A Transport Protocol for Best-Effort Content-Based Networks

Doctoral Dissertation submitted to the
Faculty of Informatics of the *Università della Svizzera Italiana*
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

presented by
Amirhossein Malekpour

under the supervision of
Prof. Antonio Carzaniga, Prof. Fernando Pedone

November 2012

Dissertation Committee

Prof. Matthias Hauwswirth	Università della Svizzera Italiana, Switzerland
Prof. Mehdi Jazayeri	Università della Svizzera Italiana, Switzerland
Prof. Benoît Garbinato	University of Lausanne, Lausanne, Switzerland
Prof. Bettina Kemme	McGill University, Montreal, Canada
Prof. Peter Triantafillou	University of Patras, Rio, Greece

Dissertation accepted on November 2012

Prof. Antonio Carzaniga
Research Advisor
Università della Svizzera Italiana, Switzerland

Prof. Fernando Pedone
Research Advisor
Università della Svizzera Italiana, Switzerland

The PhD program Director *Prof. Antonio Carzaniga*
PhD Program Director

I certify that except where due acknowledgement has been given, the work presented in this thesis is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; and the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program.

Amirhossein Malekpour
Lugano, November 2012

To my parents

On any conceivable horizon -
I'll say until about 5 billion
years from now, when the sun
explodes we're not going to
run out of discoveries.
- Paul Romer

Abstract

Content-based publish/subscribe (or simply content-based) networking is a relatively new communication paradigm compared to IP networking, with a different approach to addressing network hosts. In content-based networking addressing as well as information dissemination center around information and interests. A host's address is represented by its interest and information is routed by a network of brokers to hosts with relevant interests. We advocate the idea that content-based networking should be implemented and offered as a basic network service and be accessible everywhere and to all applications.

Best-effort content-based systems which substantiate this idea are high-throughput low-delay publish/subscribe systems in which messages are treated as datagrams without end-to-end or in-network acknowledgments and flow control. In such systems, publishers (senders) do not avoid or even detect congestion, and brokers respond to congestion by simply dropping overflowing messages. These systems are therefore unable to provide reliable message delivery and fair resource allocation and to properly handle traffic anomalies, which may degrade the overall quality of service. Furthermore, extensive usage of concurrent matching and routing algorithms in broker software causes frequent message reordering in the brokers and ultimately out-of-order delivery of messages to clients. These shortcomings of best-effort content-based systems impede their deployment for many applications and environments. For instance, the lack of congestion control hinders the deployment of such systems in the open Internet, where congestion-unaware flows are considered a major threat to the stability of the network.

With this motivation, we propose an end-to-end "transport protocol" that improves three key properties of best-effort content-based systems namely, message ordering, reliability, and congestion control. These essential requirements have been previously addressed at the protocol and broker-level, at the cost of much higher complexity and lower messaging efficiency. On the contrary, our guiding principle in designing this transport protocol is to view the underlying best-effort publish/subscribe network as a black box and only assume that it provides a general publish/subscribe API. This design principle has multiple advantages. First, the transport protocol works with virtually every best-effort content-based system, for it makes no assumptions about the underlying network. Second, individual clients can optionally take advantage of this

transport protocol for better quality of service while those with relaxed requirements can opt out and avoid unnecessary overheads. More importantly, the end-to-end protocol has a minimum impact on the favorable performance and inherent simplicity of the best-effort network since it does not require the involvement of the broker network.

Our methodology in developing this transport protocol, on the one hand, is based on understanding the body of related research on end-to-end solutions for FIFO ordering, reliability and congestion control, and on the other hand, dealing with challenges in adopting them for our purpose. This requires understanding the key distinguishing characteristics of traffic in content-based networks, and their practical implications on the design of the transport protocol. In particular, message loss detection is a canonical issue which we discuss in detail in the thesis.

Our transport protocol is comprised of three separate components for FIFO ordering, reliability, and congestion control, where the FIFO ordering protocol is prerequisite for the other two. Our FIFO ordering protocol, which is also responsible for loss detection, is probabilistic in that it does not guarantee deterministic FIFO ordering on all messages but it eliminates a vast majority of out-of-order deliveries. This compromise in the precision of message ordering is to minimize its overhead on the network. The reliability and congestion control protocols are inspired by existing proposals for IP multicast. We will detail the challenges in adopting these protocols for our purpose and propose methods and enhancements for their application in our target context. In brief, the reliability component of our protocol is based on a technique, similar to reliable multicast, that takes advantage of collaborative caching and retrieval of messages by the clients through a request/repair process. The proposed congestion-control protocol is also inspired by an existing rate-control scheme for IP multicast. This protocol uses an equation-based flow-control algorithm that reacts to congestion in a manner similar to and compatible with TCP. For the purposes of fairness and efficiency, this protocol is also content-aware, meaning that it modulates specific content-based traffic flows along a congested path.

We have a complete implementation of the proposed transport protocol. Experimental results with a best-effort content-based system and our transport protocol reveal that the protocol improves message ordering and reliable delivery by up to 95%. Also, congestion control protocol eliminates a large number of messages losses and provides considerably better resource sharing among concurrent content-based and TCP flows. Most importantly, our results show that these improvements come with a minimum compromise in the throughput and end-to-end delay of the messaging service. In summary, the experimental results are encouraging and confirm our fundamental hypothesis that the basic service provided by best-effort content-based networks can be significantly improved in terms of key quality measures, using an end-to-end transport protocol.

Acknowledgements

I am thankful to my advisors Antonio Carzaniga and Fernando Pedone for their patience, friendship and invaluable involvement in the development of my research. With Fernando and Antonio on board, one never longs for better technical insight, sharper and more constructive criticism and interesting ideas.

I am grateful to my dissertation committee members, Mehdi Jazayeri, Matthias Hauwswirth, Peter Triantafillou, and, Bettina Kemme, for their insightful comments. In particular, I would like to thank Benoît Garbinato for his constructive feedback and also for the collaboration during the first two years of my studies.

During the past five years I was privileged to enjoy the company of a group of friends and colleagues in the Distributed Systems lab and the Faculty of Informatics. Thanks to Vaide Narvaez, Lasaro Camargos, Marcin Wieloch, Nicolas Schiper, Marco Primi, Daniele Sciascia, Alex Tomic, Eduardo Bezerra, Parisa Jalili, Ricardo Padilha, Daniele Sciascia, Leandro Pacheco, Cyrus Hall, Jeff Rose, Shima Gerani, Mehdi Mirzaaghaei, Navid Ahmadi, Hamid Ghods, Parvaz Mahdabi, Mark Carman, Giovanni Toffetti, Mircea Lungu, Aliaksei Tsitovich, Marco Pasch, Edgar Pek, Lile Hattori, Paolo Bonzini, Giovanni Ciampaglia, Giovanni Ansaloni, Elisa Larghi, Janine Caggiano, Danijela Milicevic and many others for their contributions to my social and academic life. Mostafa Keikha, thank you for being a great pal, for helping me improve my math and for the Friday afternoon statistics group reading. Adina Mosincat, my beloved colleague and friend, you are missed and I will never forget you. Thanks to Mouna Allani for our fruitful collaboration on Streamline project.

Last but not least, I am grateful to my family for their patience and love, for being there whenever I needed them. Dad, thank you for asking me, every time we had a phone conversation, if I was working on a new paper, and mom, thanks for not asking. Leila, Parisa, Mehran and little Raha, thank you for cheering me up and always reminding me of what a wonderful family I have.

Contents

Figures

Tables

Chapter 1

Introduction

In publish/subscribe communication the addressing of messages is implicit and controlled by receivers. Receivers express their interests through subscriptions that state conditions on the content of messages, while senders simply publish messages without any set address. Each message is then delivered to all receivers whose interests match the content of the message. This messaging paradigm aims to eliminate some of the inefficiencies of IP networks that became evident with the exponential growth of available information and the number of connected devices on the Internet. For one, in IP networks a host's address pertains to its location while in publish/subscribe networks a client's addresses is tied to its interests and hence mobile with the client itself. For another, in IP networks access to desired content is usually facilitated through multiple tiers of centralized or distributed services that work atop IP, such as the domain name system (DNS), directory services and web services. Publish/subscribe communication eliminates or reduces the need to many of these services, for the addressing and routing protocols in these networks evolve around content and interests instead of devices and physical locations.

Publish/subscribe communication is already used in several types of applications and seems generally useful to virtually all data-driven applications. Examples of such uses include system monitoring and management (e.g., management of a large data center), information dissemination (e.g., news), resource discovery and sharing (e.g., service discovery in a "cloud" computing infrastructure, proactive search in on-line auctions), stream processing (e.g., analysis of financial data), and distributed simulations (e.g., multi-player gaming) [BRS02; EFGK03].

The successful deployment of this communication paradigm has motivated the development of such standards as Java Messaging Service (JMS)[1] and Advanced Message Queuing Protocol (AMQP)[2], followed by enterprise-level implementations like Apache ActiveMQ and IBM MQ Series. Publish/subscribe networking has also been the subject

[1] http://java.sun.com/products/jms
[2] http://www.amqp.org

of substantial research in academia to devise and implement robust and efficient publish/subscribe systems [BCM+99; CRW01; CDKR02; PB02; FJLM05; BMVV05; RPS06].

For the purposes of scalability and fault tolerance, a publish/subscribe system is usually implemented as a distributed service, and falls into either of the two major architectural categories: *peer-to-peer* and *broker-based* systems. In the broker-based network design, which is the focus of this thesis, a network of brokers functions as a routing substrate which efficiently routes messages to their intended receivers. Content-based publish/subscribe networks (or simply content-based networks) are a type of publish/subscribe system with rich addressing schemes that allow clients to express fine-grained details about the information they are willing to receive.

Content-based networks like any other communication system face an array of challenges such as scalability, security and privacy, authenticity, message ordering, reliability, fair resource sharing, and efficiency in terms of throughput and communication delay. Unfortunately, these are conflicting goals, in the sense that an improvement for one of them typically implies a loss for another. Therefore, system designers often target and optimize a subset of these factors based on the application requirements. In the set of requirements for a content-based network, three crucial elements are message ordering, reliability, and fair resource usage. The importance of these three measures is that they are among the basic properties of any communication system and directly affect its quality of service.

In this dissertation, by message ordering we mean *FIFO ordering* which is a basic ordering guarantee in communication and distributed systems. According to FIFO ordering, for any given sender, messages must be delivered in the same order they were sent. Many applications rely on FIFO ordering to ensure the consistency of actions between a sender and a receiver or among different receivers. FIFO ordering is also a building block for more involved ordering semantics such as causal ordering. By reliability, we mean reliable delivery, that is ensuring that messages are received by their intended recipients. Naturally, the reliability of a communication system is crucial to the effective functionality and often to the ultimate correctness of an application that relies on it. Furthermore, since a communication system is often a shared resource among multiple clients, it is imperative to ensure that all clients have access to their fair share of the network resources. In other words, it is often desirable to prevent a single client from exhausting network resources, causing other clients to suffer from inability to communicate. An important observation is that these requirements are in many ways intertwined, meaning that satisfying or improving one is often a prerequisite for another. As a result, these problems are often addressed in unison, in the form of a set of protocols with complementary functionalities.

Based on the aforementioned quality measures, we can classify the current broker-based architectures into two categories. In the first category, FIFO ordering and reliable message delivery are built in the protocol and implemented by the broker network. While these properties are favorable advantages from the client's perspective, provid-

ing such guarantees complicates protocol and broker design and is costly at run-time. In particular, providing such guarantees usually requires compromising throughput, scalability, and message delivery delay. Obviously, mission-critical applications with stringent message ordering and delivery requirements opt for this class of systems while applications that need to scale to a very large set of loosely coupled clients or require rapid message delivery opt for an alternative design.

On the contrary, in the so called *best-effort* design, protocols do not provide such guarantees but instead aim for maximum throughput and minimum end-to-end delivery delay. So, typically, these systems do not log messages to a persistent storage, nor do they implement any mechanism to guarantee message ordering or fair resource usage. Acknowledgment messages are not used and no explicit congestion control mechanism is in place. The advantage of these systems is that they allow for more streamlined message processing, with simpler protocols and with broker designs closer to those of network routers. This design allows for various types of performance optimizations, hardware-assisted implementations, and extension through modular design. Moreover, these protocols thanks to a simple core specification have better prospects for widespread adoption on the Internet. One of the primary reasons for this is that service providers are reluctant to adopt protocols which impose potential high costs, are difficult to analyze, and interoperate and require complex configurations. Still, despite their better performance and simplicity, the unreliable nature of best-effort systems seems to limit their deployment significantly, especially in critical application domains.

1.1 Motivation and Rationale

This dissertation centers around the idea that content-based publish/subscribe (or simply content-based networking) should be seen as a basic networking protocol with efficient and scalable message routing as its primary functionality, much like in conventional IP networking. Such a design opts for a layered and modular protocol stack resembling that of TCP/IP, in which the client side network stack or middleware builds optional enhancements atop the underlying unreliable communication primitives. In essence, we look at the best-effort publish/subscribe protocol as an underlying high-throughput but unreliable network service, and build a *transport protocol* that targets end-to-end FIFO ordering, reliable delivery and congestion control. By an end-to-end solution we mean considering the broker network as a black box and having clients involved in providing these services without making any particular assumption about the internals of routing and forwarding infrastructure.

This modular approach will allow a wider range of applications to take advantage of a best-effort service with optional additional costs in return for a better service. For instance, maintaining FIFO ordering within brokers can be memory intensive and would delay all messages without distinction. By contrast, when ordering is handled

by end-points, it is up to the client to decide the right balance between strictness of the ordering and cost in terms of additional delivery delay. Moreover, this design is applicable to virtually every publish/subscribe system, regardless of their architectures, routing protocols, and broker technologies.

The principles of end-to-end protocol design for data networks can be traced back to the first years of ARPANET [Bar64; MW88] where end-to-end protocols were devised to eliminate out-of-order reception of messages. Later, Saltzer et al. [SRC84] in their seminal work titled "End-to-end arguments in system design" maintain "end-to-end argument" as an essential design principle in communication systems:

> *"In a system that includes communications, one usually draws a modular boundary around the communication subsystem and defines a firm interface between it and the rest of the system. When doing so, it becomes apparent that there is a list of functions each of which might be implemented in any of several ways: by the communication subsystem, by its client, as a joint venture, or perhaps redundantly, each doing its own version. In reasoning about this choice, the requirements of the application provide the basis for the following class of arguments: The function in question can completely and correctly be implemented only with the knowledge and help of the application standing at the endpoints of the communication system. Therefore, providing that questioned function as a feature of the communication system itself is not possible, and moreover, produces a performance penalty for all clients of the communication system. (Sometimes an incomplete version of the function provided by the communication system may be useful as a performance enhancement.) We call this line of reasoning against low-level function implementation the end-to-end argument."*

A natural question follows this statement: *is the end-to-end argument applicable to content-based networks?* The answer to this question requires answering the following canonical question: *is it possible to implement message ordering and reliable delivery* ***completely and correctly*** *with an end-to-end approach?* In IP networking, TCP is a very successful adoption of the end-to-end argument that is, reliable message delivery and message ordering are completely and correctly provided by end-to-end protocols. At a first glance one might think that a transport protocol for content-based networks can implement the same solutions that exist in TCP. Unfortunately, there are some inherent properties of data flows in content-based networks that obstruct adoption of TCP mechanisms for content-based networks. In particular, precise end-to-end loss detection which is necessary for reliable message delivery, FIFO ordering and congestion detection requires a set of changes that cause significant overhead to the network and fall into contrast with the design principles of the publish/subscribe communication.

Thus, we find a middle ground between precision and practicality of loss detection and propose a loss detection mechanism that is prone to errors but has practical values

to our transport protocol. As a result, the reliability and ordering components of our transport protocol are probabilistic in the sense that FIFO violations or message losses are not completely eliminated, but their probability of occurrence is reduced. In other words, the end-to-end transport protocol eliminates the vast majority of FIFO violations and recovers many of the lost messages, effectively enhancing service quality that is experienced by the application. Similarly, our congestion control protocol may exhibit suboptimal resource sharing due to its reliance on loss detection, but we will show that it is able to largely reduce the negative effects of congestion and improve fairness. Therefore, we emphasize that the primary purpose of this research is not to replace reliable publish/subscribe networks but to propose, implement, and evaluate a set of end-to-end protocols that improve the quality of service of best-effort content-based networks with minimum sacrifice in their favorable properties. Moreover, we hope that the contributions of this work will pave the way in designing "complete and correct" end-to-end protocols for CBPS networks in the future.

1.2 Challenges and Solution Overview

The three components of our end-to-end transport protocol are FIFO ordering, reliability and congestion control. These problems have been extensively studied and addressed in the domain of distributed systems and data networks. Throughout the thesis we will study the applicability of the existing relevant solutions to our problem. This requires understanding the distinguishing characteristics of traffic in a content-based network and its practical implications for the components of our end-to-end protocol.

In unicast or multicast IP networks, traffic originating from a given source is channeled in a flow whose recipients are interested in all of its constituting packets (e.g, a media stream or a file). On the contrary, in a content-based network, each published message is usually an independently meaningful object which based on a subscriber's subscription might be of interest to it (selective reception). This property is the root of two key distinguishing factors between IP networks and content-based networks. First, as a consequence of selective reception, source-based sequence numbers are inapplicable to detect message loss on the receiver side. This is a major challenge in the adoption of mechanisms used in TCP or any of the existing reliability schemes for IP multicast. Second, publish/subscribe networks although categorized as one-to-many communication systems, do not incorporate the notion of group membership nor do they have the means for explicit addressing of a set of nodes. This is in contrast to IP multicast, in which group membership is a well defined concept where all group members are addressed using the multicast address dedicated to the group.

In this work, we introduce mechanisms to tackle these two problems. We propose a probabilistic loss detection algorithm that takes advantage of an efficient encoding scheme and augments messages with information that enables receivers to detect message losses with a configurable precision. We also elaborate on how the underlying

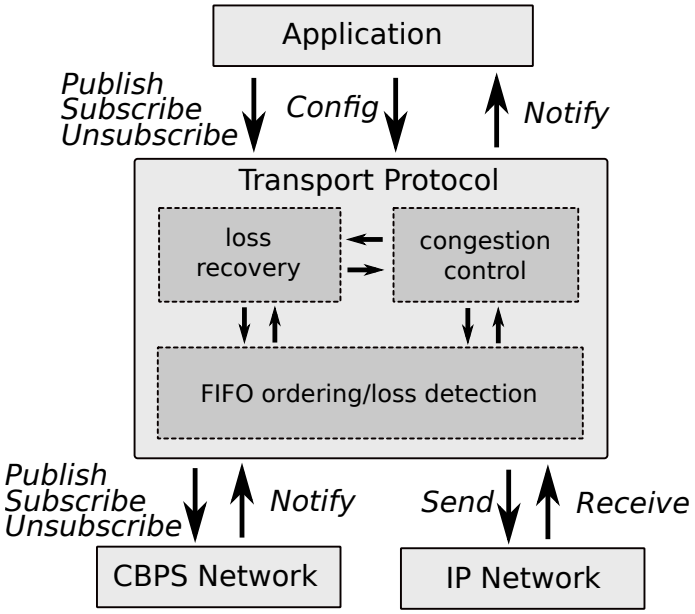


Figure 1.1. Schematic diagram of the transport protocol and its interface with application and the underlying IP and content-based publish/subscribe (CBPS) networks

publish/subscribe network can be used to route control messages to a set of subscribers. These solutions enable us to tailor the existing protocols for reliability and congestion control for IP multicast to content-based networks.

Figure 1.1 illustrates how the transport protocol sits between the application and the publish/subscribe network. The transport protocol communicates with other endpoints through the publish/subscribe network as well as the IP network to exchange control messages. Publish/subscribe API calls (subscribe, unsubscribe, publish) by the application are intercepted by the transport protocol and relayed to the network after necessary processing and adding required transport headers. The publish/subscribe API is extended to provide means for adjustment of the protocol parameters.

Notifications are first received by the transport protocol, processed, possibly buffered by the FIFO ordering component for ordered delivery, and finally delivered to the application after removing their transport header. The FIFO ordering component is also responsible for the probabilistic loss detection that is essential to the functionality of reliability and congestion control protocols. Reliability and congestion control protocols can be independently disabled or enabled. When both of these protocols are in use, the reliability component consults congestion control before sending out any control message to ensure that congestion control policies are not violated by the generated outgoing traffic.

The FIFO ordering component of our transport protocol is a probabilistic mechanism that attempts to reduce the number of out-of-order deliveries that are visible to

the application, at the cost of a minimum additional delay. In other words, having received a set of messages whose delivery to the application might cause potential FIFO violations, the protocol treats messages in a way that the probability of the potential FIFO violations after the delivery of the involved messages is reduced to a configurable value. More specifically, this is the probability of a single FIFO violation and can be chosen by the client as a configuration parameter. This parameter also affects the extra cost (in terms of delivery delay) of using the ordering protocol. This scheme builds a model for the distribution of end-to-end delay as a random variable, which is then used to determine a *latch time* for each message. This probabilistic design implies that the protocol is not able to guarantee the FIFO delivery of messages but instead, aims at reducing the probability of each instance of a FIFO violation. The end result is that the total number of FIFO violations that the application incurs is significantly reduced, subject to a minimum extra cost in message delivery delay.

The reliability component of our transport protocol is an adoption of Scalable Reliable Multicast[FJL+97] which is a generic reliability scheme for applications running on IP multicast. Loss recovery is performed through a request/repair process during which nodes recover lost messages from other receivers. Our protocol includes a novel approach to routing request messages to their intended receivers through the publish/subscribe network.

The third component of the transport protocol is a content-aware congestion control protocol that is designed to prevent persistent congestion and to improve fairness when scarce network resources are shared among concurrent flows. This protocol is inspired by TCP friendly multicast congestion control [WH01] and is TCP friendly in the sense that its long term throughput is not more than that of TCP under the same network dynamics.

1.3 Structure of the Dissertation

Chapter 2 reviews the core properties of the publish/subscribe technology and highlights the points of distinction between different systems: namely language and architectural models. We then discuss the related research with a particular focus on the problems that we intend to study in this thesis. The next three chapters discuss the main components of the transport protocol. In Chapter 3 we discuss the FIFO ordering component of our transport protocol. In Chapter 4 and Chapter 5 we detail reliability and congestion control components respectively. Finally, Chapter 6 concludes the thesis and outlines future research directions.

Chapter 2

Background and Related Research

In this section, we review the relevant literature and discuss the state of the art in publish/subscribe systems with an emphasis on the topics that relate to the contributions of this thesis. We first review language and architectural models, describing how different systems define syntax and semantics of publications and subscriptions. There is a large body of work on subscription representation, processing, and summarization. Here we briefly mention the main topics and notable publications on these areas. We will review the existing prominent architectures, highlight major challenges in designing such systems, and discuss proposed solutions. We then turn our attention to the three specific aspects that relate to the thesis, namely message ordering, reliable message delivery, and congestion control. Finally, we review Siena B-DRP, an implementation of the Siena publish/subscribe system which we use as the underlying network for all the experimental analyses presented in this thesis.

2.1 Language Models

The *language model* of a publish/subscribe system specifies how subscriptions and publications are expressed. It also defines rules according to which messages match subscriptions. Generally, there are two main categories of language models: *topic-based* and *content-based*. In the topic-based model [CDKR02; OB06; CMTV07a; CMTV07b; BBQ+07; JZR+09] subscriptions and publications are associated with a topic (tag). Topics usually have a hierarchical structure like *topic.sub-topic.sub-sub-topic* in which parts of the topic are delimited by a "." and each part covers a number of subtopics. An example of a topic is *sports.football.uk.liverpool*. In subscriptions, usage of wildcards are usually allowed, for instance *sports.football.** where "*" matches any word. Matching in this model is easily performed by a character-by-character comparison between the subscription and the publications.

On the other hand, the content-based language model [SA97; BCM+99; GKP99; CDNF01; FJL+01; CRW01; PB02; FMMB02; FGKZ03; GSAA04; TA04; FJLM05; CS05;

RPS06; DGH+06; BFG07; JHMV09; AT11] provides a rich set of operators that facilitates advanced and fine grained filtering and selection of messages. The set of operators and language syntax differs from a system to another, though in essence all of them provide similar facilities. In this thesis we will use the expressive and general model proposed by Carzaniga et al. [CRW01]. In this language model, *events* or *messages* are assigned a tuple of *attributes*, each attribute composed of a name and a value. These attributes are typed with one of the three main types: numeric, boolean, and string. For instance, *(symbol="AMD", price=100, market="NYSE")* exemplifies a message with three attributes.

A *constraint* defines a filtering rule on the type, name, and value of a single attribute. For example, *(price < 120)* mandates existence of an attribute in the message, named "price" with a value less that 120 (type of the attribute is implied). A conjunction of constraints is called a *filter* which allows for defining conjunctive filtering rules on multiple attributes. For instance, *(symbol="IBM"* $\wedge$ *price < 120)* is a filter with two constraints. Finally, a *predicate* (commonly called subscription) is a disjunction of multiple filters which represents the complete form of a subscriber's interests. An example of a predicate is *((symbol= "IBM"* $\wedge$ *market="NYSE"* $\wedge$ *price < 120)* $\vee$ *(symbol="AMD"* $\wedge$ *market="NYSE"* $\wedge$ *price < 100))*. In addition to equality, some systems provide string matching operators including substring, string prefix and suffix [CRW01; TE04; AT07].

While most common content-based publish/subscribe systems offer operators of numerical, boolean, and string type, there are proposals for richer operators to support specific applications. For instance, to facilitate dissemination of geographical information (e.g., position of a user or an object) with the aid of a publish/subscribe network Konstantinidis et al. [KCW11] propose enhancements for supporting 2D spatial objects. Some systems define the notion of a *composite event* which reflects occurrence of a given pattern in a set of events [Cou02; LJ05; JE10]. Accordingly, there are composite subscriptions to detect correlations among events or to find a given pattern in them. For instance, *((symbol="AMD"* $\wedge$ *price=$X) & (symbol="AMD"* $\wedge$ *price < $X))* yields a match only if the two subsequent events indicate a decrease in the value of the attribute "price" (the operator "&" specifies two consecutive events). Similarly, *parametric subscriptions* [JJE10] allow a subscriber to parametrize its subscription (e.g., (symbol="AMD" $\wedge$ price=$X), where $X is the parameter) and update the corresponding parameter during runtime, instead of unsubscribing and subscribing anew for the new value.

A third approach to publish/subscribe takes advantage of extended markup language (XML) to describe events in XML format along with XPath[1] or XQuery[2] to describe subscriptions. [SCG01a; CF04; DRF04]. The major drawback of these systems is the processing overhead of the XML files which is an obstacle to system's scalability.

Finally, in this subsection we explain two notions of *subscription covering* and *sub-*

[1]http://www.w3.org/TR/xquery/
[2]http://www.w3.org/TR/xpath/

scription overlap that will be frequently referred to, in this thesis. Considering two subscriptions S_1 and S_2 and the set of all messages that match these two subscriptions, M_1 and M_2 respectively, we say S_1 covers S_2 if $M_2 \subseteq M_1$. Also, S_1 and S_2 are said to be overlapping subscriptions if $M_1 \cap M_2 \neq \emptyset$.

2.2 Subscription Representation and Matching

The publish/subscribe communication paradigm is intended to facilitate development of high throughput, large-scale applications. Obviously, efficient event matching is crucial to this goal. Since the appearance of the first publish/subscribe systems, much research has been done to devise processor- and memory-efficient matching algorithms that take advantage of commodity hardware and multicore technology [ASS+99; PFLS00; CCC+01; FJL+01; CW03; FFTJ09; CTCHW09; RCF+09]. Deployment of specialized hardware has also drawn attention recently. Margara and Cugola [MC11] propose using Graphical Processing Units (GPU) while Sadoghi et al. [SLS+10; SSJ11] use FPGAs for high speed event matching.

Beside the large body of research on matching static subscriptions (subscriptions whose format does not change), there are also works on matching dynamic subscriptions like composite subscriptions ([LJ05]) and stateful subscriptions ([DGH+06]) which require stateful matching. This means that in addition to the subscriptions, brokers also need to maintain soft state about previously matched messages and account for this state for matching the next incoming messages.

Subscription matching is an interconnected problem with subscription representation where the goal is to represent and aggregate subscriptions in a compressed manner with minimum sacrifice in accuracy. This helps preserve memory at each individual broker, reduces network bandwidth usage and leads to better matching throughput [TE04; LHJ05; CTCHW09; BFG07; JE11].

2.3 Architecture and Event Routing

Typically, a publish/subscribe system is a network of processes that take on the role of publisher, subscriber, or event router (or a subset of these). The interaction among these components is usually intertwined with the event routing mechanism. Broadly speaking, publish/subscribe systems fall into one of the two architectural types: peer-to-peer systems and broker based systems (See Figure 2.1).

In broker-based systems, a network of brokers routes subscriptions and messages among publishers and subscribers. Examples of such systems are Siena [CRW01], JEDI [CDNF01], Hermes [PB02], PADRES [FJLM05] and various implementations of

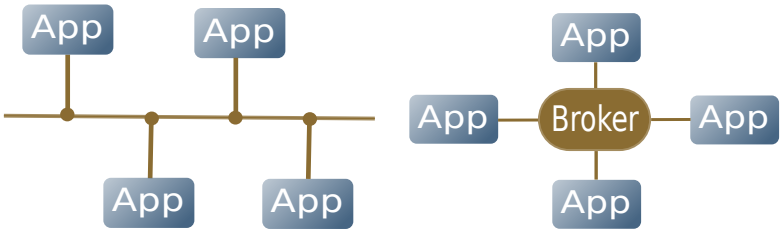


Figure 2.1. Two types of publish/subscribe architecture. (a): Peer-to-peer (b):Broker-based

JMS such as Apache Active MQ[3] and IBM WebSphere MQ.[4] Cluster-based architecture is a variation of the broker-based model proposed in [JMV08] where brokers are grouped into clusters in order to achieve better scalability.

In peer-to-peer publish/subscribe systems subscriptions and publications are routed by peers without reliance on any broker network [GSAA04; BEG04; AT06; CW06; CVJ11; AT11]. *Data Distribution Service (DDS)*[5] is a prominent architecture of peer-to-peer publish/subscribe system, defined as a formal specification developed by the Object Management Group.[6] There is a variety of proprietary implementations of the standard such as PrismTech's OpenSplice and RTI DDS. These systems are usually targeted at specific applications such as real-time processes and military purposes.

Predicate and event routing algorithms are directly influenced by architectural design and the underlying assumptions about the network topology e.g., if the broker network is an acyclic graph or a general mesh. Most routing algorithms for publish/subscribe systems account for covering relations among predicates and filters. in order to reduce the amount of state that has to be propagated throughput the network and maintained in the brokers' routing table. [CRW01; CRW04; LHJ05; TK06]

For broker-based systems a variety of routing protocols has been proposed, many of which adopt a variation of reverse path forwarding [BCM+99; CRW01; FMMB02; CW03; LMJ08]. In these systems, publishers issue advertisement messages that are propagated throughout the network, effectively creating a tree, rooted at the publisher's home broker [CRW01; LHJ05; LMJ08]. Filters are then routed towards brokers with matching advertisements on the formed trees. This process enables the intermediate brokers to populate their routing tables to route messages. There are also broker-based systems that incorporate routing schemes similar to those used in IP networks where a variant of a link-state [MRI+80] or distance vector [GLAM97] routing protocol is used. Siena B-DRP [CTCHW09] is an example of such systems that works

[3] http://activemq.apache.org
[4] http://www-01.ibm.com/software/integration/wmq
[5] http://portals.omg.org/dds
[6] http://www.omg.org

atop any conventional distance vector or link state routing protocol.

In a separate class of broker-based systems, certain topics or contents are associated with rendezvous points in the network, typically using a predefined hash function. Subscriptions and events are routed to the rendezvous point, ensuring that they will intersect at some intermediate broker or eventually at the rendezvous point from which they will be propagated along a tree rooted at that rendezvous point. Optimizations are applied to avoid unnecessary routing of events towards the rendezvous point where routing decisions can be made locally. Most of the systems in this class rely on a DHT to map topics or contents to certain brokers in the network [PB02; BMVV05; JHMV09].

Peer-to-peer publish/subscribe systems rely on conventional query and message routing algorithms previously proposed for structured and unstructured peer-to-peer networks, including distributed hash tables (DHT) and epidemic algorithms. In particular, DHTs have been shown to effectively facilitate event routing in large scale peer-to-peer networks, while providing favorable reconfigurability in response to failures. For instance, Scribe [CDKR02], PastryString [AT06] and Marshmallow [GLZ11] are based on Pastry [RD01]; SPICE [CQL08] works atop Tapestry [ZHS+04]; FEL [CW06] and the work by [TBF+03] are based on Chord [SMLN+03]; and Meghdoot [GSAA04] relies on CAN [RFH+01]. There are also DHT-independent publish/subscribe systems that view the underlying DHT as a black box with a common lookup API [TAaJ03; TA04; BMVV05; AT05; BFDG07; AT11].

