Computer Networks 54 (2010) 1864–1884

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

K-Tree: A multiple tree video multicast protocol for Ad hoc wireless networks

Bheemarjuna Reddy Tamma [a,*], Anirudh Badam [b], C. Siva Ram Murthy [c], Ramesh R. Rao [a]

[a] California Institute for Telecommunications and Information Technology, University of California San Diego, CA 92093, USA
[b] Department of Computer Science, Princeton University, NJ 08544, USA
[c] Department of Computer Science and Engineering, Indian Institute of Technology Madras, TN 600036, India

ARTICLE INFO

Article history:
Received 16 July 2008
Received in revised form 5 February 2010
Accepted 24 February 2010
Available online 4 March 2010
Responsible Editor: Prof. Qian Zhang

Keywords:
Ad hoc wireless networks
Video multicasting
Multicast trees
Path-diversity
Multiple description coding

ABSTRACT

In this paper, we address the problem of video multicast over Ad hoc wireless networks. Multicasting is an efficient means of one-to-many communication and is typically implemented by creating a multicast tree. Video multicasting demands high quality of service with a continuous delivery to receivers. However, most of the existing multicast solutions do not guarantee this because they are not resilient to mobility of the nodes and do not exploit error-resilient nature of recently available video coding techniques. Uninterrupted video transmission requires continuous reachability to receivers which emphasizes the usage of path-diversity. Hence, we propose a multiple tree multicast protocol which maintains maximally node-disjoint multicast trees in the network to attain robustness against path breaks. We further enhance the robustness by using the error-resilient multiple description coding (MDC) for video encoding.

We prove that finding a given number of node-disjoint multicast trees for a multicast session in a given network is NP-Hard. Then we propose a protocol called *K*-Tree which maintains the maximal node-disjointedness property of *K* trees by using a distributed online heuristic. Through extensive simulation experiments, we show how the proposed protocol improves the video quality as we use two or three trees instead of a single tree for multicasting video stream. We also show, through simulations, that the protocol efficiently, in terms of overhead, provides high quality video as compared to an existing two tree video multicast protocol and a well known mesh-based multicast protocol.

1. Introduction

With the recent advances in wireless technologies, wireless networks are becoming a significant part of today's access networks. Ad hoc wireless networks come under the category of wireless networks that utilize multi-hop radio relaying and operate without the support of any fixed infrastructure. These are zero configuration, self-organizing, and highly dynamic networks formed by a set of mobile hosts (nodes) connected through wireless links. As these are infrastructure-less networks, each mobile node should act also as a router. As a router, the mobile node represents an intermediate node which forwards traffic on behalf of other nodes. If the destination node is not within the transmission range of the source node, the source node takes help of the intermediate nodes to communicate with the destination node. Tactical communication required on battle fields, among a fleet of ships, or among a group of armored vehicles are some of the military applications of these networks. Civilian applications include peer-to-peer computing and file sharing, collaborated computing in a conference hall, and search and rescue operations. Some of the

* Corresponding author. Tel.: +1 858 822 2564.
E-mail addresses: btamma@ucsd.edu (B.R. Tamma), abadam@cs.princeton.edu (A. Badam), murthy@iitm.ac.in (C. Siva Ram Murthy), rrao@ucsd.edu (R.R. Rao).

1389-1286/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2010.02.013

applications demand for providing video service for users of Ad hoc networks, such as first responders, search and rescue teams, and military units. Such content-rich service is more substantial than simple data communications: it will add value to and catalyze the widespread deployment of Ad hoc networks. With increased sophistication of applications and growing demand for resources, it has become more challenging to provide adequate Quality of Service (QoS) for multimedia applications being delivered over Ad hoc networks.

