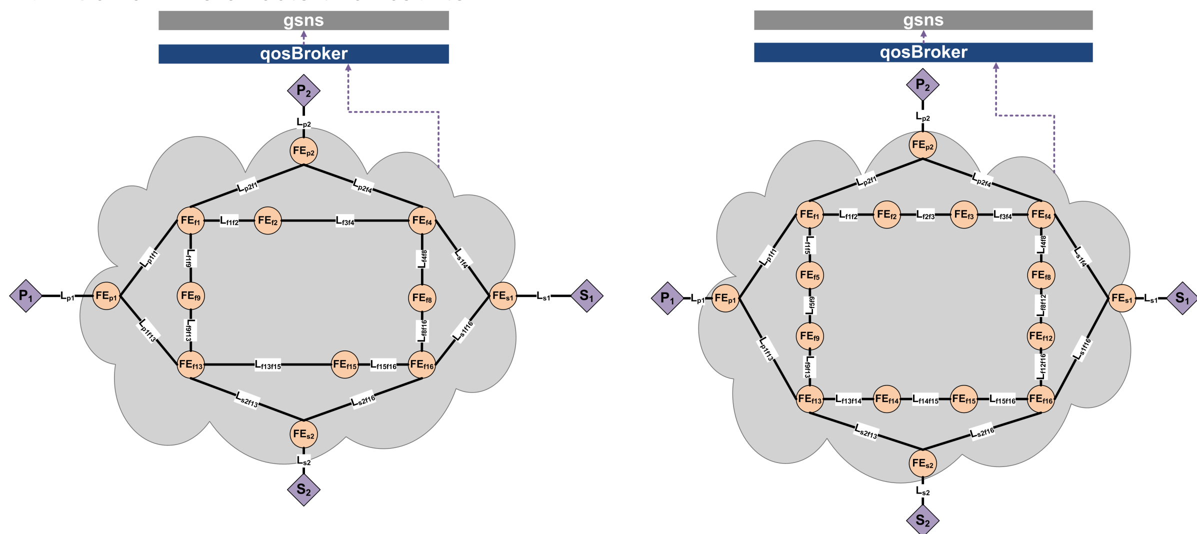

Figure 6.6. Left: DETER testbed set-up for 12 FEs. Right: DETER testbed set-up for 16 FEs.

These new set-ups are very similar to Figure 6.1, but have more internal FEs in both of them. Figure 6.6 shows both of these set-ups. Results between the different

set-ups in relation to the number of FEs will present a measure of the cost of increasing the FE count in a system. Additionally, similar to the precision tests, the rates for subscriptions in this experiment are the same from Figure 6.1 with the same time out for heartbeats and rate at 400 milliseconds.

6.4.2. Procedure.

Using the same basic procedure as the other tests, this experiment individually fails every FE in the testbed with both Crash and Rate failures. The rate failure is again three times the desired rate.

Additionally, a measurement of the end to end latency of subscriptions will be taken from the different scale experiments using failure detection and an instance of GridStat without failure detection. This is an important measurement to show the cost of having the failure detection service operating at all. This will all be done by using a rate of 60 for each subscription to maximize the interference of the Failure Detection's additional work.

6.4.3. Expected Results.

Overall the hypothesis is that while the testbeds get bigger, execution time for detecting and locating a failure will scale linearly based on the number of components.

For the 8 FE set-up, it is likely that everything will perform the shortest amount of time due to not too many flows or components to worry about. Meanwhile, the 12 FE set-up will perform slightly slower than the small set-up, and the 16 FE set-up will be the slowest of the experiments. It is not expected that the additional number of FEs will scale the experiment more than linearly, since additional FEs will only add to the size of the relationship graph.

6.4.4. Results.

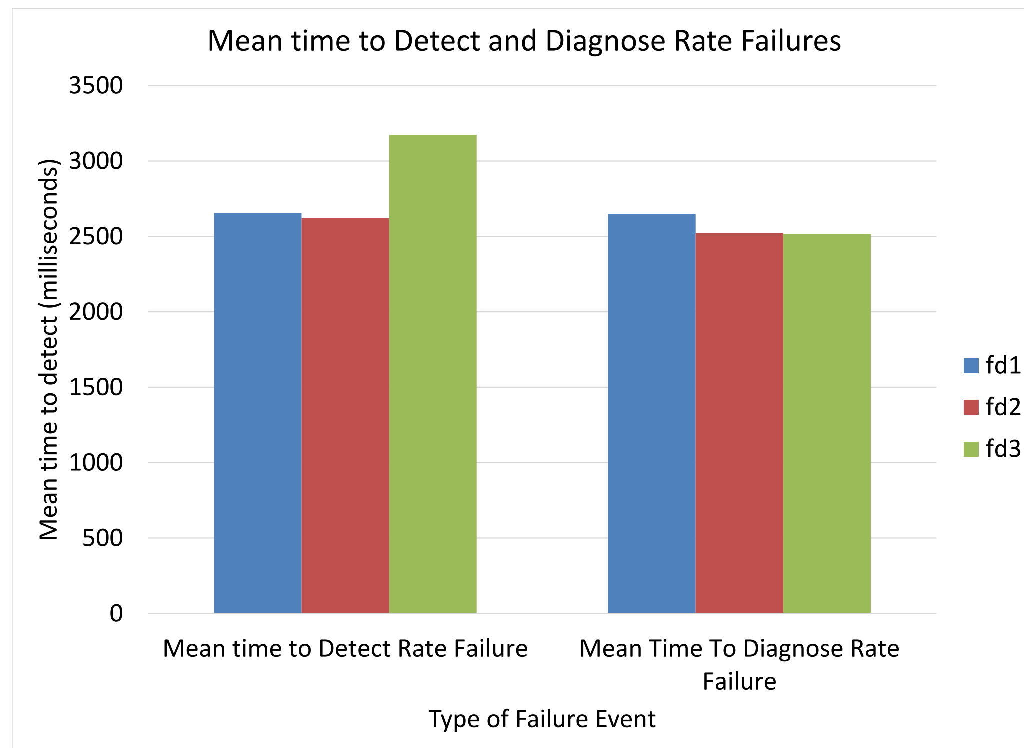

Figure 6.7. Mean time to diagnose and detect a rate failure.

