Content-Based Routing in Mobile Ad Hoc Networks

Milenko Petrovic[†], Vinod Muthusamy[†], Hans-Arno Jacobsen[†‡]
[†]Department of Electrical and Computer Engineering
[‡]Department of Computer Science
University of Toronto
{petrovi,vinod,jacobsen}@eecg.toronto.edu

Abstract

The publish/subscribe model of communication provides sender/receiver decoupling and selective information dissemination that is appropriate for mobile environments characterized by scarce resources and a lack of fixed infrastructure. We propose and evaluate three content-based routing protocols: CBR *is an adaptation of existing distributed publish/subscribe protocols for wired networks,* FT-CBR *extends* CBR *to provide fault-tolerance, and* RAFT-CBR *provides both fault-tolerance and reliability. Using network simulations we analyze the applicability and test the tradeoffs of these algorithms. We show that* RAFT-CBR *can guarantee 100% delivery to small groups, at the expense of transmission delay.* CBR*, with a low message overhead and low delay, is more suitable for larger groups at the expense of reliability.* FT-CBR *provides comparable delivery rates to* RAFT-CBR*, as well as low delay, at the expense of increased message cost.*

1. Introduction

Mobile ad hoc networks (MANET) have attracted a lot of attention in the past few years. The main advantage of ad hoc networks is that they do not depend on any infrastructure for operation. Such peer-to-peer networks are created spontaneously as soon as two participants are within the transmission distance of each other. As more participants join, leave or just change geographic position, the network seamlessly reconfigures allowing all participants to communicate. The network uses peer routing to allow two nodes to communicate even if they are not within transmission distance of each other.

The flexibility of ad hoc networks comes at a price. Ad hoc networks are highly dynamic; network topology changes can be very frequent because of user mobility, network links often go down intermittently and their bandwidth is limited because of wireless channel errors (usually an order of magnitude lower than wired links; for long multi-hop connections bandwidth reduction is even more dramatic). In addition, nodes in a mobile ad hoc network are powered by batteries, thus placing further restrictions on communication and processing capabilities.

Such a resource constrained and highly variable environment is challenging for tightly coupled distributed applications. Communication failures are the norm rather than the exception in such an environment, and hence the need arises for new communication protocols that take this into consideration. Data-centric networking protocols use content addressing instead of host (e.g., IP) addressing for participating nodes, thus decoupling application communication from the underlying network [1, 2]. Equally important for mobile ad hoc networks is the possibility of routing protocols that can provide highly selective information dissemination between peers, thus making group communication more efficient: the routing protocols perform timely filtering of data, thus minimizing the amount of data that needs to be communicated between peers.

Many applications can benefit from selective and decoupled dissemination including stores sending coupons to shoppers at a mall, reporters at a news conference sending real-time photos and news to interested readers, teammates communicating with each other in a strategic game, and commuters propagating traffic jam information to drivers behind them. Other more critical applications include sending information about arriving patients in a hospital or disaster scenario to those doctors and staff who are able to treat the patients' symptoms, and disseminating traffic accident data to nearby police and ambulance.

It has been shown that the publish/subscribe (P/S) communication model is well suited as a model for a wired, data-centric content-based routing network [1, 2]. However, the performance of content-based routing (CBR) in ad hoc, highly dynamic networking environments offering a P/S style of interaction is not known. It is the objective of this paper to establish this through the following contributions:

- We design two new CBR algorithms specifically targeted for ad hoc networks: CBR and the fault-tolerant

Paper appears in 2nd International Conference on Mobile and Ubiquitous Systems *(MobiQuitous 2005)*, San Diego, CA, USA, July, 2005.

version, FT-CBR. The two algorithms are the first general-purpose CBR solutions in ad hoc networks.

- We describe a new *reliable*, *content-based* explicit multicast (xcast) protocol: RAFT-CBR. To the best of our knowledge, this is the first reliable and fault-tolerant content-based xcast for ad hoc networks.
- To cope with the inherent unreliability of MANET systems, the algorithms use soft state. Leases are introduced as a means to keep routing information fresh.
- We examine the effectiveness of the algorithms by thoroughly evaluating their performance using network simulation.

The three algorithms, CBR, FT-CBR and RAFT-CBR, all provide P/S CBR semantics to the application, although they differ in the implementation. Some applications, such as those involving financial transactions or the transmission of safety critical notifications in a hospital, functionally require strong reliability and fault-tolerance guarantees, such as those provided by RAFT-CBR, while others, can enjoy improved performance with effective fault-tolerance techniques, such as FT-CBR, without incurring the costs of full reliability. Despite the lack of reliability guarantees in FT-CBR, its performance is fairly close to that of RAFT-CBR but without the extra overhead. The main difficulty in creating a reliable CBR protocol stems from the decoupling of publishers and subscribers.

In the next section we briefly introduce P/S and multicast, followed by related work review in Section 3. In Section 4 we describe the three protocols in detail and then evaluate them using simulation in Section 5. Section 6 summarizes the main contributions and identifies directions for further research.

2. Background

The P/S communication model includes three entities: publishers, subscribers and brokers. Publishers are data producers, subscribers are data consumers and brokers forward data from publishers to subscribers.

To inform the broker about its interests, a subscriber sends a subscription message: a query on the content of the data. The broker uses the subscription to filter all incoming data and only forwards the data that is of interest to the subscriber. The basic unit of data is an event, and the subscriptions are filter expressions on the event content. This model provides uniform access to data, in the form of subscriptions, regardless of where the data is located in the network.

The data (event) flow creates loose coupling between publishers and subscribers. In other words, neither the subscribers nor the publishers are aware of where that data originated or who is receiving it, respectively. This makes the P/S model highly attractive for information dissemination applications characterized by timely filtering of data for multiple subscribers and support for intermittent connections.

An application of this model to wide area information dissemination in wired networks has been studied in the context of content-based routing [1, 2]. In this context, the functionality of the broker is implemented as a network of cooperating brokers rather that as a single centralized broker. The primary task of the broker network is the same as with the centralized broker approach, namely filtering of events based on subscriptions. However, due to the decentralized nature of the system, efficient delivery of events from publishers to subscribers now becomes an important task for the broker network as well. The broker network is a virtual, application level network, which is usually made to coincide with the actual (physical) network for efficiency purposes, but could equally be created as an overlay network.

