End-to-End Congestion Control for Content-Based Networks

Amirhossein Malekpour, Antonio Carzaniga, and Fernando Pedone
University of Lugano
Lugano, Switzerland
{*malekpoa,antonio.carzaniga,fernando.pedone*}*@usi.ch*

***Abstract*—Publish/subscribe or "push" communication has been proposed as a new network service. In particular, in a content-based network, messages sent by publishers are delivered to subscribers based on the message content and on subscribers' long-term interests (subscriptions). In most systems that implement this form of communication, messages are treated as datagrams transmitted without end-to-end or in-network acknowledgments or without any form of flow control. In such systems, publishers do not avoid or even detect congestion, and brokers/routers respond to congestion by simply dropping overflowing messages. These systems are therefore unable to provide fair resource allocation and to properly handle traffic anomalies, and therefore are not suitable for large-scale deployments. With this motivation, we propose an end-to-end congestion control for content-based networks. In particular, we propose a practical and effective congestion-control protocol that is also content-aware, meaning that it modulates specific content-based traffic flows along a congested path. 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. We demonstrate experimentally that the protocol improves fairness among concurrent data flows and also reduces message loss significantly.**

***Keywords*-congestion control; content-based networking; publish/subscribe;**

I. Introduction

In content-based publish/subscribe communication, each published message goes to the receivers that subscribed for the content of the message. One common architecture to implement this form of communication uses a network of content-based routers (or "brokers"). Among other things, content-based publish/subscribe communication systems differ in the ordering and reliability guarantees they offer. Some systems offer end-to-end guarantees such as FIFO and reliable delivery above and beyond what is provided by the underlying communication primitives, and they do so typically by operating as store-and-forward networks [3], [14], [8], [12].

Other systems do not offer such additional guarantees and instead try to maximize throughput and minimize end-to-end delay by operating as best-effort networks [11], [6], [17]. Such systems typically process and forward messages as fast as they can, without taking into account the balance between independent flows, and without feedback (acknowledgments) or other flow-control mechanisms.

In short, 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.

Congestion and its adverse effects on traffic and applications have been studied extensively [4], [5], [19]. 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 [4]. 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. For this reason, communication systems that are oblivious to congestion are considered unfit for deployment in open networks such as the Internet and in dedicated networks such as data centers [19].

These considerations motivate the work presented in this paper. Our high-level goal is to develop a congestion control mechanism for best-effort content-based publish/subscribe systems. In essence, we would like 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 a congestion control protocol based on the design of a 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, our solution is an end-to-end protocol in which each receiver (subscriber) measures the end-to-end delay and loss rate within each message flow from a sender (publisher). The receiver then uses these measures to determine a proper maximal rate for that flow using a "TCP-response" function. The receiver then feeds the prescribed maximal rate back to the sender that in turn aggregates rate requests 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 throughput.

However, 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 router software and in particular to its routing and forwarding algorithms.

In Section II 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 III; we discuss its salient features in Section III-C; we present an experimental evaluation of the protocol in Section IV; and we conclude with some final remarks in Section V.

II. Context and High-Level Design

Our problem is end-to-end congestion control in best-effort publish/subscribe networks. For the publish/subscribe network, we assume the following basic content-based publish/subscribe behavior: publishers publish *messages* whose content is characterized by *attributes*; subscribers request messages of interest by specifying *constraints* on the values of their attributes. In particular, the term *constraint* refers to an individual condition on an attribute value, the term *filter* or *subscription* refers to a logical conjunctions of constraints, and the term *predicate* is a logical disjunction of filters [6]. At each given time, the publish/subscribe communication system is configured, and its behavior is fully determined by the association of predicates to receivers, so that each message published at that time should be delivered to all receivers associated with a predicate matching the message.

As in traditional networking, controlling and avoiding congestion in a publish/subscribe network amounts to withholding 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 and possibly interdependent flows.

