Opportunistic Overlays: Efficient Content Delivery in Mobile Ad Hoc Networks

Yuan Chen and Karsten Schwan

College of Computing, Georgia Institute of Technology, USA
{yuanchen,schwan}@cc.gatech.edu

Abstract. Current content-based publish/subscribe systems assume a fixed network environment in which nodes do not move and where the network topology remains relatively stable. For mobile environments, the resulting problem is a mismatch between static broker topologies and dynamic underlying network topologies. This mismatch will result in inefficiencies in event delivery, especially in mobile ad hoc networks where nodes frequently change their locations. This paper presents a novel middleware approach termed *opportunistic overlays*, and its dynamically reconfigurable support framework to address such inefficiencies introduced by node mobility in publish/subscribe systems. The opportunistic overlay approach dynamically adapts event dissemination structures (i.e., broker overlays) to changes in physical network topology, in nodes' physical locations, and in network node behaviors, with the goal of optimizing end-to-end delays in event delivery. Adaptation techniques include the dynamic construction of broker overlay networks and runtime changes of mobile clients' assignments to brokers.
Opportunistic overlays and the adaptive methods they use are realized by a set of distributed protocols implemented in a Java-based publish/subscribe infrastructure. Performance evaluations use simulation, system emulation, and also involve running a representative application on an experimental testbed. Experimental results demonstrate that the opportunistic overlay approach is practically applicable and the performance advantages attained from the use of opportunistic overlays can be substantial.

1 Introduction

Publish/subscribe is a widely used method for providing anonymous, inherently asynchronous group communications in distributed settings. A publish/subscribe system has three main elements: (1) an event producer generating (publishing) events (messages), (2) an event consumer who declares its interest in receiving certain events via a subscription, and (3) an event broker responsible for collecting and processing event subscriptions and for routing processed events to corresponding consumers. Most publish/subscribe architectures employ an application-level broker network(i.e., overlay) to perform content-based routing of events at application-level.

Past work has created numerous publish/subscribe systems, in industry and in academia [27, 2, 8, 26, 12, 29, 10, 36]. With the increased availability of powerful mobile computing devices like laptops and IPAQs, and the widespread deployment and use of wireless data communications, there is a pressing need to extend such middleware to the mobile computing domain. Moreover, certain features of publish/subscribe make it well-suited to mobile environments, including asynchronous event delivery, anonymity, multipoint communication and content-based routing [15, 7, 11]. Current systems targeting Internet-based communications, however, commonly assume a fixed network environment in which clients do not move and where the network topology remains relatively stable. Stated more technically, they assume statically deployed broker networks(i.e., overlays) mapped to static network topologies. The resulting problem for mobile environments is a mismatch between static broker topologies and dynamic underlying network topologies. This mismatch will result in inefficiencies in

event delivery, a simple example being a shortest path in the original overlay and physical network turning into an inefficient path when the same logical overlay is used with a different physical network topology. Another example is that a node's movement(either a client, or a broker or an intermediate node) affects the underlying network topology and changes the distance between a mobile client and its assigned broker, hence resulting in sub-optimized event delivery.

This paper proposes *opportunistic overlay* approach to manage overlays for mobile nodes in mobile ad-hoc networks. The idea is to dynamically optimize content-based event delivery by adapting event dissemination structures (i.e., broker overlays) to changes in physical network topology, in nodes' physical locations, and in network node behaviors. The term 'opportunistic' denotes the fact that the solution is one in which each broker opportunistically acts to improve its relations with both other brokers and with its clients. The key points characterizing opportunistic overlays may be summarized as follows: (1) dynamically constructing broker network topologies to match the underlying physical network, (2) dynamically changing a mobile client's broker assignment based on the client's physical location and the broker's current capabilities, and (3) when broker topologies or clients' home brokers change, recalculating overlay routing paths and then using the newly computed paths.

Opportunistic overlays are implemented with the JECho Java-based publish/subscribe infrastructure [36]. A unique attribute of this implementation is that with JECho, dynamic topology adjustments can be coupled with runtime techniques for event filtering, thereby also permitting the system to match event rates and sizes to the currently available levels of bandwidth of physical communication channels. This is particularly effective in JECho because it uses general Java functions for event filtering rather than the predicate-based filters supported elsewhere. That is, JECho subscriptions are based on general functions deployed by event consumers 'into' event producers or brokers. These functions not only implement the event processing needed for the complex data conversions present in multimedia, business, or scientific applications, but they can also realize application-specific tradeoffs in performance vs. desired functionality (e.g., event contents) in order to address some of the resource limitations existing in pervasive systems.

The performance evaluations reported in this paper are performed both on actual hardware, to assess basic performance properties and penalties and via emulation and simulation, to assess the effects of mobility and to better understand the scalability of our approach. Experimental results demonstrate that the performance advantages attained from the use of opportunistic overlays can be substantial. For instance, simulation results indicate that the delay of sending a message can be improved by up to 100%. In a set of emulation experiments, the opportunistic overlay approach is able to both optimize path lengths and address broker overloads. Measurements on a small testbed comprised of three laptops running the AODV protocol [18] show more than six fold improvements in the end-to-end delay experienced by events in the flood watch application. The overheads experienced by opportunistic overlay behavior are moderate, For example, the average per broker bandwidth used for state propagation is less than 2Kbps for a moderate size broker network with 40 broker and for a broker update interval of 100 seconds.

The remainder of this paper is organized as follows. We present the system model, protocols, and algorithms used by opportunistic overlays in Section 2. The prototype architecture and some implementation details are discussed in Section 3. Section 4 presents evaluation results. Related work appears in Section 5, followed by conclusions and future work in Section 6.

2 The Opportunistic Overlay Approach

We first outline the system model assumed by opportunistic overlays, followed by descriptions of the adaptation protocols and algorithms underlying the approach.

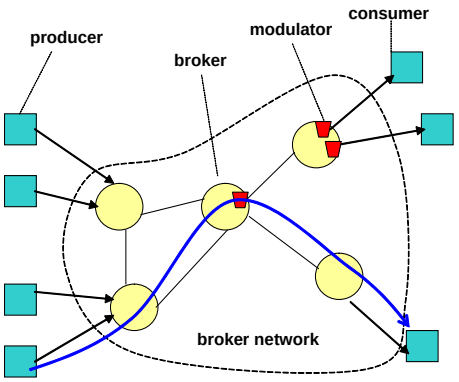

Fig. 1. System Model

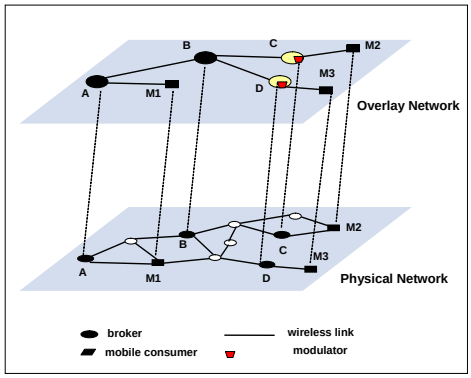

Fig. 2. A Sample Event System in Mobile Ad Hoc Networks

2.1 System Model

Our system model adopts an overlay network approach. As illustrated in Figure 1, an event system consists of producers, consumers, and a broker network. The latter is an overlay across the physical network, composed of broker processes connected via links. Each overlay link is a network path between a broker node pair in the physical network. Each producer/consumer (mobile client) connects to one of the brokers (usually the nearest one) via one or multiple wireless links. This broker is called the client's *home broker*. A consumer also provides a content-based subscription function termed *modulator*, which operates on event contents to dynamically tailor them to the consumer's current needs. A consumer's modulator executes in an intermediate broker's address space on behalf of the consumer. The intermediate broker can be any broker(typically its home broker) on the overlay path between producers and the consumer. An event generated by a producer is first sent to the producer's home broker, then routed from the producer's home broker to the consumer's home broker, processed using the consumer's modulator, and then delivered to the consumer via some wireless network links.

An event system with four broker nodes (A, B, C and D), one producer M1 and two consumers (M2 and M3) in a wireless ad hoc network is depicted in Figure 2. A mobile ad hoc network(MANET) is an autonomous system where all nodes are free to move, organizing themselves to form an on-the-fly underlying physical network without relying on any fixed network infrastructure. Two nodes typically communicate with each other by a multi-hop wireless link between them. In mobile ad hoc networks, all brokers, producers and consumers reside on mobile nodes. A broker link of the broker network is a multi-hop wireless path on the underlying physical wireless network. A producer/consumer connects to its home broker via a multi-hop wireless path, too. A broker can reside on the same nodes as producers/consumers or on separate nodes. One extreme case is for each node to act as both broker and producer/consumer, and as home broker. Since most nodes in wireless ad-hoc networks have limited resources, however, it is not practical to assume that processors can run arbitrarily complex modulators. Instead, it is often advantageous to use as brokers additional nodes and their resources, such as PCs installed on higher end nearby platforms (e.g., 'command post' vehicles [6, 31]).

