

TypeCast: Type-Based Routing in Wireless Ad-hoc Networks

Jinsong Lin

*Department of Computer Science
University of California, Los Angeles
jinsong@cs.ucla.edu*

Thomas Phan*

*IBM Almaden Research Center
phantom@us.ibm.com*

Rajive Bagrodia

*Department of Computer Science
University of California, Los Angeles
rajive@cs.ucla.edu*

Abstract

Type-based communication is proposed as an effective paradigm to enable group communication in wireless ad-hoc networks (MANETs). In this paradigm, type is used as the fundamental construct for addressing and routing messages. Type hierarchies are used to dynamically control group size; and object-oriented principles such as subtyping and multiple inheritance are utilized to construct new groups from existing ones. We present the design of TypeCast, a routing protocol that directly supports type-based communication. TypeCast leverages efficiency and mobility management provided by MANET multicast protocols and extends them by adding a Bloom filter-based type encoding and routing mechanism. TypeCast is fully decentralized and supports subtyping and type-composition. We implement TypeCast on top of ODMRP and conduct a detailed performance and scalability study of TypeCast through simulation. The results show that TypeCast demonstrates good resiliency to mobility and group size. When the number of types in the network increases, TypeCast achieves good scalability thanks to type aggregation provided by Bloom filters.

1. Introduction

Mobile ad-hoc networks (MANETs) are envisioned as new platforms for deploying applications in diverse areas such as emergency response, homeland security, and classrooms where a group of people or devices conduct collaborative work in a spontaneous fashion without network infrastructure. These ad-hoc applications typically have the following characteristics:

- The collaboration groups are dynamically formed, and members of the groups do not necessarily know each other's personal identity or network addresses. They address others by high level application properties, such as the role(s) of the participants (e.g. student vs. teacher), or types of service the participants provide.
- These high level application properties usually form hierarchical structures to reflect social and organizational relationship, which in turn constrain informa-

tion flow. For instance, in a dynamically-formed military squad consisting of personnel with different ranks (e.g. Private, Sergeant, Lieutenant, or Captain), communication flow must be properly directed so that lower ranking personnel should not receive messages meant for higher ranking officers while higher ranking personnel should be able to receive messages meant for lower ranks [1].

- In dynamically-formed groups, communication patterns are more often one-to-many, many-to-one, or many-to-many rather than simply one-to-one.

From the perspective of programming languages and distributed systems, each entity in the communication group can be modeled as a distributed object which can receive messages and provide services. Further, the role or characteristics of a distributed object can be abstracted as its type, which we use to drive our notion of type-based communication. In this paradigm, messages are addressed to a type rather than a specific object: the message will be routed to all the objects of the target type(s), and the operation specified in the message will be executed by receivers.

Type-based communication provides numerous benefits for ad-hoc applications: First, it requires senders and receivers to maintain only application-layer knowledge of each other. Communication can be initiated as long as one knows the types of the intended targets. The complexity of routing the messages to the matching objects is handled by the underlying infrastructure and completely hidden from the applications. Second, complex social and organization structure can be naturally modeled as type hierarchies, which provide rich expressivity for the applications to specify the communication targets. Third, object-oriented principles, such as subtyping and multiple inheritance, enable system designers to build distributed ad-hoc applications in an open and evolvable way by incrementally extending the type systems.

Consider a disaster relief system where workers from different organizations dynamically form an ad-hoc group to exchange real time disaster information and coordinate relief efforts using the wireless devices they carry. To support such a scenario, we can build a type hierarchy (shown in Figure 1) to model the relationship among group members. With type-based com-

* This work was performed while the author was at UCLA

munication, applications running on the devices can directly address receivers by their types - such as *PoliceOfficer* or *FirstReponseWorker* - without knowing any other detailed information about the receivers. This scheme also enables applications to dynamically control the scope of the communication by targeting types at different levels in the hierarchy. Specifying a target type at an upper layer of the hierarchy broadens the scope of the communication; for instance, *EmergencyResponseWorker* includes all *PoliceOfficer*, *FireFighter* and *Paramedics*. On the other hand, specifying a target type at a lower tier reduces the scope of communication and allows applications to send more specialized messages via additional methods provided by subtypes.

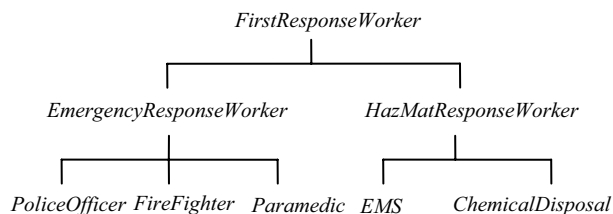


Figure 1: Type Hierarchy for Disaster Relief System

Furthermore, the type hierarchy is open and dynamic: new types can be “plugged-in” at any point in the existing hierarchy and will automatically inherit properties of their parent types. For instance, suppose that a new type, *BioterrorResponseWorker*, is added as a subtype of *FirstResponseWorker* in Fig. 1. As a result, any messages targeted to *FirstResponseWorker* should also be automatically delivered to instances of *BioTerrorResponseWorker*, even though the prior application is not aware of this new type. Lastly, the expressive power of type-based communication is further strengthened with type composition, which allows applications to send messages to objects with multiple types. For instance, assuming that a gender based hierarchy like Figure 2 has also been deployed, applications can subsequently send messages to all policewomen, which are of the type *Female* and *PoliceOfficer*.