A. Related Work

Congestion control has not been studied extensively in the context of content-based publish/subscribe systems. Pietzuch and Bhola [15] designed the first and only specific congestion-control mechanism we know of. In essence, this mechanism limits publication rates to the level of the slowest link in the publish/subscribe network. Congestion is detected on the subscriber side by measuring decreasing delivery rates that do not correspond to decreased publishing rates. Congestion then triggers rate-limitation requests that are propagated upstream towards the publishers. This protocol targets *reliable* broker-based publish/subscribe systems and it involves every broker along the path between publisher and subscriber in the congestion-control process, since each broker aggregates the feedback sent from downstream brokers. By contrast, our goal is end-to-end congestion control over a best-effort network. Also, the protocol of Pietzuch and Bhola does not provide fairness among concurrent flows between separate endpoints that share the same path, nor does it distinguish between different content-based portions of the same flows. As we will see later, these are some of the most salient features of the protocol we propose in this paper.

Beyond publish/subscribe systems, 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. So in other words, the goal is to make multicast flows behave like TCP flows.

The protocols that are most relevant to this paper can be characterized as single-rate protocols: a receiver, usually called the *acker,* measures end-to-end delay and message loss, and sends feedback to multicast sources, which adjust their 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 [10], [16]. 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 [2], [19].

An important representative of this latter category, 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 [9]. 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) [19]:

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

The resulting rate T is the maximal rate for that flow compatible with TCP behavior. When the receiver measures a delivery rate exceeding the computed maximal rate, the receiver communicates its desired rate to the sender. Each sender then adjusts its transmission rate to the lowest requested rate 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 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.

B. 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 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 $A \to B$ flow with sending rate λ_A is $\lambda_{A \to B} = \lambda_A \cdot \rho_{A,B}$.

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 $\lambda_{A \to B}$ and $\lambda_{A \to C}$ 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 $A \to B$ and $A \to C$, A could reduce $\lambda_{A \to B}$ and thereby avoid congestion while still maintaining a high rate $\lambda_{A \to C}$. 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. This situation is illustrated in Figure 1.

In practice, content-based communication introduces two requirements for an efficient and fair end-to-end congestion control. First, rate-limitation 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.

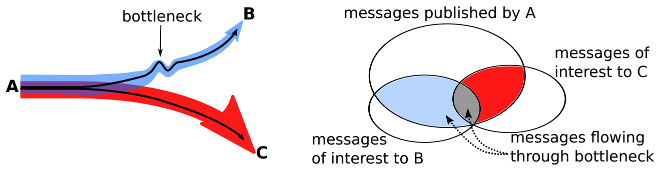

bottleneck
B
A
C
messages published by A
messages of interest to C
messages of interest to B
messages flowing through bottleneck

Figure 1. Content-aware rate control

In order to meet these 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, 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.*

C. 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 (we discuss this problem in detail later) 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 [1] 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) [16]. 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.

III. Content-Aware Congestion Control

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 congestion control headers and the operations of subscribers and publishers.

A. *Content-Based Flows*

As explained in Section II-A, 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 already 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 also determines the rate limitation for that flow.

B. *Congestion Control Protocol*

1) 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 congestion control header to each publication that belongs to a controlled flow (see Figure 2).

2) 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. To reduce the overhead of this evaluation and also to reduce the overhead of transmitting filters in feedback messages, we take advantage of an encoding scheme 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 [7], [13]. This encoding also allows for an equally fast evaluation of the covering relation between filters. This efficiency in matching comes at the cost of false positives. Yet our empirical evaluation shows that the performance gains justify the lower precision. In the following sections we explain how our protocol takes advantage of this scheme. Notice however that the encoding is an optimization and a modular part of the protocol, and can be replaced or even removed altogether.

3) Loss Detection: Subscribers must measure the loss event rate in order to compute the appropriate rate for each flow. However, in content-based networking there is no simple and effective way of detecting losses. Sequence numbers, which are typically used in unicast and multicast protocols to detect packet losses, are not applicable in the case of content-based communication. This is because a gap in the sequence does not necessarily evidence a message loss, as it might well indicate that the missing messages did not match the receiver's interests. In our protocol, we use a probabilistic loss detection method that we developed in prior work [13]. Essentially, this method augments each message (publication) with a *publication record* consisting of an encoded summary of the latest k messages published by the publisher. This summary is in fact produced using the same message-encoding technique that transforms a message into a Bloom filter. Figure 2 shows a message carrying a publication record of size $k = 4$.