Among the publish/subscribe systems that are built upon unstructured networks, [CF05; CP05; VRKS06; CMTV07b; BBQ+07] are notable works that rely on epidemic algorithms like CYCLONE [VGS05] that facilitate dynamic self management for the overlay. One of the primary challenges in this category of systems is the optimal construction of the overlay in a way that peers with similar interest shape clusters in order to reduce event propagation delay and network resource usage [BBQV04; OR10; GCV+10; CJV10; CVJ11; OR11].

Finally, few proposals leverage generic multicast protocols like IP multicast for event dissemination. Event space is partitioned into smaller partitions and each partition is assigned to a distinct multicast group. Here again, the primary concern is to reduce the number of partitions (and hence the number of multicast groups). Unfortunately, the lack of wide availability of multicast protocols obstructs the deployment of such protocols on the Internet [OAA+00; CS05; EGN08; Hol11].

2.4 Reliable Delivery and Ordering

Message oriented middleware and in particular, content-based publish/subscribe systems, have a wide range of applications in enterprise environments. Mission critical applications usually have stringent requirements in terms of delivery guarantees and ordered delivery, so to support such applications, system architects opt for publish/subscribe systems that provide such guarantees. As we will discuss in this section,

most publish/subscribe systems address these two issues within the same technical framework, with logging on persistent storage and hop-by-hop acknowledgement messages inside the broker network.

Of the numerous types of ordering guarantees that have been defined in the context of fault tolerant systems [HT93; DSU04], some can be directly applied to publish/subscribe communication. In this thesis, we consider the most basic form of ordering, that is known as *FIFO ordering*. This means that if two messages m_1 and m_2 sent by the same sender are delivered to a receiver, then they must be delivered in the same order they were sent. Violation of this property can cause wrong or inconsistent decisions by different receivers. Hereafter, we refer to FIFO message ordering simply as message ordering. Also, various forms of reliability have been studied, along with various methods to achieve them [BSB+02; BZA03; PB02; FJLM05; KJ09]. However, in this chapter we refer primarily to a form of reliable delivery also known as *message persistence* in such standards as the Java Messaging Service specification. According to this delivery mode, the service must guarantee not to lose messages due to failures of message brokers.

2.4.1 Reliable versus best-effort systems

Reliable publish/subscribe systems strive to ensure that any subscriber will receive all of the published messages that match its subscriptions and the receiver will not receive messages "out-of-order" (i.e., according to some ordering semantics)[BSB+02; BZA03; PB02; FJLM05; KJ09]. Guaranteed delivery and service availability are provided by broker replication, deployment of redundant links and logging of messages on durable storage. Moreover, reception of messages by the intermediate brokers and the final recipients are acknowledged at each hop. These solutions are known as *store and forward* systems. Persistent storage and ordered delivery are clearly favorable (and for some applications essential) service qualities. Nevertheless, providing these guarantees come at the cost of lower messaging throughput and higher delivery delay.

On the other hand, systems in the "best-effort" class [JZR+09; CRW01; CDNF01; SCG01a] do not offer guaranteed delivery but instead try to maximize throughput and reduce end-to-end delay through a simple design that satisfies a minimal set of tasks. Messages in these systems are similar to IP datagrams, receiving a best-effort treatment. So, typically these systems do not log messages to persistent storage, nor do they guarantee ordered delivery of any kind. Also, unlike most systems in the previous category, the best-effort systems do not use any sort of acknowledgment messages or flow control. This simplicity in protocol allows for a simple broker design that makes optimum usage of multicore processors through concurrent matching and forwarding algorithms. In fact, brokers in such systems have functionalities and tasks similar to those of routers in IP networks, with a primary task of routing and forwarding messages. This category of systems in their basic form are suitable for interactive and realtime applications or applications that require scalability and are tolerant to

occasional message losses or FIFO violations.

As we mentioned before, the purpose of this thesis is to combine the best of both worlds i.e., to bring better reliability, message ordering and congestion awareness to best-effort systems with an end-to-end solution that requires a minimum sacrifice in terms of messaging throughput and delivery delay. With this objective, we will next review reliable publish/subscribe systems and detail various approaches to offer guaranteed and ordered message delivery in this context.

2.4.2 Reliable delivery

Kazemzadeh and Jacobsen [KJ11] tackle the problem of partitioning in a network of brokers, which can happen as a result of broker failures. The underlying assumption in this work is that brokers are structured in a tree rather than a mesh. To facilitate tolerance against network partitioning, brokers maintain local knowledge about their immediate neighboring brokers as well as brokers located within a given distance (in terms of broker hop). This allows a broker to bypass network partitions and directly connect to working brokers in its downstream. As a result, the broker network is able to rapidly reconstruct itself from a set of disconnected partitions. The protocol also incorporates a mechanism to ensure that publications are forwarded to a given subscriber only when all of the intermediate brokers are aware of the complete subscription set of that subscriber. This is a method to disallow gaps in the stream of messages delivered to a subscriber.

Yuanyuan [ZSB04] et al. address the same issue by placing two or more brokers in small broker groups. Broker groups are connected through multiple propagation trees which provide redundant routing paths to counter link failure and facilitate load balancing. However, utilization of multiple propagation trees creates the possibility of incomplete or inconsistent subscription information as well as gaps in delivered messages (e.g., in case of using different trees to route messages to a given subscriber). To address this challenge, this approach relies on virtual time (VT) vectors to convey temporal consistency in propagating incremental subscription information along different trees. Brokers maintain VT vectors for their subscription information and messages carry a VT vector that indicates their publication timestamp. Brokers decide if they have the necessary information to route the message, by comparing their VT vector with that of a message. If the test indicates insufficient or out-of-date information, the brokers conservatively flood the message to all neighbors on a routing tree (hence eliminate the possibility of causing a gap by falsely filtering a message). Although the proposed method does not require subscription state agreement across redundant paths it requires separate protocols to maintain the broker network in groups and multiple overlays.

Ostrowski and Birman[OB06] divide the publish/subscribe network to scopes where reliability is provided within each scope using a mechanism that may vary from one scope to another. The authors briefly mention the possibility of combining reliable

multicast protocols like Scalable Reliable Multicast (SRM) [FJL+97] and Reliable Multicast Transport Protocol (RMTP) [LS96] with an unreliable publish/subscribe protocol to provide reliability within one scope. The idea is that scopes are arranged in a tree, forming a hierarchy of scopes. Within each scope there is a second tree of receivers where delivery of each message is acknowledged to the root of the tree and lost messages are recovered from the root. If a lost message can not be recovered by the local root it is forwarded to the upper node the in scope tree, and the recovery process continues in a recursive fashion (similar to RMTP). However, the proposed general approach is limited to topic-based publish/subscribe systems. Even so, this work lacks a precise and concrete plan on how it can be implemented.

Bhola et al. [BSB+02] propose a mechanism in which publishers and subscribers together with brokers form a *knowledge graph*. Soft state information labeled as *knowledge* and *curiosity* flow downstream and upstream among the nodes of the tree. Using these state messages, nodes decide about the delivery order of the events and ensure one-time guaranteed delivery even in the presence of failure. While this method can guarantee FIFO and total order delivery, it introduces complexities in the implementation of the broker and does not integrate with any broker technology in a seamless manner.

Yoon et al. [YMJ11] study algorithms for instantaneous restructuring of broker overlay in response to failures or in order to optimize overlay in terms of delivery delay, based on the current subscription or publication pattern. The authors first define a set of requirements for a publish/subscribe network, in terms of ordered and guaranteed delivery and then propose algorithms for overlay reconfiguration that respect the defined properties. To this end, a set of primitives and operations are worked out (for instance to remove or add an inter-broker link) using which the broker topology can be restructured. The limitation of this work comes from the assumption that the broker network is a tree, instead of a general overlay.

Costa et al. [CMPC03] propose using epidemic and peer-to-peer techniques to provide reliability in publish/subscribe networks. This work includes two separate protocols both relying on gossip-based message dissemination. The first protocol called *active push with positive digest*, is a push-based gossip protocol in which a receiving node[7] periodically gossips a digest of its recently received messages which match one of its subscriptions s. The choice of s is based on a configurable strategy (e.g., randomly chosen from the set of local subscriptions). The gossip message is labelled with s and then routed towards other nodes with similar interest. Each receiving node that has s as one of its local subscriptions compares the digest with the list of its received messages. If the comparison indicates a lost message a request is sent to the source of the message to recover the lost event.

The second protocol proposed by Costa et al. is named *reactive pull based with neg-*

[7]This work does not explicitly distinguish between a broker or an ordinary client, hence we call it a node here.

ative digest, requires publishers to tag each message with the unique identifier of the subscription that matches the message, assuming that the all publishers have global knowledge about all subscriptions in the network. The message also bears a sequence number that is associated with the subscription that matched the message. This requires the publisher to maintain a separate sequence number for each distinct predicate in the network. Subscribers, upon reception of a message, compare its sequence number with their expected sequence number, which enables them to detect a message loss. A lost message is then recovered either from the publisher or from another subscriber using gossiping. The first protocol provides reliability in probabilistic terms in the sense that loss detection (and hence message recovery) is not guaranteed. The second protocol is prone to poor scalability due to the fact that publishers need to maintain a separate sequence number for all predicates in the network.

Esposito et al. [ECG09; ERB+12] try to provide both reliability and timely message delivery in best-effort topic-based systems targeted for Internet scale deployment. The proposed protocol is comprised of two phases. First, a dissemination phase in which the publisher transmits the original data messages along with redundant messages which are a linear combination of the original messages (through network coding), in order to provide redundancy. The second phase is recovery, when a subscriber detects a messages loss (using timeouts) and starts a gossip-based recovery process to recover the lost event. This approach is in fact similar to that of Ricochet [BBPP07] which targets reliability and timeliness for overlay multicast. Unfortunately, the protocols proposed in the two previous works suffer from inherent limitations of gossip based communication, specially substantial resource usage and propagation delay in larger networks.

2.4.3 Ordered delivery

Aguilera and Storm [AS00] propose a *bias algorithm* to deterministically provide uniform total order of messages. In this scheme, some of the nodes act as *merger* nodes, each one responsible for a subset of subscribers. All of the events go through the merger nodes to be ordered in a globally uniform manner and then they are forwarded to subscribers. This algorithm functions based on the assumption that publishers have access to synchronized clocks and they have a known publish rate. Although this algorithm has the interesting ability to determine an upper bound on the delivery delay, it may cause substantial delivery delays and confine scalability. Furthermore, there needs to be a smart assignment of subscribers to merger nodes in order to share the balance among them and prevent overloading of some mergers.

The method described by Lumezanu et al. [LSB06] provides causally ordered message delivery to the members of any two overlapping subscription groups. A subscription group is a set of subscribers with equal subscriptions and it is possible for subscription groups to have common members. This work is based on the observation that when events are routed to overlapping subscription groups via different routing

paths it is possible for subscribers to receive events in an inconsistent order. To address this problem the authors utilize a set of *sequencing atoms*, that is a set of receivers in the network, that sequence messages by assigning globally consistent sequence numbers. The challenge is to place sequencers at the intersection of different routing paths so that messages en route to overlapping groups undergo the same sequencing operation. This work is based on the assumption that links guarantee FIFO message ordering.

Platina [Pla11] argues that when sequencers are used to assign sequence numbers to all events published to a given topic, it is possible for a set of subscribers whose subscription include multiple topics, to receive events in a non-uniform order. This is due to the fact that sequence numbers assigned to events of two different topics have no ordering relation. The author then describes an algorithm to provide *Total Notification Order (TNO)* in which for any two events e_i and e_j, all subscribers that receive both of them must receive them in the same order. Thus, TNO is semantically similar to *Weak Total Order* where the ordering agreement is limited to a given view of the network [DSU04]. This work assumes that channels provide FIFO message ordering and topics have a predefined total order. To achieve TNO, Platina uses a set of *topic managers* where the manager of a given topic has a knowledge about all subscriptions (as well as their respective subscribers) that include that topic. Before publishing a message to a given topic T, publishers obtain a sequence number from the topic manager M_T assigned to that topic. To issue a sequence number, the topic manager sends a request to the topic manager responsible for the immediately preceding topic T' in the ordering sequence, provided that T and T' are both included in more than one subscription. This process continues recursively, while each topic manager adds the sequence number of the latest message published to that topic to a vector attached to the message. The result is sent back to the publisher in the form of a vector of sequence numbers which enables the receiving subscribers to deliver events of different topics according to a uniform order.

Zhange et al. [ZMJ11] study and propose methods to implement total order in publish/subscribe systems. They introduce three levels of total order in publish/subscribe networks: First, *per publisher total order* in which for any two messages m and m' published by a given publisher all subscribers that deliver m and m' must agree on their delivery order. Second, *local total order* where for any two messages, m and m' if they have the same set of receivers then those receivers must agree on the delivery order of m and m'. Finally, *pairwise total order* is defined for any two messages irrespective of their publisher or their set of receivers. Total ordering is optional i.e., it is enabled at the request of a publisher or subscribers. The process involves a *conflict detection phase* where by checking the content of a message published by a publisher and advertisements of other publishers in the network, a broker determines if there is the possibility of an ordering conflict. In the second phase, the *resolution phase*, brokers that route the conflicting messages to the subscribers defer their delivery until there is indication that all brokers agree upon the delivery order of messages. This protocol

is designed to work with routing protocols based on advertisement and reverse path routing [CRW01] and assumes an acyclic broker topology.

2.5 Scalability and Load Balancing

Like any other communication paradigm, scalability is one of the main factors by which the usability of a publish/subscribe system is measured. Two main components of scalability in these systems are the ability to serve a large number of clients (or subscriptions) and the capacity to handle high publication rates. In large networks with many clients and/or high aggregate input load, parts of the broker network may not be able to handle the high rate of incoming messages.

Cheung et al. [YCJ06] propose a method to handle the excessive load when there are a large number of clients associated with a broker. Load balancing is achieved by moving the responsibility of some of the subscribers to the brokers with underutilized capacity. In [LMJ08] the authors propose a routing scheme that routes messages through different paths in order to distribute forwarding and routing load among multiple brokers and bypass overloaded links and brokers.

Jafarpour et al. [JMV09] propose a load balancing scheme for their cluster-based architecture. Brokers are statically grouped into clusters in such a way that clients whose subscriptions are highly probable to match popular events (and hence require more matching and forwarding effort) are distributed equally among clusters. Moreover, during the operation, the overloaded brokers can offload some of their tasks to other brokers in the same cluster. The downside of this technique is the administrative overhead associated with clustering the brokers; not to mention that there needs to be prior information or assumptions about the popular events.

2.6 Congestion Control

Congestion control and fair resource allocation have been one of the primary challenges in any data communication system including publish/subscribe networks. The work by Pietzuch and Bhola [PB03] is the only published work we know of that explicitly addresses congestion control for publish/subscribe networks with a focus on reliable broker-based systems. The authors propose two orthogonal protocols: first a protocol to address congestion caused by high rate publishers and a second protocol to control the rate at which brokers recover from failure by requesting lost messages from the publishing end (pubend).

This first mechanism tries to keep the publish rate to the rate that the slowest link in the broker network can handle. This scheme requires involvement of the broker network in the congestion control process where there are upstream and downstream congestion control messages along the route between the publisher's home broker and

the subscribers' home brokers. The rate at the publisher's home broker is adjusted to the lowest rate which is requested by the broker that suffers from congestion the most. Subscriber hosting brokers (SHBs) frequently check if they receive messages at the same rate at which the pubend sends messages; if there is an indication of congestion, through feedback messages the pubend is requested to slow down. These feedback messages are propagated upstream toward the publisher hosting broker (PHB) while the intermediate brokers aggregate feedback messages coming from different downstream brokers. The PHB in response to feedback messages deploys a combination of additive and multiplicative increase with a multiplicative decrease to adjust its sending rate.

The second protocol allows for smooth recovery of brokers that lost messages due to failures. The recovery phase involves requesting lost messages from the pubend while retransmission of the lost messages has to be rate-controlled to prevent congestion during the recovery phase. Rate limiting is facilitated through a NACK window which indicates which part of the message stream has to be requested. The size of this window is adjusted in a similar fashion to TCP Vegas [BOP94], according to the level of congestion in the network.

The main drawback of this protocol is its requirement to modify the broker software. Moreover, the protocol is not capable of providing fairness among concurrent flows between separate endpoints sharing the same PHB and SHB.

2.7 Siena B-DRP

Siena B-DRP [CTCHW09] is a recent implementation of the Siena publish/subscribe system which we have used for all the evaluations in this thesis. Siena B-DRP is a best-effort protocol and broker software which implements a high-throughput, low latency content-based network. It does not order nor store messages at intermediate brokers, and does not use acknowledgments to confirm delivery. As such, this system provides an appropriate platform upon which we develop and test our transport protocol.

At its heart, D-DRP uses a probabilistic encoding that transforms filters and messages into Bloom filters, and that admits to a matching algorithm consisting of a simple bit-wise operation between the two Bloom filters. The matching of a message against a predicate in this encoding scheme is defined by *covering relation* as follows: if the Bloom filter of a message *covers* the Bloom filter of at least one filter in the predicate, then the message matches that predicate. The covering relation between two Bloom filters of size M is denoted by $B_1 \subseteq B_2$ (B_2 covers B_1) and is defined as below:

$$B_1 \subseteq B_2 \Leftrightarrow (\forall i \in 1..M : B_1[i] = 1 \Rightarrow B_2[i] = 1)$$

This encoding scheme and the covering relation is extensively used in the design of our transport protocol. Throughout the thesis, we will detail some other properties

of this encoding method and how it is used in the transport protocol.

In Siena B-DRP, brokers advertise (broadcast) the set of Bloom filters that represent the interests of the subscribers that they serve. As a result, all brokers have a global information about all subscriptions and their associated home broker. Publisher home brokers encode messages to Bloom filters and using a simple matching algorithm find the set of brokers with an advertised Bloom filter that match the message. The identifier of such brokers is added to the message header and the message is then routed towards the destination brokers using a generic destination routing protocol. If an intermediate broker is a split point in the dissemination tree, then the message is duplicated for each separate downstream subtree and the header that represents the set of destination brokers is adjusted accordingly to only contain the set of brokers in that specific subtree.

Chapter 3

FIFO Ordering

Message ordering is a fundamental element of many transport-level services in traditional networking and fault-tolerant distributed systems. Of the numerous types of ordering guarantees that have been defined and extensively studied [HT93; DSU04], some can be directly applied to content-based communication. One such guarantee is first-in-first-out (FIFO) ordering which requires that messages sent by a given sender be delivered in the same order they were sent. The importance of FIFO ordering in messaging systems stems from the fact that individual messages can be independently meaningful, and reception of different messages by different subscribers in non-FIFO orders can result in faulty or inconsistent actions. For example, consider a temperature monitoring system whose responsibility is to react to any temperature increase, where temperature is measured and reported by a single sensor. If two consecutive messages that indicate a temperature rise (e.g., the second published message reports a value higher than what was reported by the first message) are received out-of-order, the monitoring system will fail to detect the correct pattern.

Beside its important impact on application logic and semantics, FIFO ordering is a primitive upon which more complex ordering and reliability semantics are defined and implemented [DSU04]. For instance, as we discussed in Section 2.4.3, there are proposals to implement causal and total ordering in reliable publish/subscribe systems, with the assumption that communication channels provide FIFO message ordering. Finally, in the context of a transport protocol, FIFO ordering is interrelated with message loss detection which is itself a prerequisite for loss recovery and congestion detection.

Intuitively, FIFO violations are caused by short-term variations of the end-to-end delay of messages, which may occur in the presence of different delivery paths or if the forwarding process is parallelized and therefore does not itself maintain FIFO ordering. The out of order reception of messages due to parallelism and queuing complexities has been acknowledged and studied by the networking community. In particular, Bennet et al. suggest that IP packet reordering is not a pathological behavior but rather an inevitable outcome of highly parallelized processing [Bol93; Pax97; BPS99]. Best-

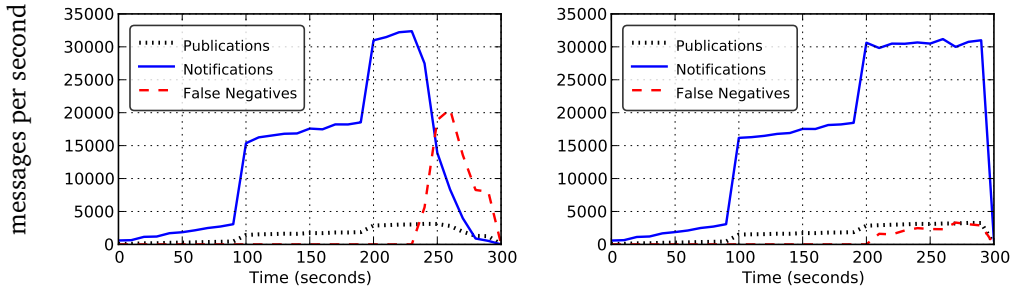
effort content-based networks are also prone to the same issue due to their heavy usage of multithreading for the matching and forwarding process.

FIFO ordering is typically implemented using sequence numbers set on the sender side to reflect the sending order, and checked on the receiver side to enforce the same order for delivery [HT93]. When the network delivers a message with a higher-than-expected sequence number, the receiver must decide whether to wait for the missing message or to proceed by delivering the message it has received. In practice, protocols like TCP assume a bound on end-to-end delay and trigger a timeout on a missing message after a given threshold. This is called the *timed asynchronous distributed system model* and is showed to have practical values for implementing a variety of distributed systems [CMA97; CF99]. Unfortunately, in the context of content-based networks, this simple mechanism is inapplicable. More precisely, because of the implicit addressing induced by the content-based model, the receiver does not know whether the hole in the sequence is due to a message that was delayed along the delivery path, or to a message that does not match the receiver's interests.

In this chapter we present a probabilistic method to achieve FIFO ordering in content-based communication. We illustrate this method using Siena B-DRP that we introduced in Section 2.7. B-DRP's design is intended to achieve high delivery rates thanks to an efficient routing scheme as well as highly parallelized matching and processing within brokers. Thus, B-DRP is arguably an ideal testbed to experiment with message ordering. Yet, the method we present is generic, as it applies to end-points (publishers and subscribers) and treats the whole network as a black box.

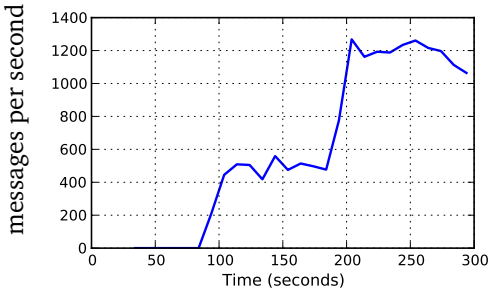
At a high-level, our approach is to measure the delay variations and then compensate for their effect. Consider two messages m_1 and m_2 published within δ seconds by the same sender. If the delay variation on the path to a receiver is comparable to δ, then m_2 may be delivered before m_1, thereby leading to a FIFO violation. Upon receiving m_2, the receiver must decide whether to deliver m_2, thereby assuming that m_1 will never arrive perhaps because it did not match the receivers' interest, or to hold m_2 and wait for m_1. Our approach is to give the receiver some effective means to inform this decision. In particular, if the receiver knew δ and the distribution of delay variations, then it could determine an optimal holding time for m_2 so as to reduce the probability of incurring a FIFO violation.

To understand and measure delay variations, we study the dynamics of B-DRP and show that the end-to-end delay of messages along a specific path follows a *hypoexponential* distribution. We also develop a way to measure the parameters of this distribution dynamically, and therefore a method to calculate the probability of a FIFO violation upon the observation of a hole in the sequence numbers. We also use the same model and technique to estimate the necessary latch time (i.e., deferring the delivery of messages to the application) to reduce the probability of a FIFO violation. We then enhance the receiver's decision algorithm with a method to estimate the relevance of missing messages, to prevent the unnecessary holding of a message when none of



(a) ActiveMQ: Throughput (b) B-DRP: Throughput

Figure 3.1. Throughput of ActiveMQ and B-DRP in an 8-broker network.



B-DRP: FIFO Violations

Figure 3.2. FIFO violations of B-DRP in an 8-broker network corresponding to Figure 3.1

the missing messages matches any local interest. Our extensive experiments with networks of up to 46 brokers and 2500 clients reveal that our model reflects the dynamics of the network in various working conditions, and is able to avoid more than 95% of FIFO violations while keeping the extra delay caused by latching to a minimum.

In Section 3.1 we begin by motivating this work and overviewing the problem and our proposed solution. We then detail the model and our probabilistic FIFO ordering algorithm in Sections 3.2 and 3.3. Section 3.4 formalizes our protocol with an algorithmic description. We present an experimental evaluation of the proposed algorithm in Section 3.5. We conclude this chapter in Section 3.6.

3.1 Overview of Problem and Solution

We motivate this work through an experimental comparison between B-DRP and Apache ActiveMQ.[1] Our purpose here is to exemplify the problem at a high level, so we give

[1]ActiveMQ (http://activemq.apache.org/) is a very popular and reportedly very efficient messaging system that also implements a content-based publish/subscribe service as part of the Java Messaging

only a cursory description of the experiment, focusing on the comparative analysis of the two systems. In Section 3.5.1 we discuss this and other experiments in greater depth. The experiment measures the throughput and the rate of FIFO violations in a content-based publish/subscribe network of 8 brokers subjected to increasing message traffic. The network topology is a graph of diameter 3. Each broker runs on a dedicated machine and serves 50 clients running on the same machine. All 400 clients are subscribers, but only 50 also act as publishers. In order to simulate a wide-area network, we apply transmission delays and bandwidth limits on inter-broker links. In particular, the bandwidth limit is 10Mbps and the transmission delay is 50ms with a dynamic runtime variability of $\pm$5ms which is typical of the Internet, based on different Internet measurements.[2]

Clients generate a synthetic workload of subscriptions and publications, and the generation algorithm is parametrized so as to induce an intense stream of messages to a few subscribers combined with a steady but slower flow to all other subscribers. The experiment modulates the publication rates over a period of 300 seconds using two groups of high-rate and low-rate publishers, respectively. Two high-rate publishers, each one publishing 1200 messages per second, join the network at 90 and 190 seconds, respectively, and cause the two noticeable increases in the aggregate input and output rate. The remaining 48 are low-rate publishers, with a publication rate that slowly ramps up from about 1.5 messages per second at the beginning of the experiment to a maximum of 25 messages per second at the end of the experiment. This mixed workload is intended to show, to the extent possible in a single experiment, the general reaction of the broker network to abrupt as well as gradual increases of publication rates, and also to congestion, since the workload is also designed to reach the maximum (collective) delivery capacity of the network for both systems.

Figures 3.1a and 3.1b show the throughput measurements. In particular, we plot the rates of publication, message deliveries (i.e., notifications) and false negatives for both ActiveMQ and B-DRP. (A false negative occurs when a subscriber does not receive a published message that matches its subscriptions, and is typically caused by congestion.) Both networks gracefully handle gradual and sudden increases of the aggregate input load during the first 200 seconds of the experiment. At this point B-DRP reaches its delivery limit of about 30000 messages per second, and from then on it starts dropping messages, as evidenced by a raise in the rate of false negatives in the diagram. ActiveMQ handles the growth of the input rate up to time 240, when it delivers 32500 messages per second. However, after this peak point, it exhibits a sudden reduction in delivery rate. Also, about 20 seconds after this congestion point, ActiveMQ brokers start blocking publishers, presumably to counter congestion.

As for ordering, ActiveMQ delivers all its notifications in FIFO order—as it should,

Service.

[2]For example see measurements by RIPE Network Coordination Centre (RIPE) available at http://www.ripe.net/data-tools/stats/ttm/ttm-data

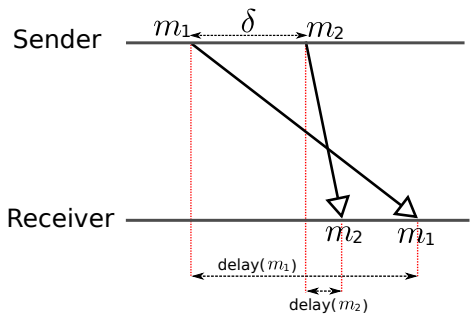


Figure 3.3. Illustration of how a FIFO violation occurs

according to the Java Messaging Service specification—while B-DRP, which is designed as a best-effort network, does not prevent FIFO violations, and in fact incurs a rate of violations that is roughly proportional to the delivery rate (see Figure 3.2).

In summary, we observe that ActiveMQ maintains FIFO ordering at all times but suffers an almost catastrophic reaction to congestion, while B-DRP reacts more gracefully to congestion but incurs a significant number of FIFO violations. Our goal is to combine the best behaviors of these two systems. In particular, we argue that FIFO ordering can be best supported as an optional, end-to-end service implemented on top of a best-effort publish/subscribe system. Notice that, while this approach is consistent with well-established principles in network design, it is practically incompatible with the specification of the Java Messaging Service and other similar publish/subscribe standards.

3.1.1 FIFO ordering

FIFO is a simple ordering condition defined for each sender/receiver pair: considering a sender s and a receiver r, for every pair of messages m_1 and m_2 sent by s and received by r, a FIFO violation occurs whenever s sends m_1 before m_2 but r receives m_2 before m_1. It is also useful to express this condition in terms of the total travel time of each message: let $departure(m)$ be the departure time of a message m, and assume $\delta = departure(m_2) - departure(m_1) > 0$; let $arrival(m)$ be the arrival time, and $delay(m) = arrival(m) - departure(m)$ the total travel time of a message m. Then, a FIFO violation occurs when $delay(m_1) - delay(m_2) > \delta$ (see Figure 3.3).

Furthermore, the total travel time of a message can be expressed as the sum $delay(m) = delay^*(m) + vardelay(m)$ of a nominal delay, $delay^*(m)$, representing the long-term-average link, queuing, and processing delays, plus a short-term-variable delay $vardelay(m)$. This distinction is useful because, ignoring pathological cases, FIFO violations occur only when the departure interval δ is small, and therefore when the long-term-average delays of m_1 and m_2 can be reasonably considered constant. This means that FIFO violations are essentially a function of the short-term-variable delays

and the departure interval δ. Specifically,

$$\text{FIFO violation} \Leftrightarrow \mathit{vardelay}(m_1) - \mathit{vardelay}(m_2) > \delta \tag{3.1}$$

Equation (3.1) expresses the essence of the problem as well as the idea upon which we develop a solution.

FIFO ordering is typically implemented with sequence numbers attached to each message by the sender to reflect the sending order, and used by receivers to follow the same order for delivery. One might try to apply the same sequencing technique to content-based communication. However, in this case, holes in the sequence (e.g., receiving m_7 immediately after m_5) must be treated differently. Specifically, because of the implicit addressing of the content-based model, the receiver does not know—and in some cases it can not know—whether a hole in the sequence is due to a message being delayed along the delivery path (e.g., due to its longer processing time) or whether that message was not supposed to be delivered at all because it does not match the receiver's interests.

Therefore, in order to avoid (or minimize) FIFO violations, we must solve two problems: first, a receiver must decide whether or not to wait for a missing message; second, if the missing message is determined to be likely to arrive, the receiver must determine an appropriate buffering time (or "latch" time) for the message(s) received out of order. One might argue that the receiver can always wait until the missing messages arrive. However, due to the best-effort nature of the service, those messages may get lost, leaving the receiver in a live-lock condition. So eventually, the receiver must timeout after a certain latch time and drop or deliver the buffered messages.

Our approach is to give receivers a way to answer these questions on the basis of the condition defined in Equation (3.1). In particular, we propose to carry with each message its departure time, the departure time of the preceding message, and a summary of the content of some previous messages. With departure times (time-stamped by senders) and arrival times (recorded locally) a receiver can continuously measure the distribution of end-to-end short-term-variable delays. Also, upon receiving message m_2 in the absence of the preceding m_1, a receiver can use the time stamps on m_2 to compute the sending interval δ between m_1 and m_2. Then, with δ and the measured distribution of delays, the receiver can decide, up to a set error probability, whether m_1 may have been delayed and, using the content summary carried by m_2, whether m_1 may arrive. If so, the receiver determines, also based on δ and the delay measurements, how long to hold m_2 so as to avoid a FIFO violation without delaying the delivery of m_2 excessively. Our approach does not require synchronized clocks. We do however, assume that clock drifts are small enough so that changes in clock skews are negligible in short periods of time (e.g. less than one second).

