FaReCast: Fast, Reliable Application Layer Multicast for Flash Dissemination

Kyungbaek Kim[1], Sharad Mehrotra[1], and Nalini Venkatasubramanian[1]

Dept. of Computer Science
University of California, Irvine, USA
{kyungbak,nalini,sharad}@ics.uci.edu

Abstract. To disseminate messages from a single source to a large number of targeted receivers, a natural approach is the tree-based application layer multicast (ALM). However, in time-constrained *flash dissemination* scenarios, e.g. earthquake early warning, where time is of the essence, the tree-based ALM has a single point of failure; its reliable extensions using ack-based failure recovery protocols cannot support reliable dissemination in the timeframe needed. In this paper, we exploit path diversity, i.e. exploit the use of multiple data paths, to achieve fast and reliable data dissemination. First, we design a forest-based M2M (Multiple parents-To-Multiple children) ALM structure where every node has multiple children and multiple parents. The intuition is to enable lower dissemination latency through multiple children, while enabling higher reliability through multiple parents. Second, we design multidirectional multicasting algorithms that effectively utilize the multiple data paths in the M2M ALM structure. A key aspect of our reliable dissemination mechanism is that nodes, in addition to communicating the data to children, also selectively disseminate the data to parents and siblings. As compared to trees using traditional multicasting algorithm, we observe an 80% improvement in reliability under 20% of failed nodes with no significant increase in latency for over 99% of the nodes.

1 Introduction

Our work is motivated by the need for highly reliable data dissemination that delivers critical information to hundreds of thousands of receivers within a very short period of time. An example is the flash dissemination of disaster (natural or man-made) warning messages that must be rapidly delivered to urban populations in a matter of a few seconds to enable citizens to take self-protective measures (e.g. duck-cover-hold on for earthquakes, shelter underground for tornadoes). The key challenge lies in delivering messages scalably (reaching large populations), reliably (despite network outages and message losses) and efficiently (low operational cost during non-disaster times with quick ramp-up when needed). Given the rarity of these events, it is unlikely that a dedicated infrastructure, such as communication connections between peoples and the source of the messages, that is operational and available 24/7 will be deployed. Our objective, therefore, is to leverage any and all available infrastructure and exploit knowledge of network connectivity to ensure that interested recipients are able to actually receive the messages within a very short period of time.

In this paper, we develop an ALM-based solution for the flash dissemination problem above using a peer-oriented architecture where the peer nodes are formed by those interested in receiving the messages. We argue that the problem lends itself well to a peer-based architecture; an ALM-based solution is attractive due its potential for easy deployment (no changes to lower layer network protocols at the participating nodes)[3] and its ability to deal with a variety of recipients. Challenges arise since (a)end-devices are autonomous and hence unreliable; (b)end-devices are nodes that are executing other tasks and must typically be rapidly repurposed to deal with the rare warning messages when it occurs. Our ALM-based solution must deliver the messages reliably and in time, while handling the node churn as well as minimizing the maintenance overhead.

We view the ALM approach as consisting of two key aspects - (a) the ALM structure and (b)the multicasting algorithm deployed on the structure. Participating nodes in the ALM structure organize themselves into an overlay topology (typically a tree or mesh) for data delivery - each edge in this topology corresponds to a unicast path between two nodes in the underlying Internet. Once the ALM structure is constructed, data from the source node is delivered to all multicast recipients using the implemented multicasting algorithm. Note that all multicast-related functionality is implemented at the nodes (instead of at routers as in network-layer multicast). Typical ALM applications include file sharing and content streaming; specialized protocols have been designed for these applications[4][5][6][7][8][9][10]. Our prior work on flash dissemination focused on fast dissemination of medium to large sized data. We leveraged a dynamic mesh-based overlay network constructed using a random walker protocol, CREW[11] to concurrently disseminate multiple chunks of a large message. The Roulette protocol extended the random walker implementation to enable low dissemination overhead despite catastrophic failures; it was used to implement a P2P webserver, Flashback [12] to deal with flash crowds. However, the cost of metadata propagation and overlay maintenance in CREW/Roulette is unwarranted in the current scenario where (a) the end recipients are known and (b) the message size is small.

Our target scenario consists of a single source and several receivers; A tree-based ALM structure (where every node has a single parent and multiple children) seems appropriate for fast data dissemination[3][4][16][25]. A tree-based structure exploits concurrency in that all nodes that have received data can communicate with their children concurrently. As the *fan-out*,i.e. the number of children per node, of the tree increases, the data delivery time decreases logarithmically with the number of fan-out as the base. Unfortunately, the tree structure is highly vulnerable to failures since every intermediate node in a tree structure is a point of failure[14][15][22] that can block delivery of the message to the entire subtree under it. In fact, this problem is further aggravated as the fan-out of a tree structure increases, since the number of nodes impacted by a failure increases exponentially.

Typical reliable multicast approaches detect such failures during operation and recover from them (usually by reconstructing parts of the tree). Protocols have been designed to recover the tree structure by substituting the failed node with an existing online node[3][4][5][6] - this requires tree reconstruction which is a time-consuming operation; techniques have also been proposed to use backup information to reduce recovery time to some extent[16][20][21][22][23][24]. Many

of these approaches rely on TCP retransmission to detect and retransmit lost packets and support reliability at the communication layer. In our flash dissemination scenario, i.e. dissemination of disaster warning messages, delivery times are highly constrained; existing approaches to detect and recover from failures or employ per-packet acknowledgements require interactions and handshakes with other nodes, which makes it difficult to respond within the required time constraints.

In this paper, we present FaReCast, an ALM protocol for fast and reliable data dissemination that exploits the use of multiple data paths between nodes judiciously by (a) designing a new ALM structure and (b) developing a new multicasting algorithm that efficiently exploits the proposed structure. Specifically, we design a forest-based M2M (Multiple parents-to-Multiple children) ALM structure where each participant node has multiple parents as well as multiple children. Care is taken in the construction of the M2M structure to support path diversity (through the choice of unique parent node sets); this supports increased reliability by minimizing the probability that nodes with failed common parents lose data. To complement the M2M ALM structure, we design a multidirectional multicasting algorithm that effectively utilizes the multiple data paths in the M2M ALM structure. In addition to top-down communication characteristic of traditional multicasting, the multidirectional multicasting algorithm deploys bottom-up data flow and horizontal data flow carefully. Since there is prior knowledge of the M2M structure at each node, nodes can trigger communication and send data to parents from which it has failed to receive the expected data. Additionally, in the case of leaf nodes, FaReCast forwards data to the other leaf nodes which it anticipates may have not received the data. A key design issue for FaReCast is addressing the tradeoff between two conflicting factors - higher fan-out (increased speed) vs. higher fan-in (increased reliability).

Multiple fan-in ALM structures are not new; approaches such as [7][9][13][14][15][18][19] all increase the number of data paths to a target node to stream large data concurrently and efficiently. Here, the large data is divided into smaller data chunks that are disseminated through the multiple paths concurrently. The chunking techniques are combined with loss tolerant decoding/encoding [7], e.g. erasure coding[17] to ensure that all recipient nodes get the entire large data correctly. Such expensive coding schemes are unnecessary in our scenario where the information delivered to target nodes is small (order of a few bytes/Kbytes). FaReCast exploits path redundancy to send the same data through multiple paths and thereby improves dissemination reliability. Another aspect of performance is the number of duplicate messages received by a node (any duplicate messages are considered to be unnecessary overhead).