The publication record allows a receiver to determine if any of the k previously published messages was of interest for the receiver and therefore which of them was lost. Unfortunately, this loss detection method is imprecise, particularly when the matching probability is small (the probability that a message matches a subscriber's predicate), and thus there are large gaps in the sequence of messages delivered to the subscriber. In Section III-C we discuss the effects of this approximate loss detection on the protocol's performance and we show how potential problems can be mitigated.

4) 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

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

Figure 2. Congestion control header in a publication message.

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 3. Per-publisher state maintained by a subscriber.

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 always goes through the publish/subscribe network, since its purpose is to measure the latency at that level.

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 2); at the same time, the subscriber measures the travel time of the same publications using the publisher's time stamp ($timestamp$ in Figure 2); 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.

5) Subscriber's State and Operations: Subscribers maintain a session with each publisher from which they receive publications. Figure 3 shows the information associated with sessions. 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 1) for each session (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 its priorities. In our implementation we use a max-min

```
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 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 $R$}

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:     $e.quota \leftarrow e.reception\_rate$
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 unused quota by n. 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($\alpha \cdot e.reception\_rate, q$)
                 {increase quotas by at most $\alpha\times$ 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 1. Congestion monitoring and control run by a subscriber for each publisher from which there is an incoming message flow.

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 and therefore to reduce instability. Our experiments demonstrate that a value of α between 0.5 and 1 is appropriate. Finally,

filter	Bloom filter	CLR	quota	tokens	feedback time	queue
F_1	$0101\cdots00$	S_1	Q_1	T_1	t_1	$\cdots$
F_2	$0010\cdots10$	S_2	Q_2	T_2	t_2	$\cdots$
F_3	$1001\cdots01$	S_3	Q_3	T_3	t_3	$\cdots$
F_4	$1011\cdots00$	S_4	Q_4	T_4	t_4	$\cdots$

Figure 4. A publisher's congestion control state