2.2 Basic Idea

The idea behind opportunistic overlays is to continually optimize event delivery, by dynamically changing broker networks and mobile clients' home brokers. Updates occur in response to changes in physical network topology and in nodes' physical locations. Potential broker overloads are avoided by judiciously choosing clients' home brokers. The key points characterizing opportunistic overlays may be summarized as follows:

Resource awareness. An opportunistic overlay is aware of the underlying network topology used for transporting events from producers to consumers. It is also aware of the respective locations of both and of its current state (e.g., CPU Load, Memory availability).
Dynamic construction of broker overlay networks. Dynamic broker network topology construction uses a global state routing protocol [4]. Each broker maintains a local view of the broker network topology. At runtime, an opportunistic overlay dynamically monitors client location, physical network topology, and resources (e.g. latency, bandwidth, broker computation load). Periodically, each broker updates its local view of broker network topology, by changing its neighboring brokers and by propagating changes to its neighbors. Neighbor broker information is acquired by querying the network protocols' routing tables or via neighbor discovery operations [13]. When a broker receives propagated information from its neighbors, it updates its broker topology accordingly.
Dynamic change of home broker. A mobile client periodically checks the brokers in its vicinity via a 'nearest broker search'. It identifies to its current home broker a found candidate broker closer to its current physical location than said home broker. Upon receiving such a report from a client, the home broker initiates a broker selection protocol. This protocol uses an approach that combines shortest path selection with load balancing methods. Specifically, the home broker first calculates the path length from the producer to the candidate broker, and then determines whether or not to change the client's home broker based on both the network distance and the candidate broker's current capabilities. Preference is giving to the closer broker unless that broker is currently overloaded.
Dynamic overlay routing. Changes in broker network topology and in mobile clients' home brokers will result in rebuilding broker-level routing tables. Opportunistic overlays use source routing for event delivery, where whenever a broker's local view of broker topology changes or whenever a client receiving events from a broker changes its home broker, new event paths are calculated using a shortest path algorithm. We next discuss some of these protocols in more detail.

2.3 Dynamic Construction of Broker Networks

Broker network topologies are kept congruent with the underlying physical network topology by periodically re-constructing global knowledge about the broker network, using a global state routing protocol [4]. Toward this end, each broker maintains its knowledge about the current broker network topology in a topology table T. Periodically, each broker receives its neighboring broker's T, updates its own T, and then propagates found topology updates to its neighbors. Each broker keeps track of the other brokers in its vicinity by querying the routing table maintained by the wireless network routing protocol used in each broker machine, or via a neighbor discovery protocol like the Expanding Ring Search described in [13]. The broker topology update protocol can be summarized as follows.
Step 1: Broker Neighbor Discovery. Each broker periodically updates its neighboring brokers using the Expanding Ring Search. If a neighbor broker moves too far away from a broker, then the original overlay link between the broker and that neighbor is removed from the broker network. If a broker moves into the vicinity of another broker, a new broker link between them is created.
Step 2: Broker Topology Propagation. Once a broker completes updating its topology table by neighbor discovery, the broker sends to its neighbors those items in its topology table that have changed since the prior propagation period. A sequence number is associated with each such update.
Step 3: Broker Topology Update. When a broker receives updated information from its neighbor, it compares the sequence number of the incoming message with its topology table's corresponding items, replaces old items with new ones, and marks the items changed if the incoming items have a higher sequence number.
Step 4: Broker Routing Table Rebuilding. A broker's topology table T changes either due to its own execution of the periodic neighbor update or due to the receipt of topology propagation from

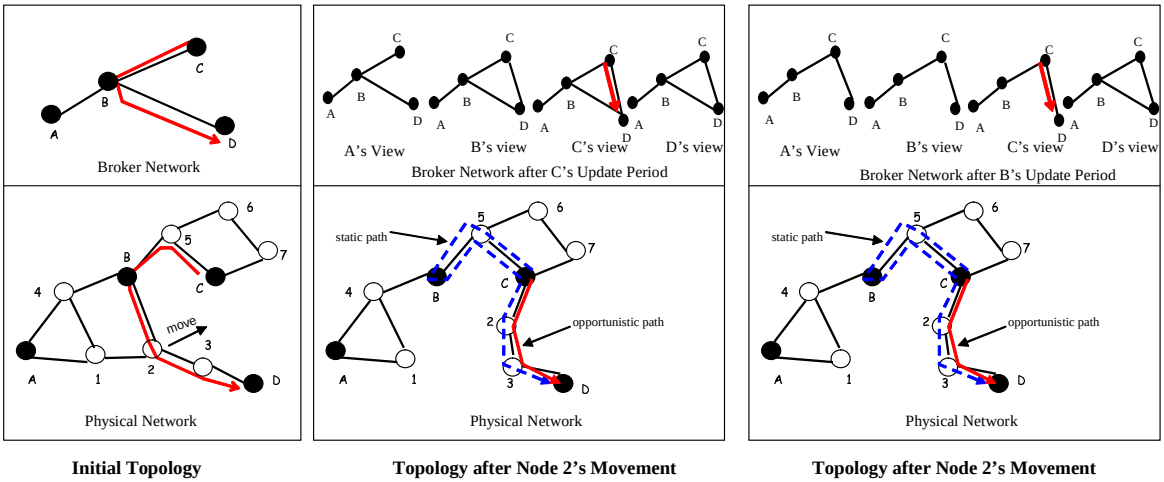

Fig. 3. An Example of Dynamic Broker Network Construction

its neighbors. When such changes occur, the broker rebuilds its routing table by recalculating its shortest path to other brokers henceforth uses the new routing table for delivering events.

Figure 3 depicts an example. At the beginning, all four brokers(A, B, C and D) have the same view of the global broker network topology A—B, B—C and B—D. At some point, node 2 moves away from B and 1, and closer to C. As a result, two old wireless links 2—B and 2—1 are removed and one new wireless link 2—C is created. Let's assume C is the first to start its update period. C adds D's as its new neighbor. Then C updates it topology table accordingly and propagates the change to its neighboring brokers B and D. After receiving updated information from C, each of B and D updates its topology table by adding the broker link C—D. B, C and D will rebuild their overlay routing tables based on the new broker network topology A—B, B—C, B—D and C—D. As a result of C's routing table update, opportunistic overlays deliver events from C to D using the wireless path C→2→3→D, compared with the static broker approach's C→5→B→5→C→2→3→D. Let's assume B is the next to start its update period. B removes D from its neighbor list and updates its broker topology knowledge accordingly. B then sends its changes since previous period to its neighbors A and C. Upon receiving updated information from B, both A and C update their topology knowledge by removing broker link B—D. A also adds the broker link C—D to its topology table. At this time, each of A, B and C has the latest broker topology knowledge. D's topology knowledge is outdated, and D's topology table will be updated either through C's propagation or via D's own running of the neighbor updates protocol, whichever comes first.

Our current protocol assumes that there are no network partitions. One solution is described next. Consider a broker that discovers that the next broker or the client on the event delivery path is not reachable, using for example, the network partition detection algorithm described in [14]. If the destination is still reachable in some way, then simply compute an alternative path to the destination. The result is that opportunistic overlays continue to operate correctly within the same partition. For those destinations that cannot be reached, our current thoughts are to store undeliverable events for later delivery and/or ask applications to provide functions that implement event discard. The event ordering issues caused by dynamic routing and network partition can be handled at application-level using higher-level (e.g. JECho's) synchronization support [34]. We leave to future work topics like disconnection, reconnection, and related reliability issues.

2.4 Dynamic Home Broker Change

End-to-end latency not only depends on the network distance between a producer's home broker and a consumer's home broker, but also on the distance between producers/consumers and their home brokers. By dynamically constructing a broker network, we aim to optimize the former. By dynamically changing home brokers, we improve the latter. Toward these ends, opportunistic overlays act as described next.