While our primary goals are scalability, speed and reliability, our secondary goal is to reduce maintenance overhead during normal times when there is no event. In FaReCast, we minimize the client-side maintenance overhead by storing the snapshots of current network status at the configuration manager. Each individual user retrieves accurate parent/children information from the configuration manager periodically. The configuration manager detects node failure based on this periodic update request and updates the snapshot asynchronously. Through both simulation and implementation-based evaluations, we show that FaReCast tolerates over 40% random failures while meeting the latency con-

straints, i.e. FaReCast can endure high user churn as well as a certain level of snapshot inconsistency.

The rest of the paper is organized as follows. In Section 2, we describe the design and management of the M2M ALM structure having multiple fan-in and fan-out. The multidirectional multicasting algorithm achieving high reliability with small data latency is described in Section 3. We conduct a simulation-based performance evaluation of FaReCast in Section 4. In Section 5, we present an implementation of the FaReCast system as well as its evaluation by using a campus cluster platform and a emulated wide area network. Finally, we present concluding remarks in Section 6.

2 The Forest-Based M2M ALM Structure

In this section, we present our forest-based M2M ALM structure. Figure 1 depicts the overall architecture of the envisioned system and illustrates the proposed ALM structure. The system primarily consists of (a) target nodes interested in receiving the dissemination, (b) an originating server where the dissemination is initiated and (c) a configuration manager that maintains and manages the structure and connectivity information (discussed later). We will construct an overlay structure consisting of the originating server and target nodes and effectively use that structure to address our dual needs of reliability and timeliness in information dissemination. To maintain separation of concerns, we distinguish two clear-cut steps in enabling reliable, fast ALM: (a) construction of an overlay structure that enables concurrency and reliability (b) use of the constructed overlay in a multicast protocol that efficiently implements concurrency (aka speed) and reliability tradeoffs in the dissemination process under dynamic conditions.

The process begins with the construction of the forest-based M2M overlay structure which consists of the originating server (or a well-established representative) as the root-node and target nodes (interested recipients) that form the sub-tree rooted at the originating server. The goal is to organize the target nodes into levels and establish overlay connections between them to enable concurrency and reliability for dissemination. The overall design philosophy is as follows: to support fast dissemination, every node in the structure will have multiple children from which concurrent content dissemination can occur. To handle reliability, every node in the structure will have multiple (redundant) parents that it receives content from. Determining the arities and connections of nodes in the M2M (Multiple parents to Multiple children) overlay structure to maximize speed and reliability while minimizing redundant transmissions is the key challenge. Prior to establishing the properties of the M2M structure, we provide some definitions and assumptions that will be used in creating the M2M structure.

Level of a node (L) : The level of a node is the length of the path, expressed as number of hops, from the root to this node. The level of the root node is assumed to be 0, children of the root node are at level 1 etc.

Sibling Nodes : Nodes having same level are referred to as sibling nodes. Sibling nodes at a level belong to a *level-group*. N_L is the number of sibling nodes at level L.

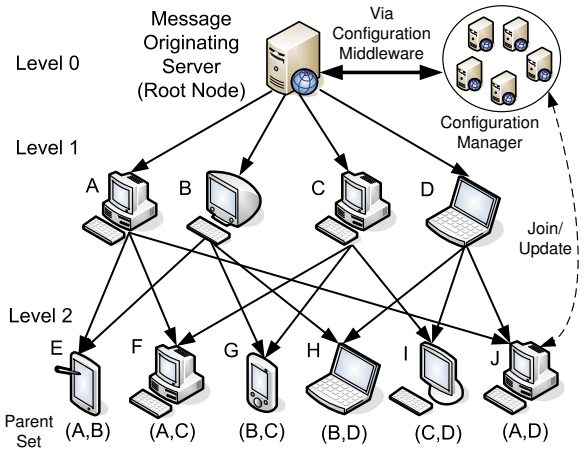


Fig. 1. The forest-based M2M ALM structure (F_o =2, F_i =2).

Fan-in (F_i) : The fan-in of a node is the number of parents of a node. All participating nodes (except those at levels 0 and 1) have F_i parents.

Fan-out factor (F_o) : The fan-out factor is the ratio of N_{L+1} to N_L, and is a measure of the minimum number of nodes at a level, i.e. $N_L \geq (F_o)^L$.

The **Configuration manager** enables the management of the M2M structure by performing the following tasks: (a) management of information on nodes, level-groups and sibling nodes in a level-group; (b) construction and maintenance of the overlay structure, (c) answering of queries about the M2M structure - for example, it responds to update requests from target nodes with information on their current parents/children. The update requests also serve as periodic heartbeat messages to the configuration manager indicating that a node is alive; in FaReCast, we keep the frequency of such update requests very low - ($\sim$ 1 hour) in our experiments. Communications between the configuration manager, target nodes and the originating server occur via a pre-established API.

2.1 Properties of the M2M ALM structure

- **Root-node Reliability:** The root node (at level 0) is expected to be continuously available.
- **Fan-in constraint:** Each participating node should have F_i distinct parents just one level above. If the level of a node is L, all the parents are picked from the same level, $L-1$, as illustrated in Figure 1. The only exceptions are nodes at level 1 that have only one parent since the level-group for level 0 has only one node, the root node.
- **Loop-free nature:** We assume loop-free transmission when data flows from parent nodes to child nodes. In particular, we assume that the assignment of levels in the M2M ALM structure is done in such a way that all paths from the root to a node have the same length. Assuming that the latency of message transmission over any two links is not significantly different[26], the M2M ALM structure can guarantee that data reaching a node from different parents arrive close to each other in time. That is, in the M2M ALM structure

with F_i parents, each node should get F_i data items within a short period of time under the failure-free situation. This property is later exploited (and selectively relaxed) by the multidirectional multicasting protocol to improve reliability and make up for missing messages from failed parents.
- **Parent Set Uniqueness:** All nodes in the M2M ALM structure (with $level \geq 2$) have a unique set of parents. That is, there is at least one different parent in the parent-sets of two nodes at the same level. Without this property, concurrent failures of nodes at a level can potentially block out messages to the level below; parent-set uniqueness enforces path diversity in the propagation of the message and consequently improves the chances of reliable delivery.

Achieving the parent-set uniqueness property brings out an interesting relationship between fan-in and fan-out factor as described below. The following equation should be satisfied to guarantee parent-set uniqueness:

$$N_{L+1} \leq C(N_L, F_i) \quad \text{where } F_i > 1,\ L > 0 \tag{1}$$

In the equation 1, $C(n, k)$ represents the number of k-combinations from a given n elements. That is, the number of distinct subsets with F_i-combinations that represent the parent sets of nodes in level L should be equal or greater than the number of nodes (N_{L+1}) in level $L+1$. If we assume that the number of sibling nodes is $N_L = (F_o)^L$, the equation 1 can only be satisfied when the value of L is large enough or F_o is greater than F_i. With this assumption (i.e. $N_L = (F_o)^L$), when $F_i = 2$, $F_o = 2$, the equation 1 is satisfied only if $L > 2$. In other words, N_L calculated using only the fan-out factor may not be sufficient to guarantee the parent-set uniqueness property at a level - additional nodes may therefore be needed at a level L - we call these nodes *complement nodes*.