In the results seen in Figure 6.7, the *fd1* results correspond to the original setup used in the previous two test suites. The setups *fd2* and *fd3* correspond to the new 12 FE and 16 FE setups, respectively.

In these results we can see the effect of growing testbed sizes on the detection and diagnosis time for rate failures. The rate at which the detection time grew is a bit more than expected. Though the relationship appears linear, it seems that by adding four additional FEs on two different flows added about a second more time to failure diagnosis. The reason for this, lies in the way the tests were done. Because the results

seen here are the average of all of the events together, we can see that when larger testbed setups were used to test for failures, there were more FEs being affected by the rate failures. Because of this, rate-based monitors that were an extra step away from the failed component added larger detection times to the test, thus skewing the results to be larger than we expected.

The time for diagnosis shows, however, that it does not have a strong relationship with the size of the testbed. As mentioned previously, this result is expected, due to the relationship graph being the main factor impacted by the testbed size. This is a good sign that the failure detection service presented in this work scales well enough to operate well in large setups.

To futher test the scalability of the failure detection in this work, more variations on the testbed should be done. An even larger setup could be attemped with 30 to 50 FEs to truly rule out data-plane size as an effect on failure detetion. Other variables to change and consider include the number of subscriptions, disjoint paths, and domains. These futher variations on the system need to be carried on in future work.

7. CONCLUSIONS AND FUTURE WORK

7.1. CONCLUDING REMARKS

In order to maintain the power grid in a stable state, it is essential to have available and reliable communication of status information. Because of this, there is a need for a reliable system that will detect QoS failures in data delivery. Using GridStat as a base, a failure detection service that is capable of monitoring and reporting these QoS failures has been designed.

To detect QoS failures on status information flows, a rate-based failure detector that is usable in middleware-level forwarding engines has been designed. This failure detector takes advantage of a strictly managed middleware communication system in order to find deviations and monitor flows. The rate-based failure detector uses exponentially weighted moving averages as a rate estimation mechanism, which has previously been used in manufacturing control systems[3]. Through evaluation of this failure detector, it has been shown that monitors at the forwarding engine level are accurate and provide this monitoring a low cost to overall system latency. This has helped determine that exponentially weighted moving averages is a viable failure detector for this application.

Using these lower level rate-based failure detectors, a higher level algorithm, known as suspect graphs, was developed to determine the general locations of these rate failures. By using pre-known mappings of monitors to components, through the strict middleware managment, suspect graphs are able to correlate events in order

to determine suspicious components. However, while suspect graphs are designed to not miss any potential failure, the technique is prone to suspecting several more components as than necessary.

This work is the beginning of development for a self-monitoring middleware service capable of handling the requirements of critical-infrastructure communication. While there are many aspects that should be explored for future research, it has been shown that this service is capable of detecting QoS failures in the data delivery.

7.2. FUTURE WORK

7.2.1. Improvements to RBFD.

With the rate-based failure monitoring using EWMAs, there are several potential areas to improve upon in future research. The first of these areas involves a limitation with identifying rate failures at FEs where redundant paths meet. Note that the rate measurements are only made on the first of the unique packets that arrive for a flow, and not the duplicates. This means that failures in rate must occur on all packets in order to be detectable by RBFD at the subscriber. Currently, this is the best that can be done at the edge FE where this happens, because there is no identifier on the packets being sent that allows the FEs to differentiate which packet came from which redundant flow. An identifier could be added to each flow in order to aid the subscriber in monitoring each flow individually, but the added size to packets is a significant cost that should be explored.

Another area that could be improved, is distinguishing low and high rate failures. Recall that when a high rate failure occurs, FEs will prevent the issue from

spreading further due to rate decimation. However, this makes them difficult to diagnose on their own, in case only one monitor is able to observe the failure. With a distinction of high and low rate failures, this can be taken into account when diagnosed by the failure detection service.

In addition to monitoring the rate of arriving packets, the same EWMA based monitoring could be used to detect latency based failures. With a latency-based monitor it would be possible to detect links that have their delay reaching too high of a value due to network load. The subscriber is also a point end-to-end latency measurements that can be determined to diagnose failures of QoS to the subscriber application. In order to do latency failure detection accurately, it is important to assume that the clocks of the publisher and subscriber are synchronized. This can be achieved in the power grid through use of synchronized GPS clocks for PMUs, relays, and other power equipment.

Another problem that can be seen in this work is related to the performance of detecting rate failures. The issue is when a low rate failure occurs, the rate-based failure detection will not quickly detect it due to the failed rate being longer than the timeouts of the monitor. If this type of situation occurs, then evaluating the EWMA for rate between packets cannot occur on every timeout due to lack of information. A future fix to this is to instead evaluate the rate between the last packet received time and the time of the timout. Given that the timout is larger than the desired rate, this will still be large enough to trigger a failure event at the monitor, and greatly increase the performance of failure detection in the system.