3.2 Probabilistic FIFO Ordering

The method we propose is probabilistic in nature, since it is based on a probabilistic model of delay variations. We now detail this model and how we use it to reduce FIFO violations.

3.2.1 Model of end-to-end delay

We model the end-to-end delay of a message m as the sum of a long-term average delay plus a short-term variation $vardelay(m)$. Since we are interested in comparing the end-to-end delay of pairs of messages sent by the same publisher within a short interval, we consider the long-term-average component of these delays to be the same. Thus, we focus on the delay variation $vardelay(m)$. In particular, we model $vardelay(m)$ as a random variable with a probability distribution whose parameters are also constant during the short interval that separates two consecutive messages.

In general, the variable component of the processing time (including queuing) and the transmission times at each hop in the publish/subscribe network contribute to the end-to-end variable delay. Typically, a broker has a set of tables to store subscriptions and routing information, and forwarding a message involves comparisons against the entries of the subscriptions table and/or a lookup in the routing table. As a result, the processing time may vary according to several factors, including the number of subscriptions, their constraints, the number of attributes in the message, and the matching algorithm, which might itself be randomized.

Furthermore, in a typical modern implementation on multi-processor hardware, the forwarding process is usually parallelized for maximum throughput, at a minimum with each message handled by a separate thread, and possibly with finer-grained parallelism. Therefore, since forwarding incurs minimal (if any) contention on shared data, the processing times for two different messages are mostly independent. Similar considerations apply to the transmission time, although typically with much less variability, to the point that transmission time for two messages published within a small time frame can be considered equal.

In summary, considering two messages m_1 and m_2 that might give rise to a FIFO violation, we model their short-term variable delays $vardelay(m_1)$ and $vardelay(m_2)$ as two independent and identically distributed random variables whose distribution depends essentially on the processing time in brokers. (We validate this model experimentally and discuss our findings in Section 3.5.1.) Thus, our goal is to characterize this distribution in general, and then to measure and parametrize it at run-time.

3.2.2 Measuring delay differences

Measuring end-to-end delays with a significant precision requires synchronized clocks, and therefore is not practical outside of a tightly controlled environment. On the

other hand, the *difference* between the delays of two messages m_1 and m_2 can be readily computed, without synchronized clocks, using the time stamps associated with messages. In practice, a receiver stores the departure time $departure(m_i)$ stamped on m_i by the sender, records its arrival time $arrival(m_i)$, and also records departure and arrival times, $departure(m_{i-1})$ and $arrival(m_{i-1})$, for the previous message m_{i-1}. With this information, it computes $delay(m_i) - delay(m_{i-1}) = [arrival(m_i) - arrival(m_{i-1})] - [departure(m_i) - departure(m_{i-1})]$. The crucial point here is that, by subtracting a departure time from a departure time, and an arrival time from an arrival time, the result is not affected by the lag between the two clocks. We do assume though that the imprecision due to clock drift during delivery is negligible.

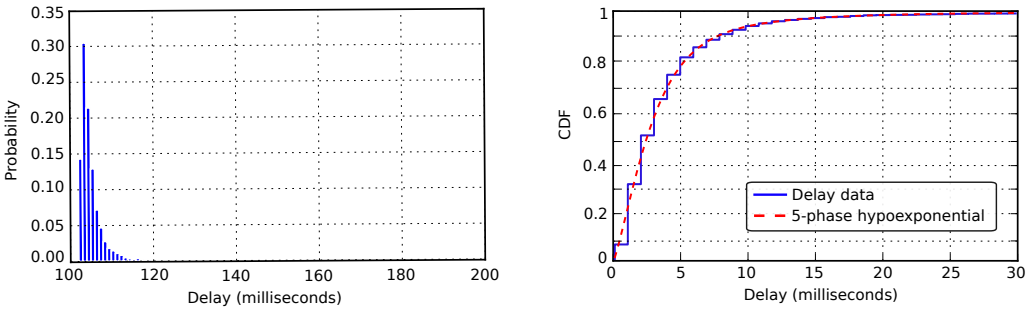
In summary, a receiver can measure the distribution of the difference between end-to-end delays, and can then use it as the basis for the estimation of the probability of FIFO violation and the estimation of the optimal latch time. In our model, any two messages that a subscriber receives from a specific publisher go through the same route and hence the same number of brokers. Consider two messages m_x and m_y received by a subscriber that is k brokers away from the publisher. Subtracting the delay of two messages cancels out the constant component of the delay and the subtraction reduces to subtracting two random variables. Writing each variable in terms of its components we have:

$$\begin{aligned} &delay(m_x) - delay(m_y) = \\ &(X_1 + X_2 + \cdots + X_k) - (Y_1 + Y_2 + \cdots + Y_k) = \\ &(X_1 - Y_1) + (X_2 - Y_2) + \cdots + (X_k - Y_k) \end{aligned} \tag{3.2}$$

where X_i and Y_i, $1 \leq i \leq k$, are the independent and identically distributed random variables representing the processing time of messages m_x and m_y at each broker i. Observe that in the last form of Equation (3.2) each term of the summation (i.e., $X_i - Y_i$) is itself the difference between two independent and identically distributed random variables and hence is a symmetric random variable with a mean of zero.

As such, without making any further assumption about any of the random variables involved in this equation we can find probabilistic bounds on the value of the above delay difference using probabilistic inequalities such as Chebyshev's inequality. In more specific cases, when the broker-hop count is known (e.g., as a field in the message header similar to IP header) we can also use Bernstein inequalities or Hoeffding's inequality to find better bounds. This is indeed of great advantage because as we will detail later, to find the latch time we need to find the probability of delay difference being more than a given value. Furthermore, as the number of terms in the second form of Equation (3.2) grows, the resulted distribution converges to a Normal distribution, due to the central limit theorem. Thus, we can also use properties of Normal distribution to find probabilistic bounds on the resulted summation.

Here, instead of using the aforementioned probabilistic inequalities, we focus on finding a more accurate characterization of the delay difference distribution and how



(a) (b)

Figure 3.4. (a) End-to-end delays for a sender/receiver pair 3 brokers apart. (b) Cumulative distribution of end-to-end delay samples fitted in a 5-phase hypoexponential distribution.

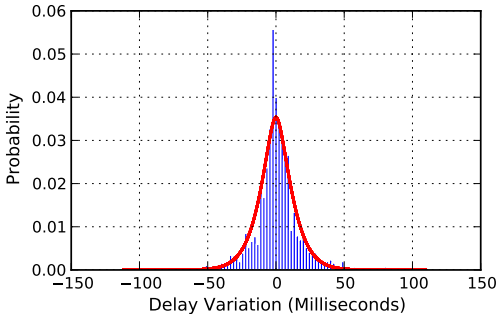


Figure 3.5. Histogram of the delay difference for a sender and receiver separated by 5 brokers. The thick line is the approximation with the sum of two Laplacian random variables.

to measure its parameters. To this end, finding the distribution of end-to-end delays will enable us to find the distribution of delay differences. In the next section we will analyze the distribution of end-to-end delays in more detail.

3.2.3 End-to-end delay distribution

In order to model the difference between end-to-end delays, which is the observable distribution for a receiver, we start from the distribution of end-to-end delays. In the context of IP networks, various researchers have proposed different methodologies and distributions to model end-to-end IP-level packet delay. For instance Zhang et al. found that a power-law distribution offers a good model [ZGG05], while Mukherjee [Muk92] reported that Internet packet delays can be represented by a shifted Gamma distribution whose shape and location factor depend on traffic load and path length.

To extend some of these results to the case of content-based publish/subscribe

systems, and to study the dynamics and distribution of end-to-end delays, we conducted experiments with a variety of parameters such as network size and topology, subscription and publication patterns and rates, and link delays. We then synchronized publishers and subscribers via NTP up to a clock difference of less than one millisecond, which enabled us to accurately measure the end-to-end delay of messages with a negligible error.

Our first observation is that neither Gamma nor power-law distributions properly fit the traces of the end-to-end message delay. Figure 3.4a shows the delay distribution of messages received by a subscriber from a publisher through 3 brokers. The inter-broker links have an assigned delay of 50ms. The links that connect subscriber and publishers to their local brokers have no delay. Since there are two inter-broker links, all the delays have a constant component of 100 milliseconds. This distribution has two pronounced characteristics: a long tail, which is composed of low frequencies at large values, and a large density around its mean.

As mentioned in Section 3.2.1, the variable component of the delay of a message is the sum of the processing delays at all the brokers it passes through. So, we start the analysis of the distribution of end-to-end delays (Figure 3.4a) with a specific experiment to measure the distribution of processing times in a single broker. In a typical setup with a few brokers and 10 clients per broker, we observed that most messages take a few milliseconds to be processed while a few of them need longer processing times. More specifically, in the case of our subject system B-DRP, measurements with different combinations of workload parameters (number and sizes of subscriptions, number and sizes of messages) reveal that the processing time is best fit by an exponential distribution.

We therefore proceed to model the variable component of the end-to-end delay as the sum of n exponentially distributed random variables, where n is the number of brokers between the publisher and the subscriber. This distribution is called a *hypoexponential distribution* which is a member of a general class of distributions called *phase type distributions*. To test this modeling hypothesis, we used the method described by Asmussen et al. [ANO96] to fit the measured end-to-end delay in a hypoexponential distribution with the appropriate number of phases, where a phase corresponds to a hop in the network. Before fitting each data set into the distribution, we removed the constant component of the samples (i.e., the delay caused by the inter-broker links). We performed the sampling and fitting process with a variety of configurations and different number of brokers and topologies with diameters of up to 10. In all cases, the data closely follow the theoretical distribution.

Figure 3.4b shows the cumulative distribution function of 6500 samples of end-to-end delay measurements, for a given publisher-subscriber pair, fitted into a hypoexponential distribution with 5 phases (in the experiment, the publisher and the subscriber were 5 brokers apart). Next, we detail how we use this model to find the distribution of delay *differences*.

3.2.4 Distribution of delay differences

We established that the end-to-end delay of a message is a hypoexponential random variable, resulting from the sum of exponentially distributed random variables, each representing the processing time at a broker.

Therefore each term in the second form of the Equation 3.2 (i.e., $X_i - Y_i$) is the difference of two identically distributed exponential random variables, which is known as a *Laplacian* random variable. It follows that the distribution of differences of end-to-end delays is the sum of independent Laplacian random variables. The probability density function of a Laplacian random variable is $f(x) = \frac{1}{2b} e^{\frac{-|x-\mu|}{b}}$ where μ is the mean of the distribution and b is its scale parameter. Due to the linearity of expectation, μ is zero for all of the above Laplacian random variables that are the result of subtracting two exponentially distributed random variables (e.g., $X_i - Y_i$). This is because all the brokers run the same forwarding algorithms, thus we can assume that b is similar for all the Laplacian random variables.

So, our analysis shows that the difference between end-to-end delays for a given publisher/subscriber pair can be modeled as the sum of k Laplacian random variables, where k is the number of hops between the publisher and the subscriber. However, unfortunately, determining an analytical expression of the distribution of the sum of $k > 2$ Laplacian random variables with different scale parameters is still an open problem [NK05]. So, as an approximation, we use the statistical properties—namely, the probability distribution, cumulative density, and quantile functions—of the sum of *two* Laplacian random variables. Even though this model is an approximation, we have empirical evidence that the two-sum distribution also fits reasonably well the sum of up to 8 Laplacian random variables. The sum of two independent and identically distributed Laplacian random variables with mean $\mu = 0$ and scale factor b has probability density $f(x) = \frac{(|x|+b)}{4b^2} e^{-\frac{|x|}{b}}$ and cumulative distribution $F(x) = \Pr[X < x]$ for $X > 0$:

$$F(x) = 1 - \frac{(2b + x)}{4b} e^{-\frac{x}{b}} \qquad (x > 0) \tag{3.3}$$

The estimation of the scale factor b based on a set of n samples is possible with maximum likelihood estimation, which yields $b = \frac{2}{3n} \sum_{i=1}^{n} |x_i|$. In Appendix A we illustrate the steps we took to derive Equation A.3 and how we estimate parameter b in this equation.

Figure 3.5 shows the histogram of delay differences for messages received 5 hops away from the sender. The thick line represents the sum of two Laplacian random variables whose parameter is estimated from the data. The sharp spike around zero falls outside of the approximate distribution because of the approximation of sum of 5 random variables to only two. In other words, as the number of broker-hops increases, the density of the real distribution increases around the mean and the tails become shorter, while the approximation is less dense around zero but has longer tails. This does not cause a problem though, since in determining the latch time, the likelihood of

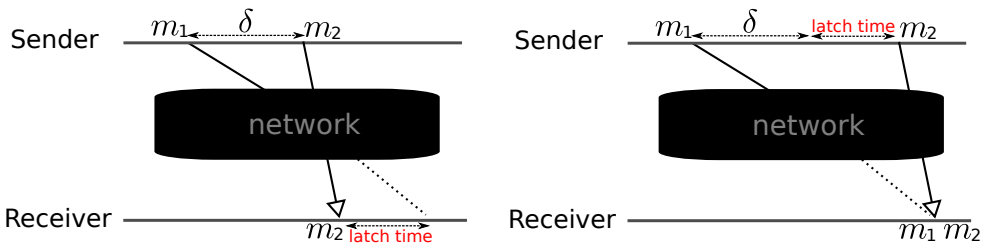


Figure 3.6. By virtually increasing δ, we avert a FIFO violation

the extreme values of delay difference is used (i.e., the tails of its distribution) which we will detail next.

3.2.5 Determining the latch time

Based on the model we developed, we now go back to Equation (3.1) to estimate the probability that m_1 and m_2 are received out of order (a FIFO violation). This probability is a function of the difference between their departure times, $\delta = departure(m_1) - departure(m_2)$. In particular,

$$\begin{aligned}\Pr[\text{FIFO Violation}] &= \Pr[delay(m_1) - delay(m_2) > \delta] \\ &= 1 - \Pr[delay(m_1) - delay(m_2) < \delta] = 1 - F(\delta)\end{aligned}$$

where $F(\cdot)$ is the cumulative distribution of delay differences.

Whenever the receiver detects a gap in the sequence numbers, it can virtually increase δ by latching the messages whose delivery would cause the FIFO violations so that the probability of a FIFO violation drops below a given threshold (Figure 3.6). More precisely, we would like to determine a latch time τ that reduces the FIFO violation probability below a given threshold P_t for a pair of messages m_1 and m_2 published δ time units apart from each other. Thus

$$\tau = F^{-1}(1 - P_t) - \delta \tag{3.4}$$

where F^{-1} is the quantile function of the delay variation. Intuitively, τ is the *minimum* amount of time that the receiver has to hold m_2 and wait for the missing message m_1 based on the sampled delay difference. We call P_t the *FIFO violation coefficient*. Higher values of P_t map to smaller latch times and more FIFO violations. F^{-1} is the inverse of Equation (A.3) and corresponds to

$$F^{-1}(p) = b[\omega(4e^{-2}(p-1)) + 2] \qquad (0.5 \leq p \leq 1) \tag{3.5}$$

where $\omega(\cdot)$ is the *Lambert Omega Function*, and can be efficiently computed using several existing numerical methods. Appendix A shows how the above equation is derived.

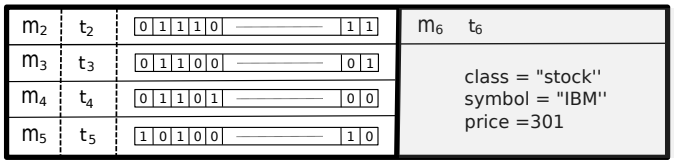


Figure 3.7. Message m_6 carrying a publication record of size 4

Now let us consider cases with more than one message missing (e.g., a receiver receives message m_6 immediately followed by m_{10}).

Let $\Phi(m,n)$ denote the occurrence of a FIFO violation for messages m and n, let $\delta_{m,n} = departure(m) - departure(n)$ denote the time difference between the publication time of two messages m and n, and let $\tau_{m,n}$ be the latch time given by Equation (3.5) for messages m and n. Since $\delta_{10,9} \leq \delta_{10,8} \leq \delta_{10,7}$ it follows that $\Pr[\Phi(m_9,m_{10})] \geq \Pr[\Phi(m_8,m_{10})] \geq \Pr[\Phi(m_7,m_{10})]$ and therefore $\tau_{9,10} \geq \tau_{8,10} \geq \tau_{7,10}$.

In words, in this probabilistic model, the latch time is independent of the number of messages in a chain of missing messages. In such cases, in order to calculate the latch time, the receiver only considers the time difference between the latest received message and the latest missing message.

3.2.6 Publication record

So far, we have assumed that whenever there is a gap in the message sequence number, the missing messages would match the interests of the receiver. This assumption enforces the assessment of a latch time upon every message that causes a gap in the sequence, even when the missing messages are not even supposed to be received because they do not match the subscriber's interest. Obviously, this may introduce unnecessary delivery delays.

To eliminate (or reduce) this problem, we propose to attach to each message some information about previously published messages along with their publication timestamps (see Figure 3.7). We call this information the *publication record* of the publisher. As a simplistic example, consider attaching to each message a copy of the previous 4 messages sent by the same publisher. In this case, a receiver receiving m_6 right after m_2, and therefore detecting a gap of three messages, might be able to deliver m_6 immediately after checking that none of the missing messages (whose Bloom filter is attached to m_6) matches its subscriptions. The question then becomes how to compile a compact and yet informative publication record.

In topic-based publish/subscribe systems this is easily achieved by attaching the topic of the last k messages to each new publication. Things are not as simple in content-based publish/subscribe systems, although it is possible to attach a summary of the content of the previous k messages. A good encoding for this summary is a message representation based on Bloom filters that was developed for B-DRP and we

introduced in Section 2.7.

The salient properties of this encoding, which are detailed in [CTCHW09], are that it is compact and it admits to a fast matching algorithm, but it may incur false positives, meaning that an encoded message may be found to match the interests of the receiver while the original message would not. This does not compromise correctness but may lead to unnecessary delays. Nevertheless, given that in general only a small percentage of the publications of a publisher match the interests of a given subscriber, in most cases this simple method is effective in preventing unnecessary delivery delays. We call this the *enhanced mode* of the probabilistic FIFO ordering protocol as opposed to the *basic mode* in which messages do not carry any publication record.

A publication record is attached to a message m_k by its source s and consists of R entries representing the previous R messages $m_{k-1}, m_{k-2}, \ldots, m_{k-R}$ published by s. Each entry B_i is a Bloom filter obtained by encoding message m_{k-i}. The encoding works as follows: first, a message m is mapped into a set of "categories" or "tags." For example, a message that contains the attributes (event=disk-failure, cause=overheating, priority=high) might be associated with tags "disk-failure," "overheating," and "high-priority." Then the set of tags is simply represented as a Bloom filter. In addition to defining sets of tags for messages, the encoding scheme also defines tags for subscriptions with the intended semantics that, if a message m matches a subscription S, then the tags associated with m are a *superset* of the tags associated with S. In summary, a subscription S is encoded as a Bloom filter B_S (representing a set of tags) and a message m is encoded as a Bloom filter B_m (representing a set of tags), and if m matches S then $B_m \supseteq B_S$ (where a Bloom filter B is interpreted as a set of bits).

When a subscriber receives a message m_k that carries a publication record $\langle B_1, \ldots B_R \rangle$ the subscriber checks whether it has received messages $m_{k-1}, \ldots, m_{k-R}$ from the same publisher. Then, for each message m_i that was not received, the subscriber checks whether the B_i entry in the publication record matches any of its subscriptions S. That is, the subscriber checks whether there is one of its subscriptions S such that $B_i \supseteq B_S$. (Of course, the subscriber does not have to recompute the encoding of its subscriptions for each message.) If one such subscription is found for B_i, then the subscriber concludes that message m_{k-i} was of interest and may decide to latch message m_k (see Section 2.7 for a formal definition of covering relation).

As mentioned at the end of Section 3.2.5, when the sequence number gap contains more than one message, in basic mode the receiver has to consider only the latest missing message. Instead, in enhanced mode, the receiver has to consider only the latest missing message that is found to match local subscriptions. Referring to the example where a receiver receives message m_6 immediately followed by m_2, if the receiver detects that m_5 does not match local interests (through the publication record attached to m_6) but m_4 is of interest, it takes m_4 into account to calculate δ in Equation (3.4) since m_5 will not be received anyway.

The size of the publication record attached to each message is controlled by the

publisher. A larger record translates into shorter delivery delays, at the expense of a greater bandwidth consumption. In general, high publication rates require large publication records because the interval between publications is smaller and thus the probability of out-of-order deliveries is higher.

3.3 Loss Detection

A publication record that is attached to each message has a second purpose in our transport protocol, that is loss detection. Recall that a publication record of size k determines which one of the latest k published messages are of interest to a receiver (match its subscription). Assume for instance that the publication record attached to message m_i determines message m_j to be of interest while m_j itself is not received. In this case, the FIFO ordering protocol latches m_i for a time given by equation Equation (3.4). After this time, m_i is delivered and m_j is determined to be lost. This event is then sent to reliability and congestion control components of the transport protocol (if they are enabled) to recover the lost message or enable congestion control policies if necessary.

A major limitation of this loss detection method stems from the limited size of publication record. In other words, with a publication record of size k a receiver is able to detect only the k latest published and lost messages. We will discuss this limitation and its effects on reliability and congestion control in the next two chapters.

3.4 Algorithmic Description

Algorithm 3.1 shows the core of our probabilistic FIFO ordering mechanism. A receiver (subscriber) executes a separate instance of this algorithm for each sender (publisher) from which it receives messages, and maintains a separate set of variables associated with that sender. The configurable parameters of the algorithm are the FIFO violation coefficient P_t and the size q of the ring buffer that stores the delay-difference samples.

The main algorithm (starting on line 1) is executed for each message received from the publisher. A variable called *base* stores the previously received message with the highest sequence number. The algorithm starts by computing the delay difference with respect to *base*, and uses that to update the parameters of the distribution of delay difference (line 7). If the received message m_n causes a gap in the sequence number, the algorithm assesses a latch time based on the difference in departure time between m_n and the latest missing message that is known (in enhanced mode) or assumed (in basic mode) to match the local subscriptions. In particular, in basic mode the relevant message is assumed to be the immediate predecessor m_{n-1} whose departure time is attached to m_n. Conversely, in enhanced mode, the receiver looks in the publication

```
1: procedure initialize
2:     base ← ⊥                                   {last in-order received message}
3:     β ← 0                                      {scale factor of the distribution of delay difference}
4:     Q ← RingBuffer(q)                          {ring buffer of size q}

5: upon receiving m_n from publisher do
6:     if base ≠ ⊥ then
7:         x ← [arrival(m_n) − arrival(base)] − [departure(m_n) − departure(base)]
8:         update(x)                              {update the distribution}
9:     n ← sequence(m_n)
10:    if n = 0 or n = sequence(base) + 1 then
11:        schedule(m_n, 0)                       {schedules m_n for immediate delivery}
12:    else if n < sequence(base) then
13:        schedule(m_n, 0)
14:        return
15:    else
16:        if publication_record(m_n) ≠ ⊥ then
17:            R ← publication_record(m_n)
18:            m_* ← latest message in R that matches local subscriptions and was not
               already received
19:            if m_* = ⊥ then
20:                m_* ← earliest message in R
21:            if sequence(m_*) < sequence(base) then
22:                schedule(m_n, 0)               {none of the missing messages is relevant, deliver m_n now}
23:                base ← m_n
24:                return
25:            t_* ← read departure(m_*) from R
26:            δ ← departure(m_n) − t_*
27:        else
28:            t_{n−1} ← read departure(m_{n−1}) from m_n
29:            δ ← departure(m_n) − t_{n−1}
30:        τ ← latch_time(δ)
31:        schedule(m_n, τ)
32:    base ← m_n

33: function latch_time(δ)
34:    τ ← β[ω(4e^{−2}(−P_t)) + 2] − δ            {(P_t ≤ 0.5)}
35:    return τ

36: procedure update(x)
37:    Q.append(x)                                {add the sample to the ring buffer}
38:    β ← (2/3q) Σ_{x_i∈Q} |x_i|                 {update scale factor of the distribution}
```

Algorithm 3.1. Probabilistic ordering FIFO algorithm run by a recipient for each publisher

record (attached to m_n) for the latest missing message that matches its subscriptions (line 18). If no such message is found in the publication record then the receiver conservatively assumes that the latest matching message was published at the same time as the earliest message in the publication record (line 20).

For instance, suppose that the size of the publication record is three and the recipient receives m_{10} but the last 5 consecutive messages prior to m_{10} (e.g., m_5 to m_9) are missing. The publication record attached to m_{10} covers the last three messages (m_7, m_8, and m_9). If none of these three messages appears to match the interests of the recipient, the algorithm assumes that message m_6 matches local interests and assumes that its publication time equals the publication time of its proceeding message, i.e., $departure(m_6) = departure(m_7)$.

Having computed an appropriate latch time τ, the receiver schedules the message for delivery to the receiver application. This is done through a procedure $schedule(m, \tau)$ that also assures the ordered delivery of all scheduled messages. This procedure (not shown in the listing) is analogous to the mechanisms found in most sliding-window protocols. In essence, the scheduler maintains a queue of pending messages and a set of corresponding timers. A message is queued until its timer expires or all earlier messages have been delivered, at which point the message is also delivered to the application and removed from the queue. Also, in the enhanced mode of the protocol, if a relevant message is not received before a timer expires, then a message loss is signaled to both reliability and congestion control components for further actions. The signal also contains relevant information about the lost message like its sequence number and its Bloom filter for usage in the recovery process.

Since the ordering protocol is probabilistic, the latch time of a set of messages may not be large enough to cover the time needed for the missing messages to arrive. In such cases, the timer associated with the latched messages expires and the messages are delivered. When the missing messages are received, they might be simply dropped or delivered to the application, thereby causing a FIFO violation (see line 13). We have implemented this treatment as a configurable parameter of the ordering protocol.

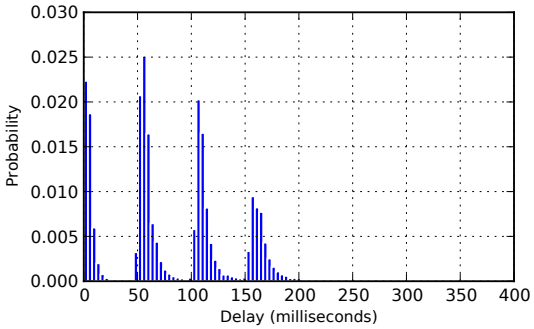


Figure 3.8. Distribution of end-to-end delay for all of the delivered messages during the first 90 seconds of the experiment.

3.5 Evaluation

We implemented the recovery protocol as a pluggable module which integrates into any publish/subscribe application and protocol. Specifically, the publication record and other metadata that is required by the ordering protocol is attached to messages as an array of bytes, perceived by brokers as application payload.[3] Our experiment testbed is a cluster of Dell PowerEdge with two dual-core 2GHz AMD Opteron processors and 4GB of main memory running Linux with a 2.6 kernel. Connectivity is through an isolated high-throughput Gigabit Ethernet switch. Broker software, client, and transport protocol are run on the 64-bit open-JDK VM.

We now present the experimental evaluation of the proposed FIFO ordering method. This evaluation addresses three high-level questions. First, it validates the statistical models, developed in Section 3.2, upon which the method is built. Second, it evaluates the benefits, costs, and scalability of the method in its basic and enhanced form. Third, it evaluates the ability of the method to respond and adapt to dynamic workloads.

3.5.1 Network delay model validation

This first experiment corresponds to the scenario described in Section 3.1. However, whereas in Section 3.1 we compared global statistics of throughput and FIFO violations for ActiveMQ and B-DRP, here we focus on B-DRP (because we are interested in modeling best-effort delivery) and look closer at the distribution of end-to-end delays and their variations.

Figure 3.8 shows a histogram of the end-to-end delay for nearly 250000 messages that were delivered across the network during the first 90 seconds of the experiment. Recall that this initial phase is characterized by a slow (and slowly growing) flow of publications. Below we also examine the later phases of the experiment when high-rate

[3]Most implementations of the Java Messaging Service (JMS) are capable of carrying an opaque payload.

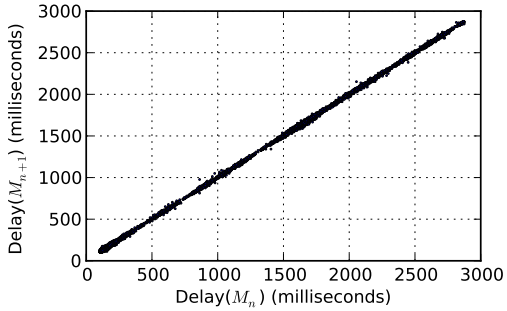


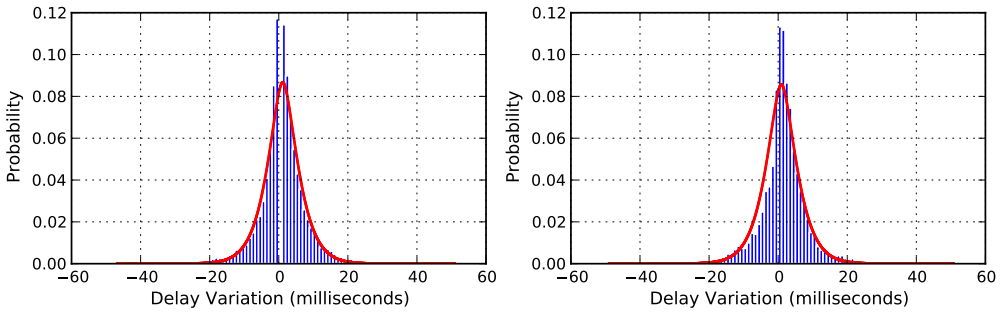
Figure 3.9. End-to-end delays of every two consecutive messages for a chosen pair of sender and receiver.

publishers cause congestion, and therefore cause a significant shift in the delay distribution. The histogram shows 4 distinct clusters corresponding to the hop-distance between publisher and subscriber. For example, messages that pass through 3 brokers (two broker-to-broker hops) have a transmission delay of around 100 milliseconds. The histogram of Figure 3.8 is qualitatively consistent with our model of end-to-end delay. Now, in order to validate the model more precisely, we isolate a single publisher/subscriber pair and measure end-to-end delays and delay differences.

We first examine the delay of pairs of consecutive messages recorded over the entire duration of the experiment. The results are reported in the scatter-plot of Figure 3.9. The plot highlights two facts. First, the delays of two consecutive messages are highly correlated; second, the delays vary significantly throughout the experiment, and since the data refers to a single sender/receiver pair, this indicates the effect of significant queuing delays. We also note that we purposely select a sender/receiver pair that experiences an intense flow of messages that ultimately causes congestion in the intermediate brokers.

We now take a closer look at the effect of delays and congestion on delay variation. In particular, we test our intuition that queuing delays do not have any substantial effect on the distribution of delay variation. To do that, we compute the distribution of delay variations between consecutive messages for pairs of messages subject to delays within two ranges, corresponding to the areas marked with dotted lines in Figure 3.9. The resulting histograms, plotted in Figure 3.10, demonstrate that the delay variations are essentially independent from the delay.

To confirm this visual analysis, we use Wilcoxon rank-sum test to test whether this data is consistent with our hypothesis that the two datasets follow the same distribution. The resulted p-value of the Wilcoxon test is 0.35 which confirms that the evidence is compatible with our hypothesis with a high statistical significance.



(a) (b)

Figure 3.10. Delay variation distribution for messages with end-to-end delay of (a) $delay(m) \leq 1500ms$ and (b) $delay(m) > 1500ms$ respectively.

3.5.2 Effectiveness of the ordering protocol

We evaluated the effectiveness of the ordering protocol through various experiments. In general, these experiments are intended to measure both the reduction in FIFO violations and the additional latency incurred by the protocol. Specifically, to characterize the trade-offs between these benefits and costs, and also to obtain a comparative baseline, we juxtapose the performance of our probabilistic protocol with that of a simpler protocol that uses a static latch time. This protocol latches each message that creates a gap for a fixed amount of time. However, to obtain the most conservative comparison, we first select the parameters of our probabilistic protocol and measure its performance in terms of FIFO violations, and then configure the static protocol with the *optimal* latch time that achieves the same (or nearly the same) level of FIFO violations. (We determine the optimal static latch time experimentally with a trial-and-error search.)

We set up and perform each experiment so that receivers run multiple instances of static and probabilistic ordering protocols with different parameters. This enables us to compare the efficiency of the protocols and the effect of different parameters in the exact same scenario.