When a client subscribes to a broker network for the first time, it must connect to some home broker that receives events (via the broker network) on behalf of the client and delivers received events to the client via some wireless network link. Intuitively, we should choose the nearest broker as the client's home broker, thereby optimizing the delay between the client and its home broker. In ad-hoc mobile environments, therefore, home brokers must be chosen repeatedly, whenever nodes substantially change their locations. The procedure used by opportunistic overlays may be described as follows. Each client periodically (or in response to changes indicated by the underlying physical network protocol [6]) executes a protocol that searches for the broker nearest its current location. If the nearest broker is not its home broker, it notifies the current home broker of its discovery. Upon receiving this news from its client, the home broker selects new home broker based on average path length between producers and the client and the cpu load of candidate brokers. If the home broker must be changed, the modulator relocation protocol is performed.

An interesting aspect of our approach is overload control, which is particularly important because end-to-end event delay from a producer to a consumer depends not only on the length of the network path, but also on event processing times at brokers. Processing times are determined by how fast modulators can be executed on home brokers which in turn depends on the home brokers' loads and capabilities. In mobile ad-hoc networks, with clients changing locations, broker loads are subject to substantial runtime variation. One reason would be the sudden arrival of large numbers of local users, exemplified by many mobile units converging at a meeting. Another reason is the use of complex modulators by 'thin' clients, such as modulators that implement the flexible data transcoding required by such clients [36, 31]. In fact, the processing time of a modulator on moderately to highly loaded brokers can exceed network delays by an order of magnitude. The protocol followed to change home brokers can be summarized as follows.

Step 1: Nearest Broker Search. Each client searches the broker nearest its location, periodically, using the same algorithm in broker neighbor update as described in Section 2.3. When a client finds the nearest broker that is not its current home broker, it shares with its current home broker the newly discovered broker along with its distance to that broker.

Step 2: Home Broker Selection. When a broker receives a nearest broker discovery message from its client, it executes the following home broker selection algorithm.

1. Check current home broker's (its own) and the discovered broker's CPU loads.
2. If one of the broker's CPU load exceeds the specified threshold, the other broker is selected as the client's new home broker.
3. If both brokers are overloaded, the broker with the lighter load will be used as the new home broker.
4. If neither broker is overloaded, calculate the average path from each producer to the client through each of two candidate brokers; then select the broker on the shortest path.
5. If the home broker must be changed, perform the modulator relocation protocol.

Step 3: Modulator Relocation. The relocation protocol relocates a client's modulator from its current home broker (source broker) to a new broker (destination broker), asks all producers' home brokers to compute paths to the new home broker, and switches event delivery from the old to the new paths. The following protocols guarantee event order, prevent event duplication or event loss, and ensure consistent event state. For applications that do not need such strict semantics, variations

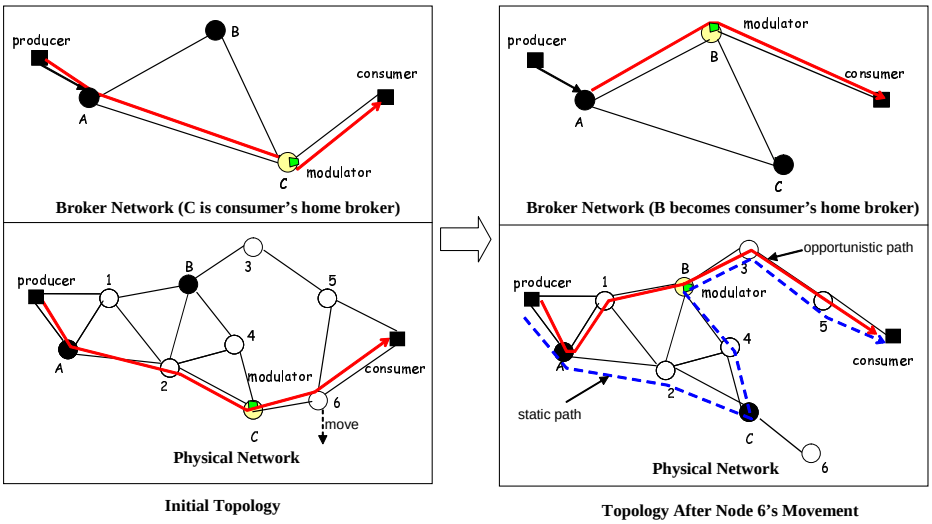

Fig. 4. An Example of Dynamic Home Broker Change

of the above protocols are possible. Lighter weight protocols used in our work loosen the requirements of event state consistency or modulator consistency or both.

1. The source broker initiates a handoff by sending a HANDOFF request to the destination broker.
2. Upon receiving the handoff request, the destination broker adds the mobile client to its consumer list and sends an ACK to the source broker.
3. After receiving the ACK from the destination broker, the source broker sends a DETOUR request, which includes the name of the destination broker, to all event producer.
4. Upon receiving the DETOUR request, each producer computes the shortest path from to the destination broker, and then atomically switches event delivery from old to new path via changes to its routing table; it then sends ACKS to the source and destination brokers.
5. The source broker receives events from each producer, applies the client's modulator to these events, and forwards them to the next broker, until it receives the ACK from the producer.
6. The destination broker buffers events from each producer after it receives the ACK.
7. After ACKs from all producer are received by the source broker, it sends a HANDOFF along with the current modulator to the destination broker and removes the modulator.
8. Upon receiving the HANDOFF, the destination broker applies the modulator received from the source broker to the buffered events and starts forwarding events to the client using the new path.

An example of dynamic home broker change is shown in Figure 4. A consumer originally receives events from broker C via the path producer→A→C→consumer corresponding to the physical network path producer→A→2→C→6→consumer. Broker C is the consumer's home broker. The consumer's modulator is also placed on C and executes there. At some point in time, with node 6 moving away from the consumer and node 5, during the consumer's nearest broker discovery period, it discovers that it is closer to B than C. When the consumer detects this, it sends a home broker change request along with B's information to C. C will choose B as the consumer's new home broker, since A→B→consumer is shorter than A→C→consumer under the new network topology. C then performs the modulator relocation protocol. After the change, the event delivery path from producer to consumer becomes producer→A→1→B→3→5→consumer. In contrast, without changing home brokers, the old overlay path producer→A→C→consumer corresponds to the physical network path producer→ A→2→C→4→B→3→5→consumer, which is 2 hops longer than the path used by opportunistic overlays.

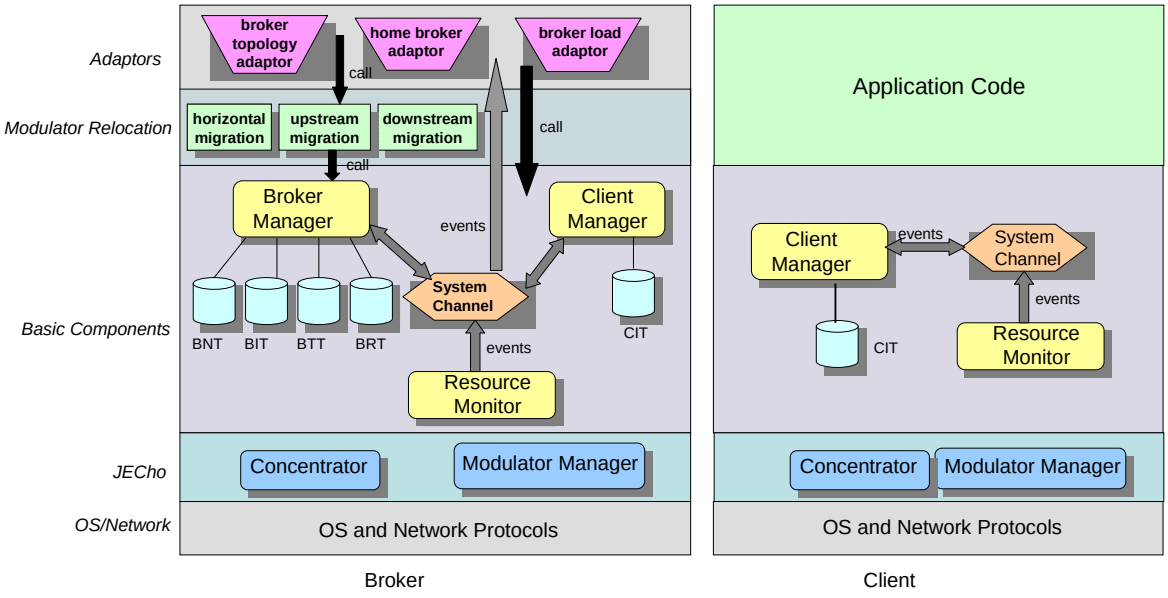

Fig. 5. Opportunistic Overlays Software Architecture

3 Software Architecture and Selected Implementation Detail

3.1 Overview of JECho

Opportunistic overlays are realized with the JECho distributed event system[36]. JECho implements a publish/subscribe communication paradigm, providing services to distributed, concurrently executing components via event channels. JECho's implementation is in pure Java, its group-cast communication layer is based on Java Sockets, and it runs with both standard and embedded JVMs. Using JECho's *modulators* [36, 34], individual event consumers can dynamically tailor event flows to their own needs, thereby adapting to runtime changes in component behaviors and needs and/or changes in platform resources. Modulators are implemented as Java objects, executed in a source's or broker's address space on behalf of clients. Client-defined customization with modulators, including event conversion or transformation, is performed prior to delivering the event to the consumer. Event conversion may reduce their sizes and hence reduce network traffic. Modulators can also be used for offloading computation from resource-constrained mobile devices.