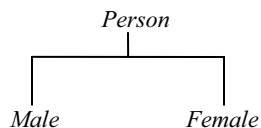


Figure 2: Type Hierarchy for Person

Compared with type-based communication, traditional distributed object paradigms such as RPC, CORBA, DCOM, and Jini [25] are not directly suited for mobile environments because they focus on supporting one-to-one communication pattern, typically rely on a centralized naming/lookup service to find target ob-

jects, and assume a relatively reliable network infrastructure.

The challenge of supporting type-based communication is providing efficient network support without losing the expressiveness of type; this efficiency requirement is especially difficult to achieve in MANETs where low overall network capacity, high link error, and frequent node mobility are the norm. In this paper, we introduce TypeCast, a network routing protocol that directly supports type-based communication in wireless ad-hoc networks. The protocol leverages routing efficiency and mobility management of existing MANET multicast protocols and embeds additional data structures in the routing layer to preserve the expressivity and intrinsic properties of types.

Our contributions in this paper are: (1) we suggest type-based communication as a fundamental paradigm to support ad-hoc applications; (2) we identify subtyping and type-composition as important principles for building open distributed collaborative applications; (3) our design of TypeCast bridges two research domains that have traditionally been separated - networking and programming languages - to support emerging ad-hoc applications with required efficiency; and (4) we present a detailed performance study of TypeCast with simulation, showing its resiliency against increasing node mobility, group size and number of types.

This paper is organized as follows: In section 2 we formally define type-based communication and identify its core requirements. We discuss the design of TypeCast in section 3 and presents the simulation result in section 4. Section 5 compares our work with previous research. We conclude in section 6.

2. Type-Based Communication

2.1. Background

Type is a fundamental concept in object-oriented languages. It represents the behavior of an object; for distributed objects, it specifies the set of messages that the object can process. The power of type lies in the following:

1. Type represents the interface that an object supports via its set of methods and hides the implementation of the object from the application. This abstraction allows the evolution of the service implementation without affecting the applications that depend on the service.
2. Type supports inheritance: new types can be created by inheriting from existing ones. An object of a subtype should have all the behavior and functionality of its supertypes in addition to its own. Subtyped objects can be used in any place that expects its supertypes;

this property is usually formally called *principle of substitutions* [16][17]. Subtyping allows developers to incrementally introduce new behavior or functionalities into existing systems.

3. New types can be created by using multiple inheritance or *mixin* [4] from two or more supertypes. A sophisticated type hierarchy can be created in this manner to mimic social structure or the physical world.
4. A programming language based on type allows type checking to ensure software correctness and to encourage modular and reusable software.

2.2. Requirements

In our proposed type-based communication paradigm, type is explicitly used as a first-class language construct to identify group membership and to specify the destination address of messages. Our purpose is to provide a more useful and expressive communication pattern to facilitate dynamic group collaboration in multi-hop wireless networks. To attain this goal, it is important that the semantics and properties of type be preserved in type-based communication in order to seamlessly incorporate object-oriented principles. Thus, we identified the following requirements of type-based communication:

1. In type-based communication, a message targeted to type T will be delivered to all reachable objects of type T in the network.
2. Type-based communication must adhere to the *principle of substitution*: a message addressed to type T must be delivered to all reachable objects of T and all its subtypes.

There are two important implications of this principle. First, if we think of type as the representation of a group of distributed objects sharing common behaviors, then type and subtype relationship can be transformed to group and subgroup relationships according to the substitution principle. Applications can dynamically shrink or grow the size of the collaboration group by sending messages to different levels of the type hierarchy.

Second, the significance of the substitution principle is more apparent in the context of open systems, where new types of objects can be incrementally developed and deployed. Since objects of new types will inherit the behavior of parent types, they will automatically participate in collaboration with existing members without requiring older members to be aware of the fact that the type hierarchy of the system has been extended.

3. Type-based communication should support type composition to allow applications to send messages to multiple types. Type composition allows applications

to dynamically combine existing type definitions in order to more accurately specify characteristics of group members. In the absence of type composition, developers are forced to predefine a type hierarchy that covers all the possible combinations of types to enable type-based communication in every scenario. This alternative not only results in type explosion but also is infeasible in open systems, where there is an unlimited number of ways of creating new types from existing ones.

Subtyping and type composition provide expressiveness in type-based communication and are the key properties that separate type-based communication from traditional IP multicast. In IP multicast, multicast groups are independent of each other, and messages sent to one group should not reach members in any other groups. This is in contrast to type-based communication, where types have a hierarchical relationship and messages sent to one type must be delivered to all members of its subtypes. Furthermore, in IP multicast, there is no equivalent operation as type-composition, by which messages can be sent to nodes that are members of the intersection of multiple groups.

4. Since type-based communication is deployed in MANETs, the network and OS should provide efficient support for handling mobility and link errors. Ideally the message should only be delivered to nodes hosting the objects with target type(s) following optimal paths based on some predefined performance matrix.

The above requirements for type-based communication call for strong support from the network. One naive approach is to directly use MANET multicast by mapping each type to a multicast group; sending a message to a type would be equivalent to sending a message to a multicast group. This approach has several problems. First, dynamically assigning a multicast address to a type and having all group members agree on it in a decentralized fashion is itself a challenging problem. Statically assigning multicast addresses to types is only feasible when the type structure is small and closed. Second, type composition and subtyping requirements imply that every message sent via type-based communication is in fact addressed to shared members of multiple multicast groups; IP multicast does not directly provide efficient support for such an addressing scheme. Third, since the number of multicast groups in the network must be at least the number of types in the network, the multicast group management cost (including control packets for setting up the routing paths and the memory for storing routing tables) will grow linearly as the number of active types in the network increases.

The overhead can be very expensive in ad-hoc wireless networks when the number of types increases.