To get the proper number of complement nodes, we take into account F_i as well as F_o. Consider the case where a node A at level L attempts to get F_o children with F_i parents each. To come up with a conservative estimate of the number of complement nodes, let us further assume that none of the pre-existing nodes at level L (besides A) are available to meet the fan-in need. In other words, the fan-in requirement must be satisfied by additional complement nodes at level L. Since A is already a parent for its children, we need at least $F_i - 1$ complement nodes for the first child. To guarantee parent-set uniqueness for the remaining $F_o - 1$ children, we need $F_o - 1$ additional complement nodes. Consequently, $(F_o - 1) + (F_i - 1)$ nodes is a sufficient condition for a node having F_o children to assign a unique set of parents to each child node. To satisfy the equation 1, N_L is determined by the following equation.

$$N_L = (F_o)^L + F_o + F_i - 2 \quad \text{where } F_o > 1,\ F_i > 1,\ L > 0 \tag{2}$$

Lemma 1. *Given $F_o > 1$, $F_i > 1$ and $L > 0$, $N_L = (F_o)^L + F_o + F_i - 2$ is enough to satisfy the parent set uniqueness property.*

Proof. When the equation 2 is applied to the equation 1, we get the following expanded equation, $(F_o \cdot (F_o)^L + F_o + F_i - 2) \leq \prod_{x=1}^{F_i}(A/x + 1)$, where $A =$

$(F_o)^L + F_o - 2$. Since A is always greater than zero, and $F_i > 1$, the right hand side of the expanded equation has the minimum value when $F_i = 2$. Also note that the right hand side always increases as F_i increases. Below, we show that the minimum value of the right hand side is always equal to or greater than the left hand side, i.e. the expanded equation is always true.

We apply $F_i = 2$ to the above expanded equation followed by some straightforward algebraic manipulation. This yields

$$((F_o)^L + 1) \cdot F_o \leq ((F_o)^L + F_o) \cdot ((F_o)^L + F_o - 1)/2. \quad (3)$$

Comparing terms on the right hand side and the left hand side, we can see that the equation always holds true for $F_o > 1$ and $L > 0$.

2.2 Node Operations and Structure Maintenance

Given an initial root node (possibly the originating server), the creation of an M2M ALM structure that is compliant with the properties described above proceeds as a series of node joins. Subsequent evolution of the structure consists of a series of node join/leave requests coordinated by the configuration manager. Interspersed with node joins and node leaves, the configuration manager detects node failures (i.e. an unstructured node leave) and reorganizes the M2M ALM structure to recover from the failure. FaReCast employs periodic *update request* messages from participant nodes to the configuration manager to communicate that a participant node is still alive. The CM responds to a nodes *update request* with the current parents/children of the node. As discussed earlier, the frequency of *update messages* is kept low to prevent bottlenecks at the configuration manager and reduce structure maintenance cost. The details of each operation are described below.

Node Join: When a new node, N_{new}, joins the system, it is first given a new nodeID. the node sends a join request to the configuration manager in order to determine (a)its level and (b)its parent/children nodes. In accordance with equation 2, the configuration manager determines current information about each level group and selects the lowest level requiring more nodes as the *level* of the new node in the M2M ALM structure. Parent selection for the node proceeds as follows. If the selected level is $L + 1$, the configuration manager randomly picks F_i different nodes among the level L nodes as the parents of the new node. The configuration manager next determines whether the selected parent set satisfies the *parent set uniqueness property* at level L. The configuration manager maintains a *parents pool* structure managed by each level-group - that contains entries of the form {child, parent-set}. If the newly selected parent-set is not unique, we choose to replace one of the nodes (the one with the maximum number of children) in the current parent set with another distinct randomly selected node at the same level (excluding the most recent selection). This uniqueness checking operation is repeated until a unique set of parents is found. According to Lemma 1, every new node can find a unique set of parents. After finding a unique set of parents, the configuration manager registers the new node with the new nodeID and its selected parent-set in the corresponding parent pool, and each parent of the new parent-set registers the new node as a child using its nodeID. The configuration manager then responds to the join request with the nodeID, the level, and parents/children information.

Node Leave: When a node leaves the M2M ALM structure, it notifies the configuration manager to coalesce the hole caused by it in the ALM structure. All the impacted nodes - parents-sets that include the leaving node, children of the leaving node and parents of the leaving node should be invalidated and updated. Also the number of nodes in the level-group of the leaving node reduces by one. The key idea is to quickly find a replacement node for the leaving node with minimum maintenance overhead including communication and processing cost. To do this, we invalidate the registered node information of the leaving node. The invalidated node information is retained at the configuration manager for a while (this is captured by a *retainment-timer*) with the expectation that a new node will join quickly and it can simply replace the invalidated node. If this happens, the configuration manager validates all the impacted nodes to accommodate the newly arrived node. If there are not enough new nodes to cover the invalidated nodes, the configuration manager must repair these holes with other existing nodes. A retainment-timer is associated with each invalidated node; when the retainment-timer of an invalidated node expires without any replacement, the configuration manager picks a random leaf node to replace the invalidated node and subsequently invalidates the leaf node information. The configuration manager does not notify the updated information to all the impacted nodes right away, but responds with it to their periodic requests for the recent structure information.

Node Failures: Failed nodes are those that leave the M2M ALM structure without any notification. The configuration manager detects the failed nodes by an *update-timer*. Each participant node sends a periodic update request in order to refresh the parents/children information. Once a node joins the system and its information is registered, the configuration manager sets the update-timer of the node and resets the timer whenever it receives an update request from the node. If the update-timer of a node is expired, the node information is invalidated, and the repair of the M2M ALM structure proceeds similar to the case of a leaving node. The nodes in the lower level-groups use shorter update-timers than the nodes in the higher level-groups, because lower level nodes are critical to preserving the reliability of the M2M structure.

Maintenance Overhead and Reliability: In our target scenario, an interesting event occurs rarely and participating nodes join/leave the system far more frequently than the event. It is therefore imperative that the management overhead (e.g. network bandwidth, processing power) incurred by the M2M ALM structure is low, especially during non-event times. In order to reduce the maintenance overheads of the participating nodes, we implement a cooperative solution in which the configuration manager and nodes participate. Firstly, we effectively and efficiently use the configuration manager to maintain the M2M ALM structure and required meta-data; participating nodes send update requests to the configuration manager for current structure information. At the node end, we increase the length of timers (retainment-timer and update-timer) to keep neighborhood information updated. There is an obvious tradeoff between the overhead and reliability since delaying timers implies that the M2M ALM structure from the view of participant nodes is not as updated as desired. In the next section, we propose an effective multidirectional multicasting protocol (when the event occurs) that can tolerate stale information in the M2M ALM structure - allowing us to achieve low maintenance overhead with high reliability.