We report the results of our experiments in Figures 3.11 and 3.12 for an 8-broker setup and Figures 3.13 and 3.14 for a large scale experiment with 46 brokers. Each data set corresponding to the static protocol is labeled "CST-t," where t is the constant latch time (milliseconds); and each set of the probabilistic FIFO ordering protocol is labeled "P-x-y," where x is the probabilistic FIFO coefficient P_t, and y is the number of previously published messages whose encoded Bloom filters are attached to each message (in the enhanced version of the algorithm). The probabilistic FIFO ordering protocol also uses a sample buffer Q of size 25 in all the experiments, and uses an encoding of the publication record that uses 16 bytes per message, so for example, "P-0.2-5" and "P-0.2-25" indicate experiments in which the enhancement of the publi-

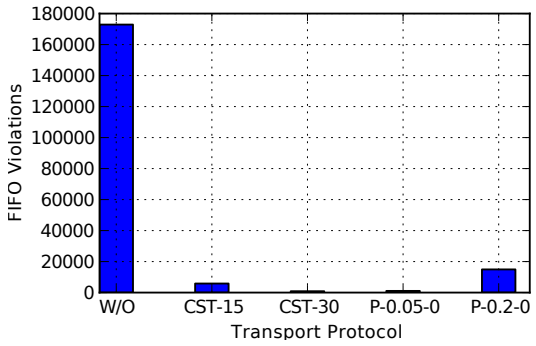


Figure 3.11. Effectiveness of different FIFO algorithms in an 8-broker setup: the total number of incurred FIFO violations with and without ordering.

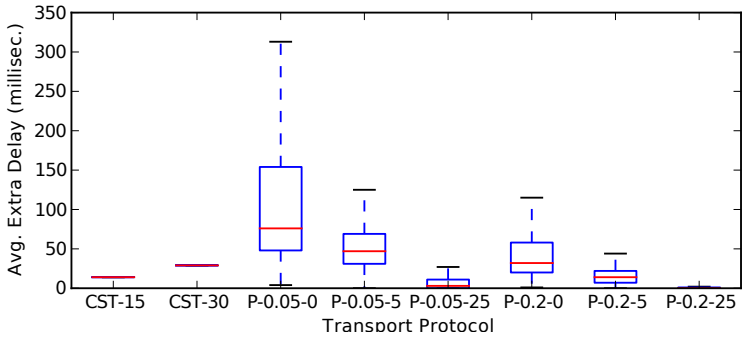


Figure 3.12. Effectiveness of different FIFO algorithms in an 8-broker setup: the average extra delay caused by different ordering algorithms.

cation record introduces an overhead of 80 and 400 bytes, respectively.

Figure 3.11 compares the total number of FIFO violations observed by 400 receivers, with and without ordering mechanisms. Note that for a given P_t, the size of the publication record does not have any effect on the performance of the protocol in terms of the number of reduced FIFO violations. Recall that the use of the publication record allows the receiver to assess a reduced latch time, but does not change the behavior of the protocol in terms of FIFO violations. Hence, we only plot the number of FIFO violations for the basic mode of the probabilistic protocol. For example, in Figure 3.11 with P_t equal to 0.05, the number of FIFO violations for P-0.05-0 is the same for P-0.05-25; what differs is the extra delay, as we will show later. In total there were almost 173000 out-of-order receptions. The probabilistic FIFO ordering with P_t set to 0.05 mitigated more than 99% of the FIFO violations, performing as well as CST-30. With $P_t = 0.2$, the probabilistic FIFO algorithm reduced FIFO violations by about 91%.

To demonstrate the benefit of the probabilistic FIFO ordering algorithm with respect to minimizing the overall delivery delay when compared to the static protocol, we first define a measure we call the *average extra delay*. If M is the set of all the mes-

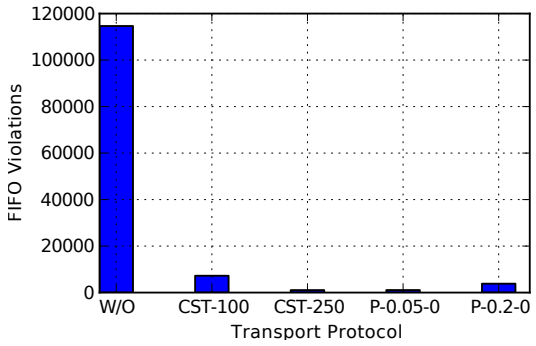


Figure 3.13. Effectiveness of different FIFO algorithms in a 46-broker setup: the total number of FIFO violations with and without ordering.

sages that a given subscriber received from a given publisher, we define the average extra delay for the messages of that stream as $\frac{1}{|M|}\sum_{m\in M} latch\_time(m)$.

Figure 3.12 presents the average extra delay caused by the static and probabilistic FIFO algorithms for all of the pairs of sender and receiver. Unsurprisingly, the static algorithm with constant latch time induces the same average extra delay for all nodes. P-0.05-25 performs better than CST-30 in terms of average extra delay. P-0.05-0 (basic mode) causes much larger extra delays, but when enhanced with a publication record of only 5 messages (P-0.05-5), it incurs a considerably smaller extra delay. Note that the large average extra delay in basic mode is largely due to the intentionally extreme scenario in which two high-rate publishers are combined with subscribers configured to receive a very high portion of their publications (more than 80%). This is an extreme case that is relatively uncommon in a publish/subscribe network. In other words, whenever the publisher sends messages at lower rates, the difference in the average extra delay between the static protocol and our adaptive protocol increases significantly. This is because the adaptive protocol is sensitive to the time difference between two consecutive publications while any static protocol latches messages irrespective of the publication rate.

To demonstrate the performance of the protocol in larger networks, in Figures 3.13 and 3.14 we present the performance of the probabilistic ordering in a network of 46 brokers where each broker serves 10 clients. We use a workload with similar characteristics to the 8-broker experiment, except that this one runs for 240 seconds. We choose a low-degree network topology with a graph diameter of 15. Due to this larger diameter, constant latch times of 30 or 50 milliseconds are not effective. Indeed, like in the previous experiments, we ran several setup experiments to find the optimal constant latch time for our comparative analysis. On the other hand, the adaptive nature of our algorithm captures very well the dynamics of larger-diameter networks. In this experiment, there are more than 115000 out-of-order receptions without any ordering algorithm in place, and our probabilistic ordering algorithm with $P_t = 0.05$ is able to

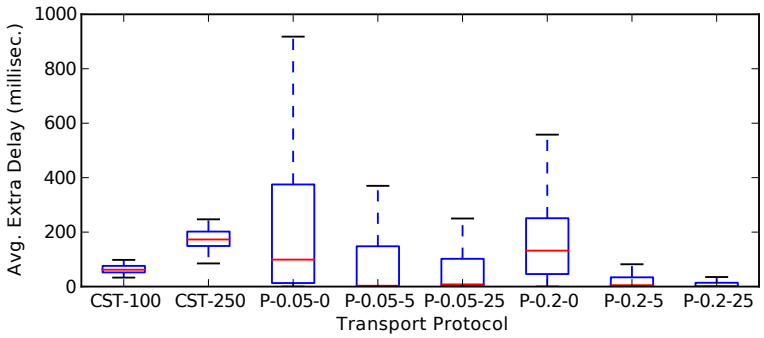


Figure 3.14. Effectiveness of different FIFO algorithms in a 46-broker setup: the average extra delay caused by different ordering algorithms.

prevent 99.5% of such FIFO violations. This mitigation in basic mode comes at a cost of a maximum 960 milliseconds in average additional delivery delay, while with a publication record of 25, this cost is nearly zero for more than 50% of the nodes and less than 190 milliseconds for 90% of them.

3.5.3 Adaptivity of the protocol

Figure 3.15 shows the dynamics of the FIFO-ordering protocol for a publisher/subscriber pair in response to changes in publication rate. The top frame pin-points out-of-order deliveries in the message stream; the second frame shows the publication rate of the publisher (messages per second); the third frame plots the changes in FIFO-violation probability calculated by our algorithm; and the two bottom frames show the changes in latch time when the publication record is 0 (basic mode) and 25 (enhanced mode).

The FIFO violation probability and the latch time follow the trend in the changes of publication rate. This is the result of delay variation and change of time gap between two consecutive messages. Observe that in the basic version of the protocol, where there is no publication record attached to messages, the latch time spikes more frequently, for there are many cases when the missing message does not match the subscriptions of the receiver but the ordering algorithm latches the messages for that specific time frame.

3.6 Conclusion

In this chapter we presented our approach to enhancing FIFO ordering in best-effort, content-based publish/subscribe networks. Our general idea is to implement an end-to-end, probabilistic algorithm to avoid FIFO violations. More specifically, first we studied and modeled the causes of FIFO violations, and showed experimentally that the major cause of FIFO violations is the variation in end-to-end delays. Then, based on

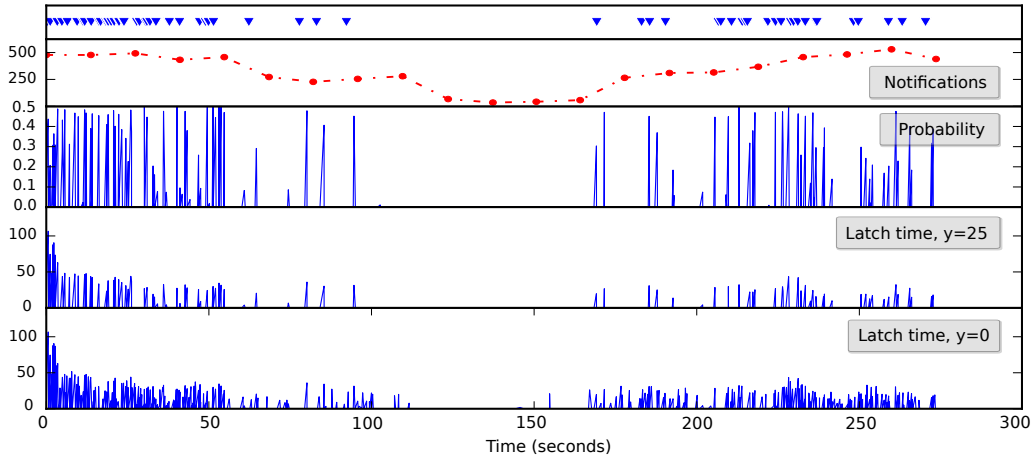


Figure 3.15. From top to bottom: timestamps of out of order receptions; publication rate; probability of a FIFO violation; latch time in enhanced mode with a publication record of size 25; latch time in basic mode.

a simple analytical model of the end-to-end delay, we developed a method to quantify its variation, which we also validated experimentally. This allowed us to devise an algorithm to estimate the probability of a FIFO violation whenever there is a gap in the sequence number of an incoming message stream. The same estimation also allows us to find an adequate latch time for some of the received messages in order to reduce the FIFO-violation probability below a desired threshold. Through experiments, we showed that this method can mitigate up to 99.5% of the FIFO violations while keeping the unnecessary delivery delay to a minimum.

Chapter 4

Reliability

In this chapter we introduce and detail the second component of our transport protocol that provides end-to-end reliability for best-effort content-based networks. By reliability, we mean a type of message persistence which enables subscribers to recover lost messages. Obviously, an end-to-end approach requires the involvement of end-points in providing message persistence and a collaborative mechanism for the recovery of lost messages. In designing the end-to-end reliability protocol we had a few key requirements in mind. First, for scalability reasons, it should require the least compromise of the loose coupling between publishers and subscribers. For instance, direct communication between end-points should only happen as a reaction to a message loss, instead of a correct message reception (NACK instead of ACK). Second, given the multicast nature of content-based communication, the protocol should incorporate mechanisms for efficient recovery of shared losses (cases where a lost message has multiple intended receivers). Third, the recovery protocol should be flexible enough to allow for optimizations like location aware caching/retrieval of messages for performance and efficiency reasons.

These requirements are common to any reliability scheme for multicast protocols. Indeed, in the development of our reliability protocol we borrow ideas from reliability protocols for IP multicast, namely, Scalable Reliable Multicast (SRM) [FJL$^+$97]. Briefly, SRM works as follows: a receiver that detects a message loss (using sequence numbers) tries to recover a copy of that message from other group members by multicasting a request to the group. A group member that has already received and cached the message will send a copy of the message back to the requesting node. As we will detail later, SRM's properties make it an attractive basis for our reliability protocol. However, SRM is not directly applicable to content-based publish/subscribe networks because messages are not explicitly addressed to a multicast group, and therefore it is not immediately clear how to address a request and replies for a lost message. In fact, for the same reason, it is not even clear how to *detect* a message loss. This is because there is no clearly identifiable stream of messages other than what is published by a

single source, and gaps in such a stream are very often due to the legitimate filtering of the content-based selection. In other words, simple sequence numbers do not allow a receiver to distinguish a lost message from a message that was legitimately filtered out by the receiver's subscriptions.

In Section 3.3 we discussed a probabilistic loss detection mechanism based on a synthetic publication record that is attached to messages. This information allows receivers to detect message losses that occurred within a limited time frame in the past. In this chapter, we build a probabilistic reliability protocol atop this loss detection method and then study the effects of this imprecise loss detection on the performance of the reliability protocol. The major contribution of this chapter is to propose an approach to routing requests to relevant clients in the publish/subscribe network, in a manner similar to SRM, in order to retrieve a lost message. We also extend SRM with a simple scheme for better cache management. Finally, through experiments we show that the protocol is capable of recovering from a large number of message losses in networks of different size and with different dynamics.

In Section 4.1 we first set the context of our work by discussing reliable IP multicast, and then present the problem we address and give an overview of the reliability protocol we propose as a solution. In Section 4.2 we detail the reliability protocol and its implementation, with an in-depth discussion of its performance in Section 4.3. Section 4.4 presents the experimental evaluation. Finally in Section 4.5 we offer some concluding remarks.

4.1 Context and Preliminaries

Since our reliability protocol is inspired by reliable IP multicast, in this section we first give a summary about Scalable Reliable Multicast which is the basis of our protocol and then give a bird's eye view of the problem we intend to solve and the solution.

4.1.1 Reliable IP multicast

A best-effort content-based network offers a service that is similar in nature to IP multicast. Since our goal is to improve the reliability of a best-effort content-based network, and specifically since we propose to do that with a pure end-to-end solution, we model our solution after the existing end-to-end reliability protocols developed for IP multicast. As we will discuss later, such protocols are not immediately applicable to a content-based addressing because of its greater expressiveness. Nevertheless, we review such protocols here and in particular we focus on Scalable Reliable Multicast (SRM) [FJL+97] as a basis for our reliability protocol.

In the context of IP networks, a number of reliability mechanisms have been proposed in the form of additions to the standard IP multicast. Among the most notable ones, Scalable Reliable Multicast (SRM) [FJL+97] and Reliable Multicast Transport

Protocol (RMTP) [LS96] provide reliability without reliance on the routing infrastructure (i.e., "end-to-end") while Pragmatic General Multicast (PGM) [SCG+01b] proposes additional functionalities to routers in order to provide reliability. In the terminology of these protocols, a *request* is a (broadcast or multicast) message whose function and meaning is similar to that of a negative acknowledgment (NACK), which is to request a missing message. The term *repair* refers to a reply (to a request) that carries the missing message. We also use these terms in rest of this chapter.

We chose SRM as a basis for our reliability protocol for two reasons. First, SRM makes limited assumptions about the application logic, and second, it only relies upon a basic multicast service to recover lost messages without requiring any addition or modification to the underlying multicast service. Since in this chapter we often refer to SRM semantics and internals, we now give a cursory description of how SRM achieves end-to-end reliability. For a detailed description we refer the reader to the original paper by Floyd et al. [FJL+97].

SRM is a general-purpose reliability protocol designed for large-scale applications that use IP multicast. To be generic, SRM does not make any particular assumption about the formats and sizes of application-level messages. Thus, message loss detection is not embedded in the protocol but is instead assumed to be part of the application logic. Once the application detects the loss of a message within a group, it multicasts a *request* to the same group. Other applications in the same group that received (and cached) the message respond by multicasting a *repair* to the group. In order to reduce duplicate requests and repairs for the same message, nodes hold their requests and repairs for an initially random delay. A node holding a request would then progressively back off by doubling the delay every time it receives a request for the same message. A node holding a repair would simply cancel the repair upon receiving the same repair.

The random delays are chosen uniformly in the interval $[C_1 d_{s,a}, (C_1 + C_2) d_{s,a}]$ for request and $[D_1 d_{s,a}, (D_1 + D_2) d_{s,a}]$ for repair messages where $d_{s,a}$ is the average message trip-time from the multicast source to the node and C_1, C_2, D_1, and D_2 are adjustable parameters. The protocol has an algorithm for automatic tuning of these parameters so that the likelihood of duplicates and the time to recover lost messages are lowered to a minimum.

4.1.2 Problem and overview of the solution

To extend the simple idea of cooperative message recovery to content-based networks, we must overcome two main technical problems. The first problem is to enable receivers to *detect* message losses. For some applications, specifically when messages are channeled into identifiable streams and subscribers receive all messages in a stream, this can be easily done by marking each message within its stream with a sequence number. In this case, a gap in the sequence numbers indicates a message loss. However, such streams do not exist in a content-based publish/subscribe network, where messages are delivered only if they match the interests of subscribers, and where such

interests may partially overlap. In other words, with partially overlapping receivers' interests, it is impossible to assign sequence numbers to messages so as to obtain continuous sequences for all receivers. Therefore, in practice, a receiver can not distinguish a message that was lost from a message that was not delivered because it does not match the receiver's interests.

We address this problem by adding some information to each message that allows a receiver to determine, with some probability, if any of the latest publications of the sender that were not received was in fact of interest for the receiver. In Section 3.2.6 we briefly introduced this entity that we call *publication record* which consists of a set of Bloom filters, each encoding one of the sender's most recent publications.

The second problem is to *recover* a lost message that was determined to be of interest. As in SRM, we propose a cooperative recovery scheme whereby a lost message is recovered from some other application in the network that might have received and cached the message. In SRM a receiver would multicast a request for a lost message to the same multicast group to which the message was sent, which conveniently identifies all potential caches from which the message might be recovered. Unfortunately, in a content-based publish/subscribe network it is not as easy for a receiver to address other receivers of a given message. One way to reach potential caches is to send a request to *all* caches, which can be done by having all caching applications subscribe for a generic "request" message. However, that solution might incur an unacceptable overhead for caching applications and for the network in general, especially in situations where the network is already congested.

We address this second problem using the same publication record attached to messages. A receiver that receives a Bloom filter B_m from the publication record of a message m', and therefore determines that m was of interest, creates a request message that includes B_m. This request is intended for other applications interested in the same message m that are willing to serve as caches for lost messages, and therefore that would subscribe for request messages that match B_m. This subscription in effect creates a way to explicitly address such clients. Thus, any node that incurs loss of m can send a request to other potential receivers, by simply publishing a request for m.

Lastly, an application that has a cached copy of a lost message and that receives a request for a repair must somehow deliver that message to the requesting application. This can be done in a straightforward way through a direct (unicast) connection, or also through the publish/subscribe API by effectively republishing the message for the requesting receiver. Each of these solutions has advantages and disadvantages that we discuss below.

4.2 End-to-end Loss Recovery

4.2.1 Message loss detection

In Section 3.3 we detailed how the FIFO ordering component of the transport protocol performs loss detection. If the reliability protocol is enabled, upon every loss detection the incident is signaled to the reliability component with a method call that in turn the triggers recovery process for the lost message. The method call also passes information about the lost message that is required for its recovery, namely the Bloom filter and sequence number of the lost message (recall that an entry in the publication record is a Bloom filter representing a message). We refer the reader to Section 3.2.6 for details of the encoding scheme that we use in generating publication record.

4.2.2 Routing requests

A subscriber may try to recover a lost message by requesting a copy of that message from other applications that received and cached the message. The problem is then to address such a request so as to reach all and only the receivers that might have cached a copy of the lost message.

We propose to transmit the request through the same publish/subscribe system, by constructing a special request message and by having receivers subscribe for those requests. So, an application with subscription S that is willing to cache messages for other applications must subscribe for requests for messages matching S. For example, a simplistic way to do that for a subscriber interested in, say, {news=sport, team=Yankees} would be to subscribe for a request message {reliability-message=request, news=sport, team=Yankees}. However, this simplistic approach does not work. For one thing, the second subscription is useless, since the first one would already match all corresponding request messages. More importantly, a subscription for a request that repeats the same constraints as a normal subscription would not work, because a receiver trying to recover a missing message may not be able to fill in the relevant attributes.

Consider in fact the following scenario. Application Alice subscribes for {team=Yankees} while application Bob subscribes for {news=sport}. Now, suppose Alice receives message m ={news=sport, team=Yankees} while the same message is lost on the way to Bob. Also suppose that Bob receives a following message m' ={news=sport, team=Mets} carrying a publication record that allows Bob to detect the loss of m. At this point, Bob can determine that the missing message m matches its own subscription {news=sport} and therefore might issue a request {reliability-message=request, news=sport}. However, that request would not reach Alice. In order for such a request to reach all potential caches, Bob would have to fill in all the attributes of m in the request. In other words, Bob would have to know m completely in order to issue an effective request to recover m.

We propose to overcome this problem by once again relying on the information contained in the publication record. In the scenario we just described, Bob does not know the content of the missing message m, but he does know its encoding B_m. Therefore, Bob could include B_m in a request message such that Alice could subscribe for it using an encoding of its own subscription. For example, suppose that Alice's subscription would be encoded in a Bloom filter B_A whose bits at positions 7 and 15 are set (all other bits are zero). Then, Alice could subscribe for {reliability-message=request, b7=true, b15=true} effectively representing B_A by means of attributes each representing one of the 1-bits in B_A. Notice that what matters is the *presence* of such attributes, not their value. Now, Bob could do the same by composing his request for m using the encoding B_m he finds in the publication record. And since the encoding is such that a message m matching a subscription S would be encoded by a Bloom filter B_m whose 1-bits are a *superset* of the subscription's Bloom filter B_S, Alice would have to receive that request.

Formally, if Alice intends to volunteer for providing repair messages that match one of its subscriptions (filters) say F, it computes the Bloom filter of this subscription which here we denote by B_F and then issues a subscription for requests, similar to the following:

$$\bar{F} : \{\textit{reliability-message=request, } b_i = \textit{true, ..., } b_k = \textit{true}\}$$

where attributes b_i through b_k correspond to those bits of B_F, the Bloom filter of F that are set to one i.e., if $B_F[i] = 1$ then there is a constraint in $\bar{F}$ in the form $b_i = true$. Later, when Bob, detects that a message m was lost, because through publication record, it has access to the encoded version of message m, (denoted by B_m), it publishes a message similar to the following

$$\bar{m} : \{\textit{reliability-message=request, } b_j = \textit{true, ..., } b_l = \textit{true}\}$$

where b_j to b_l correspond to bits of B_m with value one. Now, if the lost messages m matched F, Alice's subscription, then, B_m covers B_F and thus $\{b_i, ..., b_k\} \subseteq \{b_j, ..., b_l\}$. Subsequently, $\bar{m}$ will match the filter $\bar{F}$ and so, Alice will receive this request.

With the ability to detect lost messages and to route requests to all potential caches for those messages, the recovery process proceeds in a similar fashion as in SRM. We illustrate this process through an example. Consider a subscriber A that has an active subscription and suppose that A is willing to provide repairs for all the events matching that subscription. To do that, A encodes the subscription and issues a second subscription using the 1-bit attributes as described above. As subscribers intending to participate in the recovery mechanism, B and C also initially issue an additional subscription to receive matching repair requests. As we will detail below, this is done to suppress duplicate requests.

Now, assume that later, a publisher P publishes an event m_5 matching A's subscription. The message is received by A but due to a temporary failure it is not received by B

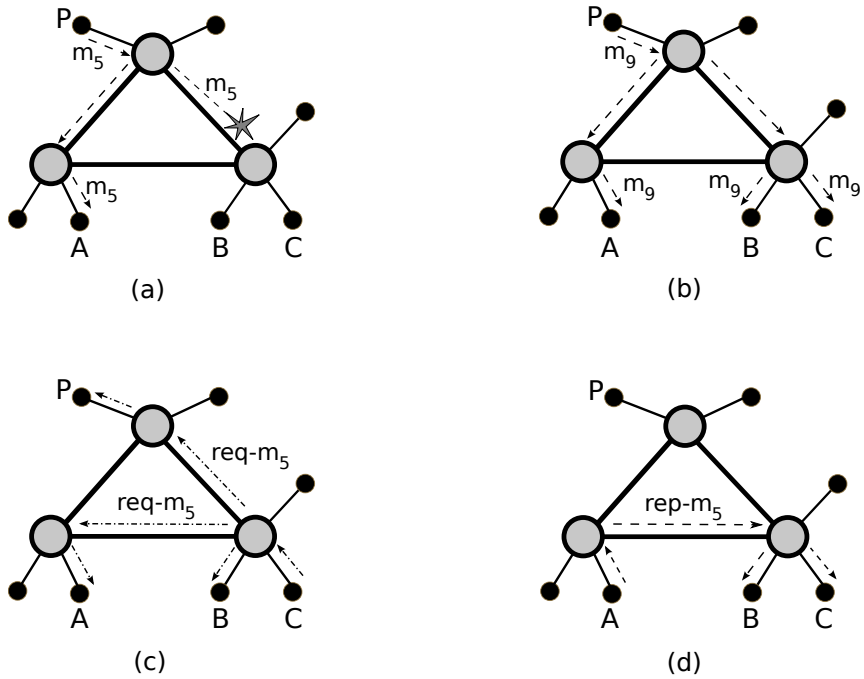


Figure 4.1. Message m_5 is lost before reaching B and C (a); having received m_9 (b), C publishes a request for m_5 (d); A replies with a repair (d).

and C whose subscriptions also match m_5 (see Figure 4.1a). Later, P publishes another message m_9 that is received by B and C (because it matches their subscriptions). This message carries the publication record of P that includes the Bloom-filter encoding of m_5. Using that data, both B and C realize that they have missed m_5 (see Figure 4.1b).

The detection of a message loss triggers the recovery mechanism at B and C. Thus, B and C issue repair requests, but in doing that, they try to suppress duplicate requests. The two nodes start to count down from a randomly generated timer (discussed in Section 4.1.1). Assume that C picks the earlier timeout between the two. Once C's timer expires, C publishes a request for m_5 by including the 1-bit attributes corresponding to the Bloom-filter encoding of m_5 as well as a unique message id for m_5 (source plus sequence number) that is also part of the publication record. This request is received by both B and A because it matches their request subscriptions (see Figure 4.1c). Upon receiving the request, B, which has a timer running on an identical request, reacts by delaying its own request by doubling its timer value.

A instead reacts by searching its message cache for a message with the given id, and when it finds it, publishes a repair message consisting of the original message m_5 with an additional attribute that indicates that it is a repair (and therefore a duplicate publication). The repair message reaches both B and C (see Figure 4.1d), at which point B cancels its pending request and both B and C deliver m_5.

Thanks to the expressiveness of content-based networking, this request routing technique allows for fine-grained control over the request messages and their receivers. On the one hand, it allows clients to precisely describe request messages they are willing to reply to, if any. On the other hand, it also allows for the implementation of special policies for the distribution of request messages, for example restricting request and repair messages within a single administrative domain. It is also easy to use this recovery protocol with designated cache nodes distributed in strategic points over the network that take on the responsibility of providing repairs, as suggested in RMTP [LS96] for reliable IP multicast.

4.2.3 Sending repairs

As explained in the example of Figure 4.1, a caching subscriber can respond to a request by re-publishing a message and by flagging that publication as a repair. Alternatively, the cache may send the repair directly to the requesting node through a unicast connection. Re-publishing is advantageous when the same message is requested by several receivers—something that might happen with overlapping subscriptions and non-local faults. Conversely, a unicast reply may be a better option when no other receivers requested the same message, and when that message would reach many receivers that already received the original copy. There may also be security and authenticity issues with the repair messages that are provided by caching nodes. This can be seen as a general trust and security management problem in the context of publish/subscribe systems, which is out of the scope of the thesis.

4.2.4 Adaptive message cache

An important parameter that is not discussed in the original paper on SRM by Floyd et al. [FJL+97] is the amount of memory a caching node (generally, a subscriber) has to allocate to its message cache. The cache should be allocated and managed so as to obtain a high cache-hit ratio while also avoiding unnecessary caching and therefore saving memory resources. In this section we elaborate on this issue.

On the one hand, for performance reasons we would like to have nodes maintain the message cache in main memory, within the limits of their memory constraints. On the other hand, it is also desirable to limit memory usage to a minimum. Therefore, nodes need to know when to drop some of the messages from their cache. The best strategy depends on various parameters, such as end-to-end message delivery delay, sender publication rate, match probability, and timer parameters of the requesting nodes. A message cache of constant size is oblivious to such network dynamics and may sometime lead to memory waste and other times to cache misses. Instead, we seek an adaptive cache that would perform well under variable network conditions. In particular, we formulate an approximation for the optimum cache size focusing on the most common scenario.

Symbol	Meaning
λ	Publication rate
C_1, C_2	SRM request timers
P_m	Match probability
$d_{Z,B}$	Trip time between requester and publisher
$d_{B,A}$	Trip time between requester and caching node
K	Cache size

Table 4.1. Parameters used in the calculation of the cache size.

We consider the message cache as a ring buffer with dynamic size, and we consider the problem of finding the optimum size of this ring buffer based on the parameters summarized in Table 4.1 (indeed, our implementation uses a ring buffer for its message cache). For this formulation we assume that message losses occur in bursts of S messages per second, where $1 \leq S < \lambda P_m$. We further assume that request messages are not lost.

Consider a publisher Z that publishes a message m_i. This message has two intended receivers: A that receives m_i and B that incurs a message loss and does not receive m_i. To find a lower bound for the size K of the message cache at A, we assume that A is closer to the publisher while B is farther away from the publisher in terms of end-to-end delay. Thus, on average, it takes $d_{Z,B} + \frac{1}{\lambda P_m} + \varepsilon$ time units for B to receive m_j, the next (relevant) message from Z, and detect the loss of m_i by investigating the publication record attached to m_j. We neglect ε, the small processing time of m_j. Therefore, the request message from B will be received by A after an expected delay of $(C_1 + \frac{C_2}{2})d_{Z,B} + d_{B,A}$ time units. Consequently, in order to be able to provide a repair for the lost message m_i, A has to store m_i in its ring buffer for at least $d_{Z,B} + \frac{1}{\lambda P_m} + (C_1 + \frac{C_2}{2})d_{Z,B} + d_{B,A}$ time units.

The minimal caching time we just computed, together with the arrival rate at A, determines a minimal cache size. Since A receives messages from the publisher with an expected inter-arrival time of $\frac{1}{\lambda P_m}$, and hence must overwrite m_i in its ring buffer approximately after $\frac{K}{\lambda P_m}$ time units. Thus, in order for A to be able to provide repair for m_i, the request must arrive at A before A overwrites m_i in its ring buffer. So we have:

$$d_{Z,B} + \frac{1}{\lambda P_m} + (C_1 + \frac{C_2}{2})d_{Z,B} + d_{B,A} \leq \frac{K}{\lambda P_m}$$

solving the inequality, we find a lower bound for the cache size K:

$$K > \lambda P_m[(C_1 + \frac{C_2}{2} + 1)d_{Z,B} + d_{B,A}]$$

```
1: if closets_repair_provider = True then
2:     a ← ((C_1 + C_2/2 + 1)d_{Z,B} + d_{B,A})λP_m + 1
3:     b ← 0
4:     if HitRatio < H then
5:         b ← 1.1K
6:     c ← max(a, b)
7: else
8:     c ← 0.9K
9: if K_min ≤ c ≤ K_max then
10:    K ← c
```

Algorithm 4.2. Probabilistic ordering FIFO algorithm run by a recipient for each publisher

The parameters of the above formula vary over time, and therefore must be continuously estimated. Each caching node A, can directly measure λP_m which equals the reception rate from the publisher Z at A. The value of $(C_1 + \frac{C_2}{2})d_{Z,B})$ is not measurable by A and therefore is included in each request message sent by B. The auto-tuning algorithm in SRM works in a way that the requesting node is likely to be the closest node to the point of failure. Thus, the requesting node for a given publisher is likely to often be the same node. Thus, this value is averaged over received requests for message of the publisher Z with the assumption that requests are coming from the same node or nodes at the same distance from the publisher. Finally, the value of $d_{B,A}$ is easily found by the simple method described by Floyd et al. in [FJL+97].