As shown in in Figure 5, two core components in JECho are the concentrator and the modulator manager. A concentrator is a hub for all incoming/outgoing events. Each Java virtual machine(JVM) has one concentrator. In this context, a modulator manager provides an environment that permits the modulator to carry out computation with necessary access to local resources, and it provides facilities for adding, removing, and changing modulators and to provide information about selected modulator state (e.g., modulator execution time).

3.2 Overview of Software Architecture

Opportunistic overlays are implemented as depicted in Figure 5. The architecture is able to support flexible adaptation implementations and the architecture itself is easy to reconfigure and extend.

The basic component layer provides lower level functionalities of resource monitoring and broker information management necessary for implementing different adaptations. Event-driven adaptations are implemented by defining a set of actions in react to specific events received from basic components.

By using services provided by basic components, adaptation code focuses on high level protocol only without needing to handle lower level details. The interaction between basic component layer and adaptation layer uses a set of consistent program interfaces and system events. As a result, it is easy for different brokers to define different adaptations based on their capabilities and requirements. In fact, the implementation of an adaptation protocol within the current system is straightforward, as exemplified by the home broker change adaptation that has less than 50 lines Java code. It is also easy to reconfigure and extend the system with new adaptations, such as those needed to handle physical network partition. For future work, we are considering adding a policy layer that permits user to define high-level policies concerning the adaptations being carried out, somewhat like the ideas in autonomic computing presented in [20].

3.3 Basic Component Layer

The basic component layer layer is composed of resource monitor, broker manager, and client manager. Components in this layer provide the core functionality implementing the adaptation protocols described in Section 2. Each component in this layer defines a set of program interfaces for other components to access its services (i.e., a set of 'get' and 'set' functions). Each component notifies other components and high level protocols by sending events containing the relevant information to an internal 'system' channel. Other components receive this information by registering their interests about certain events.

Resource Monitor. The Resource Monitor collects, aggregates, processes, and delivers data about local resource availability and about its communication costs to other brokers. Local resource information includes CPU load, memory availability, and modulator execution time.

Broker Manager. The Broker Manager maintains four tables, which are the Broker Neighbor Table (BNT), the Broker Information Table (BIT), the Broker Topology Table (BTT), and the Broker Routing Table (BRT). The set of program interfaces provided by the Broker Manager to higher level protocols include functions for accessing and changing broker-related information, and operations that propagate its broker topology to neighboring brokers. The Broker Manager is also responsible for sending notifications to higher level components when it receives them.

Client Manager. The Client Manager maintains information about each client for which the broker is currently acting as home broker, including its name, IP address, physical location, as well as related path information (e.g., current routing path and communication overhead of the path). This information is stored in the Client Information Table (CIT). The data in the CIT is collected from mobile clients. When the Client Manager receives a nearest broker discovery message from a client, it sends that message to the higher level adaptation protocol.

3.4 Modulator Relocation Layer

On top of basic component layer are three modulator relocation operations which are horizontal relocation, upstream relocation and downstream relocation. Relocation operations perform the task of relocating a client's modulator from current broker to another broker, and changing event delivery path accordingly. Relocation operations provides necessary functionality for supporting home broker change and dynamic load balancing.

3.5 Adaptation Protocol Layer

The adaptation protocol layer implements a variety of protocols, including 'dynamic broker network construction' and 'dynamic home broker change'. Each such protocol is implemented with a Java object called an *adaptor*. An adaptor can register with the system event channel by specifying its

```
public class HomeBrokerAdaptor implements BrokerAdaptor {
  //subscribe to receive nearest broker discovery message
  public void subscribe( ) {
    registerToEvents(NEAREST_BROKER_DISCOVERY);
  }

  //adaptation code defined in resource event handler
  public void process(Object e) {
      BrokerDiscovery bd = (BrokerDiscovery)e.getContent( );
      client = bd.client;

       //control broker overload
            ...
       //select home broker based on network delay
       sourceBroker = ClientManager.getSourceBroker(client);
       delay1 = ClientManager.getLatency(client);
       delay2 = BrokerManger.getPath(sourceBroker, candidateBroker) + delay;
       if(delay2 < delay1)
            newHomeBroker = candidateBroker;
       else
            newHomeBroker = currentHomeBroker;

      //perform modulator relocation and path switch
      if(newHomeBroker != currentHomeBroker)
          RelocationAdaptor.migrate(currentHomeBroker,newHomeBroker,client);
  }
}
```

Fig. 6. Skeleton Code for Home Broker Adaptor

interests in certain events delivered by the Resource Monitor, Broker Manager, and Client Manager. For the broker topology adaptor, interesting events are a time event and a broker propagation event. The interesting event for the home broker adaptor is a broker discovery message received from a client. The code in the adaptor is implemented in the event handler method "process( )", which is invoked whenever an interesting event is received. This code implements changes, such as reconfiguring the broker network, changing the home broker, or rebuilding a routing table. Using adaptors and the services provided by basic components, system developers can create potentially complex adaptation policies.

Our prototype implementation has three adaptors: a broker topology construction adaptor, a home broker change adaptor and a broker load balancing adaptor. Each adaptor performs the task for which it is named. As an example, the skeleton code for the home broker change adaptor is shown in Figure 6. Home broker adaptor is activated when a broker receives a broker discovery message from its client. It then evaluates the network path to new broker and determines whether or not to change the home broker based on path length and the potential new broker's cpu load. If a change is necessary, the modulator relocation protocols is called. Network path length information is collected by sending queries to both the Broker and Client Managers. For simplicity, this code fragment assumes that there is only one producer. It is apparent that the implementation is a straightforward Java realization of the high-level protocol described in Section 2.4.

3.6 Client Component

The final element of opportunistic overlays are client-resident components that interact with the broker overlay. These include a Resource Monitor and Client Manager. A Client Resource Monitor performs a simpler task than its counterpart in a broker. It monitors the client's location and measures the distances to its home broker and other brokers. These items are reported to its Client Manager.

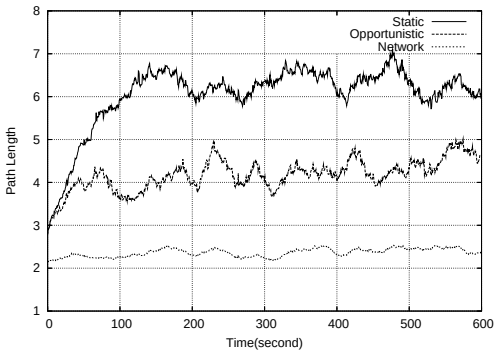

Fig. 7. Timeline of Path Length among Brokers

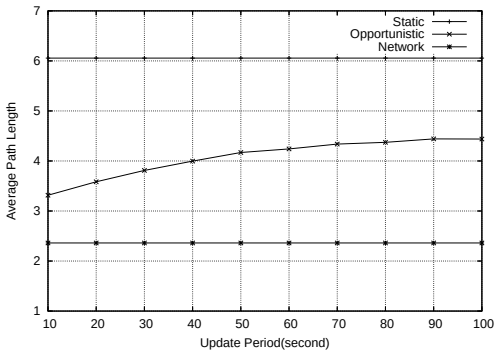

Fig. 8. Average Path Length among Brokers versus Update Period

Whenever client location changes, the Client Manager notifies its home broker's Client Manager, and it also periodically executes the nearest broker discovery protocol, as described in Section 2.3.