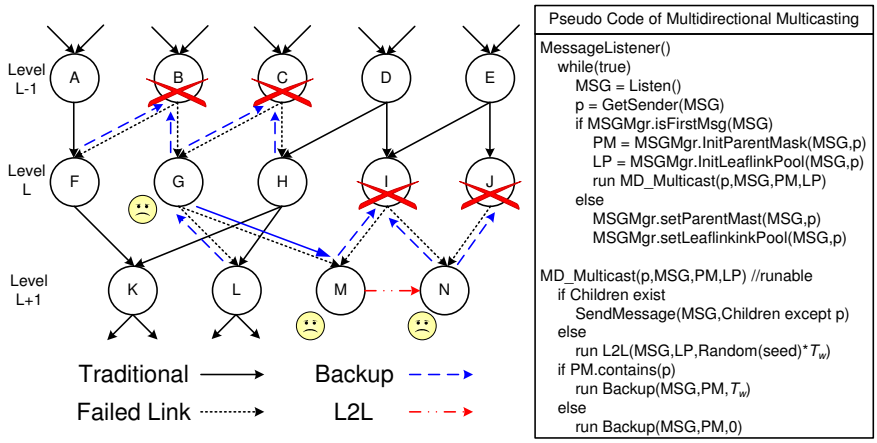


Fig. 2. Example and Pseudo Code of Mutidirectional Multicast

3 Multidirectional Multicasting Algorithms

In this section, we describe our proposed multidirectional multicasting algorithms. The key idea here is to enable reliable message delivery despite failures with limited increase in latency and messaging overhead. We do this by pushing the multicast message along directions where failures are estimated to have occurred. Recall that when a node in the M2M ALM structure receives a message (the first one) from one parent, it is expected that messages from all the other parents will arrive shortly (due to equal path length and similar delay assumption). When messages do not arrive from parents, we conservatively assume possible failures in the direction of those parents - and enhance the multicasting algorithm to send additional messages in these directions. We propose a protocol that encompasses two forms of multidirectional multicast - *Backup* and *Leaf-to-Leaf (L2L) dissemination.*

3.1 The Bypassing Problem and Backup Dissemination

In our M2M ALM structure, although a node has multiple parents, the message remains undelivered to the node if all parents of a node have failed. Note that if this node is an intermediate node, its children can still receive the message from the other parents and paths. In other words, the message may bypass this node even though the node is operational - we refer to such nodes as *bypassed nodes.* Once a node is bypassed, its children lose one data path and the probability of message loss increases. For example, in Figure 2, node G is the bypassed node and its child, node M does not get the message. One objective of our multicasting protocol design is to reduce the degradation of the reliability caused by the bypassed nodes.

Traditional multicasting protocols propagate messages unidirectionally along the tree from parents to children and cannot address the bypassing problem. If however, bottom-up propagation is permitted, i.e. messages from children to parents, the bypassed nodes can receive a message from one of its children who

have received the message from alternate parents. Since this creates possibly redundant messages - careful determination of potential bypassed parent nodes by children is essential.

The selection of bypassed nodes exploits the constant-path-length property maintained by the M2M ALM structure. According to this property, when a node receives the first message from one of its parents, it expects that other $F_i - 1$ parents will send the same message within a short period of time, T_w. When a node fails to receive a message within T_w from a parent, the node assumes that the parent node has been bypassed or has failed - and forwards the message to this parent – we refer to this selective bottom-up dissemination as *backup dissemination*.

If a parent node gets the first message from a child, it recognizes that it is bypassed. The bypassed node (a) forwards the message to the remaining children (except the one from which it received the message) and (b)initializes the backup dissemination immediately to forward the message to its parent nodes. This process is repeated iteratively until the failed node is reached. Note that since the child initiating the backup dissemination has already waited for T_w, parents in the chain do not need to wait to initiate their respective backup disseminations. This selective upward and downward message propagation initiated by backward dissemination handles the reliability degradation caused by the bypassed nodes.

3.2 Missing Leaf nodes and Leaf-To-Leaf Dissemination

While backup dissemination handles missing messages to bypassed intermediate nodes,it cannot be applied to a leaf node whose parents have all failed – we refer to such a leaf node as a *missing leaf node*. Given the M2M ALM structure (larger number of nodes at higher levels of the tree), a significant number (almost half) of the participating nodes are leaf nodes. Techniques to ensure that messages get to leaf nodes on time is critical to the reliable dissemination process.

We introduce the notion of *leaf links* to address the problem of missing leaf-nodes. Here, each leaf node maintains a *leaf link pool*, i.e. a set of links to leaf nodes sharing the same parents, i.e. *leaf-links*. When a leaf node gets the first message from a parent, the following actions are performed. First, leaf links sharing that parent are masked in the leaf link pool since it is assumed that the other leaf nodes can also receive the message from the shared parent. Second,as in backup dissemination, the node waits for a stipulated period of time T_w, to receive messages from all the other parents. After the waiting time passes, the leaf node sends the message to the other leaf nodes corresponding to the unmasked leaf links. In Figure 2, node M getting the first message from node G recognizes that node I is bypassed or failed, and sends the message to node N which shares node I as a parent. The direction of this dissemination is horizontal (leaf node to leaf node) and we refer to it as *L2L (leaf-to-leaf) dissemination*. Using L2L dissemination, therefore, makes up for reliability degradation due to missing leaf nodes .

Unlike backup dissemination, we argue that having different waiting times for starting the L2L dissemination will help reduce redundant messages. Since every path from the root to leaf nodes at the same level have the same hop-length, transitively every message that originated at the root reaches all sibling nodes within the same bounded time period T_w, assuming no failure. If all sibling

leaf-nodes use the same duration of time to wait for the other messages, they start the L2L dissemination at the similar time, resulting in multiple redundant messages. Specifically, if one missing leaf node can be detected by $F_o - 1$ other leaf nodes, the leaf node gets $F_o - 1$ messages from other leaf nodes, in other words, there are $F_o - 2$ redundant messages.

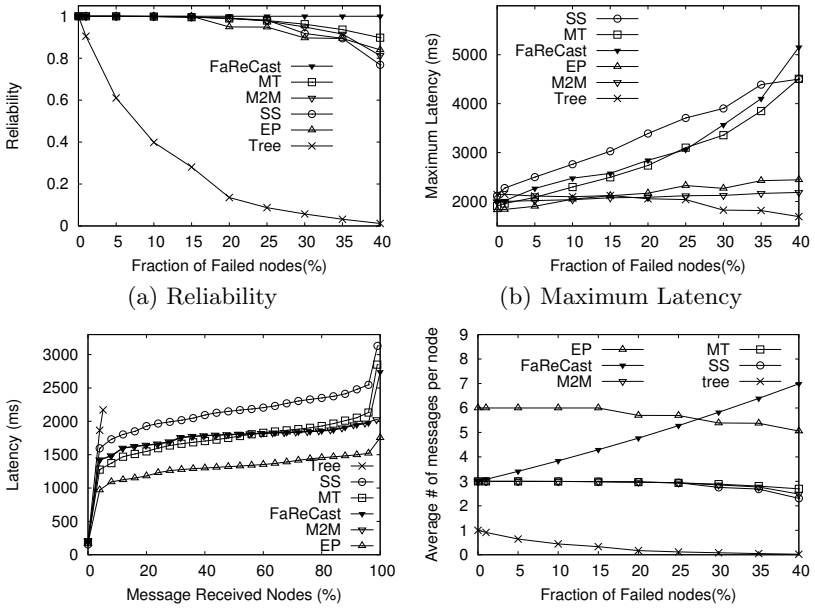
To alleviate these redundant L2L disseminations, we employ differential waiting periods at leaf-nodes, similar to exponential backoff strategies. Leaf nodes that wait for longer periods now receive messages from recovered bypassed parents (due to backup dissemination) and from other leaf nodes (due to L2L dissemination). This causes related leaf links to be masked and redundant messages are avoided. To achieve random waiting time, each leaf node generates a random integer in the range [1,N] and the waiting time is set to $N * T_W$, where T_w is the waiting time for the backup dissemination. As N increases, the variance of random waiting time increases and we more redundant messages are avoided. We set N as 4 in the evaluation.