Another consideration is that based on SRM semantics, via auto-adjustable timer parameters, the node that provides repair messages is usually the closest node (in terms of end-to-end delay) to the point of message loss, which is in fact a method to minimize recovery time. Therefore, it is better to have this node handle caching more aggressively and other nodes gradually reduce their cache size, since they are not actively involved in providing repairs for messages coming from that specific publisher. Moreover, given that the above formulation gives a lower bound for K, caching nodes also need to monitor their average hit ratio, which is updated upon reception of each request, and if this ratio is below a preconfigured value, increase their cache size. Algorithm 4.2 shows this procedure, which a caching node executes periodically. H is a preconfigured hit ratio and K_{min} and K_{max} are minimum and maximum allowed cache size, respectively, and are configurable parameters. This algorithm is quick to increase the cache size as a reaction to sudden spikes in publication rate λ, and instead reacts with a gradual increase (10%) of the cache size when the network is stable but the hit ratio is lower than the configured level H. The algorithm is also rather conservative when it comes to reducing cache size, since we would like to avoid dropping cache

entries too early as a result of abrupt but transient changes in the network dynamics.

4.2.5 Interaction with FIFO ordering protocol

We implemented the recovery protocol as a part of our transport protocol, where it functions beside FIFO ordering protocol. If the publication record of a message hints the possibility of a message loss, the received messages that proceed the missing message are latched for some time. This latch time is determined by the FIFO ordering protocol. If a missing message arrives before the latch time expires, then the delivery proceeds as usual; otherwise the recovery process for that message triggers.

4.3 Discussion

The effectiveness of the proposed recovery protocol is primarily influenced by the effectiveness of the loss detection. Obviously, the publication record carried by each message must be limited in size due to practical limitations and also to limit traffic overhead. This is in fact, where the probabilistic nature of our recovery protocol stems from.

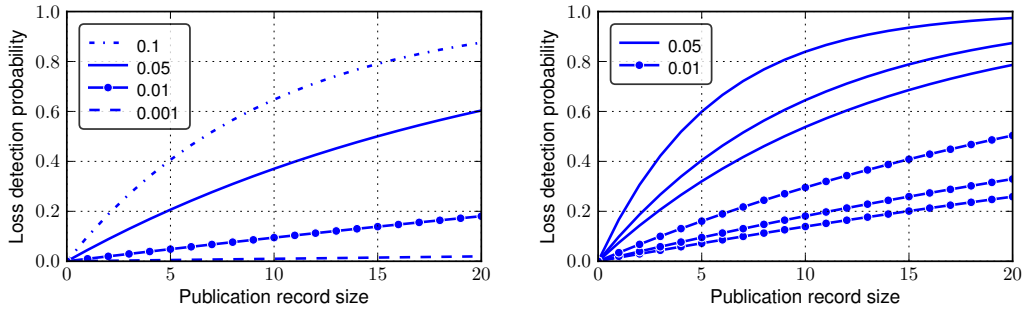


Figure 4.2. Probability of loss detection for (a) different sizes of publication record and match probability, (b) different sizes of publication record, match probability and different number of nodes sharing a loss (2, 3, 5 from bottom to top in each line category).

We now discuss the impact of publication-record size on the probability of loss detection. Considering a subscriber S and a publisher P whose publication matches S's subscriptions with probability P_{match}, the probability that S detects the loss of a message sent by P depends on the message loss probability P_{loss} on the path from P to S, the matching probability P_{match}, and the size R of the publication record. Simply put, if each message carries a publication record of size R, then the loss of a message m will not be detected by its intended receiver if none of the R messages published after m are received by that receiver. So, the probability of loss detection is:

$$1 - (1 - P_{match} + P_{match}P_{loss})^R \tag{4.1}$$

Figure 4.2a shows the probability of loss detection for different sizes of publication record and different match probabilities for a loss probability of 0.01. As the figure suggests, when the match probability is 0.001 (i.e., out of every 1000 publications, only one message is expected to match the subscriber's interests) the loss detection mechanism is very inefficient. With 5% matching probability, a publication record of size 10 makes the loss-detection possible with a probability of 0.4. At a first glance, Figure 4.2a might suggest that this loss detection mechanism would limit the applicability of our recovery mechanism. While this is true for applications with very low match probability, note that Figure 4.2a demonstrates the worst case where there is only one intended receiver for the lost message. In other words, when a message loss is shared among multiple receivers, it is sufficient that only one of them detects the loss. This happens when subscribers have overlapping subscriptions.

As an example, let us consider a case of Figure 4.2a where a node n has a match probability of 0.01. Let us also assume that there are k other nodes ($k = 1, 2, 4$ in Figure 4.2b) with the same matching probability as n and we also assume that all these nodes' subscriptions have a 50% overlap with n's subscriptions. That is, any message that matches n's subscription will match all of the other nodes' subscriptions with a probability of 0.5. Figure 4.2b (the three bottom lines) shows the growth of the loss-detection probability (the probability that at least on node detects a shared loss) with the growth of the publication-record size. The three top lines correspond to the case where nodes have a match probability of 0.05 and the rest of the scenario is similar. Notice that the growth of loss-detection probability with the size of publication record is faster for larger values of k.

Another consideration about this loss-detection mechanism is that if the publication rate (number of publications per time unit) is relatively low, detection of a message loss will be *late*, especially for subscribers with low match probability. This problem can be alleviated by periodic soft-state messages sent by the publisher. Such messages only serve the purpose of enhancing the loss detection on the receiver side. Receivers whose match probability is too low can subscribe for soft-state messages for faster and more successful loss detection.

4.4 Evaluation

In this section we evaluate the performance of the recovery protocol with a focus on effectiveness and cost. More specifically, we are interested in measuring how many lost messages are recovered and how long it takes to recover them. Another practical question we explore is how much extra traffic is generated by the recovery protocol and what is the user-tangible impact on the ordinary traffic.

We note that the performance of the recovery protocol depends significantly on the choice of topology, workload, and message-loss probability. Our choice for these experimental settings does not aim to demonstrate the best-case performance of our protocol, but rather intends to examine its effectiveness under stress, in the presence of frequent message losses, and with conservatively chosen workloads that do not necessarily contribute to increase the effectiveness of the recovery protocol.

4.4.1 Experimental setup and workload

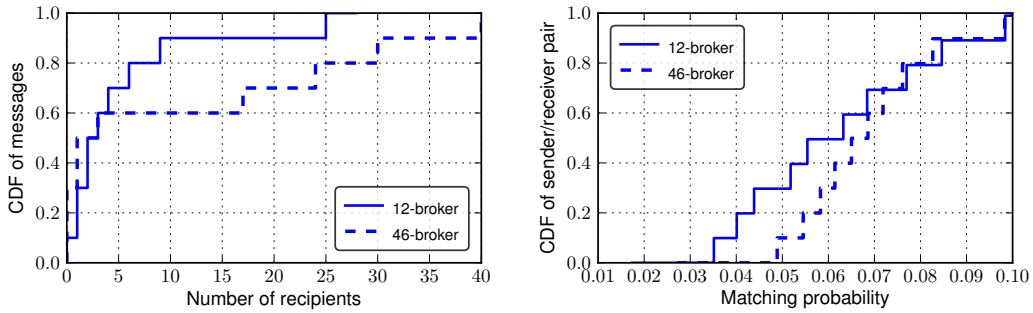
In our experiments each physical machine hosts a broker that serves 5 instances of the client (as their home broker) running on the same machine. To simulate a wide-area network, we used the Linux traffic control tools to apply delay and message loss on all inter-broker and links. Each link's delay d_i is randomly chosen in the range of $[25, 75]$ milliseconds, which also continuously varies during the execution in the range of $[d_i - 5, d_i + 5]$ milliseconds. This variation of ± 5 milliseconds is typical of the Internet, based on different Internet measurements.[1]

We present the results for two sets of experiments with two different network topologies and workloads. The first topology is composed of 12 brokers with a diameter of 5 broker-hops in which out of the total of 60 clients, 6 nodes are publishers and 54 are subscribers. Then to probe scalability of the recovery protocol, the second experiment involves a larger topology of 46 brokers with a diameter of 11 broker-hops. Among the total of 230 clients, 10 are publishers and the rest are subscribers.

Using the Linux traffic control tools, we apply a link-level message loss probability of 0.01 to all inter-broker links (i.e., each link loses approximately one message out of each 100 messages). This loss probability is rather large because for a network of diameter 11 it sums to unrealistically large likelihoods of message loss for some end-points. For example, for a sender and a receiver 11 broker-hops apart, the probability of message loss is as large as 0.1. This is a deliberate choice to stress-test the recovery protocol under very frequent message losses.

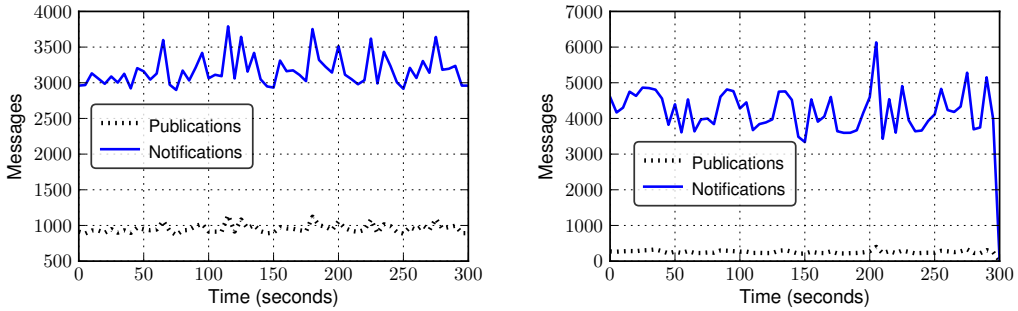
In both experiments, we use synthetic workloads with varying publication rates that simulate sudden short-term spikes in publication rate of publishers, to simulate bursty traffic that causes queuing delays. In these experiments, all the nodes participate in the recovery process. That is, they all volunteer to provide repair for messages that they receive. As shown in Section 4.2.1, two crucial factors in the loss detection and hence in the effectiveness of the recovery protocol are the matching probability and the number of receivers for a message. To be conservative in our evaluation, we choose workloads that exhibit low subscription/publication match probability (the probability that a message matches a node's subscription) and a low number of receivers per message. Figure 4.3 characterizes the two workloads for the 12-broker and

[1] For example see measurements by RIPE Network Coordination Centre (RIPE) available at http://www.ripe.net/data-tools/stats/ttm/ttm-data



(a) (b)

Figure 4.3. (a) Number of receivers for cumulative distribution of messages. (b) Match probability for cumulative distribution of subscriber/publisher pairs.



(a) (b)

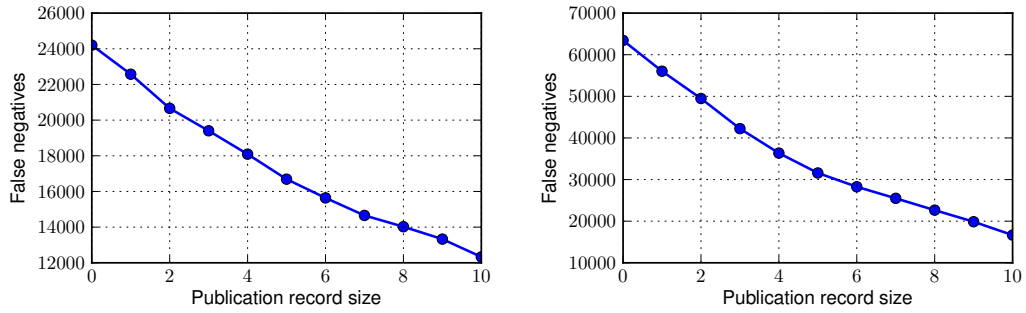
Figure 4.4. Aggregate publication and notification rates in (a) 12-broker and (b) 46-broker networks.

46-broker experiments. As Figure 4.3a shows, in both workloads 50% of the messages have at most two receivers. Figure 4.3b plots the subscription/publication match probability for each pair of subscriber and publisher. In both workloads, 80% of the pairs have a match probability of less than 0.08 while the maximum match probability is not higher than 0.1.

Figure 4.4 plots the aggregate publication and delivery rates without the recovery protocol during the course of the experiments, which runs for a total of 5 minutes. The rapid changes in delivery rate is due to spikes in publication rates.

4.4.2 Recovery effectiveness

First we look at the number of false negatives (that is, the number of messages that were not delivered to their intended receivers) with and without the recovery protocol. One determining factor in the effectiveness of the protocol is the size of the publication record. Figure 4.5 illustrates the effectiveness of the recovery protocol with different



(a) (b)

Figure 4.5. Impact of publication-record size on the effectiveness of the recovery protocol in (a) 12-broker and (b) 46-broker networks.

publication-record sizes. The y-axis shows the total number of false negatives (message losses); the zero on the x-axis represents the case where no recovery protocol is in place. The decrease in the number of false negatives is more pronounced in the 46-broker network while it is slower in the 12-broker network. Indeed, in the 12-broker experiment, growing the publication record size from 1 to 10 only halves the number of false negatives while in the 46-broker network the false negatives are reduced by a factor of 3, approximately.

Our calculation of the probability of message loss detection by Equation 4.1 as well as Figure 4.2a explain this result, which is mostly due to the characteristics of the workload, i.e., small matching probability and small subscriber/message, which hinder a more effective loss detection in the 12-broker network. Furthermore, our experiments with smaller message loss probabilities showed that the exponential effect of publication record size on the recovery effectiveness is more substantial when message loss probability is smaller, which is also explained by Equation 4.1. For instance, in the 46-broker network when loss probability is 0.001, the recovery protocol with a publication record size of 10 reduces the number of false negatives by more than 8 times.

We now examine the network dynamics and the corresponding protocol behavior over time. To do that, we focus on the experiments with the best effectiveness results (the highest recovery rate), that is, with a publication record of size 10. We choose this case because a larger size for the publication record generates larger amounts of network traffic and hence by studying this case we gain a better understanding of its impact on the network.

We begin our probe by looking at the aggregate rate of false negatives with and without the recovery protocol during runtime, shown in Figure 4.6. The reduction in the false negatives that is almost a factor of two for the 12-broker and a factor of three for the 46-broker network is persistent during the whole course of the experiment. So, at a high level, the recovery protocol does not show any pathological behavior during

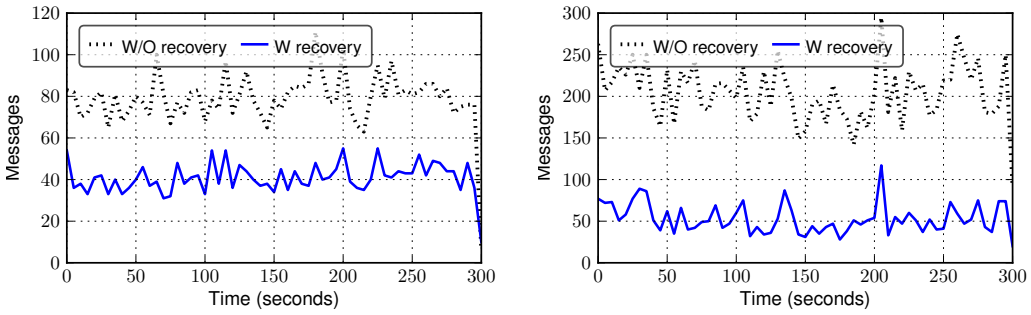


Figure 4.6. (a) Changes in the aggregate rate of false negatives (message loss) with and without the recovery protocol, for (a) 12-broker and (b) 46-broker networks.

the experiment.

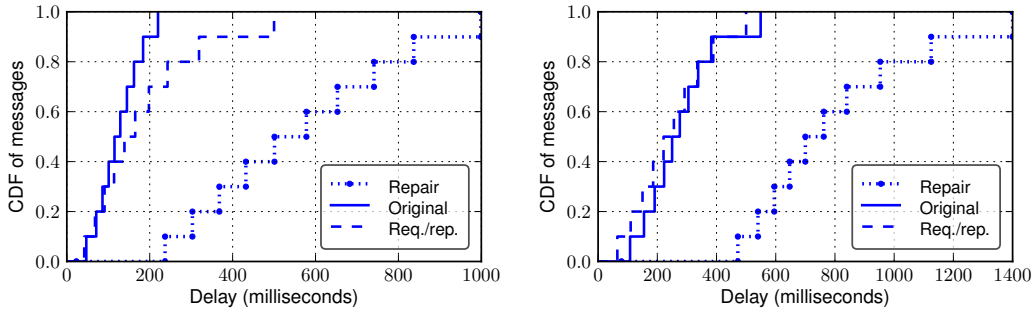


Figure 4.7. Cumulative distribution of the end-to-end delay for original and repair messages and request/repair delay for (a) 12-broker and (b) 46-broker networks.

We now proceed to examine another aspect of the recovery effectiveness, namely the time it takes to recover a lost message. Figure 4.7 shows the end-to-end delay of the original (non-repair) messages as well as the repair messages. Note that the delay of repair messages is in fact the time difference between their publication by the original publisher and their delivery to the indented receiver as a repair. Figure 4.7 also plots the request/repair delay, which is the time difference between multicast of the first request for a certain message and the first repair for that message. An interesting observation is that the request/repair delay in both networks is relatively small: for 80% of the messages in 12-broker and 46-broker networks, this delay is below 200 and 350 milliseconds, respectively, while the total time to recover missing messages is considerably larger. This means that a large part of the recovery delay is due to “late” loss detection, which is a result of low matching probability and/or low publication rate. This is not surprising, since the publication rate of each publisher varies between

20 and 500 messages per second in the 12-broker, and between 20 and 250 messages per second in the 46-broker networks, and hence, with a match probability of 0.05, it might take up to 1 second to detect a message loss.

Also, note that our choice of large message loss probability causes many of the requests to be lost and so, some requests must be reissued for a second or a third time, each time after a timeout. In fact, in other experiments with the 46-broker network when we applied smaller link loss probabilities, we observed that the request/repair delays were almost 50% smaller, which in turn resulted in smaller recovery times.

4.4.3 Performance and network overhead

We now turn our attention to the operating costs of the recovery protocol. In particular, We consider two measures: network usage and the overall impact on the receivers in terms of the extra delivery delay that the original messages incur. The extra network load is caused by the publication record attached to each message as well as the traffic of request and repair messages. In our workload all messages have 10 attributes and to produce each entry of a publication record we encoded a message in a Bloom filter of size 256 bits. Thus, a message with a publication record of size 10 carries 320 bytes of extra information. We deliberately used this large number of attributes and large-size Bloom filters to examine the negative effects of the recovery scheme in a rather extreme case. In reality though, where messages usually have smaller number of attributes, Bloom filters of size 128 or 64 would suffice and cause less network overhead.

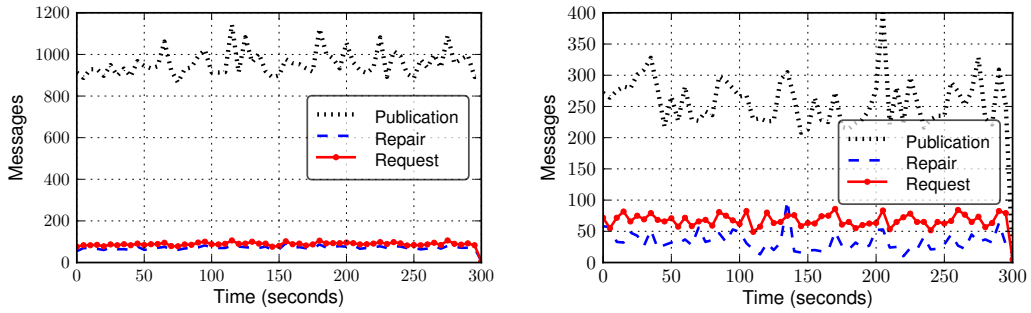


Figure 4.8. (a) Aggregate publication rate, request, and repair messages during the experiment for (a) 12-broker and (b) 46-broker networks.

Figure 4.8 illustrates the aggregate rate of request and repair messages during the experiment as well as the aggregate rate of publications to be used as a reference measure. Ideally, for each lost message there must be only one request and one repair message. However, in many cases request and repair messages are also lost, which is indeed the reason why in Figure 4.8 the number of requests is more than the number

of repairs. Interestingly, we observed that in both networks the multicast suppression mechanism built in SRM works favorably well. More specifically, in the 12-broker network more than 90% of the request and 80% of the repair messages were not duplicated while in the 46-broker network these values were 80% and 70%, respectively. This larger duplicate number in the 46-broker network is due to the network's greater diameter, which is twice the diameter of the 12-broker network. A higher diameter has a slight effect on SRM's multicast suppression mechanism, but more importantly leads to more frequent losses of request/repair messages.

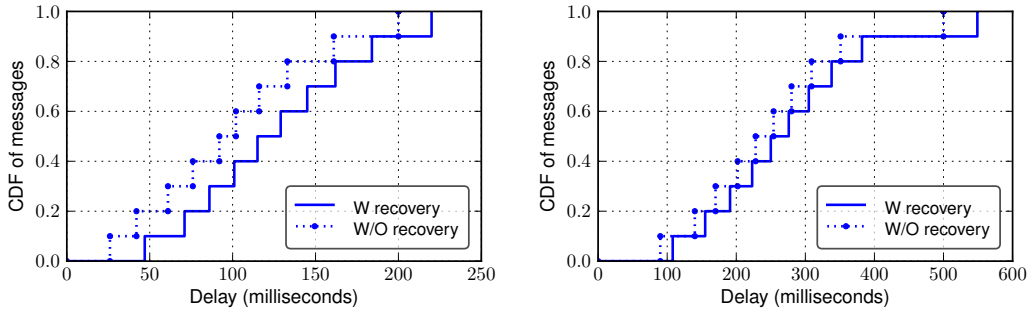


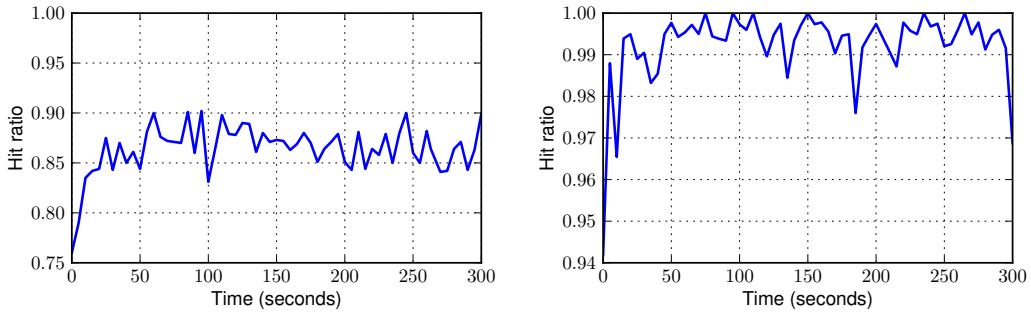
Figure 4.9. Delivery delay with and without the recovery protocol in (a) 12-broker and (b) 46-broker networks.

In effect, the overall and user-visible overhead is the change in the delivery delay of the messages when the recovery protocol is active and causes extra network traffic. Figure 4.9 shows the end-to-end delivery delay for cumulative distribution of messages with and without the recovery scheme. As the diagram suggests, the recovery protocol shifts the plot of end-to-end delay without recovery to the right, which implies a constant increase in the end-to-end delivery delay of all messages. Nevertheless, given the minimum and maximum values of end-to-end delay without recovery, an increase of 25 milliseconds in delay is not prohibitively large, since the dominant network traffic is the ordinary publication traffic and so, request/repair messages do not cause tangible impact on the overall network performance.

4.4.4 Adaptive cache

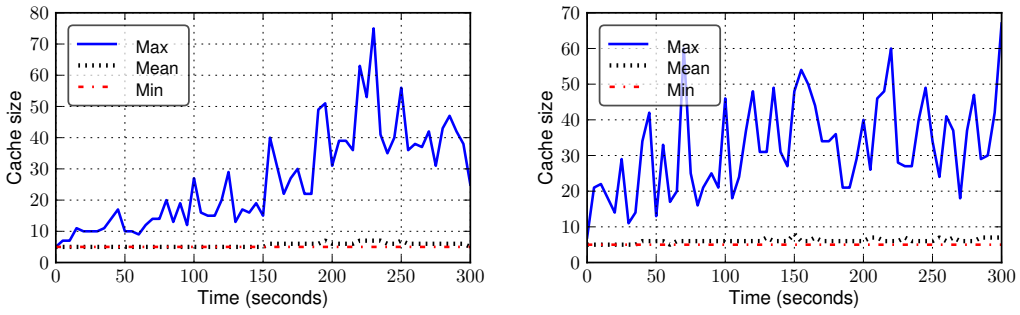
We now examine the performance of the adaptive message cache by measuring the hit rate and the size of message cache in the network. A cache hit occurs when a request for a message is sent out and at least one of the nodes that received the request is able to provide a repair. Thus, considering the whole network as a single cache, we define hit ratio as the ratio between total number of cache hits to the total number of requests.

In our experiment we assigned values of 5 and 300 to K_{min} and K_{max}, respectively,



(a) (b)

Figure 4.10. Changes in the cache hit ratio in (a) 12-broker and (b) 46-broker networks.



(a) (b)

Figure 4.11. Changes in the minimum, mean, and maximum cache size of all nodes in (a) 12-broker and (b) 46-broker networks

with an initial cache size of K_{min}. The target hit ratio H (see Algorithm 4.2) was set to 1. Figure 4.10 plots the changes in hit ratio during the experiment. In both networks, the hit ratio grows rapidly in the first few seconds of the experiment and then does not exhibit large changes during the rest of the experiment despite the continuous oscillation of the publication rates.

The effectiveness of the adaptive cache is further demonstrated by Figure 4.11, which shows the minimum, mean, and maximum cache size among all the network nodes during the experiment. Despite the high hit ratio, the amount of memory used for caching was small. This is because most of the nodes keep their cache size to a minimum, as evidenced by small value of the mean cache size. This is because the adaptive cache mechanism causes the nodes that do not actively participate in providing repair messages to reduce their cache size. On the other hand, nodes that are actively providing repairs adjust their cache size to accommodate changes in publication rate and improve hit ratio.

4.5 Conclusion

In this chapter we presented an end-to-end reliability scheme that is part of our transport protocol for best-effort publish/subscribe systems. The goal is to improve the reliability of these systems with a minimum reduction in high throughput and end-to-end delivery delay that such systems offer. In essence, our solution is an adoption of SRM with an enhancement to enable efficient routing of requests and an algorithm for cache management. The protocol is built on top of a standard publish/subscribe API, and therefore does not require any modification to the broker network.

Perhaps the main drawback of this recovery protocol is its probabilistic nature, i.e. its inability to guarantee that all lost messages will eventually be recovered. This, as we discussed in detail, is an outcome of the underlying probabilistic loss detection. We argued that when message losses are shared among multiple subscribers the chances of loss detection (and hence recovery) is considerably improved. This argument was in fact validated through experiments in which more than 75% of the lost messages were recovered despite very frequent message losses, where most lost messages had on average 5 receivers. More importantly, the experiments showed that the overhead that is the result of extra request/repair messages is insignificant and does not cause any inhibiting effect on the ordinary network traffic.

Chapter 5

Congestion Control

Best-effort systems typically process and forward messages as fast as they can, without taking into account the balance between independent flows, and without feedback (acknowledgements) or other flow-control mechanisms to avoid congestion within a single flow. On the one hand, best-effort systems are simple and fast, but on the other hand, they offer little or no support for fair resource utilization and congestion control. In such systems, publishers do not avoid or even detect congestion, and brokers respond to congestion by simply dropping overflowing messages.

Congestion and its adverse effects on traffic and applications have been studied extensively [BCC+98; BHL+00; WH01]. Congestion manifests itself when router queues fill up and ultimately overflow, which forces the router to drop packets. Congestion may be caused by transient and typically harmless traffic bursts, or by persistent high-rate flows that may cause severe delays and disruptions and even the complete lock-out of other flows [BCC+98]. In fact, network congestion is the primary cause of packet loss and long end-to-end delays, and it is a serious threat to the stability of a network.

These considerations motivate the work presented in this chapter. Our high-level goal is to develop a congestion-control mechanism for best-effort content-based publish/subscribe systems and as an integrated component of our end-to-end transport protocol. Our guiding principle in the design of the congestion control is to maintain the simplicity and elasticity of the underlying best-effort network of brokers, and at the same time provide applications with a method and a mechanism to modulate message flows according to available resources and in a fair manner.

We develop our protocol based on the design of a congestion control protocol for IP multicast called TCP-friendly multicast congestion control (TFMCC). We present the rationale for this design and discuss the challenges of implementing it in the context of content-based communication. In particular, we discuss the specificity of content-based communication with respect to flow control. We then describe our design, implementation, and the experimental evaluation of a *content-aware* congestion control protocol.

In brief, in our solution each receiver (subscriber) measures the end-to-end delay

and loss rate within each message flow from a sender (publisher). These measures are then used, through a so-called TCP-response function, to determine a proper maximal rate for that flow. That information is then fed back to the sender that in turn aggregates it across flows and adjusts its sending rates accordingly. Our protocol is TCP friendly in that, under the same network dynamics, it behaves like TCP in terms of fairness and long-term transmission rate.

Unlike the original TFMCC or any other congestion control scheme for IP networks, we implement a rate control algorithm that is *content aware*, in the sense that traffic measurement and feedback by subscribers, as well as feedback aggregation and rate control by publishers, are based upon and grouped by subscriptions and message content. This protocol only assumes the existence of an underlying content-based network with a common publish/subscribe API, and hence it is a generic protocol that can be applied to virtually any best-effort content-based network without any modification to the broker software and in particular to its routing and forwarding algorithms.

In Section 5.1 we elaborate on the problem of congestion control in the context of content-based networks and in the wider context of traditional IP networks. We then detail the internals of our protocol in Section 5.2. Experimental evaluation of the protocol is presented in Section 5.3, and we conclude in Section 5.4.

5.1 Context and High-Level Design

As in traditional networking, controlling and avoiding congestion in a publish/subscribe network amounts to witholding some publications on the part of publishers. More specifically, it amounts to controlling message flows as they are produced by publishers. Notice that, as is commonly intended in the context of congestion control, the term *flow* refers specifically to a stream of messages going from one sender (publisher) to one receiver (subscriber), even if the same stream in its entirety or in part might also reach other receivers. In this case we distinguish separate, possibly interdependent flows.

5.1.1 Congestion control for IP multicast

Congestion control has been studied in great depth in traditional networking, with some results that are very relevant to the work presented here. In particular, since content-based communication is a form of multicast communication (a message may be delivered to multiple receivers), it is natural to consider porting congestion-control protocols developed for IP multicast to content-based communication. Summarizing, existing congestion control protocols for IP multicast are designed so that multicast flows would compete in a fair way with other multicast or TCP flows, which are themselves designed to avoid congestion and to share network resources in a fair way. Existing methods of end-to-end congestion control for IP multicast are grouped into two

major categories: *single-rate* and *multi-rate* multicast.

Single-rate schemes are essentially extensions of TCP to multicast networks. In these protocols, a receiver, usually called the *acker,* measures network dynamics in terms of end-to-end delay and message loss, and sends feedback to the multicast source which in turn adjust its transmission rate accordingly. These protocols differ in the type of feedback messages and in the rate-limiting algorithm. Some protocols take a TCP-like approach where the control feedback is in the form of ACK/NACK, and rate is limited by a transmission window [WS98; GS99; Riz00]. In other protocols, receivers compute a desired rate on the basis of measures such as the delay and loss rate using a TCP-response function, and communicate the result to the source that in turn limits its sending rate accordingly [BTK99; WH01].

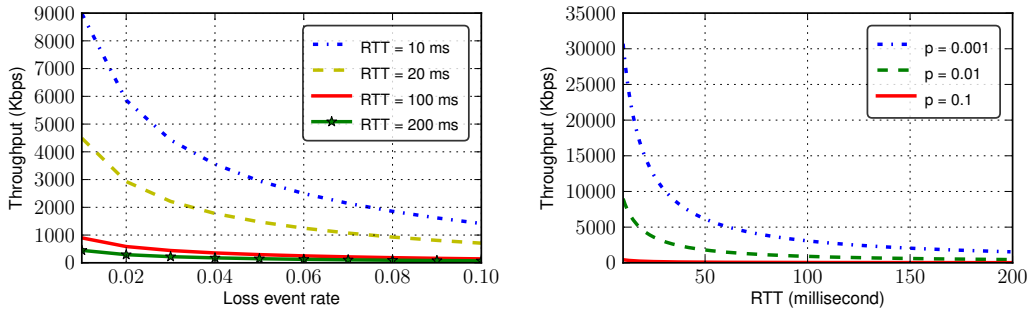
In multi-rate approaches, mostly applicable to media streaming [VCR98; BHL$^+$00; SW00], senders usually deploy a flavor of forward error correction encoding to break a data chunk into a set of stripes (sometimes with redundancy) and send each stripe to a separate multicast group. Recipients join a base group to receive a minimum amount of data, enabling them to retrieve or play the original media at a base-level playback quality. Joining additional groups leads to better-quality playback levels or faster retrieval of a file. The number of groups a receiver subscribes to is determined based on its estimate of congestion in its network area; in more congested areas, receivers downgrade their subscription to a smaller number of multicast groups which leads to a reduction of traffic in that area. In order for this method to be effective, there needs to be a coordination mechanism that coordinates all the receivers behind the same bottleneck to join the same number of groups.