7.2.2. Improvements to Suspect Graphs.

Further work to improve suspect graphs have the potential to increase its accuracy through a variety of methods.

Future research is needed to explore a tradeoff between handling multiple failures and suspecting every possible component in the flow of a rate failure. The root cause of this, is that rate monitors are only watching everything upstream from them, which has a potentially large pool of potential suspects. In a properly operating system, every rate failure detector downstream of the true failure would report an event. The problem is that doing this will cause much of the system to be suspicious for no reason. Because of this, there is a potential advantage to be gained by grouping failure events together in order to narrow down related events to a single suspect. This method of correlating failures was avoided in this work, due to the fact that in a multiple failure scenario it is possible malfunctioning components would be unsuspected. In future work, there is a potential for more analysis and testing of multiple failure scenarios with grouping of related events and comparing results against the non-grouping method presented in this work.

Another accuracy related piece of future work for suspect graphs is further research into the weights rate-based events should have. In this work, the method of making setting weights for rate-based failure events to 1/2 was chosen due to it being simple and having predictable behavior. However, as pointed out previously, this method also generates more false positives than necessary and does not take into account the distance of the monitor from the component being monitored. More work needs to be done to explore changing the weight of suspicion based on important factors, such as distance from the monitor, number of events required to be suspicious,

and number of monitors available for that component.

As noted previously, link failures could also be monitored by using suspect graphs. By adding links to the suspect graphs when GridStat starts, or subscriptions are added, it is possible to monitor for link failures as well. Detecting rate failures on a link could prove to be difficult due to rate measurements being done on the receiving end.

Another improvement to suspect graphs would be the addition of events from multiple GridStat domains. Recall that, this work was focused on failures in a single domain. For future work, another improvement similar to link failures is to add communication between multiple domains of status to coordinate failures that affect multiple domains.

The fault tolerance of the suspect graphs could also be examined in further work. More proper analizing and experimenting with fault tolerance could reveal techniques that would improve the resiliency of the failure detection service. For instance, it may be advantageous to implement a method to have the failure detection service poll for FE status occasionally to catch any lost failure event messages.

Finally another future work point has to do with the event driven nature of suspect pools. Since the failure detection service is only getting events when monitors send them, it is possible that some events are dropped on the way. Because of the emphasis on no false negatives in this work, this means that design decisions had to be made that ensured more false positives happened. A technique that would greatly reduce the number of false positives is a polling of status from each of the monitors from the failure detection service. Given a continuous poll of information, it would be grounds to assume that if an event is not triggered from a FE, then everything

is operating normally. This greatly reduces the number of false positives that could exist for a rate-based event, due to being able to rule out components that do not have monitors one hop away from them indicating a failure. This method will be explored in future work to increase the accuracy of the failure detection service.

7.2.3. Adaptation Service.

The goal of this work has been to create a reliable failure detection service, in order to move GridStat into a more failure tolerant and self-adaptable system. What this work provides is self-awareness that monitors, detects, and diagnoses failures. In the future, more work will be done to decide on actions to take when failures occur, and even act on them. Example of this could be re-routing a redundant path for a failed link, or shed low priority subscriptions to aid an overused part of the network.

1. APPENDIX: USE CASES

This work directly targets protecting the data forwarding IT infrastructure from malfunctions and attacks. In defining how to protect the data plane from failures several use cases were defined of possible failures that may occur in the data plane. The individual failure use cases are shown in the following pages.

In essence, the use cases are the combination of types of failures being protected against (rate and crash failures), type of failed component (link or FE), and the location of the crashed component (border, edge, or internal). This combination of possible failures make up what can fail individually in the data plane, and are the cases that the failure detection work in this paper are protecting against.

Each of the use cases outlines a specific failure scenario that was identified for GridStat's data-plane. In these cases the description, actors, assumptions, logic, and known issues are outlined. The *description* is a one line explanation of the scenario the use case is representing. *Actors* idenfify the key players for identifying and diagnosing the failure. *Assumptions* about the system and how it is operating are outlined in their own section. The logic of handling the failure is covered in the *Input*, *Logic*, and *Output* sections, with the required information, handling of diagnosis, and results delivered are handled respectively. Known issues are outlined in the *Issues* section, with *Security Issues* separating potential secruity flaws that are exposed by the use case's failure. Finally, *policies* describe the potential configuration values that could be set on the failure detection by the operators of the system.

Link Failures

1. Link failure within a domain

Description	A complete failure by a link to forward any traffic through itself. This link is located between two FE's that are in the same domain.
Actors	• Domain QoS Broker • Forwarding Engine A • Forwarding Engine B • The FE neighbors of A • Link AB that is in the domain controlled by the Broker
Assumptions	• A, B, and the rest of A's neighbors have a way to communicate with the QoS Broker. • The QoS Broker has complete knowledge of the graph of the domain it controls. • All actors are trusted, will do what they are supposed to.
Input	Heartbeat messages between A, B, and the Broker, known topology of the domain in the Broker.
Logic	**Problem** • Link AB fails to forward any messages **Diagnose** • Both A and B detect that they cannot hear from each other (using heartbeats) and send notifications to the QoS Broker. • The QoS Broker is notified by either A or B's message (whichever is first). • Say that A's message arrives first, then the QoS Broker will ask B and the rest of B's neighbors if they can hear from A. B will respond to the Broker that it cannot hear A, and the neighbors will respond that they can. • The QoS Broker will use this information to determine that the issue is Link AB and try to resolve the problem. **Act** • The QoS Broker will log the link issue and attempt to resolve it by rerouting messages around the link failure (this can only be done if the QoS of the subscriptions permits it)
Output	New routes for subscriptions, notifications to subscribers (could be in another domain) and notification to domain administrator of link failure.
Issues	• If rerouting fails and the link was the only redundant path for that subscription, then the subscription fails until an operator can repair the link problem.
Security Issues	• The message that the FE's send to the broker could be intercepted by a malicious node and dropped, effectively stopping the adaptation from happening. (Denial of Adaptation Service – DoAS) • Say again that A's message that it can't reach B is sent to the Broker. If A is lying it could be trying to waste the broker's resources by saying there is an attack when there isn't one. • Same as above bullet, but if B was the untrusted node. It could say it reaches A just fine even though it can't.
Policies	• How many dropped messages before notification of issue to broker?