In the following section, we introduce TypeCast, a network routing protocol that supports type based communication following the aforementioned characteristics.

3. TypeCast Design

Extensive research has been conducted on MANET multicast [3][8][9][13][21] in the past to support IP based group communication. Our strategy is to leverage existing multicast research to handle routing efficiency and node mobility while focusing on adding a novel mechanism on top of multicast protocols to support type-based routing. We achieve this goal without tying the routing scheme to any specific language.

3.1. Bloom filter

TypeCast requires efficient routing table management to support type-based routing. Our design uses Bloom filters for this purpose because (1) they allow us to easily conduct type aggregation when managing a routing table, (2) they can be used quickly and efficiently during routing process to determine whether two types have a subtype relationship, and (3) their space efficiency reduces the overhead of managing routing tables.

A Bloom filter [2] is a space-efficient representation of a set, supporting fast membership query and insertion. Given a set S consisting of n objects $\{X_1, X_2, \dots, X_n\}$ in domain U , we can convert it into an m -bit vector $B = \langle b_1 b_2 \dots b_m \rangle$ with k independent hash functions H_1, H_2, \dots, H_k ($H_i: U \rightarrow \{1, 2, \dots, m\}$) as follows:

$b_i = 1$ ($1 \leq i \leq m$) iff there exist p ($1 \leq p \leq k$) and q ($1 \leq q \leq n$) such that $H_p(X_q) = i$

The resulting bit vector B is called S 's Bloom filter representation.

To query whether an object O is a member of S using a Bloom filter, apply the k hash functions to O to obtain k integers $\{H_1(O), H_2(O), \dots, H_k(O)\}$. If one or more bits in B indexed by the k integers are 0s, then O is not in S . If all k bits are 1, then O may be in S with the false positive probability

$p_e = (1 - (1 - \frac{1}{m})^{kn})^k \approx (1 - e^{-kn/m})^k$. For fixed m and n , the smallest false positive rate $p_e \approx 0.6185^{m/n}$ when $k = \ln(m/n)$ [5].

Since the false positive rate of Bloom filter exponentially decreases when its size increases, it is a very space efficient data structure to represent a set. This unique property enables Bloom filters to be used in databases, distributed systems, and network applications

where space is a concern and the small false positive rate can be tolerated [5].

3.2. Bloom Filter Representation of Type

Syntactically, a type T can be viewed as a set of method definitions $\{M_1, M_2, \dots, M_k\}$, where each method is either directly defined in T or in one of its supertypes. By using hash functions that map a method definition to a number between 1 and m , we can convert the method set of T to a Bloom filter. This is called the Bloom filter representation of T ¹.

In this paper, we will use the following conventions and terminology:

- Symbols $T, T_1, T_2 \dots$ are used to represent types. We also use $\text{type}(o)$ to indicate the type of object o .
- We use $<$ to indicate a subtype relationship. $T_1 < T_2$ means T_1 is a subtype of T_2 . The subtype relationship is both reflective (i.e. $T < T$) and transitive (i.e. if $T_1 < T_2$ and $T_2 < T_3$, then $T_1 < T_3$).
- We use $T_1 + T_2$ to represent the composition of T_1 and T_2 . The method set of $T_1 + T_2$ is the union of the method set of T_1 and the method set of T_2 .
- The Bloom filter representation of T is written as $BF(T)$. We also use the symbols V, V_1, V_2 , etc. to indicate the bit vector of a Bloom filter, and $V(i)$ as the i th bit of V .
- $V_1 + V_2$ is the addition of V_1 and V_2 , which is defined as the bitwise-or of Bloom filters V_1 and V_2 . \sum represents the summation of multiple bloom filters.
- We define a "subfilter" relationship. A Bloom filter V_1 is a subfilter of another Bloom filter V_2 iff $\forall i, V_2(i) = 1 \Rightarrow V_1(i) = 1$. We use the symbol $<$ to represent the subfilter relationship. This relationship is also reflective and transitive.

The following two properties provide a foundation for the TypeCast design and can be easily proven from the definition of Bloom filter:

Property I: If $T_1 < T_2$, then $BF(T_1) < BF(T_2)$.

Property II: $BF(T_1 + T_2 + \dots + T_n) = BF(T_1) + BF(T_2) + \dots + BF(T_n)$

Property I asserts that the Bloom filter of a type T is a subfilter of the Bloom filter of any of T 's super types. This property can be used to test whether two types T_1 and T_2 have a subtype relationship by comparing $BF(T_1)$ and $BF(T_2)$: if $BF(T_1)$ is not a subfilter of $BF(T_2)$, then T_1 cannot be the subtype of T_2 , and T_2 cannot be the supertype of T_1 . If $BF(T_1)$ is a subfilter of

¹ Another way to encode type T to a Bloom filter is to view T as the set of names, including the name of T and those of all its supertypes. Our results shown in this paper applies to both encoding scheme. Name-based encoding is limited to languages with named subtyping, while method-based encoding can also be used for languages with structural subtyping[12].

$BF(T2)$, then it is possible that T1 is a subtype of T2, with the probability determined (reversely) by the false positive rate of the Bloom filter.

Property II states that the Bloom filter of the composition of multiple types equals the addition of the Bloom filter of each composing type. This simple approach allows us to calculate the Bloom filter of composite types. We will rely on this property to do type aggregation when setting up TypeCast routing tables.

3.3. TypeCast Routing Protocol

Our TypeCast routing protocol builds on top of MANET multicast protocols, using them to manage node mobility and achieve routing efficiency. TypeCast extends multicast protocols by adding a package filtering mechanism based on Bloom filters in order to support type substitution and type composition.