4 Simulation based Evaluation of FaReCast

To gain a better understanding of how FaReCast works under massive failures, we evaluated FaReCast along multiple dimensions, primarily in comparison to tree-based multicast, mesh-based multicast and flooding based (epidemic) protocols. These dimensions include: (a)Reliability (the ratio of unfailed nodes that finally receive the message); (b)Latency (the delay of message delivery to unfailed nodes that are able to receive the message); and (c)Efficiency/Overhead (number of duplicated/redundant messages and its impact on system overhead).

We simulated message dissemination over networks with large number of nodes based (in the order of 100,000s) - however, several input parameters for the simulations, e.g. link latency, bandwidth availability, context switch overhead were obtained from network emulators that mirror Internet scale parameters accurately. According to outcomes of this emulation in section 5, we modeled link latency between any two nodes used in the simulation. We obtained inter-node link latencies ranging from 150ms to 200ms, the average latency being 180ms. Moreover, we set bandwidth constraints (from 100Kbps to 500Kbps) and network packet loss rates (from 1% to 5%) for each link. We also measured processing delays for sending a message to multiple children. In the implementation, each node is multithreaded and the management of links incurred context switching overheads - Modelnet[1] emulation yielded processing delays per node that varied from 3ms to 6ms.

The failure models considered accommodate *node failure* and *link failure*. *Node failures* occur primarily due to node churn - i.e. at a certain time, a node cannot communicate with some of its neighbors (children/parents) which have gone offline. Node update messages capture changes in its neighborhood - further node failures may occur while these updates happen. In addition to node failures, network congestion may cause dropped messages, leading to *link failures*. A node that does not receive a message from a parent (either due to node or link failures) classifies that parent as a *failed node*. An offline leaf node is also assumed to be a failed node. We assume that the failure can happen uniformly, that is, any node can fail with same probability. To simulate message dissemination



(a) Reliability

(b) Maximum Latency

(c) Latency Distribution (20% failure)

(d) Message Overhead

Fig. 3. Performance Comparison of FaReCasts(F_i=3,F_o=3, M2M with Backup/L2L Dissemination) with other protocols under various failures with 100,000 nodes.(SS:SplitStream, MT:MultipleTree, EP:Epidemic)

under massive failures, we took snapshots of the multicast structure just before starting the message dissemination. After applying the failure model to this snapshot, the leftover online nodes/links are only used for the dissemination in our experiments. Since we are only interested in the messages received within a short timeframe, we do not assume a structure recovery process during the dissemination. In our typical experimental setup, the network size is 100,000 nodes and the fraction of failed nodes varies from 0% up to 40%. The typical waiting time for nodes to initiate backup/L2L dissemination, T_w is set to 200ms.

4.1 Effectiveness of FaReCast under Failure

In Figure 3, we compare FaReCast with other protocols, specifically, Tree, forest/mesh based dissemination such as SplitStream[7] and MultipleTree[18][19], and flooding-based approaches such as Epidemic[9][11], to disseminate an urgent message. SplitStream categorizes nodes into streams; nodes which belong to each stream form a tree with a *fan-out*, nodes also become leaf nodes of trees for other streams - thereby creating a forest. Eventually, the *given number of streams* is the given fan-in of a node. SplitStream is used to distribute a large volume of data by sending a different chunk to each stream; in our case, the root node sends the same message to all the streams. The MultipleTree protocol generates a given number of trees with a *predetermined fan-out*. A node locates a different root position for each tree, eventually the *given number of trees* is the

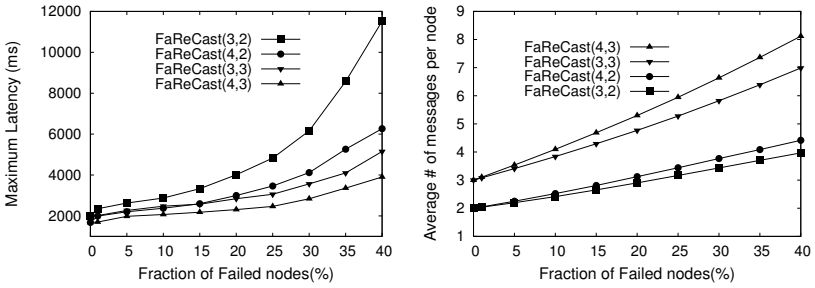
given fan-in of a node. The Epidemic protocol uses a flooding based mechanism for dissemination; every node selects a given number of *neighbor nodes* randomly. When a node gets the first message, it sends the message to all other neighbors, otherwise, it simply drops the message. To compare these protocols in a fair manner, we set the number of fan-in and fan-out as 3 and 3 for all protocols, respectively, except Tree which has only one fan-in. Also, we set the number of neighbors in Epidemic to 6, since neighbors can be parents or children.

Figure 3(a) compares the reliability of the various protocols as a function of the fraction of failed nodes. Reliability is measured as the number of nodes receiving the message over the number of online nodes. As expected, the Tree protocol is affected adversely by the failure. Protocols having multiple parents can tolerate up to 15% failures reasonably well. When the number of failures exceed 15%, FaReCast continues to achieve 100% reliability, while others including M2M (which do not use multidirectional multicasting) lose substantial reliability.

We evaluate relative latencies using two metrics - (i) maximum (worst-case) latency, i.e. latency of the last node that receives the message and (ii)latency distribution over the set of nodes that actually receive the data. Figure 3(b) plots the maximum latency as a function of the fraction of failed nodes. The Tree, Epidemic and M2M protocols appear to have lower maximum latencies - a closer look reveals that this comes at the cost of reliability (many nodes do not receive the message at all). SplitStream and MultipleTree lose reliability and exhibit increased latencies as the number of failures increase. The key reason is that the distance between the root and a node increases under failure in SplitStream and MultipleTree. M2M has a slightly reduced latency since its loop-free nature holds under failures. We observe that the maximum latency of FaReCast also increases along with failures. A deeper analysis indicates that this increase is caused by the waiting time to trigger the multidirectional multicasting. To understand general latency behavior, we plot the distribution of latency under 20% failures in Figure 3(c). We observe that FaReCast delivers the message to around 99% of the total nodes with small latencies; higher latencies are incurred by a few bypassed nodes reached by backup/L2L dissemination (more details in section 4.3). Epidemic exhibits good average latency behavior since the number of nodes reached in each dissemination step (in the early stage) is twice that of other protocols.