	• When a link fails, how to act? Does the Broker kill the Subscriptions for that path or Reroute them?

2. Link failure between domains

Description	A complete failure by a link to forward any traffic through itself. This link is located between two FE's that are in different domains
Actors	• Domain A • Domain B • QoS Broker for domain A, QBA • QoS Broker for domain B, QBB • Parent QoS Broker for QBA and QBB called PQB • link AB between domains A and B • Forwarding Engine FEA • Forwarding Engine FEB
Assumptions	• There exist other links between domains A and B • The QoS Brokers communicate with other QoS Brokers through the same physical links as the data plane. • There is a hierarchy of QoS Brokers that has a shared parent Broker between A and B. That has knowledge of the inter domain subscriptions for link AB • All actors are trusted, will do what they are supposed to.
Input	Heartbeat messages between FEA, FEB, and their respective Brokers, known topology of the domain by QBA, QBB.
Logic	**Problem** • Link AB fails to forward any more messages between domains **Diagnose** • FEA and FEB detect that they cannot hear from each other (using heartbeats) and notify their respective QoS Brokers. • Both QoS Brokers will notify their parent Brokers (and parents their parents) until a PQB is found and is notified of the link issue. • When PQB receives the first message it will ask the QB it did not receive a message from if it can hear the complaining QB through the link in question. • The other QB will respond that it cannot hear through the link either. **Act** • PQB, QBA, and QBB will all log the link issue and PQB will resolve it by rerouting messages around the link failure (possibly could even try to route around domains A and B all together).
Output	New routes for subscriptions, notifications to subscribers (could be in another domain) and notification to domain administrator of each QoS Broker of link failure.
Issues	• If rerouting fails and the link was the only redundant path for that subscription, then the subscription fails until an operator can repair the link problem. • Assuming the Hierarchy for now, but how will we will likely need to handle agreement between domains on new routes in a P2P fashion later. • Subproblems: Subscribers for subscription in the domains A or B, OR subscribers are in other domains.
Security Issues	• Any message that the FE's send to the broker could be intercepted by a malicious node and dropped, effectively stopping the adaptation from happening. (DoAS attack) • If one of the FE's are compromised then one of the two Brokers will still report to the PQB. The PQB will see that one Broker is reporting issues while the other

	is not. Who to believe? • If one of the domain QoS Brokers are compromised then the uncompromised one will still report up to the PQB, the PQB will see that one Broker is reporting issues while the other is not. Who to believe? • If the Parent QoS Broker is compromised then an attacker has control of a single point of failure. This is one of the issues with a hierarchy structure.
Policies	• How long before FE's start to complain about issues? • In the P2P Management plane, which Broker is responsible for a given Subscription? The one that the subscriber is in? Publisher is in?

3. Link failures to a Subscriber or Publisher

Description	A complete failure by a link to forward any traffic through itself. This link is located between a FE and a subscriber.
Actors	• Domain QoS Broker • Forwarding Engine A • Subscriber or Publisher S • Forwarding Engine Neighbors of S • Link AS that is in the domain controlled by the Broker
Assumptions	• All the actors in this case can be trusted.
Input	Heartbeat messages between A, S, and the Broker.
Logic	**Problem** • The link AS fails to forward any messages **Diagnose** • First the FE A linked to S will notice a lack of heartbeat messages from S. A will then send a notification to the Broker that it cannot reach S. • The Broker will check if it can reach S and if S can still hear the FE. • S will respond, but can't contact A. The Broker will ask S's neighbors if they can hear from s. • S's neighbors will let the Broker know that they can reach S **Act** • The broker will attempt to reroute the path that was traveling over the failed link. A log will be made of the issue.
Output	Rerouting of the one redundant Subscription to/from S. A log of the link failure.
Issues	• Note that this is almost exactly the same as a link failure in the domain. • This combines both Publisher and Subscriber, is there anything that may need to be handled in one of the specific cases?
Security Issues	• What if S says it can't hear from A, but A says it can hear from S or vice versa? • This adaptation could be trouble from a security perspective if a malicious attacker can imitate the FE saying it can't reach S and block any communication coming from S, the Broker would be tricked into believing that the Subscriber/Publisher failed.
Policies	• How many dropped messages before notification of issue?