Dissemination of events in a distributed P/S system is similar to a network level multicast [3]. From the system's point of view, the publisher is seen as the producer (multicast source) of events which need to be delivered to multiple subscribers (multicast receivers). However, the main difference is that the multicast groups are created dynamically based on the *content* of the event instead of an IP address. As a result, the groups tend to be *smaller* and more *numerous*, and frequently *short-lived*. This is significantly different from group membership of the traditional multicast, including reliable multicast, protocols where the groups are defined a priori and only the membership is dynamic. The ability to create and destroy multicast groups based on the content of the data to be disseminated allows for fine-grain filtering which is particularly important for resource constrained ad hoc networks.

Explicit multicast (xcast) [4, 5] is a multicasting technique which builds on an underlying unicast routing. The two main advantages of xcast over traditional multicast is the use of soft state and the ability of the sender to specify the membership of a multicast group. This makes xcast much more suited for content-based routing than traditional multicast approaches.

3. Related Work

The P/S communication paradigm has been studied extensively [1, 6]. While CBR and FT-CBR are fully content-based routing algorithms, for RAFT-CBR we make the assumption that a publisher knows about all the subscribers interested in that publisher's publications.

Distributed P/S architectures have been primarily studied in the context of wired networks [2, 7]. Our work looks at small scale wireless ad hoc networks with resource constrained publishers and subscribers and without a fixed in-

frastructure. We develop new CBR protocols that use subscription leases, a routing metric favoring subscription covering[1], and fault-tolerance mechanisms.

Multicast for ad hoc networks has been studied extensively [8, 9, 10]. In traditional multicast, group members agree on the content disseminated in a particular group a priori. Our protocols differ in two respects: during dissemination, they do not explicitly create any routing structures in the network and they allow multicast groups to be created/joined/detached/destroyed *dynamically* based on the event content. In addition, they provide fault-tolerant and reliable multicast routing.

Reliable multicast for mobile ad hoc networks has been studied previously [8]. Although CBR is similar to multicast, *decoupling* of publishers and subscribers is a distinguishing feature of CBR which precludes using existing approaches to provide reliability. Consequently, we develop FT-CBR and RAFT-CBR for this purpose. We are not aware of any existing reliable content-based xcast protocols.

Similar work in sensor networks [11], relies on subscription flooding while the protocols in this paper use a more resource efficient subscription multicast that is more appropriate for ad hoc networks.

RAFT-CBR extends the xcast idea [4, 5] to provide *fault-tolerant* and *reliable* CBR semantics.

This work—referred to as AdHoc-ToPSS—is part of the ToPSS (Toronto Publish/Subscribe System) research projects [12, 13, 14, 15, 16].

4. Algorithms

This section describes three algorithms that perform content-based P/S multicast in a MANET. The CBR algorithm is unreliable and does not tolerate any failures; it is an algorithm that serves as a foundation for other algorithms. FT-CBR is an extension of CBR that adds fault-tolerance; while this algorithm tries to recover from faults there is no way of knowing if events were actually delivered. To provide reliability, we develop RAFT-CBR, a reliable and fault-tolerant algorithm.

4.1. Content-based routing

No prior work on deploying CBR for ad hoc networks exists. The CBR algorithm is based on existing distributed P/S algorithms in wired networks [1, 2], but is extended to use soft state and use a new routing metric appropriate for ad hoc networks.

Each node maintains an `AdsTable` and a `SubsTable`, to store advertisements and subscriptions from neighbors, respectively. The fields in these and other data structures are outlined in Table 1. The purpose of these tables is described below.

1 Covering is an optimization that quenches subscriptions by forwarding a few "general" subscriptions instead of many "specific" ones.

AdsTable	Store ads from neighbors
t.pbr *	Publisher
t.pat *	Advertisement pattern
t.neigh *	Neighbor that send the ad
t.hops	Sum of hops from t.neigh to publisher(s) for t.pat
t.expire	Time to expire this entry
SubsTable	Store subs from neighbors
t.child *	Node that sent subscription
t.pat *	Subscription pattern
t.expire	Time to expire this entry
MCTable	Store MC info from neighbors
t.pbr *	Publisher
t.neigh *	Neighbor that sent the MC info
t.pat *	Subscription pattern
t.count	Number of subs at t.neigh covered by t.pat
t.hops	Sum of hops from t.neigh to publisher(s) for t.pat
t.expire	Time to expire this entry
UnackdTable	Store events forwarded to neighbors
t.evt *	Copy of event message
t.expire	Time to expire this entry
FloodTable	Store events flooded to neighbors
...	Same as UnackdTable

Table 1. FT-CBR data structures. In each table, the fields that constitute the primary key are marked.

First, every publisher must send an *advertisement* message indicating the types of events it will publish. This advertisement is flooded and stored in the `AdsTable`. The advertisement message includes the number of hops it has propagated, so nodes know which of their neighbors is on the shortest path to each publisher. Next, every subscriber must send a *subscription* message expressing its interest. The `AdsTable` is used to forward the subscriptions toward those publishers who have indicated (through advertisement messages) that they will publish events that could match the subscription. Nodes on the path from subscriber to publisher store the subscriptions in their `SubsTable`. Finally, *events* (also known as publications) published by publishers are sent along the reverse path of subscriptions to interested subscribers. The detailed contents of the various messages in the system are outlined in Table 2.

To summarize, the advertisements set up a routing layer for the forwarding of subscriptions, which in turn create a routing layer for the forwarding of events.

Notice that subscribers can be anonymous in this system. A node n receiving subscription s from neighbor c does not know if c is interested in s or is simply forwarding s from some other subscriber. Likewise, when n sends an event e to c, n cannot know if c is interested in e or is only acting as a forwarding agent.

MANET environments experience high transmission failures, and frequent topology changes. Therefore routing paths can become stale and must be refreshed over time. In CBR leases are used as a simple and inexpensive way to keep the routing tables up to date. Every

MSG	Any message below
m.sender	Node that sent the message
ADVERT	Advertisement message
a.pbr	Publisher
a.pat	Advertisement pattern
a.hops	Sum of hops from sender to publisher(s) for a.pat
SUB	Subscription message
s.sbr	Subscriber
s.pat	Subscription pattern
MC	Most-covering message
m.pbr	Publisher
m.pat	Subscription pattern
m.count	Number of subs at sender covered by m.pat
m.hops	Sum of hops from sender to publisher(s) for m.pat
ACK	Message to acknowledge events
a.pbr	Publisher
a.seq	Sequence number
EVENT	Event message
e.sender	Node that sent the message
e.pbr	Publisher
e.seq	Sequence number
e.body	Message body
e.cids	Ids of children to receive this event
e.fhops	Number of hops to flood

Table 2. FT-CBR Messages

node periodically broadcasts the contents of these tables to its neighbors, and each entry in these tables is expired if it has not been refreshed for a specified time. In this way, new routes are automatically learned, and old ones forgotten without having to be explicitly deleted.