Ad hoc wireless networks have certain unique characteristics that pose several problems in providing robust communication services [1]. First, nodes in the network do not have any restriction on mobility, therefore the network topology changes dynamically. Hence the admitted sessions may suffer due to frequent path breaks, thereby requiring such sessions to be re-established over new paths. The delay incurred in re-establishing a session may cause some of the packets belonging to that session to miss their delay targets/deadlines, which is not acceptable for multimedia applications. Second, nodes in the network maintain link-specific state information like available link capacity, link stability, and transmission delay. The state information is inherently imprecise due to dynamic changes in network topology, traffic load, and channel characteristics. Hence routing decisions may not be accurate, resulting in some of the real-time packets missing their deadlines. Finally, other peculiar characteristics of Ad hoc networks such as lack of central coordination and fixed infrastructure, error prone shared broadcast radio channel, hidden and exposed terminal problems [2], and scarcity of resources like bandwidth and battery power complicate QoS provisioning for multimedia sessions.

Multicast routing is the problem of sending data packets to more than one receiver simultaneously. Many cutting edge applications like video conferencing and digital classrooms require robust multicasting solutions for providing uninterrupted video transmission. Multicast routing plays an important role in the typical application scenarios of Ad hoc wireless networks, namely emergency search and rescue operations, disaster recovery, and network-centric military warfare in battlefields. In such hostile environments, soldiers and first responders cooperatively form groups to carry out certain tasks that require robust point-to-multipoint and multipoint-to-multipoint multimedia communication services. In battle fields, a group of cooperative soldiers moving in tankers or fighter jets form an Ad hoc wireless network whose topology changes dynamically. The major issues in designing a multicast routing protocol are as follows:

- *Robustness:* The protocol must be able to recover and reconfigure quickly from potential mobility-induced link breaks thus making it suitable for transporting multimedia traffic in highly dynamic environments such as battlefields.
- *Multicast efficiency:* A multicast routing protocol should have high multicast efficiency, which is defined as the ratio of the total number of data packets (i.e., *video frames*) received by the receivers to the total number of data and control packets exchanged in the network.
- *Control overhead:* The scarce resource availability in Ad hoc networks demands multicast routing protocol to incur very low control overhead while supporting multicast sessions.
- *Efficient group management:* Group management refers to the process of accepting multicast session members and maintaining the connectivity among them until the session expires. This process of group management needs to be performed with minimal exchange of control messages.
- *Scalability:* The multicast routing protocol should be able to scale with the number of receivers, number of sources, number of multicast sessions, and mobility of nodes.
- *Security:* Authentication of session members and prevention of non-members from gaining unauthorized information put additional security requirements on multicast routing protocol.

Based on how nodes are selected in a topology shape for establishing distribution paths among multicast group members (i.e., source(s) and receiver(s)) of a multicast session, multicast routing protocols can be divided into two main categories: tree based and mesh-based protocols. In tree based multicast routing protocols, there exists only one path between source–receiver pair. Hence, tree based protocols provide high multicast efficiency at the expense of low robustness. The tree structure can be easily broken due to movement of any participating node and packets possibly have to be dropped until the tree is reconstructed. In mesh-based multicast routing protocols, there exists mesh structures that are a set of interconnected nodes which can provide more than one path between source–receiver pair. Mesh-based protocols provide high robustness in mobile environments as they provide redundant paths from the multicast source to receivers for delivering data packets. However, mesh-based protocols sacrifice multicast efficiency in comparison to tree based protocols.

Video multicast problem has been extensively studied in infrastructure based wired and wireless networks. The solutions are centered around utilizing central coordinators in multicasting, hence it is difficult to extend them for peculiar Ad hoc network scenario. Reliable delivery of video traffic is important as there is time constraint which discourages automatic repeat request (ARQ) in case of a packet loss or packet error. Uninterrupted video delivery is not guaranteed by the existing multicast solutions for Ad hoc networks, as they have been predominantly designed for data multicast which is not sensitive to delay and delay jitter. Single tree based multicast [5,6] is not well suited for video multicast in Ad hoc networks due to the movement of nodes leading to link breaks in the tree and hence not ensuring continuous reachability. In order to increase robustness, alternate approaches like gossip [3] and mesh-based multicasting [7–9] could be considered. However, gossip based approaches are unattractive as receivers may not receive frames sequentially, hence decoding of a video frame at the receiver could be delayed unless all previous frames it (directly or indirectly) depends on are available.