Any multicast scheme with the following properties can be used to build TypeCast: (1) it allows multiple senders and receivers, (2) it is loop free, and (3) the multicast path between any two nodes is symmetric, i.e. if u multicasts a message and reaches v , then the path traversed by the message from u to v will be the reverse path when v multicasts a message that reaches u . We will relax assumption (3) in Section 3.3.2 when we discuss optimizations.

3.3.1. Basic Routing Protocol

All nodes in a MANET that participate in type-based communication should join a multicast group G . The address of this multicast group G is well-known and determined a priori. (The address can be decided based on the types of the applications. For instance, all emergency response applications are assigned one multicast address, and military applications are assigned another). The nodes in group G are called TypeCast members.

Each TypeCast member has an Object Manager that stores the type(s) of every active distributed object running on the local host. The Object Manager maintains a Local Bloom Filter (LBF) representing the aggregation of all the types on this node: $LBF = \sum_{o \in H(u)} BF(type(o))$,

where $H(u)$ is the set of distributed objects on u . In addition, each multicast routing node in group G has a type routing table that stores the *Forwarding Bloom Filter* (FBF) of each of its multicast neighbors. FBF is a Bloom filter representing all the types that can be reached via a specific neighbor. Initially LBF and $FBFs$ in the type routing table are all zeros.

When the LBF on a TypeCast member u changes due to activation or deactivation of a distributed object, u sends a TYPE-ANNOUNCE packet to each of its

multicast neighbors w in G . The payload is an *Advised Bloom filter* (ABF) calculated as follows:

$$ABF(w) = LBF + \sum_{x \in N(w)} FBF(x) \quad (3.3.1)$$

Here $N(w)$ is u 's multicast neighbor set in G to which the incoming messages from w will be forwarded.

When a multicast routing node u in G receives a TYPE-ANNOUNCE packet from its neighbor v (v is either a member node or a routing node), it retrieves ABF from the packet and sets $FBF(v) = ABF$. It then forwards the TYPE-ANNOUNCE packet to every other multicast neighbor with the new payload calculated as equation (3.3.1). This process enables the type information to be propagated and aggregated along the multicast routing paths, and eventually all the routing nodes in G will have the type information in the condensed form of Bloom filters.

To accommodate node mobility or failure, a timer is set for each FBF in the type routing table. When the timer goes off, the associated FBF will be set to zero and a TYPE-ANNOUNCE packet will be generated based on the updated ABF calculated using (3.3.1). Each TypeCast member should periodically multicast TYPE-ANNOUNCE to refresh the type routing tables of its neighbors in the network.

To send a data packet addressing to type T , a source node multicasts a TYPE-ROUTE packet to group G . This multicast packet has an optional IP header field containing $BF(T)$. The payload consists of type T , the name of the method to invoke, and the parameters. When a node u along the multicast routing path of G receives a TYPE-ROUTE packet from its neighbor v , it uses the following two rules to process the packet:

1. For each node w ($w \in N(v)$), if $FBF(w) < BF(T)$, forward the packet to w .
2. If $LBF < BF(T)$, deliver the packet to the Object Manager of this node.

Rule 1 ensures that a TypeCast data packet will not be forwarded unless there is probability that the nodes downstream contain matching types. This probability is determined (reversely) by the false positive rate of a Bloom filter.

Rule 2 is a quick way to check whether the local node has objects matching the target type. When the Object Manager receives the delivered data packet, it will unmarshall it and dispatch the method call to the matching objects.

If the packet is targeted to the composition of multiple types, the Bloom filter of the composite type will be placed in the header of TYPE-ROUTE packet. The routing process is exactly the same as that for a single type.

The above TypeCast protocol has the following characteristics:

1. TypeCast fully supports type-based communication; there are no requirements for additional system and middleware services, such as DNS or discovery service.
2. Because all TypeCast members join a shared multicast group and the intermediate routing nodes use Bloom filter's aggregation capability to propagate type information, the bandwidth and memory cost for managing the TypeCast routing tables will not increase with the number of types in the network.
3. TypeCast routing is based on the subfilter relationship between two Bloom filters, which enables TypeCast to support subtype substitution principle and type composition according to property I and II in Section 3.2. Whether Bloom filter V_1 is a subfilter of V_2 can be efficiently determined by checking if $(V_1 + V_2) \oplus V_1$ is zero, where \oplus is the bitwise-XOR of two Bloom filters.
4. TypeCast utilizes multicast protocols to manage group membership, mobility and link errors. The data packets are forwarded along multicast routing paths, so TypeCast inherits topology and path efficiency from the underlying multicast protocol.

3.3.2. Optimization

The basic TypeCast protocol can be augmented with the following optimization techniques:

- There are many MANET multicast protocols using group join messages to discover the reverse data forwarding path. For these protocols, TYPE-ANNOUNCE messages can be piggy-backed on multicast group join messages, which will remove the symmetric path assumption made in section 3.3.
- There is a possibility that TYPE-ROUTE messages can be sent along a path that does not lead to the targeted type due to node mobility and type aggregation effect. This can be mitigated by using an explicit false notification (EFN) mechanism. When a node receives a TYPE-ROUTE message that does not match *LBF* and any of its *FBFs*, it sends a TYPE-EFN message with $BF(T)$ as payload to the previous hop, which then caches $BF(T)$ to suppress any further forwarding of the message targeting $BF(T)$ and any of its subfilters along that path.
- Since a TypeCast message usually targets only a specific type or the composition of a few types, the Bloom filters in TYPE-ROUTE messages are likely to be sparse and can thus be compressed [18] to improve bandwidth utilization.