4.1.1. Covering optimization Covering is a technique to reduce subscription messages. A subscription s covers subscription s' if the set of events that match s is a superset of those that match s'. Intuitively, more "general" subscriptions are likely to cover more specific ones. A node n that has already seen and forwarded subscription s, will not forward subscription s' that is covered by s. It will simply store s' in its `SubsTable`. This reduces the number of subscriptions sent, and also provides anonymity to subscribers. With the covering optimization a node only forwards the minimal set of subscriptions S in its `SubsTable` such that the subscriptions in S cover all subscriptions in the `SubsTable`. The pseudo-code for this is shown in Figure 1.

4.1.2. Most-covering optimization We introduce a new routing metric, *most-covering metric*, that tries to build better multicast trees in an ad hoc network. Consider the network in Figure 2. There is an existing multicast tree (only a path in this case) from subscriber c to publisher p. Another subscriber c' would normally send subscriptions through the shortest path to the publisher (a). However, this ends up expanding the multicast tree. In this case, c' should send its subscription through c (b). Even though this is not the shortest path to the publisher, it builds a multicast tree that has fewer total hops and hence fewer events need to be transmitted.

To achieve this, all subscribers periodically beacon information about their subscriptions to their neighbors using an MC (most-covering) message. (When the covering optimization is used, nodes only beacon the minimal covering set of subscriptions.) Each node stores this information about its neighbors' subscriptions in an `MCTable`.

During subscription forwarding, the `MCTable` is used to forward subscriptions to the neighbor with the most covering subscriptions under the assumption that this neighbor is likely to have a covering subscription for a long time. It is dangerous, however, to only rely on the `MCTable` as subscription cycles can develop; a momentary cycle in subscription forwarding could lead a set of nodes to receive more subscriptions than expected, thereby increasing their MC count, which in turn makes them more favorable to receive even more subscriptions. Instead, the algorithm falls back on the `AdsTable` to determine where to forward subscriptions if using the `MCTable` results in a path that is h hops longer to the publisher than the path using the `AdsTable`.

Notice that the MC optimization tries to direct subscription quickly towards an existing multicast tree. When used with the covering optimization, more subscriptions can be quenched than with the covering optimization alone.

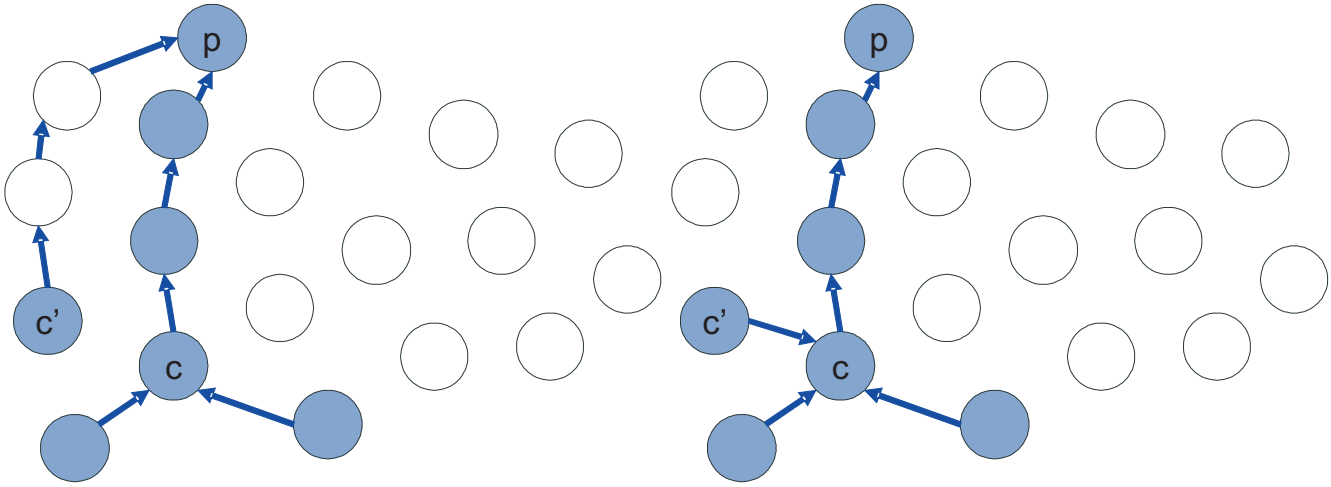
Algorithm *receive*(SUB s)
(∗ Store sub in SubsTable and forward. ∗)
1. (∗ No need to forward this sub if we've already sent a covering sub. ∗)
2. **if** !SubsTable.hasCovering(s)
3. **then** $neighs \leftarrow$ findBestNeighbors(s)
4. send($s, neighs$)
5.
6. updateSubsTable(s)

Algorithm *findBestNeighbors*(SUB s)
(∗ Return the best neighbors to forward this sub. ∗)
1. $M \leftarrow \{\ m | m \in$ MCTable $\wedge\ m.neigh \neq s.sender \wedge m.pat \supseteq s.pat \wedge (\ \nexists\ m' \in$ MCTable $|\ m'.pbr = m.pbr \wedge m'.neigh \neq s.sender \wedge m'.pat \supseteq s.pat \wedge m'.count > m.count\ )\ \}$
2. $A \leftarrow \{\ a | a \in$ AdsTable $\wedge\ a.neigh \neq s.sender \wedge a.pat \supseteq s.pat \wedge (\ \nexists\ a' \in$ AdsTable $|\ a'.pbr = a.pbr \wedge a'.neigh \neq s.sender \wedge a'.pat \supseteq s.pat \wedge a'.hops > a.hops\ )\ \}$
3.
4. (∗ For every publisher, pick a neighbor from MC or Ads table. ∗)
5. **for** $p \in \{M.pbr \cup A.pbr\}$
6. **do** $m \leftarrow m \in M | p = m.pbr$
7. $a \leftarrow a \in A | p = a.pbr$
8. $n \leftarrow$ **nil**
9. **if** $m.pbr \neq$ **nil**
10. **then** $n \leftarrow m.neigh$
11. **if** $a.pbr \neq$ **nil** $\wedge\ m.hops > a.hops + h$
12. **then** $n \leftarrow a.neigh$
13. **if** $n \neq$ **nil**
14. **then** $neighs \leftarrow neighs \cup n$
15. **return** $neighs$

Figure 1. FT-CBR subscription handler

4.2. Fault-tolerant content-based routing

Even when beacons are used to keep routes updated as nodes move, it is still possible that a node's neighbor set changes during a beaconing interval. Consider a multicast



(a) Shortest path subscription (b) Most covering subscription

Figure 2. Most covering optimization

tree with a node n that sends event e to a neighbouring subscriber c. If c has moved out of the transmission range of n, then n has no way of finding c; recall that there is no underlying routing protocol so n cannot simply route e to c.