```
1: upon receiving feedback message $M$ from subscriber $s$ do
2:   for each flow $f \in M$ 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:       $g.quota \leftarrow f.quota$ {accept quota change request from CLR}
8:       $g.feedback\_time \leftarrow$ 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:  $F \leftarrow F \cup \{(f.filter, s, f.quota, \text{TIMESTAMP}())\}$ {add $f$ to the flow table}
```

Algorithm 2. Processing feedback message M received from subscriber s

when the necessary changes are made in the local state, the subscriber proceeds to send a corresponding feedback message to the publisher.

6) 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 4). For each filter, again stored as Bloom filters for efficiency, the publisher 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 time stamp 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 2) that is then used by subscribers to estimate the RTT; and second, it allows the subscriber to discard stale entries after a set timeout.

7) 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 2, 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 f is ignored; 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.

8) 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. 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 congestion control headers (shown in Figure 2) before transmission into the network.

C. 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 (1)) and how those effects can be mitigated. Recall from Section II-A 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 $\ell \leftarrow 1$.

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 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 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 2).

Qualitatively, the optimistic assumption is generally valid in non congested and even in slightly congested network conditions, but it does not necessarily 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

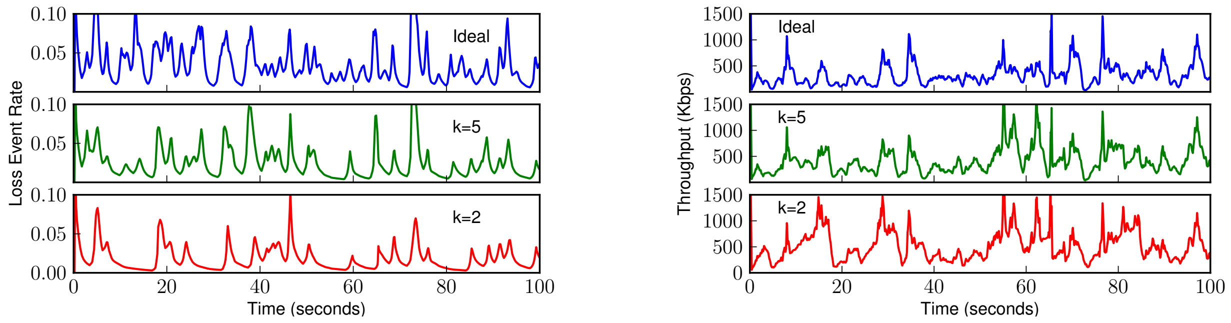


Figure 5. Loss event rate (left) and TCP response function (right) computed for the ideal receiver (top), publication record of size 5 (middle) and publication record of size 2 (bottom).

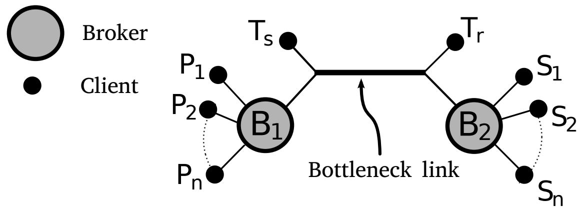


Figure 6. Experiment topology

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 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., 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.

IV. 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.

A. Experimental Setup

We have fully implemented our congestion control protocol as a Java module that integrates into the client middleware and could work with virtually any best-effort publish/subscribe system. Most publish/subscribe systems are capable of carrying a user payload which we use to add congestion control headers to each message. For the experiments presented in this section we have used a recent version of the Siena system, which implements a best-effort content-based publish/subscribe system [7].