Mesh-based multicast has some amount of robustness but the high frame rate of video traffic makes mesh-based approaches unsuitable as the inherent redundancy increases the data overhead substantially. Mesh schemes might give node-disjoint paths to some receivers, but they can not assure more than one path for all receivers in the multicast session. Even if multiple paths exist for all receivers in mesh structure, that is with partially overlapped paths for some of the receivers, in order to transmit a video stream which is partitioned into two independent sub-streams over different routes to receivers, mesh schemes require some forwarding nodes relaying more than one sub-stream (to reduce average hop length to receivers and control overhead) which could in turn complicate the forwarding process and increase data overhead significantly compared to multiple tree based multicast schemes. Hence mesh schemes do not have any clean mechanism to partition the video stream into multiple sub-streams and send to the receivers. The reception quality of video at receivers depends on the path from the source. If the path has breakups and makeups like any Ad hoc multicasting solution, the transient state of not receiving the video packets leads to interruptions. Hence to sustain path breaks we need some redundancy to resort to. By redundancy we mean multiple tree based multicast scheme that can provide multiple node-disjoint paths to receivers. The source has to multicast the video data using these redundant paths and minimize the number of interruptions there by improving the video quality. Hence we propose a multiple tree based multicast routing protocol for providing reliable video transport services in Ad hoc wireless networks. Multiple tree protocols not only assure more than one path to the receivers but also provide control on what to send on each of the trees.

Having the control, as we mentioned, on what to send on each of the trees is of paramount importance to video encoding techniques. Conventional single description coding (SDC) does not utilize the path-diversity in an efficient way. It provides redundant data at receiver, receiving the same stream along different paths does not improve the quality, but can lead to high data overhead. More recently error-resilient video coding techniques have been proposed to alleviate the problem of packet loss during transmission in networks. Examples of error-resilient video coding techniques are layered coding (LC) and multiple description coding (MDC) [4]. In contrast to a conventional SDC video encoder that generates a single bitstream, LC and MDC based coders encode a raw video stream into two or more sub-streams. The LC based coder encodes the raw video stream into layers of different importance, refer Fig. 1. The base layer (BL) sub-stream can be decoded alone to provide a basic quality of video while the enhancement layer (EL) sub-streams are mainly used to refine the quality of the video that is reconstructed from the BL sub-stream. Also, EL sub-streams can be used for decoding only if the BL sub-stream is available. Therefore, BL is the most important layer without which video cannot be decoded and hence requires more protection for LC to perform well. For example, BL sub-stream can be transported on more trees while a few trees might be used for carrying EL sub-streams. Unlike LC, MDC generates multiple equally important, and independent sub-streams, also called descriptions, refer Fig. 2. Each description can be independently decoded and is of equal importance in terms of quality, i.e., there is no decoding dependency between any two of the descriptions belonging to a video frame. The multiple descriptions contain complementary information so that the quality of the decoded frame improves with the number of descriptions that are correctly received. This is indeed beneficial as the ability to even partially recover a frame can impact whether its subsequent received frames are decodable or not. MDC even enhances the scalability of video multicast, because, based on availability of resources like bandwidth, battery power, and computing power, receivers can join to multiple trees to get multiple number of descriptions. Many recent advances in MDC have made it more widely accepted than LC for video transmission in wireless networks. Apart from continuous reachability, continuous delivery of decodable video data is important. MDC can trivially use the continuous reachability to provide the continuous delivery of decodable video data. On the other hand in LC, we have to ensure continuous delivery of the BL sub-stream and only then can we use the EL sub-streams. A more comprehensive comparative study on MDC and LC was done in [4].