4 Performance Evaluation

4.1 Simulation

Simulation techniques are used to evaluate the opportunistic overlay approach under various wireless network configurations. In all experiments reported in this section, the network consists of 100 mobile nodes that randomly roam in a 1000 x 1000 meter square. The random waypoint mobility model [19] is used with a pause time of 10 seconds. The radio transmission range of each node is 250 meters. Each simulation spans 600 seconds of simulated time.

Currently, our simulation study is limited to high level overlay routing and no link layer details and physical layer characteristics are modeled. As a result, we have not studied aspect of the systems such as control overhead, or link contention. Thus a high routing packet load does not interfere as much with the data transmissions as it would in reality. Also, there are no transmission errors and delay associated with overlay routing packets(i.e., broker network topology propagation and broker neighbor discovery). Future work should address these limitations by constructing a more comprehensive simulator with a MAC layer model (e.g., IEEE 802.11 MAC). The point of this section's simulation results is to compare the relative performance of the opportunistic approach to overlay routing vs. static approaches. Results hold for moderately loaded networks, based on the following observations. First, the bandwidth usage implied to opportunistic overlay control messages is small, e.g., 2.4 Kbps per broker with an update period of 50 seconds. Hence, the additional contention at data-link layer caused by control packets (i.e., broker network topology propagation) is low. Second, the addition of delays and overheads at lower layers does not negate simulation results. Instead, essentially, such delays increase the broker network update period, thereby causing opportunistic overlays to react more slowly to changes in physical network topology. Fortunately, simulation results shown that the opportunistic approach is fairly robust, that is, performance does not decrease rapidly with increases in update periods from 10 seconds to 100 seconds. In fact, even with an update period of 100 seconds, the opportunistic approach can still use much shorter paths than the static approach. Therefore, the changes in path length that may be experienced if more precise MAC-layer models were used are likely to be small.

Performance of the Broker Network. The first set of experiments evaluate the average lengths of network paths across broker overlays with vs. without opportunistic overlay protocols. Measurements

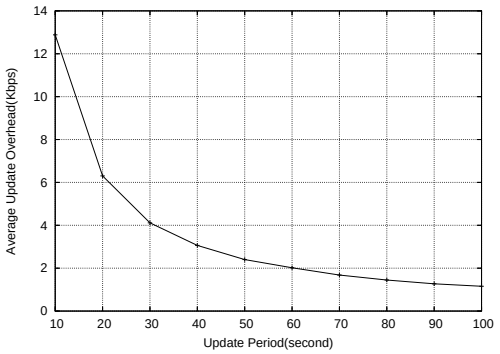

Fig. 9. Update Overhead versus Update Period

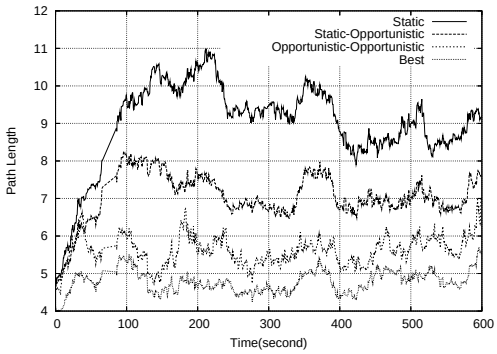

Fig. 10. Timeline of Average Path Length between Producers and Consumers

consider only brokers, not clients. 40 nodes among 100 nodes are randomly chosen as brokers. We vary the experimental configurations with different broker topology update intervals. The path length from broker B1 to broker B2 is the corresponding physical network distance of the shortest broker path between B1 and B2, which is computed based on B1's local knowledge of current broker topology. The average path length from each broker to all other brokers is computed and averaged over all brokers. In order to establish a basis for comparison, we also measure the average length of the physical network path between each pair of brokers. In addition, the overheads of broker topology update with different update intervals are evaluated.

Timeline of Path Length. Figure 7 shows how path length changes in a simulation with an update interval of 50 seconds and mobility speeds between 1 m/s and 20m/s. As shown in the figure, the opportunistic approach can deliver events more efficiently than the static approach at almost all time points. At the beginning, both the static and the opportunistic approaches have similar path lengths, since the initial broker network matches physical network topology. As time passes, the path lengths of the static approach increase rapidly because the initial broker topology cannot reflect changes in physical network topology caused by node mobility. Compared with the static approach's 6.06 hops and the network's 2.36 hops, the opportunistic approach has an average path length of only 4.17 hops.

Average Path Length versus Broker Update Period. Figure 8 shows the average path length versus the update interval, the latter varying from 10 to 100 seconds. As expected, with increased update intervals, path length increases since larger update intervals imply slower reactions to changes in physical network topology. However, as shown in the figure, the change in path length is not rapid with the increase of update periods, e.g. 4.43 hops with an interval of 100 seconds versus 3.32 hops with one of 10 seconds. Even with a relatively low update period of 100 seconds, the opportunistic approach still outperforms the static approach significantly, 4.43 hops vs. 6.06 hops.

Update Overhead versus Broker Update Period. The overheads of broker network updates are shown in Figure 9. Overhead is computed as the average bandwidth requirement of each broker for propagating its broker topology knowledge to its neighbors, the argument being that network resources tend to be scarce in pervasive systems. As shown in the figure, a total of 2.4 Kbps bandwidth is used with an update period of 50 seconds. This constitutes moderate bandwidth usage in modern network infrastructures.

Performance of Event Delivery between Mobile Clients. Most relevant to our work, of course, is the end-to-end performance experienced by end users, i.e., clients. In the following experiments, we

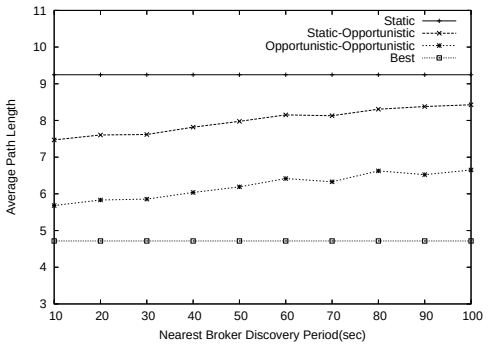

Fig. 11. Average Path Length between Producers and Consumers versus Nearest Broker Discovery Period

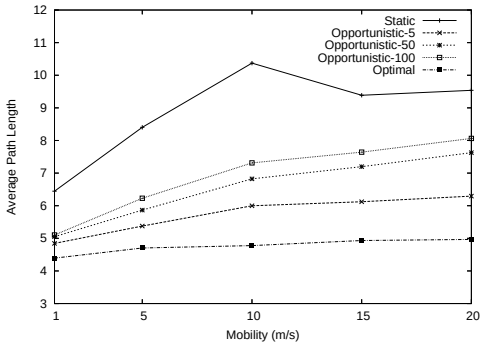

Fig. 12. Average Path Length between Producers and Consumers versus Mobility

randomly choose 20 brokers, 10 event producers, and 60 event consumers from 100 mobiles nodes. In this set of experiments, the broker network update interval is fixed at 20 seconds, and we vary mobile clients home broker discovery periods as well as mobility speed. Four different approaches are evaluated in terms of the resulting average path lengths between each pair of producer and receiver: (1) the static approach changes neither the broker network topology nor the home broker of mobile clients; (2) the static-opportunistic approach change mobile client's home broker only; (3) the opportunistic-opportunistic changes both the broker network topology and mobile client's home broker; and (4) the best approach keeps updating the broker network whenever the physical network changes and calculates shortest broker paths based on up-to-date physical network topology data. Although the best approach is not practical, we have included it to establish a basis for comparison.

Timeline of Path Length. Figure 10 shows the performance results of a simulation with a home broker discovery period of 10 seconds. We can see that the opportunistic-opportunistic approach performs best among three realistic approaches, shortening the delivery paths up to 5 hops compared with the static approach. Even the static-opportunistic approach can improve event delivery significantly compared with the static approach.

Average Path Length versus Nearest Broker Discovery Period. The results of path length versus home broker discovery period are reported in Figure 11. With increased closest broker discovery periods, path length increases slightly. As discussed in Section 2, the mobile client can find the nearest broker either by querying its routing table or by using an expanding ring protocol. Either way, the cost is small compared with the overheads of broker update operation. More frequent closest broker discovery results in more frequent home broker changes, hence more frequent modulator relocations. Additional simulation results which is not reported here due to space limitation demonstrate the fact that the modulator relocation cost is much smaller than the overhead of broker topology exchange. The interested readers can find the simulation results in [5].