5.1.2 TCP friendly multicast congestion control

An important representative of the single-rate protocols which we use as a basis for our protocol, is *TCP friendly multicast congestion control (TFMCC)*. TFMCC is a single-rate, equation-based multicast congestion control protocol, and is an extension of TCP-Friendly rate control (TFRC), a congestion control for unicast [FHPW00]. Basically, for each flow (one sender) a receiver monitors the round-trip time t_{RTT} as well as the loss event rate p (discussed below) and computes the maximum acceptable rate T for that flow using the following "TCP-response" function (s is the packet size) [WH01]:

$$T = \frac{s}{t_{RTT}\left(\sqrt{\frac{2p}{3}} + (12\sqrt{\frac{3p}{8}})p(1+32p^2)\right)} \tag{5.1}$$

The resulting rate T is the maximum rate that would assure compliance with TCP behavior for that flow. When the receiver measures a delivery rate exceeding the computed maximum rate, the receiver communicates its desired rate to the sender. The sender then adjusts its transmission rate to the lowest requested rate, which is usually dictated by the receiver with the most scarce network resources, called the *current limiting receiver (CLR)*. The sender always tracks the CLR, lowering its rate in response to



(a) (b)

Figure 5.1. The output of TCP response function: (a) for values of $p \in [0.01, 0.1]$ and $t_{RTT} \in \{10, 20, 100, 200\}$ milliseconds. (b) for values of $t_{RTT} \in [10, 200]$ milliseconds and $p \in \{0.001, 0.01, 0.1\}$.

a rate feedback lower than the current rate (possibly switching to a new CLR) and only increasing its rate in response to a corresponding request from the CLR.

The parameter p in the TCP-response function is the *loss event rate* and plays a central role in how the protocol responds to message loss. A *loss event* is the loss of one or more messages during one round-trip time, and a *loss interval,* hereafter denoted by ℓ, is the number of messages received between two loss events. Accordingly, p (loss event rate) is defined as $1/\bar{\ell}$ where $\bar{\ell}$ is the average loss interval over the last i loss events. Note the difference between *loss event rate* and *loss rate* (lost messages over total number of messages): the loss event rate is always less than loss rate. The choice of using the loss event rate in the TCP response function reflects the fact that common variants of TCP (e.g., TCP Reno) react only once to multiple message losses in a single round-trip time [FHPW00].

Figure 5.1 plots TCP response function for different values and ranges of loss event rate and RTT (p and t_{RTT} in Equation (5.1)). Both plots show that for larger values of RTT the sensitivity of the response function to changes in p declines. In other words, in presence of congestion where RTT is relatively high, the output of the response function is mostly dominated by the value of RTT. More importantly, Figure 5.1b shows that for smaller values of p (which reflects the lack of congestion), the response to increases in RTT is more pronounced. This leads to a conservative behaviour which avoids congestion by observing the fact that any increase in RTT is an indicator for congestion starting to happen [FHPW00].

5.1.3 Content-aware rate control

In a content-based network, message flows are induced by receiver predicates. The rate $\lambda_{A \to B}$ of a flow between a publisher A and a subscriber B depends on the publisher's sending rate λ_A as well as on the matching rate $\rho_{A,B}$ of the subscriber's predicate with

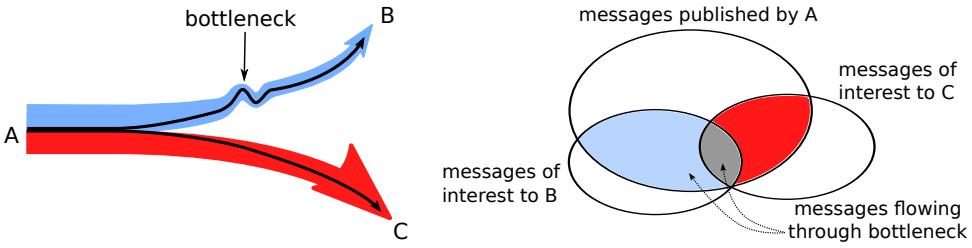


Figure 5.2. Content-aware rate control

respect to A's output. In other words, $\rho_{A,B}$ can be seen as the probability that a message published by A matches B's predicate, and thus the rate of the $A \rightarrow B$ flow with sending rate λ_A is $\lambda_{A \rightarrow B} = \lambda_A \cdot \rho_{A,B}$.

This combination of factors in the context of a content-based network makes congestion control more complicated but it also allows for more flexibility. In fact, a sender A whose publications reach two receivers B and C could modulate the rates $\lambda_{A \rightarrow B}$ and $\lambda_{A \rightarrow C}$ of the two flows independently, whereas traditional rate limitation in IP multicast would dictate that A reduce its overall sending rate λ_A. So, with IP multicast, if only B is experiencing congestion, C would also see a reduced flow from A. Instead, with the proper knowledge of the specific content-based flows $A \rightarrow B$ and $A \rightarrow C$, A could reduce $\lambda_{A \rightarrow B}$ and thereby avoid congestion while still maintaining a high rate $\lambda_{A \rightarrow C}$. In particular, this would be possible only if B's predicate does not cover C's predicate, meaning that not all messages that are of interest for C are also of interest for B (see Figure 5.2).

In practice, content-based communication introduces two requirements for an efficient and fair end-to-end congestion control protocol. First, rate-limitations must be applied (by the sender) only to those messages that are part of intense flows on congested routes. Second, the rate-control algorithm should account for the partial or total overlap between message flows, which is determined by their content-based nature. We also require such an algorithm to provide some level of fairness among competing flows, similar to TCP fairness.

To better understand the second requirement stated above, consider the example network of Figure 5.3. Assume that the only bottleneck link is B_2–B_3 and B's predicate covers that of C (i.e., the set of messages matching C's predicate is a subset of the messages matching B's predicate). Suppose that $\lambda_{A \rightarrow B}$, the rate of messages from A to B is 100 messages per second while $\lambda_{A \rightarrow C}$ is 10 messages per second. In this case, modulating the message flow that matches B's predicate will also reduce the reception rate at C. Thus, even though both B and C share the same bottleneck and both suffer from congestion, C's involvement in the congestion control process is not necessary.

In order to meet the aforementioned requirements, the rate-control algorithm must inspect message content on the publisher's side, and in particular it must be able to match messages against the subscriptions of receivers on congested paths. In essence,

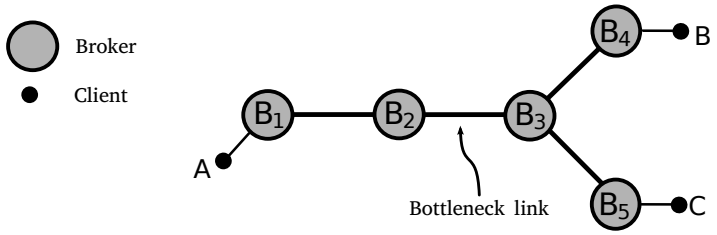


Figure 5.3. Content-aware congestion control: *C*'s involvement in the congestion control process is not necessary

the congestion-control algorithm on both ends must be informed by receiver predicates and message contents, and hence we call it *content-aware congestion control.*

5.1.4 High-level design

We designed an equation-based congestion-control protocol in which the maximum allowed throughput is a function of a set of measurable network dynamics. Three reasons motivate our choice of an equation-based rate control instead of a window-based one.

First, content-based communication does not allow for precise loss detection and equation-based rate control is less sensitive to errors in loss detection than window-based algorithms for which correct negative acknowledgments are essential. Also, irrespective of the efficiency of the loss detection method in use, equation-based rate control protocols tend to exhibit a smooth response to congestion relative to that of TCP [BBFS01] and hence are better choices for controlling traffic with frequent short bursts.

Second, we target networks with thousands of subscribers where equation-based control would scale better than a window-based control such as pragmatic multicast congestion control (PGMCC) [Riz00]. This is because, in PGMCC each message must be acknowledged by an acker. In contrast, an equation-based protocol requires only one feedback message per round trip time, which imposes a lower processing and communication overhead on the publisher and other network resources.

Third, equation-based rate control gives receivers good flexibility. For example, having computed a maximal flow rate, a receiver could prioritize some messages over others within that flow. This could be done directly using the same rate control feedback to the sender, by allocating a larger portion of the flow to some filters over others, in effect by distinguishing multiple sub-flows.

5.2 Content-Aware Congestion Control Protocol

We now describe our content-aware congestion control protocol in detail. In particular, we define the notion of content-based flows and we detail the transport headers and the roles and operations of subscribers and publishers.

5.2.1 Content-based flows

As explained in Section 5.1.2, in TFMCC, the current limiting receiver (CLR) dictates the maximum sending rate for a multicast group by sending feedback messages to the sender in that group. (Multiple senders are considered as different flows and are therefore treated separately.) The sender considers the feedback messages from all group members and elects the receiver with the lowest requested rate as the CLR of that group, and then communicates the identity of the CLR to all group members using a special header in its outgoing messages. In essence, this means that the CLR becomes a representative of the group, which makes sense because all members of the group see the same flow of messages from the sender to that group.

Unfortunately, this notion of a multicast group does not exist in content-based communication, and different subscribers might see different flows from the same publisher, and therefore no single subscriber can meaningfully represent all the subscribers in dictating a maximum sending rate. In other words, the publisher can not identify a single flow within which it can select a CLR and to which it would make sense to apply rate control.

To address this fundamental difference, we define a specific and more expressive notion of flow. We identify a content-based flow f as the stream of messages originating at a publisher P and matching a given filter s. Each subscriber may define multiple flows with the same publisher P each associated with a requested maximum rate. The publisher collects all the flow specifications sent by subscribers through feedback messages, and it processes them by merging flows from different subscribers whenever possible. (Merging flows from the same subscriber is also possible, although that can and should be done directly by the subscriber.) For each flow, the publisher then elects a CLR which determines the rate limitation for that flow.

5.2.2 Congestion control protocol

Control messages

Publishers and subscribers execute and coordinate the rate control algorithm by exchanging two types of control messages. Subscribers send ad-hoc feedback messages to define content-based flows and to control their rate. Depending on the network configuration, feedback messages could be transmitted through end-to-end IP primitives (TCP or UDP) or through the primitives of the content-based network itself. In this latter case, a sender would effectively subscribe for feedback messages addressed to it.

Publishers on the other hand transmit congestion control information to subscribers by attaching a transport header to each publication that belongs to a controlled flow (see Figure 5.4).

	publication record		time stamp	publish rate	rate controlled	CLR	$\delta_{feedback}$	message body
m_1	$10100\cdots10$	t_1						
m_2	$00011\cdots10$	t_2	917	2000	*true*	S_m	120	...
m_3	$00110\cdots01$	t_3						
m_4	$11100\cdots00$	t_4						

Figure 5.4. Transport header in a publication message.

Representation of filters and messages

Rate control at the publisher amounts to evaluating the filters in each flow specification against the messages the publisher intends to publish, so as to recognize and rate-limit the flow according to the demands of the corresponding CLR. In order to reduce the overhead of this evaluation and also to reduce the overhead of transmitting filters in feedback messages, here again we take advantage of the encoding scheme that was developed by Carzaniga et al. [CTCHW09] and we used to create publication record, as detailed in Section 3.2.6. In the following sections we explain how the congestion protocol takes advantage of this scheme. Notice however that the encoding is an optimization and a modular part of the congestion control protocol, and can be replaced or even removed altogether.

Loss detection

As mentioned in Section 3.3, upon a loss detection by the FIFO ordering component, a signal is sent to the congestion control component (if it is enabled). This signal triggers a procedure to update loss event rate, that may in turn lead to a change in the congestion control policy, in a way that we will detail later.

Round-trip time

In addition to the loss event rate, the subscriber must measure the round trip time (RTT) between itself and the publisher. To measure the RTT, we adopt the mechanism proposed in TFMCC, which is based on echo request/response messages. In our protocol, an echo request can be sent either directly or through the publish/subscribe network (similar to feedback messages). However, the echo response is always sent back through the publish/subscribe network, since its purpose is to measure the latency at that level.

filter id	Bloom filter	CLR	reception rate	quota
F_1	$01110\cdots11$	*true*	R_1	Q_1
F_2	$01100\cdots01$	*true*	R_2	Q_2
F_3	$01101\cdots11$	*false*	R_3	Q_3
F_4	$10101\cdots00$	*false*	R_4	Q_4

Figure 5.5. Per-publisher session state maintained by a subscriber.

In TFMCC and also in our protocol, echo requests/replies are used at the beginning of a session to compute the RTT when a node joins the network. Then the RTT is continuously estimated in cooperation with the publisher: the publisher measures the travel time of the feedback messages received from the CLR; it then transmits that to subscribers using publications ($\delta_{feedback}$ header in Figure 5.4); at the same time, the subscriber measures the travel time of the same publications using the publisher's time stamp (*timestamp* in Figure 5.4); finally, the subscriber adds the publisher's measured one-way delay with its own measure for the opposite direction, obtaining an estimate of the RTT in which clock differences cancel out.

Subscriber's state and operations

Subscribers maintain a session with each publisher from which they receive publications. Figure 5.5 shows the information associated with such a session. The subscriber stores the reception rate for each filter that generates an incoming flow from that publisher, and for each filter it maintains the measured reception rate and the target rate (quota) and it also remembers whether the subscriber itself is the CLR for that flow. Here again we use Bloom filters for fast matching of incoming messages against local subscriptions, though as stated before, this mechanism can be replaced by any matching algorithm. The subscriber then runs its congestion control algorithm (Algorithm 5.3) for each session (i.e., for each publisher). Once every RTT interval, the subscriber updates its measurements and in particular the reception rates and, given the average RTT and loss event rate measurements, it estimates a rate limit based on the TCP response function (Line 2). Then, based on that limit and on the current cumulative reception rate (over all filters) the subscriber initiates its rate control operations.

If the current reception rate exceeds the limit, the subscriber distributes the available rate to filters according to their priorities (which depend on the internal application logic). In our implementation we use a max-min algorithm (maximize the minimum rate) in order to favor filters with lower reception rates. If on the other hand the reception rate is lower than the limit, and if the subscriber is the CLR for at least one of its filters, then the subscriber proceeds with a rate increase. The subscriber assigns the unused rate to the filters for which it is CLR, but it limits each increase to at most a factor of α of current reception rate. This limit is intended to prevent rapid changes

1: **every** t_{RTT} **time units**
2: $R \leftarrow$ ESTIMATE_RATE() *{calculate the allowed throughput}*
3: $t \leftarrow 0$ *{overall rate from this publisher}*
4: **for** each entry e in the session state S **do**
5: $t \leftarrow t + e.reception\_rate$
6: **if** $R < t$ **then**
7: DECREASE_RATE(R) *{request a rate decrease}*
8: **else if** $R > t$ **then**
9: INCREASE_RATE(R) *{request a rate increase}*
10: SEND_FEEDBACK() *{send feedback message to publisher if necessary}*

11: **procedure** DECREASE_RATE(R)
12: **for** each entry e in the session state S **do**
13: $e.quota \leftarrow 0$
14: ASSIGN_QUOTA_MAXMIN(S, R) *{reassigns quotas based on new R}*

15: **procedure** INCREASE_RATE(R)
16: $t \leftarrow 0$ *{total reception rate}*
17: $n \leftarrow 0$ *{number of filters for which this subscriber is CLR}*
18: **for** each entry e in the session state S **do**
19: $t \leftarrow t + e.reception\_rate$ *{calculate total reception rate}*
20: $e.quota \leftarrow e.reception\_rate$ *{quota is at least the current reception rate}*
21: **if** $e.clr = true$ **then**
22: $n \leftarrow n + 1$
23: **if** $n = 0$ **then**
24: **return** *{not a CLR for any flow, no action required}*
25: $q \leftarrow (R - t)/n$ *{divide the unused quota by the number of throttled filters}*
26: **for** each entry e in the session state S **do**
27: **if** $e.clr = true$ **then**
28: $e.quota \leftarrow e.quota +$ MIN$(\alpha \cdot e.reception_rate, q)$
{never increase quotas by more than α times the current reception rate}

29: **procedure** SEND_FEEDBACK()
30: $M \leftarrow \emptyset$ *{set of filters in the feedback message}*
31: **for** each entry e in the session state S **do**
32: **if** $e.quota < e.reception\_rate$ **or** $e.clr = true$ **then**
33: $M \leftarrow M \cup \{\langle e.filter, e.quota\rangle\}$
34: send M to the publisher

35: **function** ESTIMATE_RATE()
36: $r \leftarrow \frac{s}{t_{RTT}\left(\sqrt{\frac{2p}{3}} + (12\sqrt{\frac{3p}{8}})p(1+32p^2)\right)}$
37: **return** r

Algorithm 5.3. Congestion monitoring and control run by a subscriber for each publisher from which there is an incoming message flow

filter	Bloom filter	CLR	quota	tokens	feedback time	queue
F_1	$001010\cdots00$	S_1	Q_1	T_1	t_1	$\cdots$
F_2	$010010\cdots10$	S_2	Q_2	T_2	t_2	$\cdots$
F_3	$100001\cdots01$	S_3	Q_3	T_3	t_3	$\cdots$
F_4	$100100\cdots00$	S_4	Q_4	T_4	t_4	$\cdots$

Figure 5.6. A publisher's congestion control state

and therefore to reduce instability. Our experiments demonstrate that a value of α between 0.5 and 1 is appropriate. Finally, when the necessary changes are made in the local state, the subscriber proceeds to send a corresponding feedback message to the publisher.

Publisher's state and operations

A publisher processes feedback messages and throttles message flows when necessary. A publisher maintains state describing and controlling its outgoing flows in a single table (see Figure 5.6). For each filter, again stored as Bloom filters for efficiency, the subscriber stores the identity of the CLR, the current rate limit (quota) and the current instantaneous available portion of that quota (implemented as a token bucket). The publisher also associates a filter with a *feedback time*, which is a timestamp of the latest feedback message from the CLR (set with the publisher's clock). The feedback time serves two purposes: first, it is used to compute the $\delta_{feedback}$ header for outgoing publications (see Figure 5.4) that is then used by subscribers to estimate the RTT; second, it allows the subscriber to discard stale entries after a set timeout.

Processing feedback messages

A feedback message carries a set of flow requests each defined by a subscription and an associated rate limit. The publisher collects and processes feedback messages using Algorithm 5.4, merging overlapping flows in its flow table whenever possible. Each flow f in the feedback message is processed individually. The publisher checks whether (1) f already exists in the flow table and s is already the CLR for that flow, in which case the publisher simply updates the flow; (2) f is covered by an existing flow with a lower rate limit, in which case the publisher ignores f; and (3) f covers existing flows with higher rate limits, in which case the publisher removes those flows from the table and then ultimately adds f to the table.

Rate control

Rate control is implemented on a per-flow basis with a token bucket. Tokens arrive at a rate determined by the value of the quota. Each publication sent by applications is first encoded as a Bloom filter that is then compared against all filters in the flow table.

```
1: upon receiving feedback message M from subscriber s do
2:   for each flow f ∈ M do
3:     PROCESS_FLOW_REQUEST(f, s)

4: procedure PROCESS_FLOW_REQUEST(f, s)
5:   for each flow g in the flow table F do
6:     if f.filter = g.filter and g.clr = s then
7:       g.quota ← f.quota                    {accept quota change request from CLR}
8:       g.feedback_time ← TIMESTAMP()
9:       return
10:    if g.filter covers f.filter and g.quota < f.quota then
11:      return
12:    if f.filter covers g.filter and f.quota < g.quota then
13:      remove flow g from flow table F
14:  F ← F ∪ {(f.filter, s, f.quota, TIMESTAMP())}    {add f to the flow table}
```

Algorithm 5.4. Processing feedback message M received from subscriber s

If all matching flows have enough tokens then a token is taken from each bucket (one for each matching filter) and the message is sent out, otherwise the message is queued until tokens become available. Matching messages are also tagged with the necessary transport headers (shown in Figure 5.4) before transmission into the network.

5.2.3 Dealing with imprecise loss detection

We now discus the effects of potential errors in the estimation of the loss event rate p in the TCP response function (Equation (5.1)) and how those effects can be mitigated. Recall from Section 5.1.2 that, as is done in TFRC, the loss event rate p is calculated as the inverse of the average loss interval $\bar{\ell}$, which is the average number of correct deliveries between consecutive losses. Thus, if the current loss interval is ℓ then a correct delivery would increase ℓ by 1 and a loss would insert the current ℓ into the average and then reset ℓ to 1.

Consider now a subscriber that receives two consecutive messages m_i and m_j with sequence numbers $i < j$ and therefore with a gap $g = j - i - 1$ in the sequence. Assume that messages carry a publication record of size k. If $g \leq k$ and therefore the publication record covers the gap, then the subscriber can operate as in TFRC, increasing the loss interval if the publication record does not reveal any loss, or otherwise resetting the interval. And even if the publication record does not cover the whole gap ($g > k$) but it still reveals a loss, then the subscriber should reset the loss interval.

The problem arises when the publication record does not cover the whole gap and

does not indicate any loss. In this case, the receiver could conservatively assume there was at least one message loss in the gap between m_i and m_j, then based on this assumption update the loss event rate and set ℓ to 1. In other words, the subscriber would attribute every gap in sequence number only to message loss. Obviously, this leads to substantial overestimation of p and underestimation of the available bandwidth, notably when the matching probability is low (so, gaps in sequence number are very common). Alternatively, the subscriber could assume there was not any message loss between m_i and m_j, and increment loss interval by one (i.e., $\ell \leftarrow l + 1$) without changing p. This in turn tends to underestimate p specially for small values of k and may overshoot the allowed throughput.

Our purpose is to find a balance between these two options with a higher weight on the second i.e., to attribute gaps in the sequence number to a mismatch other than a message loss. Thus, in this case we optimistically assume that no message was lost. However, to account for the expected error of this optimistic assumption, we increment the loss interval by a value that is less than one. In particular, we use an increment corresponding to the probability that the optimistic assumption is correct, which is the probability that none of the messages not covered by the publication record was relevant. This probability is $(1-\rho_{P,S})^{g-k}$ where $\rho_{P,S}$ is the matching probability, that is, the probability that each publication of publisher P matches the subscriber's predicate, which we can estimate as the ratio $\rho_{P,S} = R_P/\lambda_P$ between the reception rate R_P seen by the subscriber and the publish rate λ_P indicated by the publisher in the message header (see Figure 5.4).

Qualitatively, the optimistic assumption is generally valid in non congested and even in slightly congested network conditions, but it does not hold in the presence of persistent congestion. However, in such cases, where message losses are frequent, the receivers with higher matching ratios and hence high reception rates are likely to detect loss events anyway and therefore react with their rate control. This is because messages of a high-rate flow are more likely to be lost in persistent congestion. Also, it is more probable that the publication record covers at least one of the relevant but lost messages of such a receiver.

In order to better understand the effectiveness of our loss detection mechanism in estimating loss event rate and throughput, we conducted a simulation analysis. We simulated a simple scenario in which a flow of publications goes through a link that is unable to sustain the intensity of that flow. Figure 5.7 shows the estimated loss event rate (left) and throughput (right) during 100 seconds of simulation for three different sizes of the publication record: $k = \infty$ (top) corresponding to an ideal all-knowledgeable receiver, $k = 5$ (middle) and $k = 2$ (bottom). For a publication record of size $k = 2$ many loss events are not detected (e.g., around 15 seconds into the simulation) and the estimated throughput is often larger than that of the ideal receiver. However, with $k = 5$ loss detection is reasonably accurate which in turn results in an estimated throughput close to that of the ideal receiver.

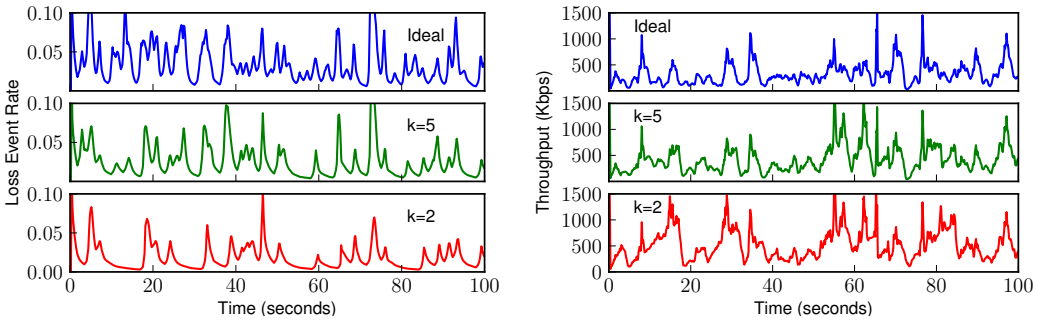


Figure 5.7. Loss event rate (left) and TCP response function (right) computed for the ideal receiver (top), publication record of size 5 (middle) and publication record of size 2 (bottom).

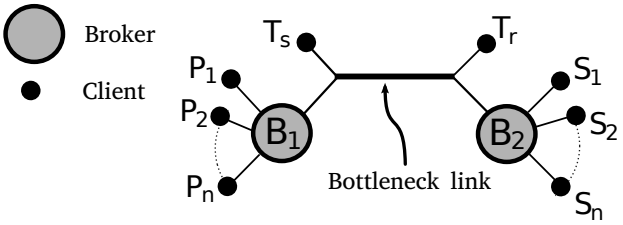


Figure 5.8. Experiment topology

5.3 Evaluation

In this section we present the results of the experimental evaluation of our protocol. The focus of the experiments is on the main functionality of the congestion control protocol. In particular, we investigate the effectiveness of the protocol in controlling congestion, responsiveness to changes in available bandwidth, fairness among concurrent content-based as well as TCP flows, and optimality of link utilization. We first analyze these quantities in a series of ad-hoc scenarios with small networks, a few clients, and specifically controlled workloads. We then demonstrate the effectiveness of the protocol in a large-scale deployment.

5.3.1 Experimental setup

We used the Linux traffic control tools to emulate bottleneck links. Figure 5.8 shows the topology we setup for all except the last two experiments. T_s and T_r are a pair of TCP sender and receiver and P_i and S_i represent publishers and subscribers respectively. As explicitly stated in each case below, different experiments use only a subset of the client nodes, while the two brokers and the inter-broker link (bottleneck) are present in all experiments.

We juxtapose the results of experiments with and without congestion control in place, so that we can better demonstrate the necessity of a congestion control mechanism and the effectiveness of our protocol in each case. In all experiments, the size k of the publication record is set to 2, implying that on average the transport header adds an extra 80 bytes to each message. To make the comparison meaningful in terms of bytes per second, in experiments without congestion control we increased the message size by adding a payload of 80 bytes. This obviously introduces a penalty that should be taken into consideration when evaluating the maximum message throughput of the best-effort network alone. However, the additional payload is relatively small and in any case does not fundamentally change the behavior of the system in terms of losses, especially in the relevant case of congestion.

5.3.2 Effectiveness, stability, and responsiveness

We start by investigating the basic properties of our congestion control protocol in a unicast scenario with one publisher and one subscriber (i.e., P_1 and S_1 in Figure 5.8). Figure 5.9 (bottom chart) shows the publication rate, message reception rate (notifi-

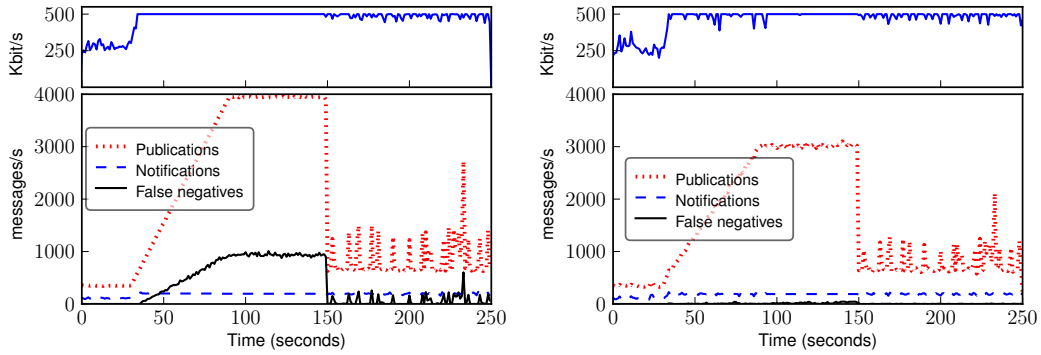


Figure 5.9. The effect of variable input load (a) without and (b) with congestion control in place. Top: traffic rate (Kbps) on the bottleneck link. Bottom: aggregate publication, reception, and false negative rate (messages per second) during the experiment.

cations) and false negatives (message losses) in terms of messages per second (mps) during 250 seconds of an experiment without (a) and with (b) congestion control. To demonstrate the optimality of resource utilization, the top graph shows traffic (Kbps) on the bottleneck link whose bandwidth is 500Kbps. In order to observe the performance of the protocol in the presence of persistent as well as transient congestion, we designed the workload so that the publisher generates traffic with an increasing, than stable, and then bursty rate. The publication rate starts at 400mps (messages per second) ramping up to 4000mps where it stabilizes for 60 seconds and then descends to 600mps continuing with short bursts of up to 2500mps. Once the publication rate

grows above 1000mps the network reaches its capacity and starts to exhibit message losses (false negatives). In all three phases, congestion control is able to mitigate persistent and transient congestion, reducing the number of lost messages by a factor of 20, particularly during the period where congestion is persistent.

Figure 5.9 also shows that the available network capacity is efficiently utilized for the most part of the experiment. More precisely, without congestion control in place nearly 46000 messages were received by the subscriber while with congestion control this figure was more than 45000 messages. Notice in Figure 5.9b that, with congestion control, the fluctuations in publication rate are still present while the rate of message losses remains almost unchanged. This is a benefit of content-awareness in our congestion control protocol that only regulates outgoing message streams that form a flow through the bottleneck link.

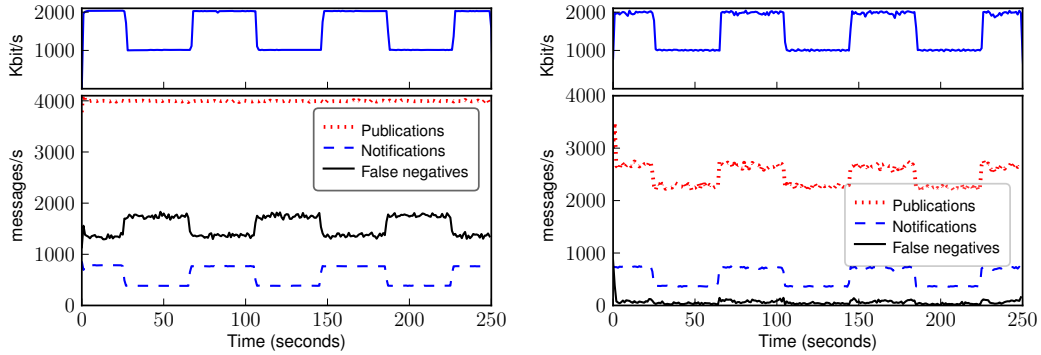


Figure 5.10. Effects of variable bottleneck link capacity (a) without and (b) with congestion control. Top: traffic rate (Kbps) on the bottleneck link. Bottom: aggregate publication, reception, and false negative rate (messages per second) during the experiment.

We now examine the responsiveness of our protocol to changes in bandwidth resources. More precisely, we want to answer the following two questions: First, when available bandwidth drops, how fast does the protocol reduce the send rate to control message loss? Second, upon an increase in the available bandwidth, how rapidly does the protocol saturate the new resources while controlling message loss? Figure 5.10 shows the results of an experiment in which we control the bandwidth of the bottleneck by alternating, in periods of 40 seconds, between 1Mbps and 2Mbps. As Figure 5.10b suggests, reaction to both decrease and increase in bottleneck link capacity is quite fast. Specifically, the flow adapts in less than two seconds to the new bandwidth and reaches a stable state while persistent message losses are barely noticeable.