Figure 3(d) shows the message overhead as a function of the fraction of failed nodes. We define the message overhead as the average number of messages sent by an *online* node. In Tree, the message overhead decreases because most of nodes can not get the message under failures. M2M, SplitStream, and MultipleTree have almost same overhead (around 3); Epidemic sends twice as many message (around 6). Note that many of these protocols lose substantial reliability in spite of the high message overhead. In FaReCast, the message overhead increases along with failures. When more failures occur, more backup/L2L messages are required to rescue the bypassed nodes. For example, notice that under 30% failure the message overhead of FaReCast is similar to Epidemic; however, FaReCast achieves 100% reliability, as compared to 90% reliability of the Epidemic protocol.

In summary, FaReCast achieves 100% reliability under various failure conditions as compared to other protocols that lose reliability; all protocols exhibit comparable latencies and message overheads. In the next section, we show how



(a) Maximum Latency (b) Message Overhead

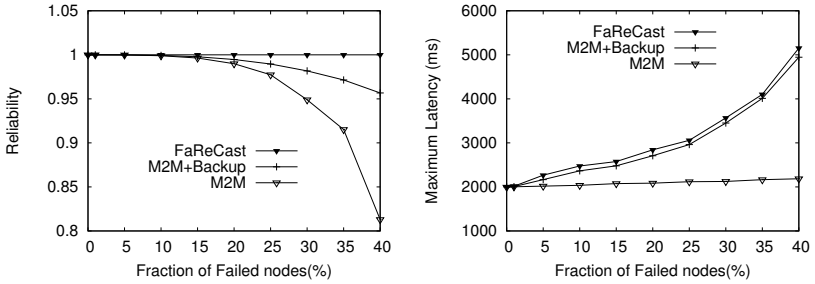
Fig. 4. Performance Comparison between FaReCasts having various fan-in and fan-out with 100,000 nodes (FaReCast(F_o, F_i)=M2M(F_o, F_i)+Backup+L2L)

FaReCast can be tuned to maintain its reliability with lower latencies and overheads.

4.2 Tuning FaReCast by adjusting Fan-in and Fan-out factor

We first explored the possibility of reducing latency and message overhead in FaReCast by adjusting the fan-in(F_i) and fan-out factor(F_o). In order to figure out the trade off between the different settings of F_o and F_i, we plot each aspect of FaReCast performance as a function of the fraction of failed nodes in Figure 4. FaReCast achieves 100% reliability regardless of F_i/F_o and the fraction of failed nodes. Even though we decrease F_i from 3 to 2, FaReCast still provides its 100% reliability. However, the latency is affected by the various F_i and F_o (See Figure 4(a)). As either F_o or F_i increases, the maximum latency decreases. Increasing F_o helps increase the number of nodes reached by each step of dissemination. Also, as F_i increases, the probability that a node detects a bypassed node with the parent information increases and a node can initiate backup/L2L dissemination faster. However, increasing either F_i or F_o causes higher message overhead (Fig 4(b)). F_i affects the message overhead significantly since each node gets $F_i - 1$ duplicated messages. Moreover, the number of leaf links used by L2L dissemination is approximately F_o*$(F_i - 1)$ – these leaf links are used when failures are detected. That is, as the failure increases, message overhead also increases and the amount of the increment is affected by both of F_i and F_o. However, since the ability to detect bypassed nodes is dependant on F_i, message overhead is more significantly affected by F_i.

What these tradeoffs indicate is that different settings can be used for the specified purposes. With big values of F_o and F_i, FaReCast supports high reliability, low latency, but incurs high overhead. With small F_o and big F_i, FaReCast supports high reliability with acceptable latency and overhead. With the big F_o and small F_i, FaReCast supports acceptable reliability and latency with low overhead - this setting can be comparable with Epidemic in terms of latency and overhead. With small F_o and small F_i, FaReCast incurs low overhead and can support acceptable reliability, but at the cost of high latency.



(a) Reliability (b) Maximum Latency

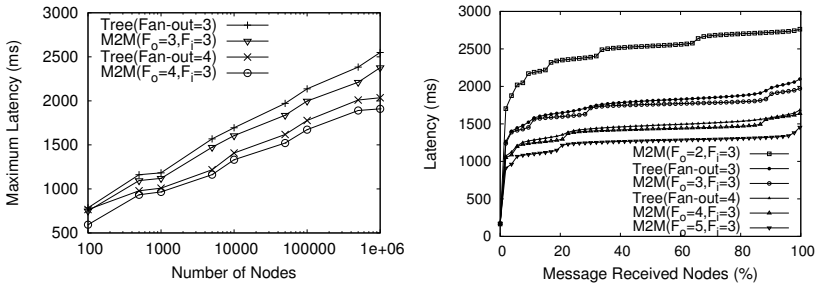
Fig. 5. Performance comparison with backup and L2L dissemination under various failures with 100,000 nodes (FaReCast = M2M+Backup+L2L) .

4.3 Effect of Backup/L2L Dissemination

We now study the impact of the various features of FaReCast on reliability, latency and overhead. We notice that a protocol that uses only the M2M ALM structure (without the multidirectional multicasting components) loses substantial reliability (loses 20% reliability under 40% failures - Figure 5(a)). The addition of backup dissemination increases the reliability to 95.5%, further addition of L2L dissemination brings the reliability to 100% under the same failure conditions. We also notice that nodes that did not receive the message with the backup dissemination were leaf nodes which were then reached by the L2L dissemination. This demonstrates that FaReCast, which includes the M2M ALM structure along with the multidirectional multicasting protocols (backup/L2L dissemination), can guarantee the very high reliability under massive failures.

In figure 5(b), the maximum latency of the M2M ALM structure slightly increases along with failures, still at the cost of some loss in reliability. When a node has multiple paths from multiple parents and there is no failure, each node obtains the first message from the fastest path. However, as the fraction of failed nodes increases, the number of multiple paths decrease, and the maximum latency under failures becomes bigger. When backup/L2L disseminations are used, the maximum latency increases rapidly. Much of this is due to the waiting time, T_w (set to 200ms in this experiment) required for initiating backup/L2L disseminations that proceed in the opposite direction to the traditional multicast. This additional latency can be further adjusted by changing the setup parameters in FaReCast such as fan-in and fan-out factor as we discussed in section 4.2.

For FaReCast to achieve 100% reliability, we need additional messages caused by backup/L2L disseminations. The message overheads of FaReCast increase linearly with the fraction of failed nodes like Figure 3(d), while the M2M ALM structure takes almost constant message overhead. The backup/L2L dissemination is triggered by the possibly bypassed nodes. According to this, as the fraction of failed nodes increases, the probability that a node is bypassed increases, and this consequently increases the number of messages FaReCast uses for the backup/L2L dissemination.