In the next section, we conduct a detailed study of TypeCast's performance using simulation.

4. Simulation and Analysis

4.1. TypeCast Implementation

We simulated TypeCast using QUALNET [23] by extending ODMRP [13][14], which is available in QUALNET. We choose ODMRP out of convenience; the ideas expressed in this paper can be easily implemented using other MANET multicast protocols. ODMRP is an on-demand MANET multicast protocol. It creates a group-shared forwarding mesh, and data packets are flooded within the mesh. Since multiple paths exist between members of the group, ODMRP has shown strong robustness against mobility and link errors [10][15].

The OMDRP protocol for setting up the mesh (the "forwarding group") is as follows: Each multicast source periodically floods the network with JOIN QUERY packets. Each multicast member responds to the flooding by sending a JOIN REPLY packet which is forwarded along the shortest path back to the source. Every node that forwards the JOIN REPLY packet becomes a member in the forwarding mesh.

The implementation of our TypeCast is by piggy-backing TYPE-ANNOUNCE messages with ODMRP's JOIN REPLY messages. Each multicast routing node in the TypeCast group G maintains a type routing table in addition to multicast routing table. The type routing table contains *FBFs* per neighbor node per multicast source. Since ODMRP already stores the neighbor information per multicast source, TypeCast only increases the routing table storage requirements by a constant factor per ODMRP routing entry regardless of the number of types in the networks.

The Bloom Filter used in the implementation is 64 bytes (512 bits) wide. For simplicity, we did not use some of the optimization technique discussed in section 3.3.2 such as EFN and compression, during simulation. Each type is encoded into a Bloom Filter with a single hash function using the name-based encoding approach.

4.2. Simulation Environment

Fifty nodes are uniformly placed within a 1000m x 1000m area. The two-ray signal propagation model is used, in which free space path loss is used for near sight and plane earth path loss is used for far sight. The wireless link capacity is 2Mbits/sec., and IEEE 802.11 is used as the MAC protocol. The mobility model is random waypoint, in which a node randomly selects a destination and moves to it at a specific speed. After the node reaches its destination, it pauses between 0 and 10 seconds, selects another destination, and repeats the process.

The traffic pattern we use in the simulation is Constant Bit Rate with a payload size of 512 bytes. The senders and receivers are both randomly chosen among 50 nodes with uniform probability. All member nodes are joined at the beginning of the simulation and remain as members through out the simulation. The simulation time is 300 seconds.

4.3. TypeCast vs. ODMRP

Since TypeCast is implemented on top of ODMRP, we are interested in seeing how much overhead it adds to ODMRP and how much performance degradation resulted from it. We assign a single type T to every TypeCast member in the simulation and let all TypeCast messages target T . We evaluate the performance by comparing it with the scenario of directly using ODMRP protocol to multicast the messages to all TypeCast members.

The following performance metrics [6] are used:

- Packet delivery ratio: The fraction of data packets transmitted that is actually received at the destination. This number represents the overall effectiveness of the protocol.
- The number of control and data packets transmitted per data packet delivered: This value measures the efficiency of the protocol.

Figure 3a and 3b shows the result for different mobility speed and group size (the number of senders is 5, and each sends the packets at a rate of 2 pkts/sec). We can see that the delivery ratio of TypeCast is close to ODMRP under low mobility speed or when the group size is relatively large. The difference increases gradually when the mobility speed becomes large or when the group size becomes small. This phenomenon is caused by the following reason: In ODMRP, the nodes of a forwarding group always rebroadcasts data packets independently of the source node, thus allowing a packet to reach a destination possibly along multiple paths. In TypeCast, the type routing table entry is per multicast source based, thus forwarding decision also depends on the source node; this lowers the path redundancy degree provided by ODMRP, which causes relatively lower packet delivery ratio when mobility speed is high and group size is small. On the other hand, since some redundant data packets are dropped, TypeCast obtains better efficiency, as shown in Figure 3b. The overhead of TypeCast is consistently smaller than ODMRP under all mobility scenarios and group size. For small group size, TypeCast can be 30%~40% more efficient than ODMRP.

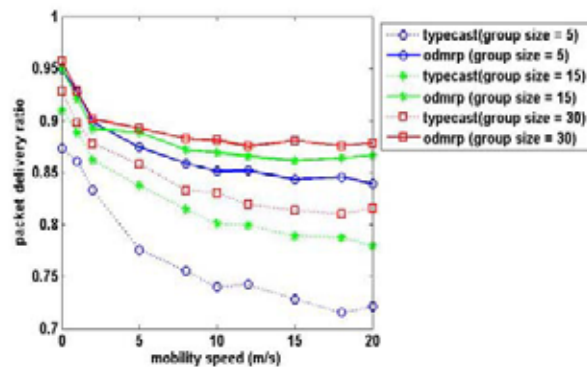


Fig. 3a: Packet Delivery Ratio of ODMRP and TypeCast

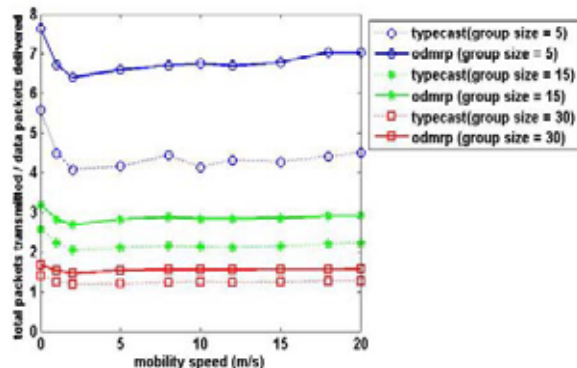


Fig. 3b: Protocol Overhead of ODMRP and TypeCast

4.4. TypeCast Performance

In this set of experiments, we study the performance of TypeCast, especially how the change of target type along a type hierarchy impacts the protocol's effectiveness and efficiency. We create 10 types, $\{T1, T2, T3, \dots, T10\}$ forming a linear type hierarchy in which $T1$ is the parent of $T2$, $T2$ is the parent of $T3$, etc.. Each TypeCast member hosts an object with type randomly chosen from $T1$ to $T10$.