The FT-CBR algorithm assumes that c has likely not moved far from n during the beaconing interval, so n floods e over a pre-determined number of hops to try to find c. In our implementation, events are flooded for two hops, so e is sent to the neighbors of n and all their neighbors. A `FloodTable` is used to remember recently sent events, so a node does not flood an event it has already flooded. In order for n to know whether c properly received e, FT-CBR requires all events to be acknowledged hop-by-hop. So flooding is initiated by n only after it does not receive an acknowledgment for some specified time. Nodes store in the `UnackdTable` recently forwarded events for which they are waiting for acknowledgments. The complete pseudocode for event handling in FT-CBR is shown in Figure 3. In the code, $e.cids$ refers to the children in event e's multicast tree, and $e.fhops$ is the number of hops to flood e.

4.3. Reliable and fault-tolerant CBR

It is very difficult to guarantee that an event is delivered to all interested subscribers in an ad hoc network because information about the set of interested subscribers is distributed over several nodes. The failure of any one of these nodes loses information that cannot be reliably recovered. Replicating this information is not sufficient; it is possible that all replicas fail simultaneously (perhaps due to a network partition).

To guarantee reliable delivery of events, we now require that the publisher knows all subscribers interested in events it publishes. This is an acceptable concession since MANETs typically only have dozens of subscribers, unlike potentially millions in wired networks. We also require that the publisher not fail; reliable delivery is not guaranteed if the publisher fails. In addition, it is assumed that there are no prolonged partitions between a publisher and its interested subscribers; that is, eventually, there will be a routing path from publisher to subscribers.

Broker	
UE	Information about unacknowledged events.
u.evt	The event itself (where u ∈ UE).
u.dests	Set of destinations that the event was forwarded to (by this node).
u.ackd	Set of destinations that have ACK'd the event.
u.ts	Timestamp of last ACK received for the event.
Publisher	
SS	Set of subscriptions for this publisher.
s.pat	Subscription pattern (where s ∈ SS).
s.dest	Subscriber associated with subscriptions.
SE	Set of events sent by this publisher that have not yet been ackd by all destinations.
PS_{dest}	Sequence number of the previous event sent to subscriber dest.
Subscriber	
RE	Set of events received at this subscriber.

Table 3. RAFT-CBR data structures

We develop a reliable content-based xcast protocol (RAFT-CBR), to route subscriptions and events between any two nodes.

In RAFT-CBR, advertisements from the publisher are flooded throughout the network and periodically beaconed, just as in FT-CBR and CBR. However, since advertisements are used to disseminate the existence of publishers rather than being used to keep routes to publishers fresh, advertisements can be beaconed much less frequently, or even

```
Algorithm receive(Event e)
(* Deliver and forward event *)
1.  wasForMe ← e.cids.contains(MyId)
2.  wasFlooded ← e.fhops > 0
3.  keepFlooding ← e.fhops > 1
4.  gotFromSelf ← e.sender = MyId
5.  seen ← false
6.  seenFlooded ← false
7.  seenForMe ← false
8.
9.  (* Update or add to flood table. *)
10. fe ← FloodTable.getMatching(e)
11. if fe ≠ nil
12.   then seen ← true
13.        if fe.cids.contains(MyId)
14.          then seenForMe ← true
15.               fe.cids.addUnique(e.cids)
16.        if fe.fhops > 0
17.          then seenFlooded ← true
18.          else fe.fhops ← e.fhops
19.
20. (* Find children with matching subs. *)
21. fwdevt ← e
22. fwdevt.cids ←
23.     SubsTable.getMatchingChildren(e.body)
24. fwdevt.fhops ← 0
25.
26. (* Ack this event. *)
27. doAck ← !wasFlooded ∧ !gotFromSelf ∧
    wasForMe
28. if doAck
29.   then ack ← new ACK message
30.        ack.publisher ← e.publisher
31.        ack.seq ← e.seq
32.        send(e.sender, ack)
33. (* Keep forwarding the event. *)
34. doFwd ← !seenForMe ∧ wasForMe
35. if doFwd
36.   then (* Deliver to own node first. *)
37.        if MyId ∈ fwdevt.cids
38.          then notify(fwdevt)
39.               fwdevt.cids ←
40.                   fwdevt.cids \ MyId
41.        (* Forward to neighbors. *)
42.        if fwdevt.cids ≠ ∅
43.          then broadcast(fwdevt)
44.               UnackdTable.add(fwdevt)
45. (* Keep flooding the event. *)
46. doFlood ← keepFlooding ∧ !seenFlooded
47. if doFlood
48.   then dupevt ← e
49.        dupevt.fhops ← dupevt.fhops - 1
50.        dupevt.cids ← dupevt.cids \ MyId
51.        if dupevt.cids ≠ ∅
52.          then broadcast(dupevt)
53.
54. (* Add to flood table if did something. *)
55. if !seen ∧ (doAck ∨ doFwd ∨ doFlood)
56.   then FloodTable.add(e)
```

Figure 3. FT-CBR event handler

EVENT	Event message.
e.pbr	Publisher of event e.
e.seq	Sequence number of event e.
e.chain	Ordered set of nodes that have forwarded event e, with the publisher's node being the first element.
e.body	Contents of event e.
e.(dest,seq)	Set of subscribers that should receive this event, and the sequence number of the last event e.pbr sent to e.dest.
SUB	Subscription message
s.sbr	Subscriber
s.pbr	Publisher
s.pat	Subscription pattern
SUBACK	Subscription acknowledgment message
a.sbr	Subscriber
a.pbr	Publisher
a.pat	Subscription pattern
PING	Ping message.
p.pbr	Publisher of the message.
p.chain	Ordered set of nodes that have forwarded the message, with the publisher's node being the first element.
p.dests	Subscribers that should receive this message.
ACK	Cumulative acknowledge message.
a.pbr	The publisher of the event this ack is for.
a.seq	The sequence number of the event this ack is for.
a.dests	The subscribers this ack if for.
NACK	A request for events that have not been received.
n.pbr	The publisher whose events this NACK is for.
n.sbr	The subscriber who sent this NACK.
n.chain	The ordered set of nodes to to forward this NACK to.
n.holes	A set of (lower,upper) sequence number ranges. Note that lower and upper may be $-\infty$ or ∞, respectively.