Average Path Length versus Mobility. The average path length at different mobility speed is shown in Figure 12. The figure shows that more frequent nearest broker discovery achieves shorter event delivery paths. In particular, when the nodes move at a very fast speed, increasing the discovery frequency can improve the performance significantly.

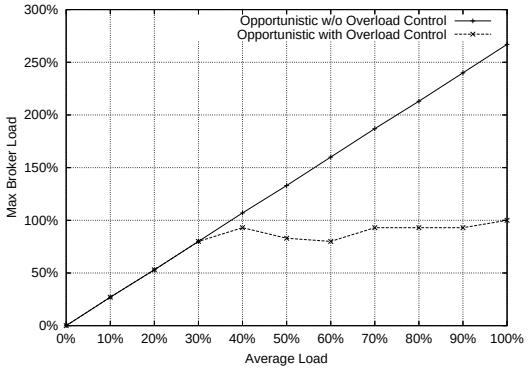

Fig. 13. Maximum Broker Load versus Average System Load

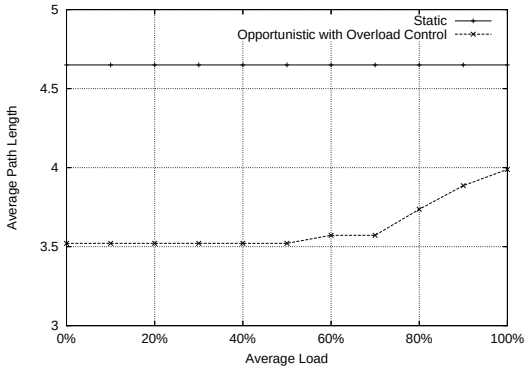

Fig. 14. Path Length versus System Load

4.2 System Emulation

In order to evaluate the effects of load balancing, we have conducted a set of experiments on an ad-hoc wireless network emulator. The mobility emulator runs on a Linux cluster of 20 nodes with MobiEmu [33] running on each node. The cluster network is a gigabit Ethernet switch. MobiEmu is a software platform for testing and analyzing ad-hoc network protocols and applications. With control software running on each node, MobiEmu mimics dynamic connectivity among nodes by dynamically installing or removing packet filters for specific MAC addresses. Since we focus on load balancing in this set of experiments, we use the 'best-case' ad-hoc routing provided by MobiEmu software. These protocols always deliver packets via shortest network paths. More detail about this system appears in [33].

The emulated mobile network consists of 20 mobile nodes, of which 5 nodes are event brokers, 5 are event producers and 15 are event consumers (5 brokers reside at event receiver nodes). In our experiments, mobile nodes move in a space of 750m x 500m and use random waypoint mobility with a pause time of 30 seconds and speeds between 1m/s and 20m/s. Due to the relatively small network size, the broker update period is set to 5 seconds and the nearest broker discovery period to 1 second. We vary the average load of the broker network and measure the maximum load during execution.

Results appear in Figure 13. As shown in the figure, when broker load is relatively light (i.e., less than 30%), there are no overloaded brokers and both approaches behave the same, where the nearest broker to a mobile client is always chosen as the client's home broker. With increased system load, without overload control, some brokers become overloaded. The load balancing algorithm ameliorates this problem, because a client's home broker will not be moved to an overloaded node, even if that node is closer than the old one. The positive outcomes of load management reported in these measurements are moderate, of course, since in random waypoint mobility, nodes move independently. Load balancing is more important and will have more significant effects when nodes move in groups, as exemplified by conference participants moving from one presentation venue to another, for instance.

Figure 14 depicts path length versus system load. When system load is less than 70%, the opportunistic approach always chooses the nearest broker as home broker, and the delivery path can be more than 1 hop shorter than in static approach. With increased system load, load balancing selects broker with lighter loads on relatively longer paths, resulting in increased path lengths. However, the opportunistic approach continues to outperform the static approach even when system load reaches 100%.

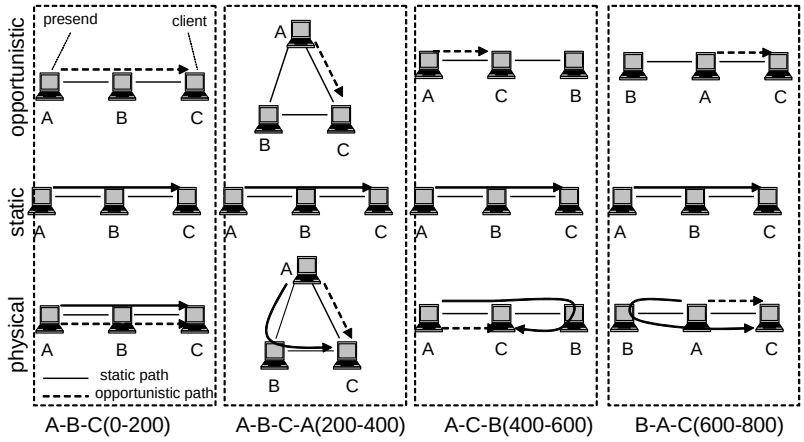

Fig. 15. Experiment Configuration

4.3 Testbed

The last set of experiments demonstrate the practical utility of our approach, by running a sample application on an actual wireless testbed. The testbed consists of 3 laptops A, B and C. A is Mobile Pentium 1GHz with 512M memory, B is a 1.7GHz machine with 512M memory and C is a 700MHz machine with 128M memory. Wireless connectivity is provided by Orinoco 802.11b cards. These cards are set to ad-hoc mode on channel 8. No WEP encryption is used. All three laptops use the UoB JAODV version 0.2, an AODV implementation in Java [18]. In order to simulate network connectivity changes, we dynamically set filters at the MAC layer. In a network consisting of 3 nodes A,B and C, there are four possible network topologies without network partitions: A-B-C, A-B-C-A , A-C-B and B-A-C.

The experiments being performed run a flood watch application on the testbed, comparing the event delivery latency of the opportunistic with the static approach using the 4 different network topologies. The application consists of two programs and works as follows. PreSend reads precipitation data from a file and places normalized precipitation data on event channel. A client program subscribes to the channel and provides a modulator that calculates water depth from precipitation data and terrain topology data using a runoff model. The client is typically interested in flood information in some specific area, which can be defined by a two-dimensional bounding box. Flood data that is outside the bounding box will be filtered (i.e., removed) before data is sent to the channel. The output of full-size flood data is a double array of 100 x 100.

We envision a scenario in which this application is used in a flood rescue action. In a flood disaster area, a rescue team is equipped with several mobile vehicles with relatively powerful computers (e.g. desktops) installed on each of them and each team member or group has a handheld (IPAQ) or lightweight laptop. All computers have communication hardware (e.g., 802.11b wireless cards) and software (e.g., the AODV protocol) so that they can communicate with each other via the ad-hoc wireless network. Brokers reside at computers on mobile vehicles and form a virtual network. They receive real weather data either from a weather center via a satellite connection or from sensors distributed across the area. They send collected data to an event channel. The client program running on each team member's mobile computer connects to a broker and receives the data of interest. Different people may be interested in receiving different data by running different client programs or by using the same client program with different modulators.

In our experiment, PreSend is running on laptop A and the client program runs on laptop C. Broker programs run on both A and B. The physical network topology changes every 200 time units. Experiment configuration is shown in Figure 15. The initial physical network topology is A-B-C.

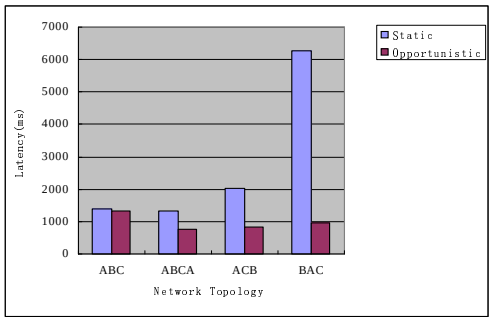

Fig. 16. Average Latency (100% data)

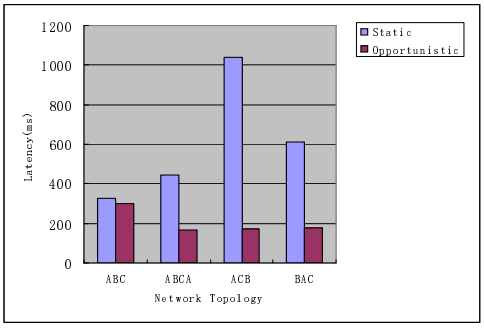

Fig. 17. Average Latency (10% data)

B is C's home broker. Both approaches use the same event delivery path A→B→C. After 200 time units, the network topology changes to A-B-C-A, so that the opportunistic approach chooses A as C's home broker and relocates its modulator from B to A, hence resulting the shorter delivery path A→C. During the time interval of 400 to 600, the network topology is A-C-B, where the opportunistic overlay approach still uses A→C as delivery path corresponding to the same physical network path A→C. The corresponding physical network path of A→B→C used by static approach becomes A→C→B→C. In the final interval, the network topology changes to B-A-C, where the opportunistic approach still uses the same path A→C, and the static approach's path A→B→C now corresponds to the physical network path A→B→A→C, with overlap at A.