We use the following performance metrics²:

- Packet delivery ratio,
- Total number of data packets transmitted. We use this metric to investigate the efficiency of TypeCast when the target type changes along the type hierarchy

4.4.1. Target Type and Mobility Speed

In this experiment, there are 20 TypeCast members and 1 sender. The network traffic load is set at 2 packets/sec. We vary the target type from $T1$ to $T10$ and the mobility speed from 0 to 20 m/s.

² We do not include the total number of control packets here because the change of the target type does not affect the number of control packets transmitted in TypeCast.

In figure 4a we see that the delivery ratio of TypeCast is almost constant with respect to target type when mobility speed is between small to medium. As mobility speed reaches the largest value, there is only a slight drop of the delivery ratio when the target type goes down the type hierarchy. Figure 4b shows the number of data packets transmitted. When the target address changes along the type hierarchy from top to bottom, the total data packets transmitted drops accordingly. As discussed in section 2, due to the substitution principle, a type/subtype relationship can be transformed to a group/subgroup relationship. When the target type goes down the type hierarchy, the size of the effective communication group becomes smaller. TypeCast's Bloom filter based routing scheme automatically detects this situation by dropping more data packets that do not lead to target nodes, demonstrating the effectiveness of the TypeCast routing protocol.

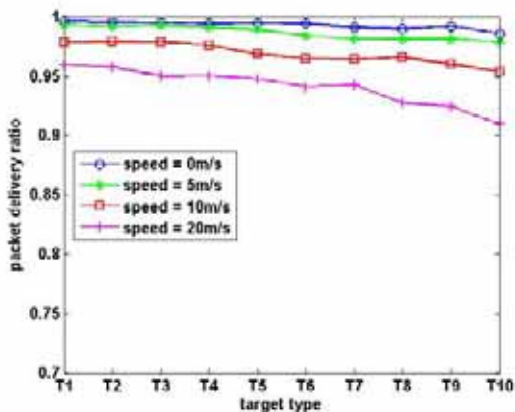


Fig. 4a: PDR of TypeCast as Function of Target Type and Mobility

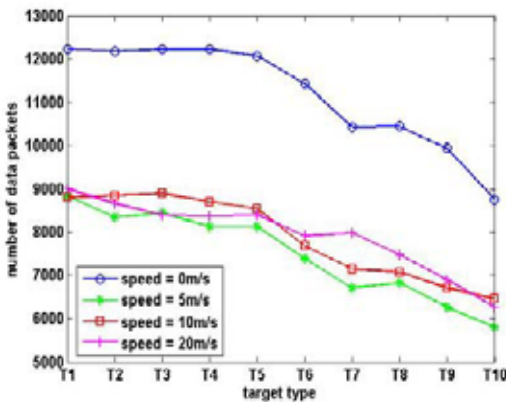


Fig. 4b: Number of Data Pkts of TypeCast as Function of Type and Mobility

4.4.2. Target Type and Number of TypeCast Members

In this experiment, 1 sender in the network transmits packet at 2 packet/sec. The mobility speed of each node

is 1 m/s. We vary the target type from T_1 to T_{10} and the TypeCast member size from 10 to 40.

Figure 5a shows that TypeCast keeps its packet delivery ratio almost constant when the target type varies along the type hierarchy under different group size. At the same time, a smaller number of data packets are transmitted in the network when the target type moves down the type hierarchy (see Fig. 5b). This reduction of data packets is observed for different group sizes.

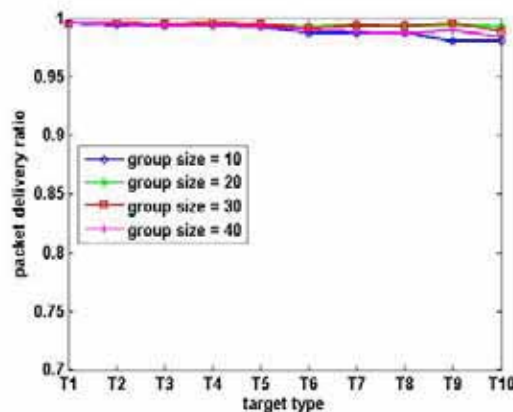


Fig. 5a: PDR of TypeCast as Function of Target Type and Group Size

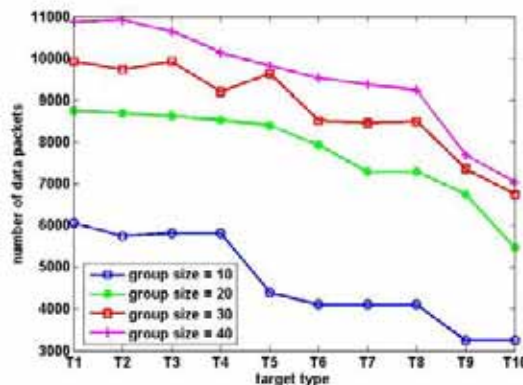


Fig. 5b: Number of Data Pkts of TypeCast as Function of Type and Group size

4.5. TypeCast Scalability

In this experiment, we study the scalability of TypeCast when the number of types in the network increases. We set the number of TypeCast senders to 5, the number of TypeCast members to 30 and mobility speed to 1 m/s. The network traffic load is 10 pkts/sec. We use 100 types divided by 10 sets: $S_1 = \{T_1, T_2, \dots, T_{10}\}$, $S_2 = \{T_{11}, \dots, T_{20}\}$... and $S_{10} = \{T_{91}, \dots, T_{100}\}$. The types in the same set forms a linear type hierarchy, and the types in different sets are independent of each other. We performed ten rounds of experiments, and in every round we added a new set of types into the network. Every

TypeCast member randomly picks a type from each of the selected sets. We repeated each test with different target types.