4. Link degradation of QoS

Description	A Link within a domain is not completely failed but has started to noticeably drop packets.
Actors	• Domain QoS Broker QB • Forwarding Engine A • Forwarding Engine B • Subscriber with a subscription on link AB • Link AB that is in the domain controlled by the Broker
Assumptions	• A and B both have a path to the QoS Broker. • The QoS Broker has complete knowledge of the graph of the domain it controls. • All actors are trusted, will do what they are supposed to.
Input	Heartbeat messages from FEs to the Broker that includes information about packet loss rates from neighbors, notifications from subscribers of QoS failure. Complete knowledge of domain topology in the Broker.
Logic	**Problem** • Link AB starts to consistently drop several packets. **Diagnose** • The Subscriber S would notify its QoS broker of the lack of QoS desired. • The QoS broker would request all the links used in the Subscription to increase monitoring to include rates of dropped messages. • The Broker would use its knowledge of the domain topology along with extra monitoring messages from FEs to identify suspect links of degradation. **Act** • The Broker could reroute some subscriptions going through that link through different links, it can also try reducing the rate of some of the subscriptions, or stop some of the subscriptions altogether. • The Broker would also tell the links to stop the extra monitoring of drop rate.
Output	Broker initiates its plan, possibly a notification of change in QoS to the Subscriber or an "I fixed it" message.
Issues	• What if the subscription spans to other domains and the problem is not in the same domain as the subscriber? • The increase of messaging for monitoring drop rates could be taken advantage of by malicious users, they could try to drop all the extra packets. • Could the instrumentation service be smart enough to send out an event when a rate of incoming packets drops unexpectedly? We could use that as an event instead of a Subscriber's notification. • What is the mechanism that we use to see if the link has recovered?
Security Issues	• Attackers could use this mechanism of adaptation to cause the system to unnecessarily restrict itself.
Policies	• Which method of fixing should be initiated? Reduce subscription rates, reroute subscriptions, or stop low priority subscriptions? • How long after 'fixing' the link before we check if it has recovered?

Subproblems: similar to crash failures (between domains, link to subscriber, link to publisher)

Forwarding Engine (FE) Failures

5. FE failure within a domain

Description	A crash failure of any FE in the data plane that is not connected to another domain.
Actors	• Domain QoS Broker QB • Forwarding Engine A • Forwarding Engine B • Forwarding Engine F. • Neighbors of Forwarding Engine F.
Assumptions	• A and B both have a path to the QoS Broker. • The QoS Broker has complete knowledge of the graph of the domain it controls. • F is a Forwarding engine between A and B. (… – A – F – B – …) • All actors are trusted, will do what they are supposed to do.
Input	Heartbeat messages from neighbors to the Broker, known topology of the domain in the Broker
Logic	**Problem** • Node F crash fails. **Diagnosis** • Node A and B detect that they no longer receive heartbeat messages from F, and notifies QB. • Say that A's message with arrive at the QB first, the QB will attempt to send a message to F (which will not get a reply). After a timeout, the Broker will determine F has failed and check this by contacting all of F's neighbors to ask if they can hear from F. If the majority of neighbors of F report they can't contact F then the QB will resolve the issue. **Act** • The QoS Broker will log the node failure and attempt to resolve it by rerouting messages around F (this can only be done if the QoS of the subscriptions permits it)
Output	New routes for subscriptions, notifications to subscribers (could be in another domain), and notification to domain administrator of node failure.
Issues	• If rerouting fails and the node was the only redundant path for that subscription, then the subscription fails until an operator can repair the node problem.
Security Issues	• The messages that the FE's send to the broker could be intercepted by a malicious node and dropped, effectively stopping the adaptation from happening. (Denial of Adaptation Service – DoAS) • Suppose that A's message arrived to the QB first but A is malicious and there is no issue. This could be used to exhaust the resources of the QB in a DoS attack. • What if we can't trust A or B to report the error in the first place? • What if F is only blocking A and B and none of the other neighbors?
Policies	• How many neighbors will create a majority? • How many dropped messages before the FE's neighbors notify the QoS Broker of the issue?

6. FE failure on the border between domains

Description	FE failure on the border between domains – a crash failure of a FE that is connected to another domain.
Actors	• Domain A • Domain B • QoS Broker for domain A, QBA • QoS Broker for domain B, QBB • Parent QoS Broker for QBA and QBB called PQB • Forwarding Engine FEA that is on the edge of domain A • Forwarding Engine FEB that is on the edge of domain B • Link AB connecting domains between A and B
Assumptions	• There still exist other nodes that have a link between domains A and B. • The QoS Broker has complete knowledge of the graph of the domain it controls. • The QoS Brokers communicate with other QoS Brokers through the same physical links as the data plane. • There is a hierarchy of QoS Brokers that has a shared parent Broker between A and B. That has knowledge of the inter domain subscriptions for link AB. • All actors are trusted, will do what they are supposed to do.
Input	Heartbeat messages between FEA, FEB, and their respective Brokers, known topology of the domain by QBA, QBB.
Logic	**Problem** • FEB crash fails. **Diagnosis** • FEA would detect it can no longer hear from FEB and notify it's broker QBA. Similarly to within domain links, neighbors of FEB would notify QBB of the issue as well. • The QoS Brokers would notify parent QoS Brokers of the issue until a common parent Broker PQB was found that is a parent of both QBA and QBB. • PQB, upon receiving the notification of failure, will attempt to confirm the issue by contacting the other QoS Broker and seeing if it has similar problems. • The other QB will respond that it does have a similar problem. **Act** • PQB, QBA, and QBB will log the node failure and PQB attempt to resolve it by rerouting messages around this node failure (by telling QBA, QBB to use a different link between their domains, or just routing through different Domains altogether)
Output	New routes for subscriptions, notifications to subscribers (could be in another domain) and notification to domain administrator of each QoS Broker of link failure.
Issues	• If rerouting fails and the node was the only redundant path for that subscription, then the subscription fails until an operator can repair the link problem. • The message that the FE's send to the broker could be intercepted by a malicious node and dropped, effectively stopping the adaptation from happening. • Assuming the Hierarchy for now, but how will we will likely need to handle agreement between domains on new routes in a P2P fashion later.