Table 4. RAFT-CBR Messages

eliminated by having subscribers pull advertisements from a neighbor upon entering the network.

Subscribers unicast subscriptions to those publishers whose advertisements intersect the subscription interest. These subscriptions are acknowledged by the publisher. Unlike the previous algorithms, subscriptions are not used to maintain a multicast tree from publisher to subscribers. Instead, since the publisher knows the subscribers' addresses, it uses reliable xcast to disseminate events, as is further explained below. For this reason, subscription beaconing is no longer required; the subscribers stop sending subscriptions once they receive an acknowledgment from the publisher.

The messages and data structures in RAFT-CBR are shown in Tables 3 and 4, respectively.

4.3.1. Event delivery In RAFT-CBR, publishers tag each event e with a locally unique sequence number, such that the (publisher id, sequence number) pair can be used as a globally unique identifier of e. The event also includes the addresses of the set of subscribers C to which the event needs to be delivered. When a node receives an event, it queries the underlying routing protocol to determine the next hop to reach every subscriber $c \in C$. The node then forwards e to each of these next hops, modifying the set C in e to include only those subscribers that a particular next hop is responsible for forwarding to. This process repeats until an event reaches all interested subscribers. The pseudo-code for this procedure is shown in Figure 4.

4.3.2. Reliability The RAFT-CBR achieves reliability by exploiting unique event sequence numbers and publishers' knowledge of interested subscribers. A publisher maintains a table $\mathrm{PS}_{\mathrm{dest}}$ that stores, for each subscriber for which it has a subscription, the sequence number of the last event that was published to that subscriber.

With every event, the publisher stores not only the subscribers' addresses, but also the sequence number of the last event that the publisher published to each of these subscribers. With these previous sequence numbers, it is possible for a subscriber to determine if it has missed any events.

```
Algorithm receive(EVENT e)
(* Broker receives an event. *)
1.  append MyId to e.chain
2.  removeCycles(e.chain)
3.  x ← new UE element
4.  x.{evt, dests, ackd, ts} ← e, e.dests, ∅, NOW
5.  UE ← UE ∪ x
6.
7.  pbr ← e.pbr
8.  par ← e.chain.myparent
9.  ds[−] ← ∅ (* (dest,seq) pair for each neighbor. *)
10.
11. (* Determine next hops *)
12. for (d, s) ∈ e.(dest, seq)
13.     do if d = MyId
14.           then mysubscriber.receive(e)
15.           else n ← getNeighbor(d)
16.                if n = ∅ ∨ n ∈ e.chain
17.                   then (* Unicast event. *)
18.                        e′ ← e
19.                        e′.(dest, seq) ← (d, s)
20.                        send(d, e′)
21.                   else ds[n] ← ds[n] ∪ (d, s)
22.
23. (* Forward event to next hops *)
24. for {n | ds[n] ≠ ∅}
25.     do e.(dest, seq) ← d[n]
26.        send(n, e)
```

Figure 4. RAFT-CBR broker event handler

If so, it sends a negative acknowledgment, NACK, up the multicast tree towards the publisher. The NACK specifies ranges of sequence numbers that the subscriber is missing. Nodes along the NACK path that have events that fall within the requested ranges unicast these events directly to the subscriber. If not, the NACK propagates up to the publisher which is guaranteed to have the event.

When a subscriber receives an in-order event, it replies with a positive acknowledgment, ACK, to its parent broker in the multicast tree. Brokers in the multicast tree collect ACKs from their children and eventually forward them up to the publisher. Notice that under normal operation, events and ACKs are multicast not unicast.

ACKs and NACKs are only sent when a subscriber receives an event. PING messages are used in cases where events fail to reach a subscriber. If the publisher does not receive an ACK or NACK from the subscriber for a specified time, it multicasts a PING to these subscribers, in order to force the subscribers to respond with an ACK or NACK.

5. Evaluation

This section presents an experimental evaluation of the algorithms introduced in Section 4. The experimental testbed, workload, and metrics of interest are described, after which the results are discussed in detail.

5.1. Methodology and Scenario

Our experiments use ns2 as the underlying network simulator, with the CBR, FT-CBR, and RAFT-CBR algorithms written on top of ns2.

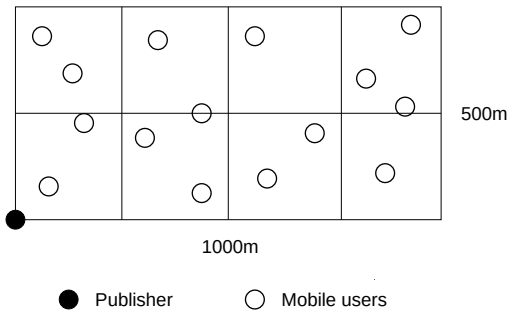


Figure 5. Shopping district scenario

Since real-world P/S or mobility traces are difficult to obtain, a plausible scenario is developed. The scenario is a shopping district in which shoppers roam about on foot. The publishers are the vendors who would like to send coupons or product advertisements to shoppers. The shoppers would only like to receive notices for those products or vendors they are likely to be interested in. In this scenario, shoppers do not explicitly indicate their interests; instead, their interest is implied by their current location. For example, users who are browsing a sporting goods store are likely to be interested in coupons for athletic footwear, and those near a movie theater are good candidates to receive movie ticket coupons. It is assumed that the shoppers' mobile devices (perhaps a PDA), contain a GPS device that can be used to determine their location. A GPS is not a requirement as other techniques to determine location can also be used.

It is important to emphasize that the stationary publisher and use of location data are features of this scenario; the algorithms in this paper can also work with mobile publishers and without location information. As mentioned in the introduction, other typical scenarios include selective data dissemination in accident scenarios, selective transmission of photos to friends and relatives, and complex data exchange among the participants of a cooperative game.