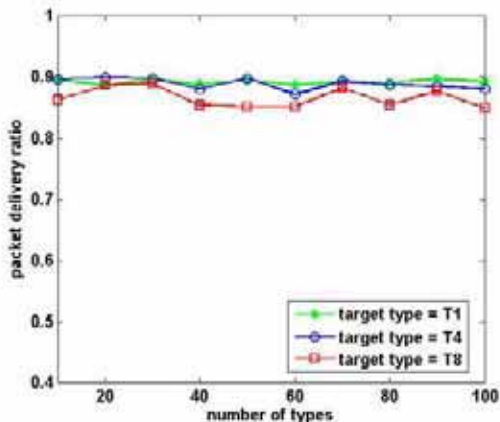


Fig.6a: PDR of TypeCast as Function of Number of Types and Target Types.

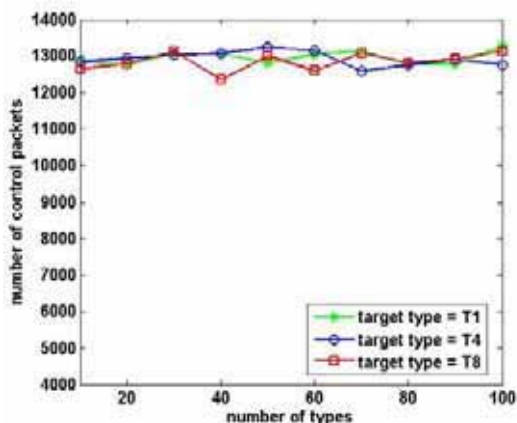


Fig. 6b: Number of Control Pkts of TypeCast as Function of Num. of Types and Target Types

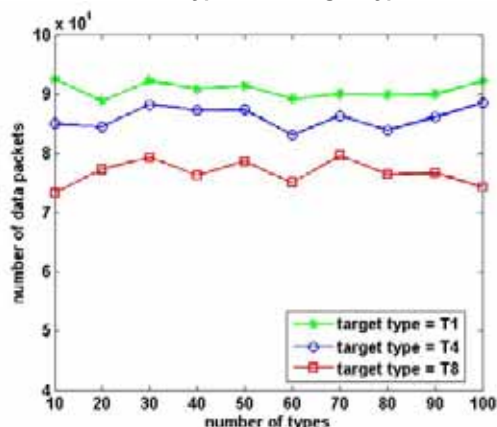


Fig. 6c: Num. of Data Pkts of TypeCast as Function of Num. of Types and Target Type

Figure 6a, 6b and 6c show that the change of the number of types does not affect the packet delivery ratio, the number of control and data packets transmitted in

the network. It confirms TypeCast's resilience to the increase of number of types in the network.

In summary, TypeCast performs well under different environments. It inherits routing topology from ODMRP, and shows robustness against the increase of mobility and group size³. TypeCast automatically adjusts the network traffic when the target types changes along a type hierarchy, and it achieves good scalability with respect to the number of types in the network thanks to the aggregation provided by the Bloom filters.

5. Related Work

Publish/Subscribe [7][20] is a data-centric communication paradigm in which receivers subscribe to a pattern of events and are subsequently notified of occurrences of these events generated by the senders. Since event matching can be based on an arbitrary value space, Publish/Subscribe is more flexible than type-based communication as a generic addressing scheme, especially when there is no hierarchical relationship among group members; however Publish/Subscribe requires developers to handle low-level event construction and processing, which can be cumbersome and error prone. On the other hand, type-based communication is a service-oriented communication paradigm in which receivers are modeled as distributed objects for providing services, and senders are service consumers. Its addressing scheme uses the natural mapping between type hierarchy and real-world social structure, enables the higher level language abstraction and the associated object-oriented principles to be directly used to develop large-scale ad-hoc applications. For existing applications built with distributed object or web service model, type-based communication provides an easy path to migrate these applications from a point-to-point, wired environment to a many-to-many, wireless environment. In addition, type-based communication can be naturally utilized to support service discovery and composition in ad-hoc networks.

M2MI [11] is a Java framework providing interface-based group communication primitives for ad-hoc collaboration in wireless networks. It uses broadcast to transmit packets, which is unlikely to be successful in multi-hop wireless network due to high contention caused by flooding. It does not address subtyping and type composition.

Content-Based Multicast(CBM) [26] dynamically routes data packets to a set of receivers determined by the relative location between senders and receivers and their mobility speed. It is a completely new routing scheme. TypeCast focuses on using language constructs

³ Due to space limit, we did not include simulation results related to sender size and network traffic load.

to dynamically determine the receiver set, while leveraging existing multicast protocols to achieve performance and deployment benefits.

In Active Networks [26], each packet carries both code and data. The code in the packet is executed on intermediate routers to decide the next hop address. In TypeCast, Bloom filters encode language constructs, but they are not executable code. The routing is still determined statically by the routing table on each node, not by the data packet. Thus, TypeCast has fewer deployment and security issues.