Figure 16 shows the latency comparisons with four topologies when the client is interested in all data. As shown in the figure, the opportunistic approach can deliver data up to 6 times faster than the static approach. The static approach has its worst performance in the configuration of B-A-C (6000ms compared with to the opportunistic approach's 1000ms), where data delivery following the path A→B→A→C results not only in a longer path but also in additional network bandwidth usage at A, since at the physical layer, every event is actually delivered twice at A.

When a client is interested in only 10% of the area data, the opportunistic approach coupled with its use of modulators shows additional improvements. Results depicted in Figure 17 show that by relocating C's modulator from B to A, the opportunistic approach not only delivers data following a shorter path but also delivers less data, hence improving the application's performance significantly: the average latency is less than 200ms under all network configurations. Note that in this scenario, the static approach performs worse under network topology A-C-B than B-A-C because only 10% of the data reaches A for the second time in the latter configuration, while all data reaches C in the former one. The fact that machine C is less powerful than A may also contribute to the observed difference in performance.

The key result of the testbed experiments presented here is that it is important to dynamically adjust the middleware overlays used in pervasive systems. The opportunistic overlay approach described and evaluated in our research is one method for runtime overlay management and by using it, significant performance improvements can be attained compared to non-adaptive approaches. However, the small scale of the testbed used in these experiments limits the generality of the results presented here. First, with larger numbers of machines, there will likely be interference and congestion effects. These will lead to increased costs and delays for managing overlays and therefore, reduce the absolute performance benefits of the opportunistic vs. static approaches being compared. Second, the testbed uses MAC address filters to simulate node mobility, but in actual adhoc networks with mobile nodes there will be additional communications compared to the filter-based communications in the testbed (since all nodes in the adhoc networked system still communicate with each other). This will result in additional congestion in actual systems vs. the testbed.

5 Related Work

5.1 Content-based Event Systems

Publish/subscribe systems [27, 3, 26, 12, 10, 36] have been investigated for many years, but most implementations have focused on systems where nodes don't move, broker networks remain fixed, and broker network topologies are defined at deployment time. As a result, their fixed event dissemination structures make them unsuitable for applications in mobile environments where physical network topology and node locations change continuously. In addition, most publish/subscribe systems perform event filtering with predicate-based subscriptions; they do not support the general event processing needed for the complex data conversions occurring in multimedia, business, or scientific applications. Opportunistic overlays are realized with the JECho pub/sub infrastructure [36, 35, 34]. JECho generalizes the capabilities of other event systems, by using consumer-provided functions, termed event modulators [35]. The intent is to address the severe resource limitations existing in many mobile and embedded systems, by permitting event consumers to deploy application-specific functions that manipulate event content into event sources and/or brokers, so as to precisely meet their current needs, and to avoid needless data transfers. Generic function-based subscription makes the opportunistic overlay system more feasible for developing applications in pervasive systems and mobile environments.

Previous research on event-based middleware for wireless networks has addressed applications in which mobile nodes make use of the wireless network to connect to a fixed network infrastructure [9, 28, 11, 1, 6]. JEDI offers moveOut and moveIn operations that enable subscribers to disconnect and reconnect at a different network, requiring applications to explicitly call those operations [9]. Similarly, the mobility extension of Siena requires explicit request by applications [1]. Elvin supports disconnection and reconnection by using a central caching proxies [28]. [11] extends Rebeca to support mobile and location-dependent applications by rebinding a client to different brokers transparently and offering a client a fine-grained control over notification delivery in the form of location-dependent filters, but it does not not target ad hoc wireless networks. [16] discusses general ideas about how to adapt a publish/subscribe system to mobile environments, but no design and implementation is described. In our previous work [6], we present an approach to support the dynamic change of home brokers in infrastructure-based wireless networks, where only the last connection between broker and client is a wireless link. The opportunistic overlays presented in this paper differ from all of the systems mentioned above in that they are designed for mobile ad hoc networks, support dynamic reconfiguration of event event dissemination structure and offer behaviors transparent to applications.

Picco et al. [24] present an algorithm for topological reconfiguration in content-based publish/subscribe due to changes in underlying connectivity. Compared with the opportunistic overlays, the reconfiguration in [24] is simpler, involving only a link removal or insertion. It has not given any detail on how to apply the proposed approach to handle changes in mobile environments. In addition, the approach assumes a tree-based topology between dispatchers, which makes it hard to achieve robustness since a single link failure partitions the tree. Baldoni et al. [30] propose another approach for the dynamic reconfiguration of the broker network. The reconfiguration in [30] aims at placing close to each other brokers that manage similar subscriptions while the reconfiguration in the opportunistic overlays is based on the nodes' physical locations and the underlying physical network topology. Similar to the approach in [24], the topology used in [30] must be maintained acyclic while the opportunistic overlay approach supports general broker overlay topologies. Hermes [25] supports limited reconfiguration by providing a repair mechanisms in case of brokers' faults.

5.2 Content-based Event Systems in Mobile Ad-hoc Networks

Steam [23] is an event-based middleware service designed for ad-hoc wireless networks. It targets application scenarios where nodes are more likely to interact when they are in close proximity to each other. Our approach aims to support general event delivery in mobile ad-hoc networks. Further, Steam uses an implicit event model without intermediate broker nodes.

Huang et al. [17] present a distributed protocol to construct optimized publish/subscribe trees in ad-hoc wireless networks [17]. Their algorithm builds multicast trees directly on top of lower level radio broadcast primitives, while we use an overlay broker network approach and depend on the underlying network infrastructure to provide basic network connectivity. Another difference is that their approach assumes a relatively stable environment with occasional reconfigurations followed by periods of stability. Opportunistic overlays do not make that assumption, and they can actually handle high levels of mobility as shown by our experimental results.

Yoneki et al. [32] propose a content-based publish/subscribe system for MANETs, which integrates an extended ODMRP (On-Demand Multicast Routing Protocol [21]) and content-based subscriptions. Similar to [17], ODMRP-PUB/SUB delivers events by creating multicast groups. The difference is that ODMRP-PUB/SUB uses a mesh-based approach instead of the tree-based one used in [17]. Since a consumer's subscription is a general function applied to events in our system, the approach of combining a multicast protocol and subscription aggregation/match is not readily applicable in our case. Further, ODMRP-PUB/SUB focuses on the routing between brokers and does not address the issue of delivery from brokers to producers/consumers. The purpose of ODMRP- PUB/SUB is to optimize network throughput while our opportunistic method focuses on providing timely event delivery.

5.3 Overlay Multicast Protocols in Mobile Ad-Hoc Networks

AMRoute [22] and PAST-DM [13] are two ad-hoc multicast protocols that use the overlay approach. AMRoute uses a static virtual mesh and has low efficiency due to the increasing mismatch between virtual topology and physical network topology, as shown in [13]. PAST-DM addresses the efficiency problem by dynamically adapting the virtual topology to changes in the physical network. That brokers need to process events distinguishes our system from multicast systems where nodes perform data routing and participate as relays. Although opportunistic overlay approach uses a similar dynamic virtual overlay construction technique as PAST-DM [13], the dynamic routing path in opportunistic approach involves not only path changes in event routing, but also subscription code relocation, which makes the existing dynamic delivery technique in past-dm is not readily applicable to our system. In addition, the processing of events will consume a broker's computational resources, which implies that brokers' computational capabilities need to be taken into account. Finally, in PAST-DM, all member nodes are considered to be equivalent peers and participate in overlay routing. In contrast, opportunistic overlays conceptually divide nodes into brokers which are organized into an overlay broker network, and clients(producers/consumers) which send/receive events via the broker network. This model is more suitable for content-based routing since overlay routing through 'thin' nodes with very limited resources will present burden on such nodes and may result in inefficiency of content delivery in mobile systems. Dynamic reconfiguration using by opportunistic overlays adapts both the overlay broker network and the connections between brokers and clients.