(a) Maximum Latency

(b) Latency Distribution, 100,000 nodes

Fig. 6. Latency Comparison without failure

4.4 Scalability

Since FaReCast must work under different network sizes (including very large networks), scalability studies are critical. Figure 6(a) plots the maximum latency as a factor of the network size (i.e. total number of nodes) under no failures. We compare latency between the tree-based ALM structures and M2M ALM structures under various fan-out factor (F_o) and fan-in (F_i). In Figure 6(a), we observe that the latency of all structures increases logarithmically with the number of nodes. The reason is that all the ALM structures exploit parallelism in message dissemination. As F_o increases, a node can send the message to more children at the same time, that is, the latency decreases. So, This illustrates that the basic M2M ALM structure is fundamentally scalable with respect to the network size.

Surprisingly, we note that the latencies of the M2M ALM structure are better than tree-based structures (in the absence of failures). Consider the case where the number of nodes is 100,000. The difference in latency between M2M and Tree having same fan-out factor is about 100ms. We argue that this is because in the tree structure, each node has only one message path from its single parent, and if the sole link is very slow, there is no alternative way to get the message faster. But, in the M2M ALM structure each node has multiple message paths since there are multiple parents. To take a closer look, we plot the latency of all the nodes for the network having 100,000 nodes in Figure 6(b). In the M2M ALM structure, each node can get the message from the fastest link among the multiple parents at each level (step), and most of the nodes in the same level have very similar latency to each other unlike the tree structure.

5 FaReCast System Implementation and Evaluation

We implemented the FaReCast protocol in a middleware suite and evaluated it on a campus cluster testbed with nodes connected through a wide-area Internet emulator (ModelNet). Modelnet[1] is a real-time network traffic shaper that allows for customized setup of various network topologies and provides an ideal base to test various systems for wide-area performance without modifying them. The cluster platform consists of multiple nodes that implement the emulator, the end-user nodes and the configuration management platform for FaReCast.

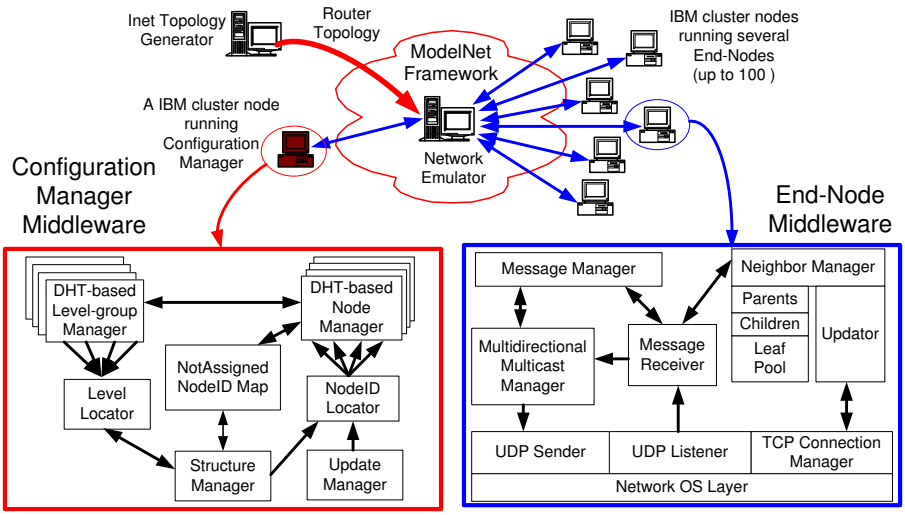
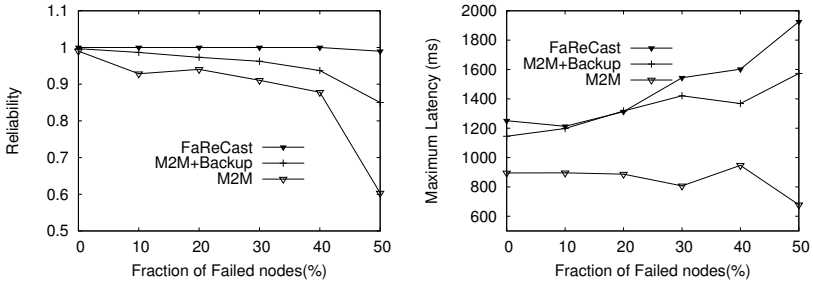


Fig. 7. FaReCast System Architecture Implemented on a ModelNet Cluster

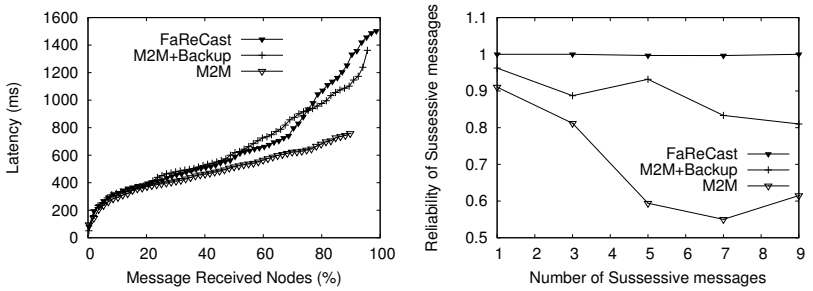
The Model emulator node has a dual 2.6Ghz CPU with 2GB RAM and runs a custom FreeBSD Kernel with a system clock at 100HZ (as required by Modelnet). End-user hosts are modeled as virtual nodes on an IBM E-server cluster with multiple (7) 900Mhz CPU with 500MB RAM. The hosts run Linux with a customized 2.6 version kernel which efficiently supports multiple threads. The hosts support Java version 1.5 and Python version 2.3.5. All hosts are synchronized to with 2msec through NTP (Network Time Protocol). All machines have Gigabit Ethernet connectivity connected by a dedicated Gigabit router.

Figure 7 illustrates the various components of the FaReCast system implementation and more detailed software architecture for two of the key modules - the configuration management module and the end-node middleware. While our initial implementation and measurements were conducted on a centralized configuration manager; we are currently working on a DHT based distributed implementation of the configuration management module.

Configuration Manager Middleware: The primary job of the configuration manager is to (a) generate and maintain the M2M ALM overlay structure given initial and updated connectivity information and (b)communicate with end-nodes to provide and gather configuration information. In the configuration manager, every participant node is mapped to a node object that contains logical information on each node - including a nodeID and node information - this includes contact information such as IP address of the node, its level, and nodeIDs of its parents/children. We are currently working on a distributed DHT-based implementation of the configuration manager where node objects are distributed based on their nodeID and managed by multiple Node Managers. Whenever the configuration manager receives the update request for parents/children information of a specific node, it gathers the nodeIDs of the requested parents/children, extracts the contact information that is mapped to the nodeIDs and supplies them with the updated information. The configuration management functionality also incorporates a Level-group Manager that maintains information on



(a) Reliability (b) Maximum Latency



(c) Latency Distribution (30% failure) (d) Sussessive Message (30% failure)

Fig. 8. Performance comparison between FareCast(F_i=3,F_o=3) under various failures with 600 nodes.

each level-group including number of sibling nodes in the same level and the parent pool. The Level group manager checks for the parent uniqueness condition when requested by a Node Manager. NotAssignedNodeID Map contains the invalidated node information. The Structure Manager and Update Manager are front-end modules that communicate with end nodes.