Security Issues	- Any message that the FE's send to the broker could be intercepted by a malicious node and dropped, effectively stopping the adaptation from happening. (DoAS attack) - If one of the FE’s are compromised then one of the two Brokers will still report to the PQB. The PQB will see that one Broker is reporting issues while the other is not. Who to believe? - If one of the domain QoS Brokers are compromised then the uncompromised one will still report up to the PQB, the PQB will see that one Broker is reporting issues while the other is not. Who to believe? - If the Parent QoS Broker is compromised then an attacker has control of a single point of failure. This is one of the issues with a hierarchy structure.
Policies	- How long before FE's start to complain about issues? - In the P2P Management plane, which Broker is responsible for a given Subscription? The one that the subscriber is in? Publisher is in?

7. Subscriber/Publisher Failure

Description	A crash failure of a Subscriber or Publisher.
Actors	• Domain QoS Broker • Forwarding Engine A • Subscriber S • Forwarding Engine Neighbors of S • Link AS that is in the domain controlled by the Broker
Assumptions	• All the actors in this case can be trusted. • The QoS Broker knows about the Subscriptions that S is subscribing/publishing.
Input	Heartbeat messages between A, S, and the Broker.
Logic	**Problem** • The Subscriber S crash fails. **Diagnose** • FE A will notice a lack of heartbeat messages from S. They will then send a notification to the Broker that it cannot reach S. • The Broker will attempt to ping S to see if it can get a response. • After a timeout the Broker will check if any other neighbors of S can still reach S. **Act** • The Broker can only act by terminating Subscriptions to/from S because there is no other way to get messages to it. It can allow forwarding to continue for a moment in case S reconnects. The Broker also logs the disconnection of S.
Output	After some timeout period, termination of Subscriptions to/from S. Other domains with subscriptions to/from S are notified.
Issues	• Other domains may need to be notified of the subscription's failure in order to recover allocated resources.
Security Issues	• If a malicious attacker can trick a majority of neighbors to think that S has failed then the Broker would be tricked into terminating Subscriptions. • What if S is malicious and says it can hear from A? How would the Adaptation service determine what is going on?
Policies	• How many dropped messages before notification of issue? • How long after a subscriber is disconnected do we wait before terminating the subscription? When do we free the resources?

8. FE degradation of QoS

Description	FE is not completely failed but has started to introduce high latency and/or drop packets noticeably.
Actors	• Domain QoS Broker QB • Forwarding Engine A • Forwarding Engine B • Subscriber with a subscription through B • Link AB that is in the domain controlled by the Broker
Assumptions	• A and B both have a path to the QoS Broker. • The QoS Broker has complete knowledge of the graph of the domain it controls. • All actors are trusted, will do what they are supposed to.
Input	Heartbeat messages from FEs to the Broker that includes information about packet loss rates from neighbors, notifications from subscribers of QoS failure. Complete knowledge of domain topology in the Broker.
Logic	**Problem** • FE B starts to consistently drop packets and introduce latency. **Diagnose** • The Subscriber S would notify its QoS broker of the lack of QoS desired (by latency or packet loss). • The QoS broker would request all the links used in the Subscription to increase monitoring to include rates of dropped messages for a short period of time. • The Broker would use its knowledge of the domain topology along with extra monitoring messages from FEs to identify suspect links of degradation. **Act** • The Broker could reroute some subscriptions going through that link through different links, it can also try reducing the rate of some of the subscriptions, or stop some of the subscriptions altogether. • The Broker would also tell the links to stop the extra monitoring of drop rate once the problem was diagnosed.
Output	Broker initiates its plan, possibly a notification of change in QoS to the Subscriber or an "I fixed it" message.
Issues	• What if the subscription spans to other domains and the problem is not in the same domain as the subscriber? • Could the instrumentation service be smart enough to send out an event when a rate of incoming packets drops unexpectedly? We could use that as an event instead of a Subscriber's notification. • When do we detect if a node came back online? Is there any attempt to return to the original state if the node comes back?
Security Issues	• The increase of messaging for monitoring drop rates could be taken advantage of by malicious users, they could try to drop all the extra packets, or turn up the monitoring to create DoS. • If attackers can drop all of the packets around a FE to make it look offline then the adaptation could be used as a mechanism to isolate the FEs.
Policies	• Which method of fixing should be initiated? Reduce subscription rates, reroute subscriptions, or stop low priority subscriptions?

Etc.

9. QoS Broker failure

Description	The QoS Broker Crash Fails while its FEs, Pubs, and Subs are actively sending subscriptions.
Actors	• Domain QoS Broker – QoS Broker • Forwarding Engines – FEs • Publishers – Pubs • Subscribers – Subs
Assumptions	• The Forwarding Engines in the domain have a way of detecting an unresponsive QoS Broker. • All of the elements in the system are trusted and will do what they are supposed to do.
Input	Heartbeat messages between FEs, Pubs, Subs of the domain and the QoS Broker.
Logic	**Problem** • The QoS Broker crashes **Diagnose** • FEs detect that they are no longer receiving heartbeat messages from the QoS Broker for their domain. • FEs use a distributed agreement algorithm with their neighbors to agree that the problem is not just them and that the QoS Broker is down. **Act** • The FEs contact an Operator to address the broken Broker. • Subs and Pubs within the domain accept no new subscriptions until they have a new Broker or the old one restored.
Output	Closed state of elements (no new subscriptions/nodes)
Issues	• Could the Forwarding Engines have a way of contacting the control center? How do they call for help if the QoS Broker is down? • Once replication is achieved in the QoS Brokers, this will become less to almost no issue because of the replication.
Security Issues	• An attack on the adaptation system could specifically bring the Broker offline in order to prevent other adaptation mechanisms from being used and new subscriptions from forming (at least in the way it's structured now).
Policies	•