6 Conclusions and Future Work

This paper presents an approach to optimizing content-based event delivery in mobile ad-hoc networks. In response to changes in physical network topology and to node mobility, the opportunistic

overlay approach dynamically changes broker network topology, clients' assignments to brokers, and event delivery paths, with the goal of optimizing end-to-end delays in event delivery. Opportunistic overlays are prototyped with the JECho pub/sub system [36]. Comprehensive performance evaluations are performed via simulation, emulation, and with representative applications on a physical testbed. Experimental results for mobile ad hoc networks demonstrate that the opportunistic overlay approach can significantly improve event delivery delays compared to static approaches, even with high levels of mobility. Results also show that the overheads of dynamic adaptation are moderate. Using a flood watch application and a wireless testbed, the opportunistic overlay approach is practically applicable in an actual ad hoc wireless network.

Future work should address some deficiencies of our current implementation, as well as generalize upon the basic concept of opportunistic overlays. First, our current implementation assumes a reliable network environment and therefore does not consider dynamic disconnection, reconnection and network partition. Future work will add application-specific failure recovery to broker overlays. Second, we will extend the opportunistic approach to optimize performance metrics other than end-to-end latency, including network bandwidth and power usage. We may also explore optimizing multi-dimensional performance metrics. Third, the location of a modulator has an impact on performance. For example, for a 'contract' modulator which typically reduces event size or filters events, placing it at a broker closer to a source broker can reduce network overheads. Toward this end, will investigate the dynamic optimization of modulator placement based on modulator semantics, network condition and broker state. A final topic of interest to us is the performance study by using a more comprehensive simulator with a MAC layer model (e.g., IEEE 802.11 MAC).

References

1. M. Caporuscio, P. Inverardi, and P. Pelliccione. Formal analysis of clients mobility in the siena publish/subscribe middleware. Technical report, Department of Computer Science, University of L'Aquila, Oct. 2002.
2. A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Achieving scalability and expressiveness in an internet-scale event notification service. In *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing(PODC 2000)*, pages 219–227, Portland, Oregon, July 2000.
3. A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, 2001.
4. T.-W. Chen and M. Gerla. Global state routing: A new routing scheme for ad-hoc wireless networks. In *Proceedings of IEEE ICC'98*, june 1998.
5. Y. Chen. *Opportunistic Overlays: Efficient Content Delivery in Mobile Environments*. PhD thesis, Georgia Institute of Technology, 2005.
6. Y. Chen, K. Schwan, and D. Zhou. Opportunistic channels: Mobility-aware event delivery. In *ACM/IFIP/USENIX International Middleware Conference (Middleware 2003)*, pages 182–201, 2003.
7. G. Cugola and H.-A. Jacobsen. Using Publish/Subscribe Middleware for Mobile Systems. *ACM SIGMOBILE Mobile Computing and Communications Review*, 6(4):25–33, 2002.
8. G. Cugola, E. D. Nitto, and A. Fuggetta. The "jedi" event-based infrastructure and its application to the development of the opss wfms. In *IEEE Transactions on Software Engineering in 2001*, 2001.
9. G. Cugola, E. D. Nitto, and G. P. Picco. Content-based dispatching in a mobile environment. In *Proceedings of WSDAAL 2000*, 2000.
10. G. Eisenhauer, F. E. Bustamante, and K. Schwan. Event services in high performance systems. *Cluster Computing*, 4(3):243–252, 2001.
11. L. Fiege, F. C. Gärtner, O. Kasten, and A. Zeidler. Supporting mobility in content-based publish/subscribe middleware. In *ACM/IFIP/USENIX International Middleware Conference (Middleware 2003)*, pages 103–122, 2003.
12. L. Fiege, G. Mühl, and F. C. Gärtner. A modular approach to build structured event-based systems. In *Proceedings of the 2002 ACM Symposium on Applied Computing (SAC'02)*, pages 385–392, 2002.

13. C. Gui and P. Mohapatra. Efficient overlay multicast for mobile ad hoc networks. In *Proceedings of IEEE Wireless Communications and Networking Conference*, March 2003.
14. M. Hauspie, D. Simplot, and J. Carle. Partition detection in mobile ad-hoc networks. In *Proceedings of 2nd IFIP Mediterranean Ad Hoc Networking Workshop (MED-HOC-NET 2003*, 2003.
15. Y. Huang and H. Garcia-Molina. Publish/subscribe in a mobile environment. In *2nd ACM International Workshop on Data Engineering for Wireless and Mobile Access (MobiDE'01)*, Santa Barbara, California, USA, 2001.
16. Y. Huang and H. Garcia-Molina. Publish/subscribe in a mobile environment. In *Proceedings of the 2nd ACM International Workshop on Data Engineering for Wireless and Mobile Access (MobiDE'01)*, pages 27–34, 2001.
17. Y. Huang and H. Garcia-Molina. Publish/subscribe tree construction in wireless ad-hoc networks. In *Proceedings of the 4th International Conference on Mobile Data Management(MDM 2003)*, pages 122–140, 2003.
18. Uob-jadhoc aodv implementation, rfc 3561. http://www.aodv.org/, 2004.
19. D. B. Johnson and D. A. Maltz. Dynamic source routing in ad hoc wireless networks. *Mobile Computing*, 353, 1996.
20. J. O. Kephart and D. M. Chess. The vision of autonomic computing. *IEEE Computer Magazine*, Janurary 2003.
21. S.-J. Lee, W. Su, and M. Gerla. On-demand multicast routing protocol in multihop wireless mobile networks. *MONET*, 7(6):441–453, 2002.
22. M. Liu, R. R. Talpade, and A. McAuley. AMRoute: Adhoc Multicast Routing Protocol. Technical Report 99, The Institute for Systems Research, University of Maryland, 1999.
23. R. Meier and V. Cahill. Steam: Event-based middleware for wireless ad hoc networks. In *In Proceedings of the 1st International Workshop on Distributed Event-Based Systems (DEBS'02)*, 2002.
24. G. P. Picco, G. Cugola, and A. L. Murphy. Efficient content-based event dispatching in the presence of topological reconfiguration. In *Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCS 03)*, pages 234–243, 2003.
25. P. R. Pietzuch. *Hermes: A Scalable Event-Based Middleware*. PhD thesis, University of Cambridge, February 2004.
26. B. Segall and D. Arnold. Elvin has left the building: A publish/subscribe notification service with quenching. In *Proceedings of A UUG97*, September 1997.
27. R. Strom, G. Banavar, and et.al. Gryphon: An information flow based approach to message brokering. Technical report, IBM TJ Watson Research Center, 1998.
28. P. Sutton, R. Arkins, and B. Segall. Supporting disconnectedness - transparent information delivery for mobile and invisible computing. In *CCGrid 2001 IEEE International Symposium on Cluster Computing and the Grid*, May 2001.
29. TIBCO Software INC. *TIB/Rendezvous*. http://www.tibco.com/products/rv/index.html.
30. A. Virgillito, R. Beraldi, and R. Baldoni. On event routing in content-based publish/subscribe through dynamic networks. In *Proceedings of the Ninth IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS 2003)*, pages 322–328. IEEE, 2003.
31. M. Wolenetz, R. Kumar, J. Shin, and U. Ramachandran. Middleware guidelines for future sensor networks. In *Proceedings of 1st workshop broadband advanced sensor networks(BASENETS 2004)*, October 2004.
32. E. Yoneki and J. Bacon. An adaptive approach to content-based subscription in mobile ad hoc networks. In *Proceedings of The First International Workshop on Mobile Peer-to-Peer Computing (MP2P'04)*, pages 92–97, Mar 2004.
33. Y. Zhang and W. Li. An integrated environment for testing mobile ad-hoc networks. In *Proceedings of the Third ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'02)*, June 2002.
34. D. Zhou. *JECho - An Efficient, Customizable, Adaptable Distributed Event System*. PhD thesis, Georgia Institute of Technology, 2002.
35. D. Zhou, S. Pande, and K. Schwan. Method partitioning - runtime customization of pervasive programs without design-time application knowledge. In *Proceedings of ICDCS 2003*, pages 610–619, 2003.
36. D. Zhou, K. Schwan, G. Eisenhauer, and Y. Chen. Supporting distributed high performance application with java event channels. In *Proceedings of the 2001 International Parallel and Distributed Processing Symposium (IPDPS 2001)*, April 2001.