Our testbed is a cluster of 46 physical machines, each one having 4 cores and 4 Gigabytes of memory and running the Linux 2.6.32 kernel. Connectivity is provided through an isolated high-throughput Gigabit Ethernet switch. Broker software and client (including congestion control protocol) are implemented in Java and run on the 64-bit open-JDK VM. We used the Linux traffic control tools to emulate bottleneck links. Figure 6 shows the topology we setup for all except the last (large scale) experiment. 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 compare the results of experiments with and without congestion control, 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 congestion control 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

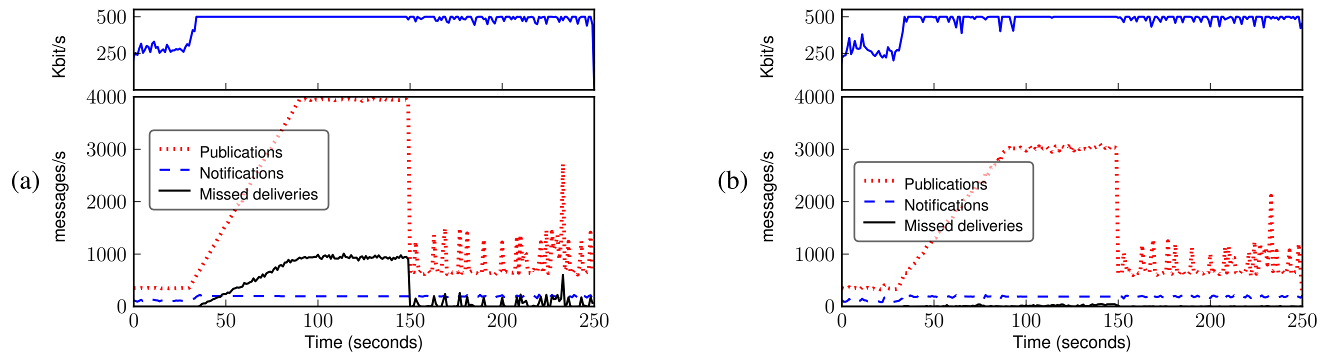


Figure 7. Effect of variable input load (a) without and (b) with congestion control. Top charts: traffic rate (Kbps) on the bottleneck link. Bottom charts: aggregate publication, reception, and missed-delivery rates (messages per second).

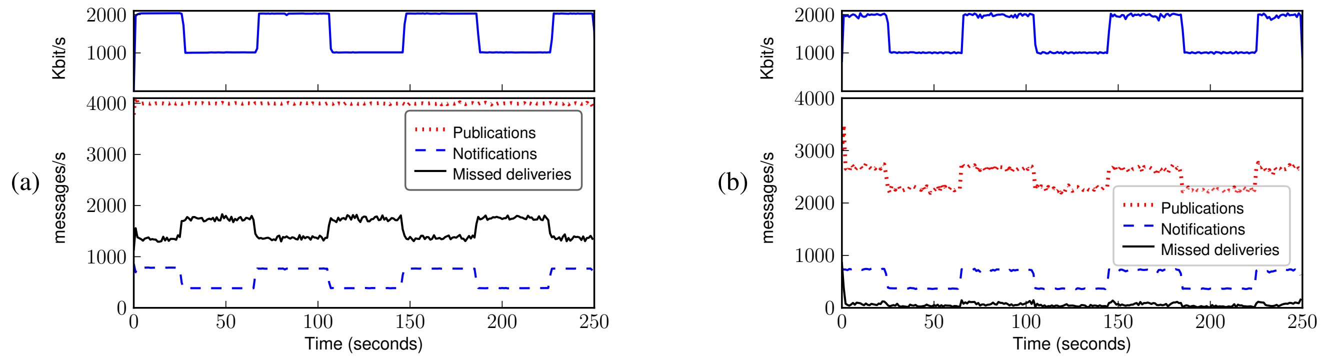


Figure 8. Effects of variable bottleneck link capacity (a) without and (b) with congestion control. Top: traffic rate (Kbps) on the bottleneck link. Bottom: aggregate publication, reception and missed-delivery rate (messages per second) during the experiment.

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.

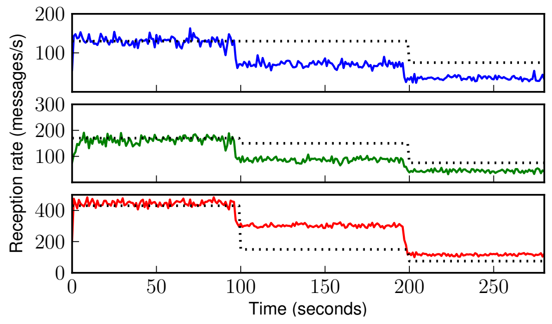
B. Effectiveness, Stability, Responsiveness

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 6). Figure 7 (bottom chart) shows the publication rate, reception rate, and missed-delivery rate in terms of messages per second (mps) during 250 seconds of an experiment without and with congestion control (figures 7a and 7b, respectively). To demonstrate the optimality of resource utilization, the top graph shows traffic (Kbps) on the bottleneck link whose bandwidth is 500Kbps. To evaluate 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, then stable, and then bursty rate. The publication rate starts at 400mps, then ramps up to 4000mps where it stabilizes for 60 seconds, and then descends to 600mps continuing with short bursts of up to 2500mps. With publication rates above 1000mps the network saturates its link capacities and starts to experience message losses (resulting in missed deliveries). In all three phases, congestion control is able to mitigate persistent and transient congestion, reducing the number of lost messages by a factor of $20\times$, particularly during the period where congestion is persistent.

Figure 7 also shows that the congestion control protocol uses the network almost to its maximum capacity. More specifically, without congestion control in place the subscriber receives nearly 46000 messages, a result that is reduced only minimally when using congestion control, with more than 45000 messages delivered correctly. Notice in Figure 7b 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-aware congestion control, which limits only those outgoing message flows that go through the bottleneck link.

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 8 shows the results of an experiment in which we control the bandwidth of the bottleneck link by alternating between 1Mbps and 2Mbps in periods of 40 seconds. As Figure 8b suggests, reaction to both decrease and increase in bottleneck