10. Link failure between FE and Broker

Description	The link between a Forwarding Engine and a Broker fails to send any heartbeat or adaptation service messages.
Actors	• Domain QoS Broker, QB • Forwarding Engine, FE
Assumptions	• All of the elements are trusted and will do what they are supposed to do. • FE's have the capability of forwarding messages to the broker from neighbor FE's
Input	Heartbeat messages between the FE, its neighbors, and the broker.
Logic	**Problem** • The link between the FE and the broker fails to send any messages. **Diagnose** • The FE and QB would both independently detect the problem because of the lack of heartbeat messages between them. • The FE sends a new heartbeat through neighboring FEs to the Broker to let the Broker know it is still alive. • The QB would receive the messages and determine that only the link has failed and not the node itself. **Act** • The QB would now send messages for the FE through its neighbors, and notify operators of the issue.
Output	Log of the issue for operators, rerouting for the Broker to the FE and the FE to the broker.
Issues	• Could a FE send a message to a neighboring FE to send to the Broker? Use the other FE to forward a message for it?
Security Issues	• How can neighbor forwarding be used against FEs? It could be possible using a compromised FE to send so many forward messages that it compromises its neighbors QoS. How could we protect against this?
Policies	• How much of other FE traffic to the broker is a given FE going to allow? • The FE with the failed link could alternate between neighbors to send messages to the Broker, how best to do that?

11. Islanded node within a domain

Description	A node becomes islanded due to link failures
Actors	• Domain QoS Broker • Forwarding Engine A • Forwarding Engine B • Forwarding Engine I
Assumptions	• A and B both have a path to the QoS Broker. • The QoS Broker has complete knowledge of the graph of the domain it controls. • I is a Forwarding engine between A and B. (… – A – I – B – …) • All elements are trusted and will do what they are supposed to do.
Input	Heartbeat messages between A,B,I, and QoS Broker.
Logic	**Problem** • A FE becomes islanded from its neighbors. **Diagnose** • The neighbors of the islanded FE will detect link/node failures (see Link and Node failure use cases) **Act** • The Broker will act in the same way as a link/node failure (see Link and Node failure use cases)
Output	New routes for subscriptions, notifications to subscribers (could be in another domain) and notification to domain administrator of link/node failure.
Issues	• (same as Link and Node Failure)
Security Issues	• (same as Link and Node Failure)
Policies	• How long does the disconnected FE wait while disconnected from the network before shutting down?

12. Domain islanded from QoS Broker

Description	The QoS Broker cannot communicate with its domain (due to all links to the domain failing).
Actors	• Domain QoS Broker – QoS Broker • Forwarding Engines in the domain – FEs
Assumptions	• The subscriptions in the domain are already set up. • All o the elements are trusted and will do what they are supposed to do.
Input	Heartbeat messages between the QoS Broker and elements in the domain.
Logic	**Problem** • The QoS Broker fails to communicate with any of the elements in its domain. **Diagnose** • The Domain QoS Broker detects that it cannot reach any of the nodes in the domain using heartbeat messages. • The FEs detect that they no longer receive heartbeat messages from the QoS Broker. • The FEs will react in a way similar to the QoS Broker failing (agree that they can't hear the broker and continue forwarding all current subscriptions), but this case is different because the Broker is actually online still. If the Broker has lost all connections, it is as good as gone, but if it can contact other Brokers then it can give them a message to let them know that no new subscriptions can be made through the domain. **Act** • Notify an operator that they need to repair the links. (for now the data plane can just continue the subscriptions it has)
Output	Log of the problem for the operator. Closed state (no new subscriptions/nodes) of FEs until they have a Broker assigned to them.
Issues	• Could this be prevented if we had a policy that tells to alert operators if the QoS Broker has fewer than *n* links to the data plane? What kind of event would take out all links at once?
Security Issues	• An attack on the adaptation system could specifically bring the Broker offline in order to prevent other adaptation mechanisms from being used (at least in the way it's structured now).
Policies	• Alert operators if the Broker has fewer than *n* links to its data plane.

13. Heartbeat but no forwarding failures

Description	A FE continues to send heartbeats to all neighbors but stops forwarding any subscriptions. This is a potential attack that could be done to a node in the system that would be difficult to detect.
Actors	• The Compromised Forwarding Engine – cFE • Neighboring Forwarding Engines – FEs • The Domain QoS Broker – QoS Broker
Assumptions	• All the other FEs and the QoS Broker are trusted and operate the way they are supposed to.
Input	Heartbeat messages between FEs, Subscriptions being forwarded by FEs
Logic	**Problem** • The Compromised Forwarding Engine sends only heartbeat messages to its neighbors. **Diagnose** • The subscribers of subscriptions passing through the cFE begin to complain that they no longer receive messages. • The adaptation service uses a mechanism in the instrumentation plane to track down which FE is not forwarding the subscription. **Act** • Declare the cFE suspect and route subscriptions around it.
Output	New routes for subscriptions that go around the cFE, a log is kept about known cFEs.
Issues	• What mechanisms are there to diagnose lack of subscription messages? Do the FEs complain or the Subscriber? • What if there is no appropriate way to route subscriptions around this FE? What if it's cross domain? Or the only link to a Pub or Sub? • This is a very targeted attack that would require a complete rewrite of the FE or complicated man in the middle attacks, unlikely that an attacker would be capable of doing this or even prefer this method to just taking the node down.
Security Issues	• Using this adaptation mechanism, an attacker could try to make a FE look suspect by intercepting and dropping its subscription packets, then the adaptation service would bring its own nodes offline.
Policies	• How long after not receiving messages before subscriber starts to complain? • If the subscription is passing through several domains, do we declare the domain suspect or just the FE in that domain?