Distributed hash tables for P2P computing (e.g. [22][24]) use an identifier as the address for routing the messages. The identifier defines the distance among nodes in the overlay network and is used to direct packet forwarding path. The identifier usually has no direct correlation with application semantics. In TypeCast, a Bloom filter encodes the hierarchical type information and thus has a direct association with programming language constructs. The data paths are determined by the topology of the physical network.

6. Conclusion and Future Work

We proposed type-based communication as a means to provide group communication applications with robust and expressive addressing and routing semantics to control the scope of message delivery. We also discussed an efficient TypeCast protocol that uses Bloom filters to represent type and to make message-forwarding decisions. Finally, we suggested how Bloom filters can naturally support subtyping, type composition, and type aggregation at the network layer.

Type-based communication is an important step towards programmable networks. Our future work will include investigating suitable language constructs to represent type-based communication and exploring how to support additional QoS semantics for type-based communication by utilizing reliable multicast protocols [19].

7. Acknowledgement

This work is funded by the National Science Foundation under NR grant ANI-0335302 *WHYNET: Scalable Testbed for Next Generation Mobile Wireless Networks Technologies*.

8. References

[1] D. E. Bell and L. J. LaPadula, "Secure Computer Systems: Mathematical Foundations and Model." The Mitre Corporation, 1976
[2] B. Bloom, "Space/time tradeoffs in hash coding with allowable errors", Communications of ACM, 13(7), July 1970

[3] E.Bommaiah, M.Liu, A.McAuley, and R.Talpad, "AM-Route:Adhoc Multicast Routing Protocol", Internet Draft, draft-talpad-manet-amroute-00.txt, Feb.1999
[4] G. Bracha, W. Cook, "Mixin-based Inheritance", SIGPLAN Notices, pp303-311, vol. 25(10), 1999
[5] A. Broder, M. Mitzenmacher, "Network Applications of Bloom filters: A Survey", Internet Mathematics, 1(4) 485-509,2004.
[6] M.S Corson and J. Macker, "Mobile Ad Hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Consideration", RFC 2501, IETF, Jan, 1999.
[7] P. Eugster, P. Filber, R. Guerraoui, A. Kermarrec, "The Many Faces of Publish/subscribe", ACM Computing Surveys, Vol. 35, Issue 2, 2003,, pp.114-131
[8] J.J.Garcia-Luna-Aceves and E.L.Madruga, "A Multicast Routing Protocol for Ad Hoc Networks", in Proc. of INFOCOM'99, pp.784-792,
[9] J.G.Jetcheva and D.B.Johnson, "Adaptive Demand-Driven Multicast Routing in Multi-Hop Wireless Ad Hoc Networks", in Proc. of MobiHoc'01, pp. 33-44.
[10]J.G.Jetcheva and D.B.Johnson,"A Performance Comparison of On-Demand Multicast Routing Protocols for Ad Hoc Networks", Technique Report, CMU-CS-04-176, School of Computer Science, Carnegie Mellon University. Dec. 15, 2004
[11]A. Kaminsky and Hans-Peter Bischof, "Many-to-Many Invocation: A new object oriented paradigm for ad hoc collaborative systems," In Proc. OOPSLA 2002.
[12] S. Kurtev, "Subtyping and Inheritance in Object-Oriented Programming," Ph.D Thesis, Viena University of Technology, 2000.
[13]S-J. Lee, W.Su and M. Gerla, "On-Demand Multicast Routing Protocol (ODMRP) for Ad Hoc Networks", Internet-draft,
[14]S-J. Lee, M. Gerla and C-C. Chiang, "On-Demand Multicast Routing Protocol", In Proc. of IEEE WCNC'99.
[15]S-J. Lee, W. Su, J. Hsu, M. Gerla and R. Bagrodia, "A Performance Comparison Study of Ad Hoc Wireless Multicast Protocol", IEEE INFOCOM'00
[16]B. Liskov, "Data Abstraction and Hierarchy", SIGPLAN Notices, 23,5 (May, 1988)
[17]B. Liskov, J. Wing, "A Behavior Notion of Subtying", ACM Transactions on Programming Languages and Systems, pp.1811-1841, Nov. 1994
[18]M. Mitzenmacher, "Compressed Bloom filters," PODC 2001.
[19]S. Paul, K. Sabnani, J.Lin, S.Bhattacharyya, "Reliable Multicast Transport Protocol(RMTP)", IEEE Journal n Selected Areas in Communications, vo.15, pp407-421, April. 1997.
[20]M. Petrovic, V. Muthusamy, H. Jacobsen, "Content-Based Routing in Mobile Ad Hoc Networks," mobiquitous'05, pp. 45-55,
[21]E.M.Royer and C.E.Perkins, "Multicast Operation of the Ad-hoc On-Demand Distance Vector Routing Protocol", in Proc. of Mobicom'99, pp.207-218
[22]A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems", in Proc. MIDDLEWARE'01
[23]Scalable Network Technologie Inc, "QUALNET Developer's Guide",
[24]I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-peer Look up Service for Internet Applications," in Proc. ACM SIGCOMM'01.
[25]Sun Microsystems Inc. "Jini Specification", <http://www.sun.com/software/jini>
[26]D. Wetherall, "Active Network Vision and Reality: Lessons From a Capsule-based system", in Proc. 17th SOSP, pp. 64-79, Dec. 1999.
[27]H.Zhou and S. Singh, "Content Based Multicast in Ad-hoc Networks", in Proc. of MobiHOC'00, pp 51-60.