5.3.3 Fairness among concurrent content-based flows

We now proceed to examine fairness properties of our congestion control protocol with regards to concurrent content-based flows. In such settings we expect the receivers with higher reception rates to start the congestion control before low rate receivers. In the next experiment we use 3 publishers and 3 subscribers (i.e., S_1 to S_3 and P_1 to P_3 in Figure 5.8) with each publisher sending messages at a constant rate of 1000mps. Each subscriber receives messages from only one publisher but matching ratios are different, inducing three content-based flows going through the bottleneck link with different average rates (all messages are of the same size). At the beginning of the experiment the bottleneck link has a capacity of 2Mbps, causing no contention among the flows. Then at $t = 100$ seconds we cut the link bandwidth in half (1Mbps), and again in half at $t = 200$ seconds (500Kbps). Figure 5.11 shows the reception rates (messages

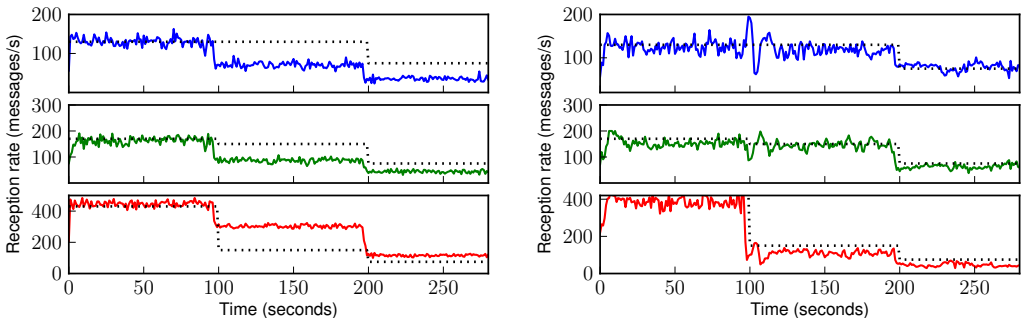


Figure 5.11. The solid lines show reception rates (mps) for 3 pairs of publishers and subscribers sharing the bottleneck link (a) without and (b) with congestion control in effect. The dotted lines show the fair share of each flow.

per second) for the three receivers (solid lines) and compares them with the fair share of each flow for the network configuration at that time. Without congestion control (Figure 5.11a), each reduction in link capacity results in a proportional rate reduction for each receiver, which amounts to an unfair allocation of resources. On the other hand, when congestion control is in effect (Figure 5.11b), the available link capacity is shared among the three separate flows so as to follow the exact demands of each client when there is enough bandwidth, and to share the available bandwidth in a fair manner when bandwidth is limited. In particular, when bandwidth is halved to 1Mbps at $t = 100$ seconds, the high rate receiver starts the congestion control process asking its corresponding sender to reduce its send rate. As a result, the average reception rate at this subscriber (Figure 5.11b (bottom)) is reduced from 400mps to 140mps, approximately the same as the other two receivers (receiving messages at 150mps). At $t = 200$ seconds when link capacity is again halved to 500Kbps, all three subscribers reduce their reception rate in a balanced way.

5.3.4 TCP friendliness

We conducted several experiments to investigate the TCP friendliness of our congestion control protocol. In all cases, we observed that the short and long-term throughput of our protocol does not exceed that of TCP when flows share a bottleneck link. In this section we present the results of a simple setup involving a TCP flow and a publish/subscriber flow (i.e., T_s, T_r, S_1, and P_1 in Figure 5.8). The bottleneck link has a bandwidth of 1Mbps. The TCP data flow starts at time $t = 60$ (indicated by the vertical dotted line). As shown in Figure 5.12a, in the absence of congestion control, the

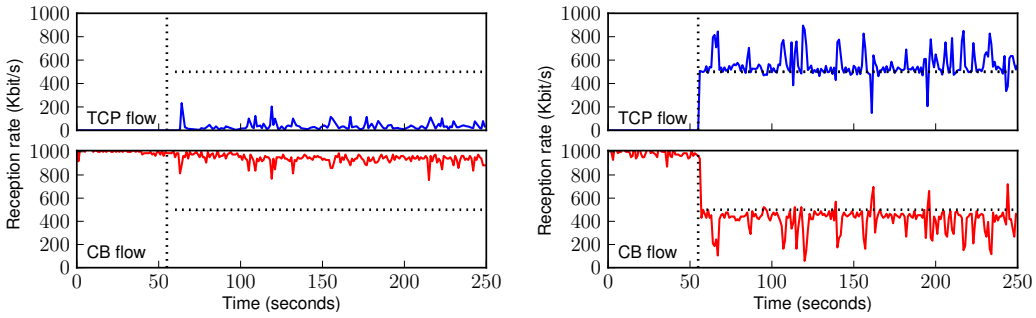


Figure 5.12. TCP and content-based reception rate (Kbps) for a TCP flow and a publish/subscribe flow sharing a bottleneck link (a) without and (b) with congestion control in place. The horizontal dotted lines show the ideal fair share.

content-based flow dominates the link capacity, pushing more than 900Kbps and leaving limited resources for the TCP connection. (The horizontal dotted line indicates an ideal fair share.) This result in fact shows the importance of a rate control scheme for deployment of best-effort content-based systems in tandem with TCP networks. With congestion control in place (Figure 5.12b) once the TCP flow starts, the content-based flow adapts its rate in less than two seconds. The average link share is slightly higher for TCP, with a difference between the two of about 5% of the total bandwidth. This is because equation-based congestion control protocols tend to be more conservative than TCP [VLB05].

5.3.5 Large scale deployment

We now study the performance of the protocol in a large network with hundreds of clients. The purpose of this experiment is to probe the protocol's effectiveness and the overhead it imposes on the publishers in a large scale deployment. This experiment involves 46 physical machines hosting 8 brokers and 760 clients where each broker runs on a dedicated physical machine and the remaining 38 hosts run client applications (20 instances per machine). The broker topology has a diameter of 6 and brokers

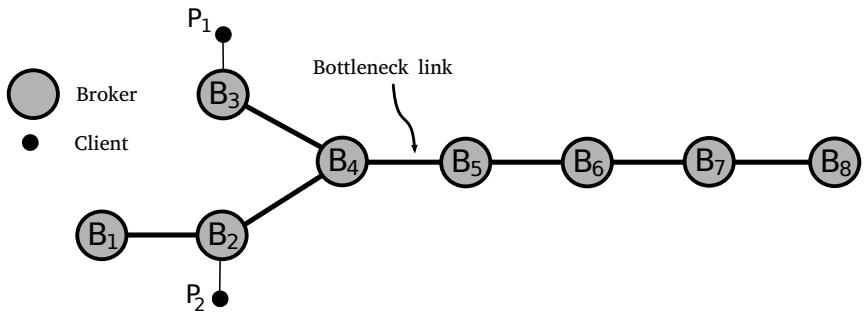


Figure 5.13. Broker topology for large scale experiment

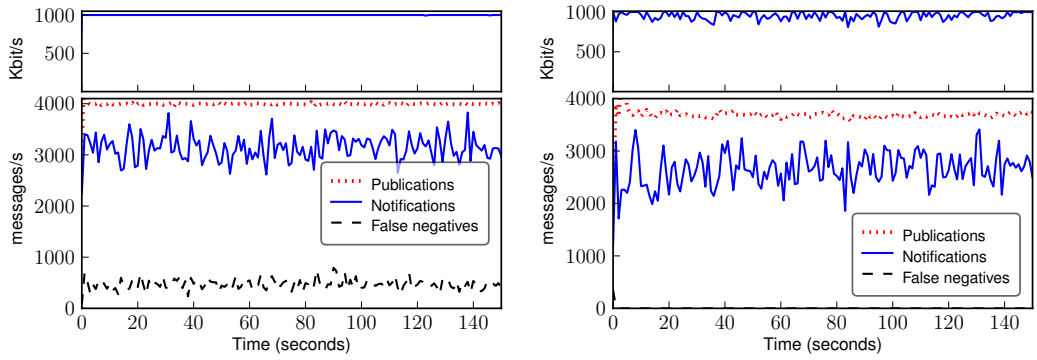
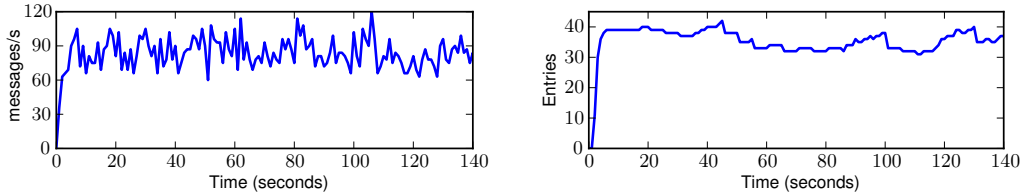


Figure 5.14. Publication, reception, and false negative rate (mps) in a large scale network (a) without and (b) with congestion control in place.

have 1 to 3 neighbors (See Figure 5.13). All inter-broker links have a bandwidth of 10Mbps except for one bottleneck link with a capacity of 1Mbps. Each broker serves 95 clients. In this setup, we have only two publishers placed behind the bottleneck link and each publishing at a constant rate of 2000mps. Each of the 758 subscribers has one filter with 1 to 5 constraints defined by a Zipfian popularity distribution so that most subscribers receive low rate flows while a few receive high-rate flows.

The bottom graphs of figures 5.14a and 5.14b juxtapose the throughput and message loss rate with and without congestion control, while the top graphs show traffic that passes through the bottleneck link. With congestion control the rate of false negatives drops to nearly zero within three seconds and remains unchanged during the entire experiment. However, the average throughput in terms of message delivery is about 12% lower with congestion control, with more than 90% of the link's capacity utilized during the experiment. Another positive effect of congestion control is that the average message delivery delay was 54 milliseconds, as compared to 102 milliseconds without congestion control, an advantage that would presumably scale with the size of the broker network.



(a) (b)

Figure 5.15. (a) the number of received feedback messages (mps) for one of the publishers. (b) changes in the number of entries in the publisher's state table.

Now turning our attention to the publisher's overhead, figures 5.15a and 5.15b show the number of feedback messages (mps) received and the number of entries in the flow table for one of the publishers. Given that 760 subscribers continuously receive messages from this publisher, the traffic and state overhead for the publisher are negligible. In fact, during the experiment we did not observe any tangible increase in memory and processing load on any of the machines where the publishers were running. Also, the relative stability of the number of feedback messages and entries in the state table reflects a stable functionality of the congestion control protocol and a smoothness of rate-limited flows.

5.3.6 Concurrent operation with the recovery protocol

We conclude this evaluation by presenting the results of an experiment with both congestion control and recovery protocols enabled to show that the two protocols interact smoothly and that the recovery does not cause traffic anomalies. Figure 5.16 shows the topology we setup for this experiment. Broker B_1 serves 5 publishers each publishing at a rate of 300mps. Brokers B_2, B_3, and B_4 each serve 5 subscribers. Links B_1–B_2 and B_1–B_3 have a limited bandwidth of 1.5 Mbps while link B_1–B_4 has a capacity of 10mbps which is ample for the given traffic. Dotted arrows show the route of messages from B_1 to other brokers.

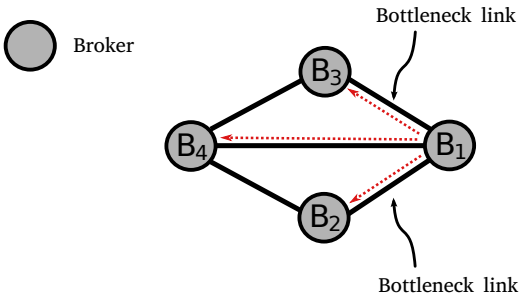


Figure 5.16. Experiment topology with both recovery and congestion control protocols in use

Looking at Figure 5.17, without transport protocol (left) the persistent loss rate is about 5500mps, while it averages around 50mps with transport protocol (right). Here the plot shows the message loss observed by the application, i.e., the losses not recovered by the recovery protocol (about 40% of the message losses were recovered by the recovery protocol). With transport protocol the total throughput is slightly lower though, since the subscribers served by broker B_4 whose subscriptions overlap with those of other subscribers on B_2 and B_3 receive messages at a suboptimal rate.

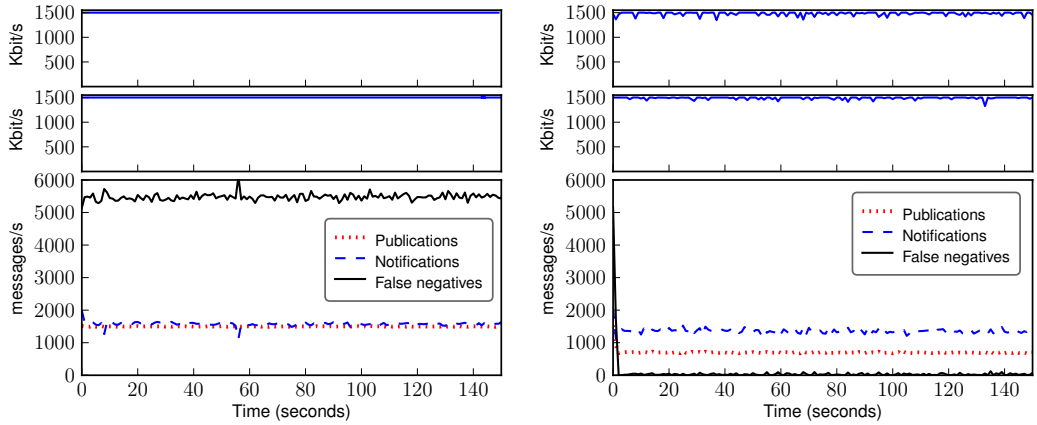


Figure 5.17. Network dynamics (a) without and (b) with transport protocol in place. Top and middle: traffic rate (Kbps) on the two bottleneck links. Bottom: aggregate publication, reception, and false negative rate (messages per second) during the experiment.

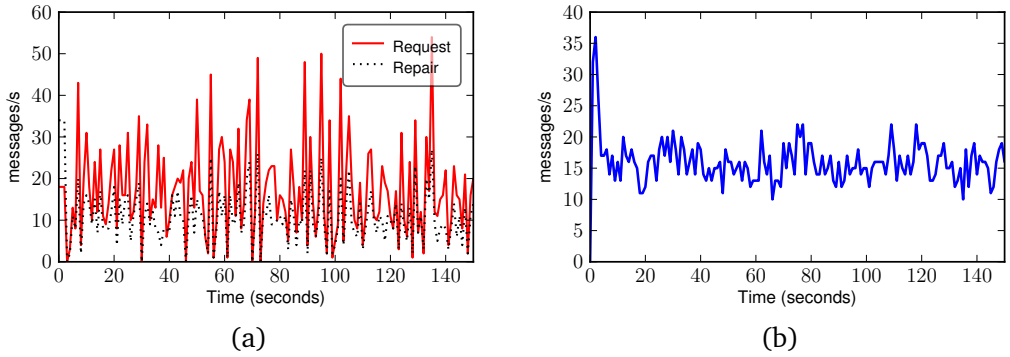


Figure 5.18. Changes in (a) total number of request/repair messages (mps), (b) number of received feedback messages for one of the publishers.

We also measure the network overhead caused by control messages. Figure 5.18(a) shows the number of request and repair messages (mps) that were sent during the ex-

periment. The number of loss recoveries (not plotted here) is roughly three times the number of repairs, due to the shared losses. The plot suggests that the request/repair traffic is negligible compared to ordinary network traffic. Figure 5.18(b) shows the number of feedback messages (mps) that were received by one of the publishers (other publishers experienced similar overhead). Except for the brief period at the beginning of the experiment, the feedback overhead averages around 15mps. Given the publication rate (300mps) and the total number of subscribers (15 clients), this feedback rate is favorably small.

5.4 Conclusion

We presented an end-to-end content-aware congestion control scheme for best-effort content-based networks. In particular we adapted an equation-based rate control scheme to the specific context of content-based communication. We did that by developing a specific notion of content-based flow. We argued that to be effective and fair an end-to-end congestion control protocol for CBPS networks should be content-aware, in that it should account for subscriptions in the congested area of the network, their covering relations and, the content of messages that flow along the congested paths.

We also presented the results of the experimental evaluation of a full implementation of our protocol. Our results show that the protocol achieves its primary objectives, namely effective resource usage, adaptability, and fairness. In reaction to changing bottleneck bandwidth, the protocol rapidly adapts to new bandwidth constraints, while keeping the loss rate to a minimum. In all experiments more than 90% of the bottleneck link bandwidth is saturated. Moreover, the protocol meets its TCP friendliness requirement, although it behaves more conservatively than TCP. We experimentally showed that in practice, even with a publication record of size 2, the protocol achieves a very good effectiveness and significantly reduces persistent message loss. Finally, we demonstrated that the loss recovery protocol interacts smoothly with the congestion control protocol.

Chapter 6

Conclusion

We believe simplicity and performance are key requirements for any data communication protocol to enjoy Internet-scale adoption. Best-effort content-based networking, thanks to its favorable performance and simple architecture, has great potential to emerge as a widespread communication service to support a variety of data-driven applications. Within a modular setting, primitives like message ordering and reliable delivery can be built atop the basic service offered by the content-based network, by involving clients and potentially other dedicated network components in the process. This thesis is an effort in this direction, to design, implement, and evaluate a set of end-to-end protocols, integrated into a single transport protocol to improve message ordering, reliability, and fairness of the basic best-effort service. Our goal was to achieve this with a minimum compromise of the high throughput and end-to-end delivery delay that such systems offer. Our transport protocol uses the publish/subscribe as a black box and so, works with any best-effort publish/subscribe network.

The problems that are targeted by this thesis have been extensively studied in the domain of IP networking and distributed systems. In developing a transport protocol for content-based networks, we believe the large body of literature on IP networking and decades of experience with TCP/IP stack are valuable assets. In fact, our protocol borrows ideas from reliability and congestion control protocols for IP multicast. However, we showed that certain characteristics of traffic and network model in content-based networking obstruct easy adoption of such solutions. We then offered solutions in three separate but inter-related components, each one discussed and evaluated in one chapter of the thesis. Below we summarize the contributions of this thesis.

6.1 Summary of Work

A major problem we dealt with in this thesis is message loss detection. Simply put, in publish/subscribe messaging, sequence numbers are incapable of indicating message loss. Loss detection is an essential requirement for correct and efficient message or-

dering, reliable delivery, and congestion control. However, correct and complete loss detection in a pure end-to-end fashion, at a minimum requires frequent acknowledgements from all subscribers back to publishers, as well as session state on the publisher's side for each subscriber. This choice would limit the scalability of our transport protocol, not to mention that is would violate loose coupling between publishers and subscribers. Thus, we opted for a probabilistic loss detection mechanism that is based on augmenting each message with a summary of the latest few messages published by the same publisher. This limited information enables subscribers to perform a limited form of loss detection with a quantifiable probability to miss a lost event.

The FIFO ordering protocol we proposed in this thesis is based on the observation that message swaps are a result of uneven processing times and heavy usage of parallel algorithms in brokers. We then developed a statistical model of delay variation with which we estimate the probability of a FIFO violation and determine a proper latch time to buffer messages upon observation of a gap in the message sequence number, in order to minimize chances of FIFO violations.

We then presented a method to enhance the reliability of best-effort publish/subscribe systems. More precisely, the goal was to enable recovery of lost messages from other end-points in the network. We used scalable reliable multicast (SRM) as a model in developing our reliability protocol with few key enhancements to facilitate adoption of SRM in our context. We proposed a method to effectively implement a request/repair recovery procedure, only using the general publish/subscribe API. We also extended SRM with a mechanism that aids network clients to adjust their cache size (the number of messages they maintain in cache to later provide repair messages) in order to reduce their memory overhead without compromise in the overall hit ratio. We analytically and experimentally showed that when message losses are shared among multiple subscribers the likelihood of loss detection and recovery is considerably improved.

In the development of our end-to-end congestion control protocol we argued that such a protocol should be content-aware, i.e., rate control should be applied only to certain message streams (content) that flow through congested paths instead of all outgoing flows of a publisher. Moreover, for efficiency reasons, the covering relation among subscriptions that belong to the congested area of the network should be considered in the congestion control process. This requires feedback by receivers and message classification by senders to center around subscriptions as well as message content. TCP friendliness was also among the requirements we had in mind, that is, coexistence of content-based flows with TCP flows, while respecting TCP fairness. Based on these requirements we presented a content-aware, TCP-friendly congestion control scheme that adopts ideas from an equation-based congestion control protocol for IP multicast, called TCP friendly multicast congestion control (TFMCC). Here again, we discussed how imprecise loss detection affects the effectiveness of our protocol and how we mitigate the problem.

Finally, we put these ideas into practice with a concrete implementation of these

components integrated into a transport protocol that sits on the client side and relays client's interactions with the network. Our implementation is in Java and was tested with Siena B-DRP, a best-effort content-based system that provided an ideal testbed for our transport protocol. The results we obtained are encouraging and show that it is possible to dramatically improve the overall service quality of a best-effort content-based network through an end-to-end protocol with small impacts on the performance of the network. Nevertheless, this protocol is not a replacement for reliable publish/subscribe systems since its probabilistic loss detection does not allow for recovery of all lost messages or deterministic FIFO ordering.

We hope this thesis will motivate and provide a basis for further research to develop more efficient end-to-end solutions for best-effort content-based networks. This research can further develop in multiple directions some of which we highlight below.

6.2 Future Research

Multipath routing and FIFO ordering. In developing our FIFO ordering protocol, we assume that all messages are disseminated on the same route and through the same set of brokers. Based on this assumption, our analysis centers around the idea that message swaps take place in brokers. However, another major cause of message reordering is usage of multipath routing (i.e., routing messages between the same end-points through multiple different paths, for load balancing and better resource utilization) [CKK12]. Although our FIFO ordering protocol is designed to capture delay variations and adaptively react to them, our statistical model is specifically tuned for delay variations caused by in-broker processing times. Therefore, it remains an open question, how the FIFO ordering protocol performs when multipath routing is in effect, and how it can be extended to better support multipath routing. Perhaps, one way to accommodate such cases is to extend the proposed statistical model, using the general probabilistic bounds we briefly mentioned in Section 3.2.2.

Improving publication record. Undoubtedly, loss detection is a major challenge in the development of end-to-end protocols for content-based networks. Thus, improvement in loss detection is essential to further enhance the protocols developed in this thesis. One way to mitigate the limitation of the proposed loss-detection mechanism is to maximize the number of messages that can be summarized into a publication record of a given size. One key to such optimizations is investigating the *temporal locality* of events, i.e., when two or more consecutive events sent out by a publisher have overlapping attribute/value pairs. We believe by exploiting temporal locality of events it is possible to enhance the encoding scheme to allow for compression that is, merging the encoded format of a few similar events in a single Bloom filter. This might further increase the likelihood of a false positive however, we believe that the overall bandwidth usage will be improved with this compression mechanism. Entries of a publication

record can be further compressed using compressed Bloom filters [Mit01] to reduce space usage and bandwidth overhead.

Alternative loss detection methods. The loss detection method proposed in this work is intended to be compatible with the concept of loose coupling between publishers and subscribers and to scale to large networks. However, it is possible to achieve more accurate loss detection mechanisms at the cost of higher load on the network and publishers. For instance, subscribers can acknowledge each received message, where an acknowledgement carries the sequence number of the messages that are being acknowledged along with the matching subscriptions. The publisher then insures that all of its previously published messages that match the filter are acknowledged by the subscriber. This of course implies large amounts of session state and processing on the publisher's side, in particular with many subscribers receiving messages from the publisher. Nevertheless, the congestion control protocol can take advantage of this in a limited way, by enabling this loss detection only for CLRs. This will improve efficiency and responsiveness of congestion control at the cost of more overhead on the publisher. Generally speaking, because loss detection has substantial impact on all performance aspects of the transport protocol, the pros and cons of such enhancements need rigorous theoretical and experimental analysis.

Another way to improve loss detection is through explicit subscription for periodic digest messages sent by publishers. These messages carry a publication record, similar to any data message, but are specifically intended to facilitate loss detection, specially for subscribers with small match probability and hence infrequent message reception rate. This can also improve loss recovery time, since our experimental results in Chapter 4 showed that recovery time is dominated by loss detection time.

Feedback suppression. Our congestion control protocol is prone to feedback implosion in cases where a large number of subscribers have the exact same subscription, share the same bottleneck link and happen to experience the same or very similar end-to-end delays to a publisher. In such rare cases it is possible that all these subscribers send feedback messages to the same publisher in a short time frame, causing large bandwidth overhead to the network and processing load on the publisher. However, this process can not last for prolonged time periods, because once a CLR is selected for a filter then the rest of subscribers will stop sending feedback messages. Yet, incorporating a feedback suppression mechanism into the congestion protocol helps its scalability and robustness. Perhaps an efficient way to implement feedback suppression is similar to the mechanism we introduced in Chapter 4 to suppress request/repair duplicates.

Analytical work on TCP friendliness of congestion control. Since the appearance of the first equation-based rate control protocols, much experimental and theoreti-

cal work has been done to understand their long-term behavior and investigate their compatibility with TCP [YKL01; BBFS01; RX05; VLB05]. Most empirical studies like [YKL01; FHPW00; BBFS01] evidence TCP friendliness of such protocols. In particular, these studies show that equation-based protocols tend to behave more conservatively than TCP with regards to the long-term throughput they achieve. Our experimental results are also compatible with such observations. However, analytical work reveals that in extreme cases such protocols can exhibit non TCP compatible behaviors [RX05; VLB05]. In particular, Vojnovic and Le Boudec [VLB05] show that the accuracy of computing loss event rate plays a central role in TCP friendliness. Unfortunately the rigorous analysis by Vojnovic and Le Boudec is not applicable to our protocol mainly due to the properties of our loss detection mechanism. Therefore, better understanding of the TCP friendliness of our congestion control protocol calls for separate analytical work. Such effort will also allow for better understanding and tuning of the protocol's parameters such as publication record size.

Window-based congestion control. In Chapter 5 we argued for equation-based congestion control and its advantages over a window-based protocol for our purpose. Nevertheless, we believe this argument needs more quantitative justification. In particular, it remains an open question whether a window-based congestion control protocol with regular acknowledgements similar to what we explained above (under the item "Alternative loss detection methods") performs better in terms of TCP friendliness and limiting loss rate. Perhaps in this protocol a publisher maintains a separate transmission window for each flow and the CLR for that flow sends regular acknowledgements to the publisher. The publisher then having the subscriptions of the CLR, estimates message loss with the help of received acknowledgements and adjusts the transmission windows in a manner similar to TCP. This protocol can be thought of as an adoption of PGMCC [Riz00], whose performance compared to our protocol remains to be seen in a separate research work.

Experimental evaluation with alternative best-effort systems. Our experiments are all conducted with Siena B-DRP. Admittedly, the one hypothesis we did not empirically validate in our evaluation is that our transport protocol works with any best-effort content-based network. Unfortunately, testing this hypothesis turned out to be a major challenge, since at the time of this writing, there was not any solid and working implementations of such systems that we could use for our purpose. Regardless, we believe more extensive experiments in large-scale settings (multiple thousand clients) and with alternative implementations will help us better understand the properties of the transport protocol and its potential weaknesses.

Better support for end-to-end solutions by publish/subscribe protocols. Our protocol is designed to be oblivious to the internals of the underlying best-effort network

and only assume a generic publish/subscribe API. However, to devise better end-to-end solutions for the problems that we attacked in this thesis, a potential research direction is to study how the underlying publish/subscribe routing and forwarding mechanisms could be enhanced or designed as to offer a better support for a transport protocol that works at a higher layer. In other words, we believe that design of an end-to-end transport protocol in parallel with the design of the underlying publish/subscribe protocol, and within a framework that adheres to a modular protocol stack, is a worthwhile research effort and should yield favorable results.

Appendix A

Statistics of the sum of two Laplacian random variables

Here we illustrates the steps to derive the sum of two Laplacian random variables and how to estimate its parameter.

The probability density function of a Laplacian random variable is of the following form:

$$f_X(x) = \frac{1}{2b} e^{-\frac{|x-\mu|}{b}} \tag{A.1}$$

where b is the scale factor and μ is called the location factor of the distribution. Assume $Y = X_1 + X_2$ is a random variable defined as the sum of two independent and identically distributed Laplacian random variables with μ equal to zero and equal scale factor of b. We will derive probability density function, cumulative distribution function and quantile function of Y. Also we derive the parameter estimator of b. Note that due to linearity of expectations, $E[Y]$ is zero.

A.0.1 Probability Density Function

In order to find the probability density function of Y we note that X_1, X_2 and Y can assume negative and positive values. Hence, we need to separate cases when Y is positive from when it is negative. Here we show how $f_Y(y)$ can be derived for positive values of y. To do that, we need to consider three cases: when $X_1 > 0$, $X_2 > 0$, when $X_1 > 0$, $X_2 < 0$ and when $X_1 < 0$, $X_2 > 0$. Now, assuming $Y > 0$, $X_1 > 0$, $X_2 > 0$, we have $X_2 = Y - X_1$ and $X_1 \in [0, Y]$. Since X_1 and X_2 are two independent and identically distributed random variables we have

$$f_Y(y) = f_{X_1,X_2}(x_1, x_2) = f_X(x_1) \,.\, f_X(x_2) \quad such \;\; that \quad x_1 + x_2 = y$$

and from there

$$f_Y(y) = \int_0^y f_X(x_1) \,.\, f_X(y - x_1)\, dx = \int_0^y \frac{1}{2b}\, e^{-\frac{x_1}{b}} \,.\, \frac{1}{2b}\, e^{-\frac{(y-x_1)}{b}}\, dx = \frac{1}{4b^2}\, e^{-\frac{y}{b}} \int_0^y dx = \frac{y}{4b^2}\, e^{-\frac{y}{b}} \quad (X_1 > 0\ ,\ X_2 > 0)$$

Now we find $f_Y(y)$ for cases when $Y > 0$, $X_1 > 0$, $X_2 < 0$. Similar to what we have above, writing X_2 as $X_2 = Y - X_1$ implies that $X_1 \in [Y, \infty)$ and hence

$$f_Y(y) = \int_y^\infty f_X(x_1) \,.\, f_X(y - x_1)\, dx = \int_y^\infty \frac{1}{2b}\, e^{-\frac{x_1}{b}} \,.\, \frac{1}{2b}\, e^{\frac{(y-x_1)}{b}}\, dx = \frac{1}{4b^2}\, e^{\frac{y}{b}} \int_y^\infty e^{-\frac{2x}{b}}\, dx = \frac{1}{8b}\, e^{-\frac{y}{b}} \quad (X_1 > 0\ ,\ X_2 < 0) \tag{A.2}$$

Due to symmetry of the Laplace distribution, the third case where $X_1 < 0, X_2 > 0$ yields the same result as Formula A.2. Now $f_Y(y)$ is the sum of three cases:

$$f_Y(y) = \frac{1}{8b}\, e^{-\frac{y}{b}} + \frac{1}{8b}\, e^{-\frac{y}{b}} + \frac{y}{4b^2}\, e^{-\frac{y}{b}} = \frac{(b+y)}{4b^2}\, e^{-\frac{y}{b}} \quad (y \geq 0)$$

By following the same method for cases when $y < 0$ we can find the general formula of the density function for positive and negative values of y which is the following

$$f_Y(y) = \frac{(b + |y|)}{4b^2}\, e^{-\frac{|y|}{b}}$$

A.0.2 Cumulative Density Function and Quantile Function

To find the cumulative density function denoted by $F_X(x)$, we can conveniently utilize the symmetry of the density function. Therefore, we begin by finding $F_X(x)$ for cases when x is negative. We have

$$F_X(x) = \int_{-\infty}^x \frac{(b - x)}{4b^2}\, e^{\frac{x}{b}}\, dx = \frac{1}{4b^2} \int_{-\infty}^x (b - x)\, e^{\frac{x}{b}}\, dx$$

Using integration by parts, the above integral yields

$$F_X(x) = \frac{1}{4b^2}[(b - x).(b\, e^{\frac{x}{b}}) + \int_{-\infty}^x b\, e^{\frac{x}{b}}\, dx\ ] = \frac{(2b - x)}{4b}\, e^{\frac{x}{b}} \quad (x \leq 0)$$

and from there we can derive $F_X(x)$ for positive values of x which yield the following formula

$$F_X(x) = 1 - \frac{(2b+x)}{4b}\, e^{-\frac{x}{b}} \quad (x \geq 0) \tag{A.3}$$

The general form of $F_X(x)$ for both negative and positive values of x can be written as the following

$$F_X(x) = 0.5\ [1 + sgn(x)(1 - \frac{(2b+|x|)}{2b}\, e^{-\frac{|x|}{b}})\ ]$$

where $sgn(x)$ is the sign of x. In order to to find $F^{-1}(p)$, the quantile function, we note that for $p \leq 0.5$, the value of $F^{-1}(p)$ is negative and for $p \geq 0.5$ it is positive. Now assuming that $p \geq 0.5$, we find the inverse of the Equation A.3. Denoting $F_X(x)$ by p we have

$$\begin{aligned}
&p = 1 - \frac{(2b+x)}{4b}\, e^{-\frac{x}{b}} && thus \\
&1 - p = \frac{(2+\frac{x}{b})}{4}\, e^{-\frac{x}{b}} && so \\
&-(2+\frac{x}{b})\, e^{-\frac{x}{b}} = 4(p-1) && which\ can\ be\ written\ as \\
&-t\, e^{-t} = 4\, e^{-2}\, (p-1) && where\ \ t = \frac{x}{b} + 2
\end{aligned}$$

The above equation is solved by ω, the *Lambert Omega Function*. This function solves the equation $y = x\, e^x$ by $x = \omega(y)$. As such, the answer to the above equation is

$$t = -\omega(4\, e^{-2}\, (p-1))$$

By substituting t back, we get the final form of the quantile function

$$F^{-1}(p) = -b\ [\omega(4\, e^{-2}\, (p-1)) - 2)\ ] \qquad (0.5 \leq p \leq 1)$$

A.0.3 Parameter Estimation

Having a set of n samples $x_1, ..., x_n$, the parameter of the probability density function can be found trough maximum likelihood estimation. We denote the estimator of b by β. Given that the values of each sample is independent of the other samples we have

$$p(x_1,...,x_n|\beta) = \prod_{i=1}^{n} p(x_i|\beta) = \frac{1}{(4\beta^2)^n} \prod_{i=1}^{n} (\beta + |x_i|)\, e^{-\frac{|x_i|}{\beta}}$$

The objective is to find the value of β that maximizes the preceding function. Since it is a positive value (the product of some probabilities) it follows that maximizing its natural logarithm is equal to maximizing the function itself. Taking natural log from the function we have

$$\ln(p(x_1,...,x_n|\beta)) = \ln(\frac{1}{(4\beta^2)^n} \prod_{i=1}^{n} (\beta + |x_i|)\, e^{-\frac{|x_i|}{\beta}}) =$$
$$-n\ln(4\beta^2) + \sum_{i=1}^{n} (\beta + |x_i|) - \sum_{i=1}^{n} \frac{|x_i|}{\beta} \tag{A.4}$$

Formula A.4 can be seen as a function of β. To find the value of β that maximizes the value of this function, we take its derivative with respect to β. Equating the resulted function to zero and solving for β, yields the answer. Thus, after taking derivative and equating to zero we have

$$-\frac{2n}{\beta} + \sum_{i=1}^{n} \frac{1}{(\beta + |x_i|)} + \sum_{i=1}^{n} \frac{|x_i|}{\beta^2} = 0 \qquad so$$
$$\sum_{i=1}^{n} |x_i| + \beta^2 \sum_{i=1}^{n} \frac{1}{(\beta + |x_i|)} - 2n\beta = 0 \tag{A.5}$$

The above equation does not have a close form answer. In order to approximate its root we approximate the second term of the left side of the equation as seen below. Showing expected value of the random variable X by $E[X]$ we see that

$$E\,[\,\frac{1}{(\beta + |x|)}\,] = \int_{-\infty}^{\infty} \frac{1}{(\beta + |x|)}\, f_X(x)\, dx = 2\int_{0}^{\infty} \frac{1}{(\beta + x)}\, f_X(x)\, dx =$$
$$2\int_{0}^{\infty} \frac{1}{(\beta + x)}\, \frac{(\beta + x)}{4\beta^2} e^{-\frac{x}{\beta}}\, dx = \frac{1}{2\beta^2}\int_{0}^{\infty} e^{-\frac{x}{\beta}}\, dx = \frac{1}{2\beta}$$

Therefore

$$E\,[\,\sum_{i=1}^{n} \frac{1}{(\beta + |x|)}\,] = \frac{n}{2\beta}$$

Substituting this value back in Equation A.5 we will have

$$\sum_{i=1}^{n} |x_i| + \frac{\beta n}{2} - 2n\beta = 0 \qquad that\ gives$$

$$\beta = \frac{2}{3n} \sum_{i=1}^{n} |x_i|$$

Bibliography

[ANO96] S. Asmussen, O. Nerman, and M. Olsson. Fitting phase-type distribution via the em algorithm. *Scandinavian Journal of Statistics*, 23:419–441, 1996.