(a)



(b)

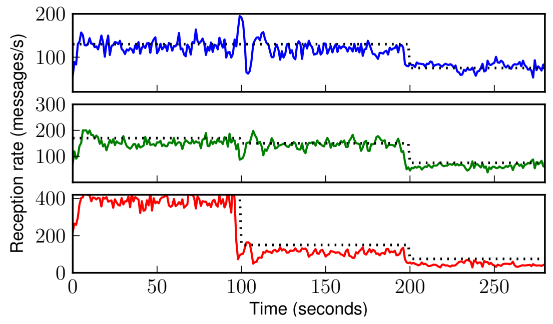
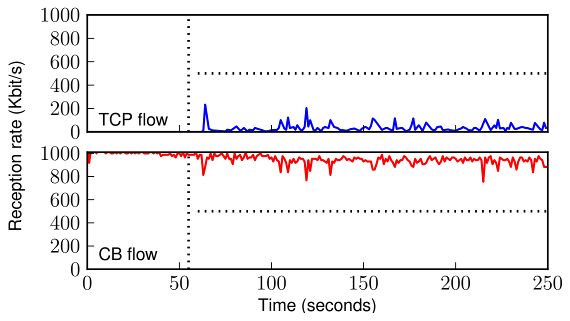


Figure 9. The solid lines show reception rates (mps) for 3 pairs of publishers and subscribers sharing the bottleneck link (a) without and (b) with congestion control in effect. The dotted lines show the fair share of each flow.

(a)



(b)

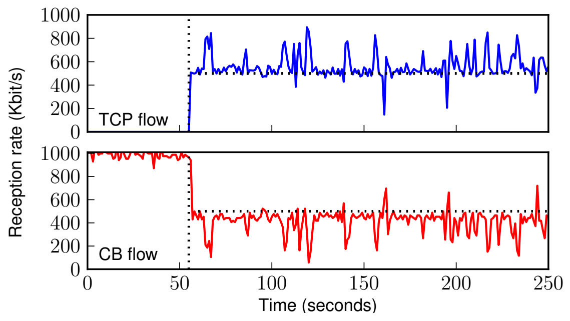


Figure 10. TCP and content-based reception rate (Kbps) for a TCP flow and a publish/subscribe flow sharing a bottleneck link (a) without and (b) with congestion control. The horizontal dotted lines show the ideal fair share.

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.

C. *Fairness Among Content-Based Flows*

We now proceed to examine the fairness properties of our congestion control protocol in the presence of concurrent content-based flows. In such settings we expect the receivers with higher reception rates to start controlling congestion before low rate receivers. In this experiment we use 3 publishers and 3 subscribers (i.e., S_1 to S_3 and P_1 to P_3 in Figure 6) with each publisher sending messages at a constant rate of 1000mps. Each subscriber receives messages from only one publisher but with different matching rates, 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, which causes no contention among the flows. Then at time $t = 100$ seconds we cut the link bandwidth in half (1Mbps) and again we do the same at time $t = 200$ seconds (to 500Kbps).

Figure 9 shows the reception rates (messages 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 (dotted lines). Without congestion control (Figure 9a) 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 9b), 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 by asking its corresponding sender to reduce its send rate. As a result, the average reception rate at this subscriber (Figure 9b, bottom) is reduced from 400mps to 140mps, which is 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 experience reduce their reception rate evenly.

D. *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 the protocol does not exceed that of TCP when flows share a bottleneck link. In Figure 10 we present the results of a simple setup involving a TCP flow and a publish/subscribe flow (i.e., T_s, T_r, S_1 and P_1 in Figure 6). The bottleneck link has a bandwidth of 1Mbps. The TCP flow starts at time $60s$ (vertical dotted lines in Figure 10).

In the absence of congestion control (Figure 10a) the

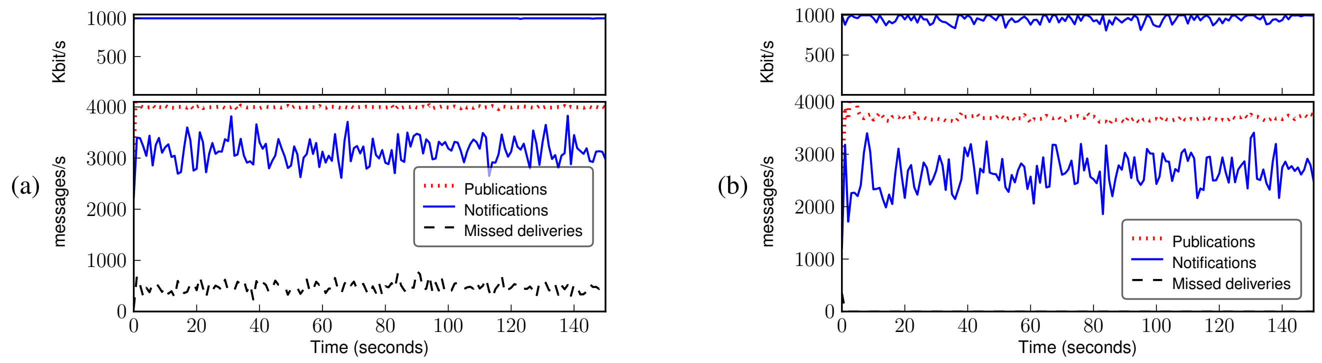


Figure 11. Publication, reception and missed-delivery rate (mps) in a large scale network (a) without and (b) with congestion control in place.

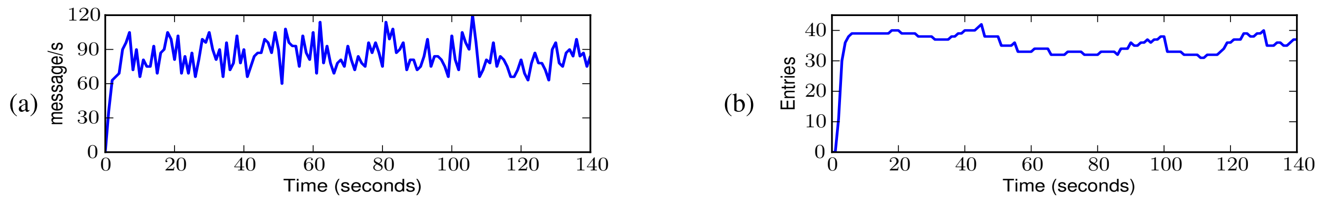


Figure 12. (a) Feedback messages (mps) received by one of the publishers; (b) entries in the publisher's state table.