End-Node Middleware: An end node has two key components (i) a Neighbor Manager, which maintains information of parents, children and leaves (only if the peer has no children) of the node and (ii) a Message Manager that verifies, authenticates and processes incoming messages to the node. Neighbor Manager information is updated periodically (every 1 hour) via requests to the configuration manager, as opposed to probing the parents/children/leaves themselves. Since FaReCast is tolerant to high levels of inconsistency between the current system status and the information of the configuration manager, the frequency of the update can be low. Message Manager is responsible for authorization and authentication of messages to ensure that there are no unintended duplicate messages. When a new message arrives at the Message Manager, it generates a parent mask to sort out the parents who have sent the message. This parent mask is exploited by the Multidirectional Multicast Manager which initiates and executes the Backup and L2L dissemination. While TCP is used for control messages, i.e. monitoring and updates, we use UDP for disseminating the actual message to deliver the messages as fast as possible.

Experimental Results from Implementation: To model the underlying Internet, we used the Inet[2] topology generator to generate Internet router

topologies of 5000 routers. Inet generates a topology on a XY plane and Modelnet uses this to generate heterogeneous inter-router (and hence inter-node) latencies. Link properties were specified separately; bandwidth constraints vary from 100Kbps to 500Kbps and network packet loss varies from 1% to 5% for each link. This length of queue of each router is set to 10 (default ModelNet value) and packet losses do not include the dropped packets by queue.

Figure 8 presents some of the experimental results from the real world deployment of FaReCast. On the whole, the implementation based evaluation yielded similar overall results to that of the simulation-based evaluation. As can be observed, reliability of dissemination was close to 100% and latency and overhead increased with an increased number of simultaneous node failures. We also experimented with multiple successive messages to mimic a scenario where a series of messages were sent in succession. In a real deployment, we notice that the pure M2M policy (i.e. M2M ALM structure with traditional multicast) exhibited poor performance - this is primarily related to the UDP-based implementation that was used to disseminate the data.

6 Conclusion

We propose FaReCast, a Fast and Reliable data dissemination system for flash dissemination scenarios such as earthquake early warning where the target network is large and potentially unreliable. FaReCast constructs a forest-based M2M ALM structure where each node has multiple children guaranteeing the fast delivery of the message as well as multiple parents ensuring the reliability of the delivery. The FaReCast M2M ALM structure is accompanied by a multidirectional multicasting algorithm which sends data not only to the children but also to the selected parents or the selected siblings. Unlike ack-based communication, the multidirectional multicasting algorithms proactively figure out the direction of failures and target message resends along the direction. As compared to trees using a traditional multicasting algorithm, we observe an 80% improvement in reliability under 20% failed nodes with no significant increase in latency for over 99% of nodes. Under extreme failures, e.g. more than 30% failed nodes, we observe that existing mesh-based protocols exhibit low reliability, FaReCast achieves 100% reliability with a small increase in latency and overhead.

A natural extension of our work is to adapt the proposed techniques to support rapid, reliable dissemination over wireless mobile devices. Our future plans are to study failure scenarios in this environment more deeply and design mechanisms for addressing timeliness/reliability tradeoffs for information dissemination using mixed wired/wireless networks.

References

1. Modelnet : http://issg.cs.duke.edu/modelnet.html.
2. Inet : heep://topology.eecs.umich.edu/inet/.
3. Y. Chu, S. G. Rao, S. Seshan and H. Zhang, "A Case for End System Multicast," in Proc. of ACM Sigmetrics 2000
4. S. Banerjee, B. Bhattacharjee and C. Kommareddy, "Scalable Application Layer Multicast," in Proc. of SIGCOMM 2002

5. D. A. Tran, K. A. Hua and T. T. Do, "ZIGZAG: An Efficient Peer-to-Peer Scheme for Media Streaming," in Proc. of INFOCOM 2003
6. A. Rowstron, A. Kermarrec, M. Castro and P. Druschel, "Scribe: The design of a large-scale event notification infrastructure," in Networked Group Communication, pp.30-43, 2001
7. M. Castro, P. Druschel, A-M. Kermarrec, A. Nandi, A. Rowstron and A. Singh, "SplitStream: High-bandwidth multicast in a cooperative environment," in Proc. of SOSP 2003
8. D. Kosti C, A. Rodriguez, J. Albrecht and A. Vahdat, "Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh," in Proc. of SOSP 2003
9. V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy and A. E. Mohr, "Chainsaw: Eliminating Trees from Overlay Multicast," in Proc. of IPTPS 2005
10. B. Cohen, "BitTorrent", 2001 http://www.bitconjurer.org/BitTorrent/
11. M. Deshpande, B. Xing, I. Lazardis, B. Hore, N. Venkatasubramanian and S. Mehrotra, "CREW: A Gossip-based Flash-Dissemination System," in Proc. of ICDCS 2006
12. M. Deshpande, A. Amit, M. Chang, N. Venkatasubramanian and S. Mehrotra, "Flashback: A Peer-to-Peer Webserver for Handling Flash Crowds," in Proc. of ICDCS 2007
13. A. C. Snoeren, K. Conley, and D. K. Gifford, "Mesh-Based Content Routing using XML," in Proc. of SOSP 2001
14. M. Bansal and A. Zakhor, "Path Diversity Based Techniques for Resilient Overlay Multimedia Multicast," in Proc. of PCS 2004
15. R. Tian, Q. Zhang, Z. Xiang, Y. Xiong, X. Li and W. Zhu, "Robust and efficient path diversity in application-layer multicast for video streaming," in IEEE Transactions on Circuits and Systems for Video Technology, Vol.15, Issue.8, 2005
16. D. Frey and A. L. Murphy, "Failure-Tolerant Overlay Trees for Large-Scale Dynamic Networks," in Proc. of P2P 2008
17. R. Rodrigues and B. Liskov, "High Availability in DHTs: Erasure Coding vs. Replication," in Proc. of IPTPS 2005
18. V. N. Padmanabhan, H. J. Wang and P. A. Chow, "Resilient Peer-to-Peer Streaming," in Proc. of ICNP 2003
19. V. Venkataraman, P. Francisy and J. Calandrinoz, "Chunkyspread: Multitree Unstructured PeertoPeer Multicast," in Proc. of IPTPS 2006
20. M. Yang and Y. Yang, "A Peer-to-Peer Tree Based Reliable Multicast Protocol," in Proc. of Globecom 2006
21. B. Rong, I. Khalil and Z. Tari, "Making Application Layer Multicast Reliable is Feasible," in Proc. of LCN 2006
22. J. Zhang, L. Liu, C. Pu and M. Ammar, "Reliable Peer-to-peer End System Multicasting through Replication," in Proc. of P2P 2004
23. T. Kusumoto, Y. Kunichika, J. Katto and S. Okubo, "Tree-Based Application Layer Multicast using Proactive Route Maintenance and its Implementation," in Proc. of P2PMMS 2005
24. S. Birrer and F. Bustamante, "Resilient Peer-to-Peer Multicast without the Cost," in Proc. of MMCN 2005
25. S. El-Ansary, L. O. Alima, P. Brand and S. Haridi, "Efficient Broadcast in Structured P2P Networks," in Proc. of IPTPS 2003
26. L. Ciavattone, A. Morton and G. Ramachandran, "Standardized Active Measurements on a Tier 1 IP Backbone," in IEEE Communications, June 2003.