Bibliography

[1] D. E. Bakken, A. Bose, C. H. Hauser, D. E. Whitehead, and G. C. Zweigle, "Smart Generation and Transmission With Coherent, Real-Time Data," *Proceedings of the IEEE*, vol. 99, pp. 928–951, June 2011.

[2] V. Paxson, M. Allman, J. Chu, and M. Sargent, "Computing TCP's Retransmission Timer," tech. rep., RFC 6298, 2011.

[3] V. A. Mahadik, X. Wu, and D. S. Reeves, "Detection of Denial-of-QoS Attacks Based On $\chi 2$ Statistic And EWMA Control Charts," *URL: http://arqos.csc.ncsu.edu/papers.htm*, 2002.

[4] R. Goodfellow, R. Braden, T. Benzel, and D. E. Bakken, "First steps toward scientific cyber-security experimentation in wide-area cyber-physical systems," in *Proceedings of the Eighth Annual Cyber Security and Information Intelligence Research Workshop on - CSIIRW '13*, (New York, New York, USA), p. 1, ACM Press, 2013.

[5] V. Irava and C. Hauser, "Survivable low-cost low-delay multicast trees," in *GLOBECOM '05. IEEE Global Telecommunications Conference, 2005.*, (St. Louis, Missouri, USA), p. 6, IEEE Comput. Soc., 2005.

[6] K. Cairns, C. Hauser, and T. Gamage, "Flexible data authentication evaluated for the smart grid," in *2013 IEEE International Conference on Smart Grid Communications (SmartGridComm)*, (Vancouver, BC), pp. 492–497, IEEE Comput. Soc., Oct. 2013.

[7] T. D. Chandra and S. Toueg, "Unreliable Failure Detectors for Reliable Distributed Systems," *Journal of the ACM*, vol. 43, no. 2, pp. 225–267, 1996.

[8] A. Doudou, B. Garbinato, and R. Guerraoui, "Encapsulating Failure Detection : from Crash to Byzantine Failures," in *Reliable Software TechnologiesâĂŤAda-Europe*, (Berlin), pp. 24–50, Springer, 2002.

[9] M. Aguilera, W. Chen, and S. Toueg, "Heartbeat: A timeout-free failure detector for quiescent reliable communication," in *Distributed Algorithms*, (Berlin), pp. 126–140, Springer, 1997.

[10] P. Felber, X. Defago, R. Guerraoui, and P. Oser, "Failure detectors as first class objects," in *Proceedings of the International Symposium on Distributed Objects and Applications*, (Edinburgh), pp. 132–141, IEEE Comput. Soc, 1999.

[11] M. Bertier, O. Marin, and P. Sens, "Implementation and performance evaluation of an adaptable failure detector," in *Proceedings International Conference on Dependable Systems and Networks*, (Washington, DC), pp. 354–363, IEEE Comput. Soc, 2002.

[12] M. Bertier, O. Marin, and P. Sens, "Performance Analysis of a Hierarchical Failure Detector," in *Dependable Systems and Networks*, (San Francisco, CA, USA), pp. 635–644, IEEE Comput. Soc, 2003.

[13] K. C. W. So and E. G. Sirer, "Latency and bandwidth-minimizing failure detectors," in *Proceedings of the 2Nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, (Lisbon, Portugal), pp. 89–99, ACM Press, 2007.

[14] B. Satzger, A. Pietzowski, W. Trumler, and T. Ungerer, "A Lazy Monitoring Approach for Heartbeat-Style Failure Detectors," in *2008 Third International Conference on Availability, Reliability and Security*, (Barcelona), pp. 404–409, IEEE Comput. Soc, 2008.

[15] R. V. Renesse, Y. Minsky, and M. Hayden, "A Gossip-Style Failure Detection Service," in *Middleware'98*, (The Lake District, England), pp. 55–70, Springer, 1998.

[16] S. Tati, B. J. Ko, G. Cao, A. Swami, and T. La Porta, "Adaptive algorithms for diagnosing large-scale failures in computer networks," in *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012)*, (Boston, Massachusetts, USA), pp. 1–12, IEEE Comput. Soc., June 2012.

[17] R. R. Kompella, J. Yates, A. Greenberg, and a. C. Snoeren, "Detection and Localization of Network Black Holes," in *IEEE INFOCOM 2007 - 26th IEEE International Conference on Computer Communications*, (Anchorage, Alaska, USA), pp. 2180–2188, IEEE Comput. Soc., 2007.

[18] S. Kandula, D. Katabi, and J.-p. Vasseur, "Shrink : A Tool for Failure Diagnosis in IP Networks," in *ACM SIGCOMM workshop on Mining network data*, (Philadelphia, Pennsylvania, USA), pp. 173–178, ACM Press, 2005.

[19] R. R. Kompella, J. Yates, A. Greenberg, and A. C. Snoeren, "IP Fault Localization Via Risk Modeling," in *2nd Symposium on Networked Systems Design & Implementation*, (Berkeley, CA, USA), pp. 57–70, USENIX Association, 2005.