[AS00] M. K. Aguilera and R. E. Strom. Efficient atomic broadcast using deterministic merge. In *PODC '00: Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*, pages 209–218, 2000. ACM.

[ASS+99] M. K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley, and T. D. Chandra. Matching events in a content-based subscription system. In *Proceedings of the eighteenth annual ACM symposium on Principles of distributed computing*, PODC '99, pages 53–61, 1999. ACM.

[AT05] I. Aekaterinidis and P. Triantafillou. Internet scale string attribute publish/subscribe data networks. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, CIKM '05, pages 44–51, 2005. ACM.

[AT06] I. Aekaterinidis and P. Triantafillou. Pastrystrings: A comprehensive content-based publish/subscribe dht network. In *Distributed Computing Systems, 2006. ICDCS 2006. 26th IEEE International Conference on*, page 23, 2006.

[AT07] I. Aekaterinidis and P. Triantafillou. Publish-subscribe information delivery with substring predicates. *IEEE Internet Computing*, 11(4):16–23, July 2007.

[AT11] I. Aekaterinidis and P. Triantafillou. Pyracanthus: A scalable solution for dht-independent content-based publish/subscribe data networks. *Inf. Syst.*, 36(3):655–674, May 2011.

[Bar64] P. Baran. On distributed communications networks. *Communications Systems, IEEE Transactions on*, 12(1):1 –9, march 1964.

[BBFS01] D. Bansal, H. Balakrishnan, S. Floyd, and S. Shenker. Dynamic behavior of slowly-responsive congestion control algorithms, 2001.

[BBPP07] M. Balakrishnan, K. Birman, A. Phanishayee, and S. Pleisch. Ricochet: Lateral error correction for time-critical multicast. In *NSDI 2007: Fourth Usenix Symposium on Networked Systems Design and Implementation*, 2007.

[BBQ+07] R. Baldoni, R. Beraldi, V. Quema, L. Querzoni, and S. Tucci-Piergiovanni. Tera: topic-based event routing for peer-to-peer architectures. In *Proceedings of the 2007 inaugural international conference on Distributed event-based systems*, DEBS '07, pages 2–13, 2007. ACM.

[BBQV04] R. Baldoni, R. Beraldi, L. Querzoni, and A. Virgillito. Subscription-driven self-organization in content-based publish/subscribe. *Autonomic Computing, International Conference on*, 0:332–333, 2004.

[BCC+98] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang. Recommendations on queue management and congestion avoidance in the internet, 1998.

[BCM+99] G. Banavar, T. Ch, B. Mukherjee, J. Nagarajarao, R. E. Strom, and D. C. Sturman. An efficient multicast protocol for content-based publish-subscribe systems. In *ICDCS '99: Proceedings of the 19th IEEE International Conference on Distributed Computing Systems*, pages 262–272, 1999.

[BEG04] S. Baehni, P. T. Eugster, and R. Guerraoui. Data-aware multicast. In *Proceedings of the 2004 International Conference on Dependable Systems and Networks*, DSN '04, pages 233–, 2004. IEEE Computer Society.

[BFDG07] S. Bianchi, P. Felber, A. Datta, and M. Gradinariu. Stabilizing peer-to-peer spatial filters. In *Distributed Computing Systems, 2007. ICDCS '07. 27th International Conference on*, page 27, june 2007.

[BFG07] S. Bianchi, P. Felber, and M. Gradinariu. Content-based publish/subscribe using distributed r-trees. In *Euro-Par*, pages 537–548, 2007.

[BHL+00] J. W. Byers, G. Horn, M. Luby, M. Mitzenmacher, and W. Shaver. Flid-dl: congestion control for layered multicast. In *Proceedings NGC 2000*, pages 71–81, 2000.

[BMVV05] R. Baldoni, C. Marchetti, A. Virgillito, and R. Vitenberg. Content-based publish-subscribe over structured overlay networks. In *ICDCS '05: Pro-*

ceedings of the 25th IEEE International Conference on Distributed Computing Systems, pages 437–446, 2005. IEEE Computer Society.

[Bol93] J.-C. Bolot. End-to-end packet delay and loss behavior in the internet. In *SIGCOMM '93: Conference proceedings on Communications architectures, protocols and applications*, pages 289–298, 1993. ACM.

[BOP94] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson. Tcp vegas: new techniques for congestion detection and avoidance. In *Proceedings of the conference on Communications architectures, protocols and applications*, SIGCOMM '94, pages 24–35, 1994. ACM.

[BPS99] J. C. R. Bennett, C. Partridge, and N. Shectman. Packet reordering is not pathological network behavior. *IEEE/ACM Trans. Netw.*, 7(6):789–798, 1999.

[BRS02] A. R. Bharambe, S. Rao, and S. Seshan. Mercury: a scalable publish-subscribe system for internet games. In *Proceedings of the 1st workshop on Network and system support for games*, NetGames '02, pages 3–9, 2002. ACM.

[BSB+02] S. Bhola, R. E. Strom, S. Bagchi, Y. Zhao, and J. S. Auerbach. Exactly-once delivery in a content-based publish-subscribe system. In *DSN '02: Proceedings of the 2002 International Conference on Dependable Systems and Networks*, pages 7–16, 2002. IEEE Computer Society.

[BTK99] S. Bhattacharyya, D. Towsley, and J. Kurose. The loss path multiplicity problem in multicast congestion control. In *IN PROC. OF IEEE INFOCOM*, pages 856–863, 1999.

[BZA03] S. Bhola, Y. Zhao, and J. Auerbach. Scalably supporting durable subscriptions in a publish/subscribe system. In *proceeding of the international conference on dependable systems and networks (DSN 2003)*, pages 57–66, 2003.

[CCC+01] A. Campailla, S. Chaki, E. Clarke, S. Jha, and H. Veith. Efficient filtering in publish-subscribe systems using binary decision diagrams. In *Proceedings of the 23rd International Conference on Software Engineering*, ICSE '01, pages 443–452, 2001. IEEE Computer Society.

[CDKR02] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. Scribe: a large-scale and decentralized application-level multicast infrastructure. *Selected Areas in Communications, IEEE Journal on*, 20(8):1489 – 1499, oct 2002.

[CDNF01] G. Cugola, E. Di Nitto, and A. Fuggetta. The jedi event-based infrastructure and its application to the development of the opss wfms. *IEEE Trans. Softw. Eng.*, 27(9):827–850, 2001.

[CF99] F. Cristian and C. Fetzer. The timed asynchronous distributed system model. *IEEE Trans. Parallel Distrib. Syst.*, 10(6):642–657, June 1999.

[CF04] R. Chand and P. Felber. Xnet: A reliable content-based publish/subscribe system. In *Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems*, SRDS '04, pages 264–273, 2004. IEEE Computer Society.

[CF05] R. Chand and P. Felber. Semantic peer-to-peer overlays for publish/subscribe networks. In *Proceedings of the 11th international Euro-Par conference on Parallel Processing*, Euro-Par'05, pages 1194–1204, 2005. Springer-Verlag.

[CJV10] C. Chen, H.-A. Jacobsen, and R. Vitenberg. Divide and conquer algorithms for publish/subscribe overlay design. *Distributed Computing Systems, International Conference on*, 0:622–633, 2010.

[CKK12] A. Carzaniga, K. Khazaei, and F. Kuhn. Oblivious low-congestion multicast routing in wireless networks. In *Proceedings of the thirteenth ACM international symposium on Mobile Ad Hoc Networking and Computing*, MobiHoc '12, pages 155–164, June 2012.

[CMA97] F. Cristian, S. Mishra, and G. Alvarez. High-performance asynchronous atomic broadcast. *Distributed System Engineering Journal*, 33, 1997.

[CMPC03] P. Costa, M. Migliavacca, G. P. Picco, and G. Cugola. Introducing reliability in content-based publish-subscribe through epidemic algorithms. In *Proceedings of the 2nd international workshop on Distributed event-based systems*, DEBS '03, pages 1–8, 2003. ACM.

[CMTV07a] G. Chockler, R. Melamed, Y. Tock, and R. Vitenberg. Constructing scalable overlays for pub-sub with many topics. In *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*, PODC '07, pages 109–118, 2007. ACM.

[CMTV07b] G. Chockler, R. Melamed, Y. Tock, and R. Vitenberg. Spidercast: a scalable interest-aware overlay for topic-based pub/sub communication. In *Proceedings of the 2007 inaugural international conference on Distributed event-based systems*, DEBS '07, pages 14–25, 2007. ACM.

[Cou02] S. Courtenage. Specifying and detecting composite events in content-based publish/subscribe systems. In *Proceedings of the 22nd International Conference on Distributed Computing Systems*, ICDCSW '02, pages 602–610, 2002. IEEE Computer Society.

[CP05] P. Costa and G. Picco. Semi-probabilistic content-based publish-subscribe. In *Distributed Computing Systems, 2005. ICDCS 2005. Proceedings. 25th IEEE International Conference on*, pages 575 –585, june 2005.

[CQL08] D. Cutting, A. Quigley, and B. Landfeldt. Spice: Scalable p2p implicit group messaging. *Comput. Commun.*, 31(3):437–451, February 2008.

[CRW01] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Trans. Comput. Syst.*, 19(3):332–383, 2001.

[CRW04] A. Carzaniga, M. Rutherford, and A. Wolf. A routing scheme for content-based networking. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 2, pages 918 – 928 vol.2, march 2004.

[CS05] F. Cao and J. P. Singh. Medym: match-early with dynamic multicast for content-based publish-subscribe networks. In *Proceedings of the ACM/IFIP/USENIX 2005 International Conference on Middleware*, Middleware '05, pages 292–313, 2005. Springer-Verlag New York, Inc.

[CTCHW09] A. Carzaniga, G. Toffetti Carughi, C. Hall, and A. L. Wolf. Practical high-throughput content-based routing using unicast state and probabilistic encodings. Technical Report 2009/06, Faculty of Informatics, University of Lugano, August 2009.

[CVJ11] C. Chen, R. Vitenberg, and H.-A. Jacobsen. Scaling construction of low fan-out overlays for topic-based publish/subscribe systems. In *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*, pages 225 –236, june 2011.

[CW03] A. Carzaniga and A. L. Wolf. Forwarding in a content-based network. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '03, pages 163–174, 2003. ACM.

[CW06] S. Courtenage and S. Williams. The design and implementation of a p2p-based composite event notification system. In *Advanced Information Networking and Applications, 2006. AINA 2006. 20th International Conference on*, volume 1, pages 701 –706, april 2006.

[DGH+06] A. Demers, J. Gehrke, M. Hong, M. Riedewald, and W. White. Towards expressive publish/subscribe systems. In *Proceedings of the 10th international conference on Advances in Database Technology*, EDBT'06, pages 627–644, 2006. Springer-Verlag.

[DRF04] Y. Diao, S. Rizvi, and M. J. Franklin. Towards an internet-scale xml dissemination service. In *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*, VLDB '04, pages 612–623. VLDB Endowment, 2004.

[DSU04] X. Défago, A. Schiper, and P. Urbán. Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Comput. Surv.*, 36(4):372–421, December 2004.

[ECG09] C. Esposito, D. Cotroneo, and A. Gokhale. Reliable publish/subscribe middleware for time-sensitive internet-scale applications. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, DEBS '09, pages 16:1–16:12, 2009. ACM.

[EFGK03] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, 2003.

[EGN08] S. Erramilli, S. Gadgil, and N. Natarajan. Efficient assignment of multicast groups to publish-subscribe information topics in tactical networks. In *Military Communications Conference, 2008. MILCOM 2008. IEEE*, pages 1 –7, nov. 2008.

[ERB+12] C. Esposito, S. Russo, R. Beraldi, M. Platania, and R. Baldoni. Achieving reliable and timely event dissemination over wan. In L. Bononi, A. K. Datta, S. Devismes, and A. Misra, editors, *ICDCN*, volume 7129 of *Lecture Notes in Computer Science*, pages 265–280. Springer, 2012.

[FFTJ09] A. Farroukh, E. Ferzli, N. Tajuddin, and H.-A. Jacobsen. Parallel event processing for content-based publish/subscribe systems. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, DEBS '09, pages 8:1–8:4, 2009. ACM.

[FGKZ03] L. Fiege, F. C. Gärtner, O. Kasten, and A. Zeidler. Supporting mobility in content-based publish/subscribe middleware. In *Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware*, Middleware '03, pages 103–122, 2003. Springer-Verlag New York, Inc.

[FHPW00] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-based congestion control for unicast applications. *SIGCOMM Comput. Commun. Rev.*, 30:43–56, August 2000.

[FJL+97] S. Floyd, V. Jacobson, C.-G. Liu, S. McCanne, and L. Zhang. A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM Trans. Netw.*, 5(6):784–803, 1997.

[FJL+01] F. Fabret, H. A. Jacobsen, F. Llirbat, J. Pereira, K. A. Ross, and D. Shasha. Filtering algorithms and implementation for very fast publish/subscribe systems. In *Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, SIGMOD '01, pages 115–126, 2001. ACM.

[FJLM05] E. Fidler, H. A. Jacobsen, G. Li, and S. Mankovski. The padres distributed publish/subscribe system. In *In 8th International Conference on Feature Interactions in Telecommunications and Software Systems*, pages 12–30, 2005.

[FMMB02] L. Fiege, M. Mezini, G. Mühl, and A. P. Buchmann. Engineering event-based systems with scopes. In *Proceedings of the 16th European Conference on Object-Oriented Programming*, ECOOP '02, pages 309–333, 2002. Springer-Verlag.

[GCV+10] S. Girdzijauskas, G. Chockler, Y. Vigfusson, Y. Tock, and R. Melamed. Magnet: practical subscription clustering for internet-scale publish/subscribe. In *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems*, DEBS '10, pages 172–183, 2010. ACM.

[GKP99] R. Gruber, B. Krishnamurthy, and E. Panagos. The architecture of the ready event notification service. In *Electronic Commerce and Web-based Applications/Middleware, 1999. Proceedings. 19th IEEE International Conference on Distributed Computing Systems Workshops on*, pages 108 –113, 1999.

[GLAM97] J. Garcia-Luna-Aceves and S. Murthy. A path-finding algorithm for loop-free routing. *Networking, IEEE/ACM Transactions on*, 5(1):148 –160, feb 1997.

[GLZ11] S. Gao, G. Li, and P. Zhao. Marshmallow: A content-based publish-subscribe system over structured p2p networks. In *Computational Intelligence and Security (CIS), 2011 Seventh International Conference on*, pages 290 –294, dec. 2011.

[GS99] S. Golestani and K. Sabnani. Fundamental observations on multicast congestion control in the internet. In *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, pages 990 –1000 vol.2, mar 1999.

[GSAA04] A. Gupta, O. D. Sahin, D. Agrawal, and A. E. Abbadi. Meghdoot: content-based publish/subscribe over p2p networks. In *Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*, Middleware '04, pages 254–273, 2004. Springer-Verlag New York, Inc.

[Hol11] V. Holopainen. Assignment of multicast groups to publish/subscribe topics in multi-domain networks. In *Computers and Communications (ISCC), 2011 IEEE Symposium on*, pages 664 –670, 28 2011-july 1 2011.

[HT93] V. Hadzilacos and S. Toueg. *Fault-tolerant broadcasts and related problems*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1993.

[JE10] K. Jayaram and P. Eugster. Scalable efficient composite event detection. In D. Clarke and G. Agha, editors, *Coordination Models and Languages*, volume 6116 of *Lecture Notes in Computer Science*, pages 168–182. Springer Berlin / Heidelberg, 2010.

[JE11] K. Jayaram and P. Eugster. Split and subsume: Subscription normalization for effective content-based messaging. In *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*, pages 824 –835, june 2011.

[JHMV09] H. Jafarpour, B. Hore, S. Mehrotra, and N. Venkatasubramanian. Ccd: efficient customized content dissemination in distributed publish/subscribe. In *Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware*, Middleware '09, pages 4:1–4:20, 2009. Springer-Verlag New York, Inc.

[JJE10] K. R. Jayaram, C. Jayalath, and P. Eugster. Parametric subscriptions for content-based publish/subscribe networks. In *Proceedings of the ACM/IFIP/USENIX 11th International Conference on Middleware*, Middleware '10, pages 128–147, 2010. Springer-Verlag.

[JMV08] H. Jafarpour, S. Mehrotra, and N. Venkatasubramanian. A fast and robust content-based publish/subscribe architecture. In *NCA '08: Proceedings of the 2008 Seventh IEEE International Symposium on Network Computing and Applications*, pages 52–59, 2008. IEEE Computer Society.

[JMV09] H. Jafarpour, S. Mehrotra, and N. Venkatasubramanian. Dynamic load balancing for cluster-based publish/subscribe system. In *SAINT '09: Proceedings of the 2009 Ninth Annual International Symposium on Applications and the Internet*, pages 57–63, 2009. IEEE Computer Society.

[JZR+09] P. Jokela, A. Zahemszky, C. E. Rothenberg, S. Arianfar, and P. Nikander. LIPSIN: Line Speed Publish/Subscribe Inter-networking. In *SIGCOMM '09: Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, pages 195–206, 2009. ACM.

[KCW11] A. Konstantinidis, A. Carzaniga, and A. L. Wolf. A content-based publish/subscribe matching algorithm for 2d spatial objects. In *ACM/IFIP/USENIX 12th International Middleware Conference*, number 7049 in LNCS, pages 208–227, December 2011.

[KJ09] R. S. Kazemzadeh and H.-A. Jacobsen. Reliable and highly available distributed publish/subscribe service. In *SRDS '09: Proceedings of the 2009 28th IEEE International Symposium on Reliable Distributed Systems*, pages 41–50, 2009. IEEE Computer Society.

[KJ11] R. Kazemzadeh and H.-A. Jacobsen. Partition-tolerant distributed publish/subscribe systems. In *Reliable Distributed Systems (SRDS), 2011 30th IEEE Symposium on*, pages 101 –110, oct. 2011.

[LHJ05] G. Li, S. Hou, and H.-A. Jacobsen. A unified approach to routing, covering and merging in publish/subscribe systems based on modified binary decision diagrams. In *Distributed Computing Systems, 2005. ICDCS 2005. Proceedings. 25th IEEE International Conference on*, pages 447 –457, june 2005.

[LJ05] G. Li and H.-A. Jacobsen. Composite subscriptions in content-based publish/subscribe systems. In *Proceedings of the ACM/IFIP/USENIX 2005 International Conference on Middleware*, Middleware '05, pages 249–269, 2005. Springer-Verlag New York, Inc.

[LMJ08] G. Li, V. Muthusamy, and H.-A. Jacobsen. Adaptive content-based routing in general overlay topologies. In *Middleware '08: Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, pages 1–21, 2008. Springer-Verlag New York, Inc.

[LS96] J. C. Lin and J. L. Sanjoy. Rmtp: A reliable multicast transport protocol. In *IEEE Journal on Selected Areas in Communications*, pages 1414–1424, 1996.

[LSB06] C. Lumezanu, N. Spring, and B. Bhattacharjee. Decentralized message ordering for publish/subscribe systems. In *Proceedings of the ACM/IFIP/USENIX 2006 International Conference on Middleware*, Middleware '06, pages 162–179, 2006. Springer-Verlag New York, Inc.

[MC11] A. Margara and G. Cugola. High performance content-based matching using gpus. In *Proceedings of the 5th ACM international conference on Distributed event-based system*, DEBS '11, pages 183–194, 2011. ACM.

[Mit01] M. Mitzenmacher. Compressed bloom filters. In *Proceedings of the twentieth annual ACM symposium on Principles of distributed computing*, PODC '01, pages 144–150, 2001. ACM.

[MRI+80] J. M. Mcquillan, I. I. Richer, M. Ieee, Eric, and C. Rosen. The new routing algorithm for the arpanet. *IEEE Transactions on Communications*, 1980.

[Muk92] A. Mukherjee. On the dynamics and significance of low frequency components of internet load. *Internetworking: Research and Experience*, 5:163–205, 1992.

[MW88] J. McQuillan and D. Walden. The ARPA Network Design Decisions. *Computer Networks*, 1(5):243–289, August 1988.

[NK05] S. Nadarajah and S. Kotz. On the linear combination of laplace random variables. *Probab. Eng. Inf. Sci.*, 19(4):463–470, 2005.

[OAA+00] L. Opyrchal, M. Astley, J. Auerbach, G. Banavar, R. Strom, and D. Sturman. Exploiting ip multicast in content-based publish-subscribe systems. In *IFIP/ACM International Conference on Distributed systems platforms*, Middleware '00, pages 185–207, 2000. Springer-Verlag New York, Inc.

[OB06] K. Ostrowski and K. Birman. Extensible web services architecture for notification in large-scale systems. In *ICWS '06: Proceedings of the IEEE International Conference on Web Services*, pages 383–392, 2006. IEEE Computer Society.

[OR10] M. Onus and A. W. Richa. Parameterized maximum and average degree approximation in topic-based publish-subscribe overlay network design. In *Proceedings of the 2010 IEEE 30th International Conference on Distributed Computing Systems*, ICDCS '10, pages 644–652, 2010. IEEE Computer Society.

[OR11] M. Onus and A. Richa. Minimum maximum-degree publish subscribe overlay network design. *Networking, IEEE/ACM Transactions on*, 19(5):1331 –1343, oct. 2011.

[Pax97] V. Paxson. End-to-end internet packet dynamics. *SIGCOMM Comput. Commun. Rev.*, 27(4):139–152, 1997.

[PB02] P. R. Pietzuch and J. Bacon. Hermes: A distributed event-based middleware architecture. In *ICDCSW '02: Proceedings of the 22nd International*

Conference on Distributed Computing Systems, pages 611–618, 2002. IEEE Computer Society.

[PB03] P. R. Pietzuch and S. Bhola. Congestion control in a reliable scalable message-oriented middleware. In *Middleware '03: Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware*, pages 202–221, 2003. Springer-Verlag New York, Inc.

[PFLS00] J. a. Pereira, F. Fabret, F. Llirbat, and D. Shasha. Efficient matching for web-based publish/subscribe systems. In *Proceedings of the 7th International Conference on Cooperative Information Systems*, CooplS '02, pages 162–173, 2000. Springer-Verlag.

[Pla11] M. Platania. *Ordering, Timeliness and Reliability for Publish/Subscribe Systems over WAN*. PhD thesis, Sapienza University of Rome, 2011.

[RCF+09] W. Rao, L. Chen, A.-C. Fu, H. Chen, and F. Zou. On efficient content matching in distributed pub/sub systems. In *INFOCOM 2009, IEEE*, pages 756 –764, april 2009.

[RD01] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, Middleware '01, pages 329–350, 2001. Springer-Verlag.

[RFH+01] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '01, pages 161–172, 2001. ACM.

[Riz00] L. Rizzo. pgmcc: a tcp-friendly single-rate multicast congestion control scheme. *SIGCOMM Comput. Commun. Rev.*, 30(4):17–28, 2000.

[RPS06] V. Ramasubramanian, R. Peterson, and E. G. Sirer. Corona: a high performance publish-subscribe system for the world wide web. In *NSDI'06: Proceedings of the 3rd conference on Networked Systems Design & Implementation*, pages 2–2, 2006. USENIX Association.

[RX05] I. Rhee and L. Xu. Limitations of equation-based congestion control. In *Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '05, pages 49–60, 2005. ACM.

[SA97] W. Segall and D. Arnold. Elvin has left the building: A publish/subscribe notification service with quenching. In *Proceedings of*

the 1997 Australian UNIX Users Group, Brisbane, Australia, 1997. http://elvin.dstc.edu.au/doc/papers/auug97/AUUG97.html.

[SCG01a] A. C. Snoeren, K. Conley, and D. K. Gifford. Mesh-based content routing using xml. *SIGOPS Oper. Syst. Rev.*, 35(5):160–173, 2001.

[SCG+01b] T. Speakman, J. Crowcroft, J. Gemmell, D. Farinacci, S. Lin, D. Leshchiner, M. Luby, T. Montgomery, L. Rizzo, A. Tweedly, N. Bhaskar, R. Edmonstone, R. Sumanasekera, and L. Vicisano. Pgm reliable transport protocol specification, 2001.

[SLS+10] M. Sadoghi, M. Labrecque, H. Singh, W. Shum, and H.-A. Jacobsen. Efficient event processing through reconfigurable hardware for algorithmic trading. *Proc. VLDB Endow.*, 3(1-2):1525–1528, September 2010.

[SMLN+03] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw.*, 11(1):17–32, February 2003.

[SRC84] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Trans. Comput. Syst.*, 2(4):277–288, November 1984.

[SSJ11] M. Sadoghi, H. Singh, and H.-A. Jacobsen. fpga-topss: line-speed event processing on fpgas. In *Proceedings of the 5th ACM international conference on Distributed event-based system*, DEBS '11, pages 373–374, 2011. ACM.

[SW00] D. Sisalem and A. Wolisz. Mlda: A tcp-friendly congestion control framework for heterogeneous multicast environments. In *Proceedings IWQoS 2000*, 2000.

[TA04] P. Triantafillou and I. Aekaterinidis. Content-based publish/subscribe over structured p2p networks. In *Proc. third Int. Workshop Distributed Event-based Systems (DEBSâĂŹ04), 16 of 16 R. BALDONI et al*, pages 24–25, 2004.

[TAaJ03] D. Tam, R. Azimi, and H. arno Jacobsen. Building content-based publish/subscribe systems with distributed hash tables. In *In International Workshop On Databases, Information Systems and Peer-to-Peer Computing*, pages 138–152, 2003.

[TBF+03] W. W. Terpstra, S. Behnel, L. Fiege, A. Zeidler, and A. P. Buchmann. A peer-to-peer approach to content-based publish/subscribe. In *Proceedings of the 2nd international workshop on Distributed event-based systems*, DEBS '03, pages 1–8, 2003. ACM.

[TE04] P. Triantafillou and A. Economides. Subscription summarization: a new paradigm for efficient publish/subscribe systems. In *Distributed Computing Systems, 2004. Proceedings. 24th International Conference on*, pages 562 – 571, 2004.

[TK06] S. Tarkoma and J. Kangasharju. Optimizing Content-based Routers: Posets and Forests. *Distributed Computing*, 19(1):62–77, September 2006.

[VCR98] L. Vicisano, J. Crowcroft, and L. Rizzo. Tcp-like congestion control for layered multicast data transfer. In *INFOCOM '98. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 996 –1003 vol.3, 1998.

[VGS05] S. Voulgaris, D. Gavidia, and M. V. Steen. Cyclon: Inexpensive membership management for unstructured p2p overlays. *Journal of Network and Systems Management*, 13:2005, 2005.

[VLB05] M. Vojnovic and J. Y. Le Boudec. On the Long-Run Behavior of Equation-Based Rate Control. *IEEE/ACM Transactions on Networking (TON)*, 13(3):568–581, June 2005.

[VRKS06] S. Voulgaris, E. Riviere, A.-M. Kermarrec, and M. V. Steen. Sub-2-sub: Self-organizing content-based publish subscribe for dynamic large scale collaborative networks. In *In IPTPSâĂŹ06: the fifth International Workshop on Peer-to-Peer Systems*, 2006.

[WH01] J. Widmer and M. Handley. Extending equation-based congestion control to multicast applications. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '01, pages 275–285, 2001. ACM.

[WS98] H. A. Wang and M. Schwartz. Achieving bounded fairness for multicast and tcp traffic in the internet. In *Proceedings of the ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '98, pages 81–92, 1998. ACM.

[YCJ06] A. K. Yeung Cheung and H.-A. Jacobsen. Dynamic load balancing in distributed content-based publish/subscribe. In *Middleware '06: Proceedings of the ACM/IFIP/USENIX 2006 International Conference on Middleware*, pages 141–161, 2006. Springer-Verlag New York, Inc.

[YKL01] Y. R. Yang, N. S. Kim, and S. S. Lam. Transient behaviors of tcp-friendly congestion control protocols. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1716 –1725 vol.3, 2001.

[YMJ11] Y. Yoon, V. Muthusamy, and H. Jacobsen. Foundations for highly available content-based publish/subscribe overlays. In *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*, pages 800 –811, june 2011.

[ZGG05] H. Zhang, A. Goel, and R. Govindan. An empirical evaluation of internet latency expansion. *SIGCOMM Comput. Commun. Rev.*, 35(1):93–97, 2005.

[ZHS+04] B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, and J. Kubiatowicz. Tapestry: a resilient global-scale overlay for service deployment. *Selected Areas in Communications, IEEE Journal on*, 22(1):41 – 53, jan. 2004.

[ZMJ11] K. Zhang, V. Muthusamy, and H.-A. Jacobsen. Total order in content-based publish/subscribe systems. Technical report, Middleware Systems Research Group, University of Toront, 2011.

[ZSB04] Y. Zhao, D. Sturman, and S. Bhola. Subscription propagation in highly-available publish/subscribe middleware. In *Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*, Middleware '04, pages 274–293, 2004. Springer-Verlag New York, Inc.