As shown in Figure 5, the scenario simulates a 1000x500m area populated with 50 mobile users moving according to a random waypoint model with a speed of 1m/s and 60s pause time. There is one stationary publisher at the corner of this area publishing at a rate of 3.2 events/s, or eight events every 2.5s. The stationary publisher is justified in this shopping district scenario because the publishers here are the various shopping outlets, and publications originate from a server in the wired network and enter the wireless network through a fixed wireless base station. To simplify the scenario and our implementation, the simulated area is divided into eight 250x250m zones, and a subscriber's current location

is specified by the zone it resides in, instead of its exact GPS location. Therefore, both subscriptions and the contents of publications are simply an integer between one and eight. Also publication messages are padded to be 512 bytes in size. The tests are run for 800s in simulated time and the results are averaged over ten runs. When computing the metrics, the first 200s of simulation are ignored in order to allow the system to stabilize.

5.2. Metrics

This paper uses three metrics to compare the performance of CBR, FT-CBR, and RAFT-CBR. The *delivery rate* is the primary metric, and it measures the ratio of successfully delivered events. At the time of publication of every event, the number of subscribers that are interested in that event is calculated, and this becomes the expected number of deliveries for this event. Then the percentage of these expected deliveries that actually occur is counted. Note that this expected count cannot be determined in a real network since information of the subscribers' interest is distributed over the network. It should be noted that this expected count is only an estimate, since it is possible for subscribers' interest to change while the event is in transit. Therefore, it is possible that a correct operation of the system results in fewer or greater deliveries, although our experiments show that this discrepancy is hardly noticeable.

The second metric is the *delay* in event delivery. This is computed as the average of the differences between the publication and delivery of an event to every subscriber. Note that when a single publication is delivered to multiple subscribers, the difference is computed once for every subscriber that delivered the event. Also note that undelivered (but expected) events are not included in this metric.

Finally, the *message cost* of the algorithms is examined. Every message transmitted, regardless of size, is counted as one message. The total number of messages transmitted by all the nodes during the simulation is presented. Furthermore, the message cost is divided into data (events), subscriptions, routing (advertisements, and any underlying routing protocol), and control (messages related to fault-tolerance and reliability, including ACKs, NACKs and PINGs).

5.3. Algorithm parameters

There are several parameters that control the behavior of each protocol. Subscription, advertisement and MC leases are used to maintain soft state. Each lease is renewed every 7 seconds, and it expires after 15 seconds. Consequently, a broker that misses two lease renewals expires the subscription state. For FT-CBR, an event is retransmitted if an acknowledgment for it is not received within 0.5 seconds. Expanding ring search is limited to 2 hops and if MC is used, the paths to the publisher are allowed to deviate up to a maximum of 2 hops from the shortest path. Recently flooded events are remembered for 5 seconds. For RAFT-CBR, negative acknowledgments are aggregated and sent every 30ms and acknowledgments are aggregated and sent every 10ms. The publisher waits for 10 seconds for acknowledgment of an event before sending a PING message to subscribers which are unresponsive. Consequently, cached events are expired after 10 seconds as well. DSDV [17] is used as the underlying routing protocol for RAFT-CBR.

5.4. Results

The experiments evaluate how the algorithms perform with varying subscribers, mobility speed, and interest locality. In addition, the impact of covering and MC optimizations on the CBR algorithm with increasing subscribers and varying interest locality is measured.

5.4.1. Scalability Figure 6 shows the performance of the CBR, FT-CBR, and RAFT-CBR algorithms with increasing number of subscribers. Note that there are always 50 mobile users; only the number of these users that are subscribers varies.

CBR can deliver about 75% of the events within less than 0.1s. These figures are largely independent of the subscriber population, because the zones in our scenario are small enough to be in the broadcast range of a node. Thus, the multicast tree depth does not change much whether there is one interested subscriber in every zone (the case with 10 users), or six per zone (50 user case). In other words, the protocol exploits the broadcast channel. Furthermore, CBR has no failure recovery mechanisms so delay will not change significantly even if there is congestion in the network (delivery rate would decrease, but remember that undelivered events are not counted in the delay metric). CBR has sub-linear growth in message cost with increasing subscribers.

FT-CBR improves on CBR by having nearly a 100% delivery rate, but at a delay slightly higher than 0.1 to 0.2s. The reason for the higher average delay is the retransmission timers used in FT-CBR. Also, FT-CBR suffers from a higher and faster growing message cost than CBR. The high message cost is largely due to the flooding of unacknowledged events in FT-CBR. Unlike CBR, the high bandwidth consumption of FT-CBR with increasing subscribers impacts both the delivery rate and delay.

RAFT-CBR, as expected, is able to deliver virtually 100% of the events for all subscriber populations shown, but it has a relatively high delay of almost one second. The delay can likely be improved with more adjustment of the timers. It is interesting that RAFT-CBR has a lower message cost than FT-CBR and yet achieves higher delivery rate. This is because the error recovery mechanism in FT-CBR (flooding) is more expensive than that in RAFT-CBR (multicast pings).