content-based flow dominates the TCP flow, pushing more than 900Kbps and almost saturating the link capacity. (Horizontal dotted lines indicate an ideal fair share.) This result shows the importance of a rate control scheme in deployments of best-effort content-based systems coupled with TCP flows. With congestion control in place (Figure 10b) the content-based flow adapts its rate within less than two seconds from the start of the TCP flow. The average link share is slightly higher for TCP, with a difference of about 5% of the total bandwidth. This is because equation-based congestion control protocols tend to be more conservative than TCP [18].

E. Large Scale Deployment

We conclude this evaluation by studying the performance of the protocol in a large network with hundreds of clients. Due to space limitations here we only focus on two important aspects of the protocol, namely its 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 3 and brokers have 1 to 3 neighbors. 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 11a and 11b juxtapose the throughput and 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 missed deliveries drops to nearly zero within three seconds and remains unchanged during the entire experiment. The average throughput (message delivery) is only about 12% lower with congestion control, with more than 90% utilization of the link capacity 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 network.

Now turning our attention to the publisher's overhead, figures 12a and 12b show the number of feedback messages received (mps) 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 or CPU load on any of the hosts running the publishers. 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.

Last, we consider the state maintained by publishers. It might seem that publisher state would be the primary threat to scalability. Indeed, since the idea is for publishers to modulate potentially many flows induced by the specific interests of each subscriber, it might seem that a publisher would have to keep track of all subscriber interests. However, note that the publisher's state (i.e., the table of Figure 4) grows sub-linearly with the number of subscriptions that generate message flows, and in practice we observed an overall low overhead in terms of state. This is because, first, only high-rate flows require rate control, and second, a high rate flow that needs modulation is represented by a single filter (of the CLR) irrespective of the number of filters that match messages in that flow. Moreover, each entry in the state table requires only a few bytes of memory. For instance, with Bloom filters of 256 bits, 2000 entries in the state table (an unrealistically large figure) consume less than 1 MB of the publisher's main memory.

V. Concluding Remarks

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 as well as a specific loss detection technique. We also presented the results of the experimental evaluation of a full implementation of our protocol. The results we obtained are very good, and show that the protocol achieves its primary objectives, namely effective resource usage, adaptability, and fairness. As a natural way to complement these empirical results, we would like to study the protocol and the problem analytically, perhaps along the lines of Vojnovic and Le Boudec [18]. We would also like to research the applicability and effectiveness of window-based congestion control for content-based networks.