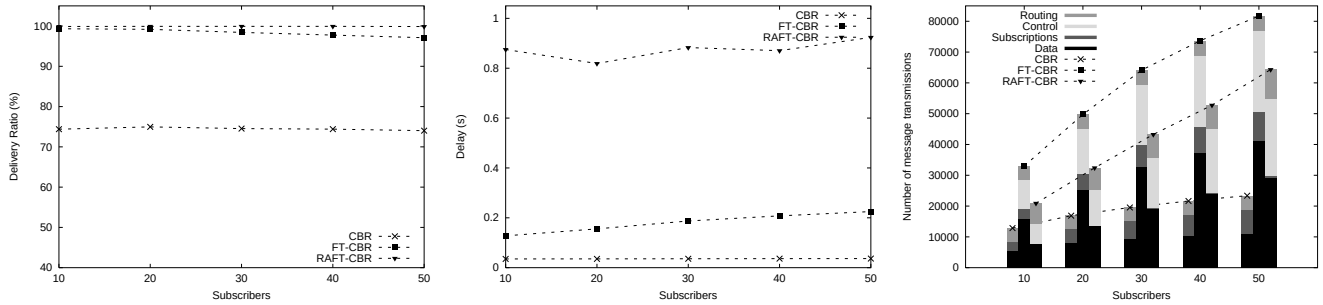


Figure 6. Performance for increasing subscribers

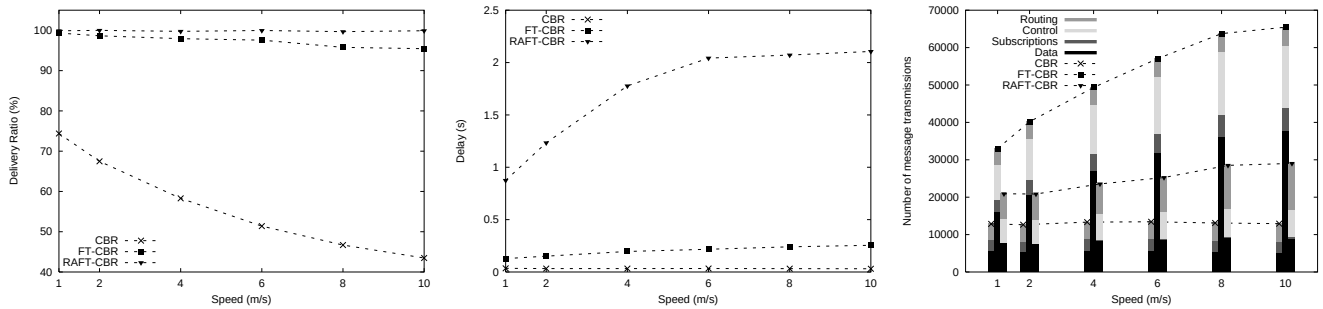


Figure 7. Performance for increasing speed

The dissection of messages in the message cost graph in Figure 6 is useful in understanding where each algorithm gains and loses. We see that CBR has the fewest transmission of events (since it never retransmits events). This, and the lack of any control messages are why CBR has the lowest message cost. We also see that even though RAFT-CBR has comparable control message cost to FT-CBR, and greater routing cost than FT-CBR, its overall cost is lower, primarily due to lower event cost. Recall that FT-CBR floods events when a node cannot be found whereas RAFT-CBR uses less expensive unicast to find the node. RAFT-CBR also gains from not needing to beacon subscriptions.

Figure 6 demonstrates that there is no clearly "better" algorithm. The choice depends on the desired delivery rate, available bandwidth, and required P/S semantics.

5.4.2. Mobility speed Figure 7 shows the performance of the algorithms with increasing maximum node mobility speed.

The CBR algorithm suffers from lower delivery rate as users move quicker. The delay and message cost are relatively constant because, even though more messages are lost when there is greater mobility, CBR has no error recovery mechanisms. Also, the multicast trees have roughly the same shape and size (since there is a constant number of randomly moving subscribers); the tree is simply changing more often as users move faster. The beacons are not fast enough to refresh the topology information when node mobility is high, leading to stale routes, and therefore lower delivery rates.

Because of these stale routes, FT-CBR experiences more message losses with higher mobility, and needs to flood events more frequently leading to increasing message cost. Indeed, most of the message cost increase in FT-CBR is due to flooded events. The ability of flooding to bypass stale, broken routes diminishes as the speed increases. However, even with 10m/s mobility (an unreasonably high speed for users in a shopping district), FT-CBR achieves about 95% delivery rates.

RAFT-CBR, reassuringly, maintains 100% delivery at high mobility, and has a scalable increase in message cost. This suggests that the routing protocol underlying RAFT-CBR—DSDV in this case—is able to maintain routes more cheaply than the beacons in FT-CBR. However, the results show that mobility has a greater impact on the delay of RAFT-CBR than FT-CBR. So, there is a trade-off here between the speed and cost of updating stale routes. Also notice that, unlike FT-CBR, most of the message cost increase in RAFT-CBR is due to the underlying routing protocol having to work harder to maintain routes; there is little increase in the event cost.

The increasing delay of RAFT-CBR with increasing mo-

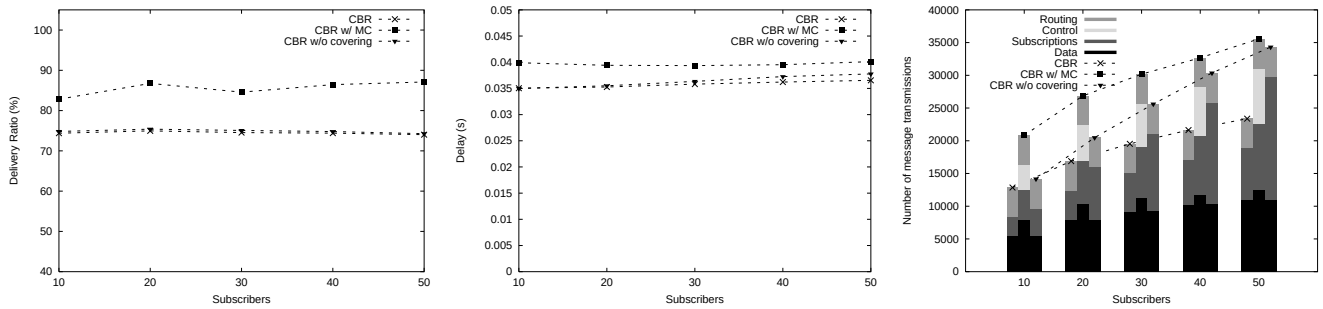


Figure 8. Performance of optimizations for increasing subscribers

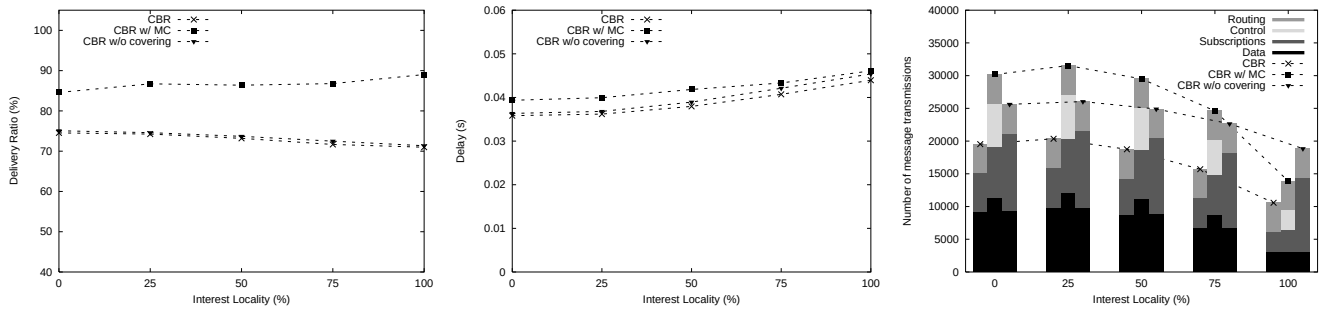


Figure 9. Performance of optimizations for increasing locality

bility might be cause for some concern. It suggests that there could be a speed at which RAFT-CBR will completely break down. Since RAFT-CBR does not give up on delivering old events, it might not be able to deliver events fast enough to keep up with a sustained event rate. This would not be a problem if the event rate is not too high, or if events were bursty, giving the algorithm time to "catch up." The delay is also highly dependent on the various timer values used in this algorithm. We plan to investigate this issue further.