More generally, we see best-effort content-based networks as a scalable base for a diverse set of content-driven reactive applications. This work is part of an effort to design an end-to-end *transport layer* for best-effort content-based networks to improve their quality of service and foster their deployment with minimal sacrifices in simplicity, performance, and scalability.

References

[1] D. Bansal, H. Balakrishnan, S. Floyd, and S. Shenker. Dynamic behavior of slowly-responsive congestion control algorithms. In *proc. of SIGCOMM '01*, pages 263–274, New York, NY, USA, 2001. ACM.

[2] S. Bhattacharyya, D. Towsley, and J. Kurose. The loss path multiplicity problem in multicast congestion control. In *Proc. of INFOCOM 99*, pages 856–863, 1999.

[3] S. Bhola, R. E. Strom, S. Bagchi, Y. Zhao, and J. S. Auerbach. Exactly-once delivery in a content-based publish-subscribe system. In *Proc. of DSN '02*, pages 7–16, Washington, DC, USA, 2002. IEEE Computer Society.

[4] 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.* The Internet Society, 1998. RFC 2309.

[5] J. W. Byers, G. Horn, M. Luby, M. Mitzenmacher, and W. Shaver. FLID-DL: congestion control for layered multicast. In *Proc. of NGC 2000*, pages 71–81, 2000.

[6] 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.

[7] 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, Aug. 2009.

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

[9] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-based congestion control for unicast applications. *SIGCOMM Comp. Comm. Rev.*, 30:43–56, Aug. 2000.

[10] S. Golestani and K. Sabnani. Fundamental observations on multicast congestion control in the internet. In *Proc. of INFOCOM '99*, pages 990–1000, Mar. 1999.

[11] P. Jokela, A. Zahemszky, C. E. Rothenberg, S. Arianfar, and P. Nikander. LIPSIN: Line Speed Publish/Subscribe Inter-networking. In *Proc. of SIGCOMM '09*, pages 195–206, New York, NY, USA, 2009. ACM.

[12] R. S. Kazemzadeh and H.-A. Jacobsen. Reliable and highly available distributed publish/subscribe service. In *Proc. of SRDS '09*, pages 41–50, Washington, DC, USA, 2009. IEEE Computer Society.

[13] A. Malekpour, A. Carzaniga, F. Pedone, and G. T. Carughi. End-to-end reliability for best-effort content-based publish/subscribe networks. In *Proc. of DEBS '11*, pages 1–12, New York, NY, USA, 2011. ACM.

[14] P. R. Pietzuch and J. Bacon. Hermes: A distributed event-based middleware architecture. In *Proc. of ICDCSW '02*, pages 611–618, Washington, DC, USA, 2002. IEEE Computer Society.

[15] P. R. Pietzuch and S. Bhola. Congestion control in a reliable scalable message-oriented middleware. In *Proc. of Middleware '03*, pages 202–221, New York, NY, USA, 2003. Springer-Verlag New York, Inc.

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

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

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

[19] J. Widmer and M. Handley. Extending equation-based congestion control to multicast applications. In *Proc. of SIGCOMM '01*, pages 275–285, New York, NY, USA, 2001. ACM.