5.4.3. Scalability with optimizations Figure 8 explores the effects of the covering and MC optimizations. The CBR algorithm is considered without any optimizations, with covering only, and with both covering and MC. CBR is used because it has the worst delivery rate and any improvements will be most evident here.

Figure 8 shows that the MC optimization improves the delivery rate by about 10%. It is expected that the benefits of the MC optimization will improve as there are more subscribers, and a larger multicast tree. This claim is supported by the graphs in Figure 9 where the effects of a larger multicast tree (obtained by increasing locality) are shown.

Understandably, the delay increases with MC. This is because the MC optimization favors paths that attach to the multicast tree quickly, even if these paths are not the shortest path to the publisher.

For the message cost, predictably, covering reduces cost by quenching subscriptions. The MC optimization has higher cost, mostly due to the introduction of messages to beacon MC information. All three algorithms experience sub-linear message cost growth with increasing subscribers, with the no covering algorithm suffering from the worst message cost scalability. The latter is due to the increase in subscription messages resulting from more unquenched subscriptions.

5.4.4. Interest locality with optimizations Figure 9 studies the impact the degree of similarity among subscriptions has on the algorithms. We vary the percentage of subscribers who have identical, and static interest; the remaining users have, as before, interest that varies with their location. This models the case where some shoppers are always interested in certain items regardless of their location.

With no locality, while subscriptions change frequently, the changes are localized in the network with the effect that all subscribers in a zone always have the same subscription. Thus, the shape of the multicast tree remains relatively stable. In the full locality case, even though subscribers do not change their subscriptions, these subscriptions move all over the network. This greatly affects the shape of the multicast tree over time; in addition, since subscriptions are not localized to a zone, the multicast tree for a particular event

must now cover a larger area of the network, whereas it only had to reach one zone before. The need to update a larger and more dynamic multicast tree reduces the reliability when there is high locality present.

The MC optimization leads to a better delivery rate as subscribers have more common interest. This is because even though the multicast tree is larger and more dynamic, MC is able to reduce message cost significantly (because it gets better at finding a quick path to join the multicast tree when the multicast tree has greater reach and has more nodes).

All three algorithms suffer from longer delay as the locality increases. This is a consequence of the multicast trees becoming more far-reaching; with no locality, events for a zone near the publisher do not propagate into more distant zones, but this is no longer true as locality increases.

The MC optimization gains the most from increased locality since, as explained above, it gets better at finding a quick path to join the multicast tree.

6. Conclusions and Future Work

Three CBR algorithms for ad hoc networks are presented. RAFT-CBR is the best choice if reliability is more important than resource preservation. However, if low message cost is a priority, and reliability is not crucial, CBR is a better choice. For example, CBR is a good choice for streaming audio dissemination, where delay and message cost are important, while reliability is not as crucial since old data might no longer be useful as it is replaced by new data. FT-CBR, is recommend for highly dynamic environments where network topology is in constant flux and node failures are frequent.

Even though multicast has been proposed to solve similar information dissemination problems in MANETs, content-based routing provides a more flexible and robust approach.

In the future, we would like to investigate subscriber locality, in which some zones have more subscribers than others. In addition, we plan to develop more scenarios to verify the performance results under varying conditions and stress the algorithms further, and also examine the state requirements of each algorithm.

References

[1] G. Banavar, T. Chandra, B. Mukherjee, J. Nagarajarao, R. E. Strom, and D. C. Sturman, "An efficient multicast protocol for content-based publish-subscribe systems," in *ICDCS*, 1999.

[2] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf, "Design and evaluation of a wide-area event notification service," *ACM Transactions on Computer Systems*, vol. 19, pp. 332–383, Aug. 2001.

[3] L. Opyrchal, M. Astley, J. S. Auerbach, G. Banavar, R. E. Strom, and D. C. Sturman, "Exploiting IP multicast in content-based publish-subscribe systems," in *Middleware*, pp. 185–207, 2000.

[4] L. Ji and S. Corson, "Explicit multicasting for mobile ad hoc networks," *ACM Mobile Networks and Applications*, vol. 8, pp. 535–549, Oct. 2003.

[5] K. Chen and K. Nahrstedt, "Effective location-guided tree construction algorithms for small group multicast in manet," in *INFOCOM*, 2002.

[6] F. Fabret, H.-A. Jacobsen, F. Llirbat, J. Pereira, K. Ross, and D. Shasha, "Filtering algorithms and implementation for very fast publish/subscribe systems," in *SIGMOD*, 2001.

[7] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron, "Scribe: A large-scale and decentralized application-level multicast infrastructure," *IEEE JSAC*, vol. 20, oct 2002.

[8] U. Varshney, "Multicast over wireless networks," *Communications of the ACM*, vol. 45, pp. 31–37, December 2002.

[9] S. Lee, W. Su, and M. Gerla, "On-demand multicast routing protocol (ODMRP)," in *Internet Draft, draft-ietf-manetodmrp-02.txt*, January 2000.

[10] E. Royer and C. Perkins, "Multicast using ad hoc on-demand distance vector routing," in *MOBICOM*, 1999.

[11] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed diffusion: a scalable and robust communication paradigm for sensor networks," in *MOBICOM*, pp. 56–67, 2000.

[12] I. Burcea, H.-A. Jacobsen, E. de Lara, V. Muthusamy, and M. Petrovic, "Disconnected operation in publish/subscribe middleware.," in *IEEE Mobile Data Management*, pp. 39–50, 2004.

[13] 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," *International Conference on Distributed Computing Systems (ICDCS'05)*.

[14] H. Liu and H.-A. Jacobsen, "Modeling uncertainties in Publish/Subscribe System," in *In Proceedings of ICDE*, 2004.

[15] M. Petrovic, I. Burcea, and H.-A. Jacobsen, "S-ToPSS - a semantic publish/subscribe system," in *Very Large Databases (VLDB'03)*, (Berlin, Germany), September 2003.

[16] M. Petrovic, H. Liu, and H.-A. Jacobsen, "G-ToPSS - fast filtering of graph-based metadata," in *the 14th International World Wide Web Conference (WWW2005)*, (Chiba, Japan), May 2005.

[17] C. Perkins and P. Bhagwat, "Highly dynamic destination sequenced distance vector routing (DSDV) for mobile computers," in *SIGCOMM*, pp. 234–244, 1994.