The rest of the paper is organized as follows. In Section 2 we present related work. In Section 3 we discuss the theoretical foundations of the multiple tree multicast problem. In Section 4 we propose the *K*-Tree protocol and elucidate it with graphical examples. In Section 5 we present the simulation scenario and the simulation results. Finally, in Section 6 we conclude with a discussion on future work possible.

2. Related work

Multicasting in Ad hoc wireless networks has been studied extensively in the literature. Different approaches include constructing structures like trees [5,6] or meshes [7–9] for multicasting. But most of these solutions are not robust enough for multicasting video traffic, because they do not guarantee continuous delivery required for

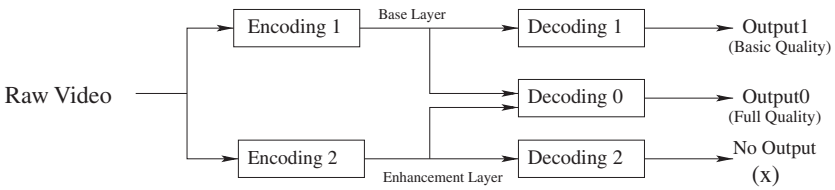

Fig. 1. Basic framework of layered coding.

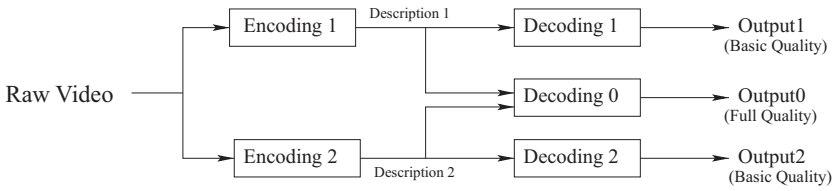

Fig. 2. Basic framework of multiple description coding.

uninterrupted video playback at receivers. Some attempts were made in the video unicasting to reduce the number of interruptions by exploiting path-diversity [10–15]. The approach they used was to analyze how to use multiple paths for robust transmission of video traffic. In [10] base layer was transmitted with ARQ while enhancement layers were sent on a different path with no protection. The authors of [11,12,14–16] describe how MDC can use multiple paths to distribute descriptions over different paths. The emphasis in all this work has been to use multiple paths for addressing video unicast (one-to-one) problem. Multi-path routing has been recently studied in [17,18]. Dynamic source routing (DSR) [19] is an on-demand source routing protocol, where the packet carries the end-to-end path information in its header. DSR obtains multiple paths for the communication pair, but because duplicate copies of RREQ (route request) packets at intermediate nodes are discarded, these paths are highly correlated [17,18], and hence are not suitable for multipath video streaming. Split multipath routing (SMR) [17] is one of the best known multipath extensions to DSR [20]. It uses a modified RREQ packets flooding scheme in the process of route discovery. The destination node returns the shortest path and another path that is most disjoint with the shortest path to the source node. A few other multipath extensions of DSR were proposed in [21]. But all these solutions have been developed for achieving robust one-to-one transmission.

One-to-one multipath solutions cannot be trivially extended to one-to-many multipath transmissions without tremendous increase in the control overhead. Robust one-to-many transmission needs structures like multiple trees. But not much work has been done in this direction. The independent-tree ad hoc multicast routing (ITAMAR) [22] creates multiple multicast trees simultaneously based on different metrics in a centralized fashion. One main problem of ITAMAR is that routing overhead might be very large to get enough information of the network to build multiple trees, and the authors only show how ITAMAR works based on perfect network information. In [23], the authors presented an architecture for supporting transmission of multiple video streams in Ad hoc networks by establishing multiple routing paths. They proposed an on-demand multicast routing protocol to transport LC video streams. The multicast routing protocol transmits layered video streaming based on a weight criterion, which is derived according to the number of receivers, delay, and expiration time of a route. But as we discussed earlier in Section 1, LC depends fully on the base layer and hence requires availability of the base layer stream for decoding enhancement layer(s). The multiple disjoint trees multicast routing (MDTMR) [24,25] protocol creates two node-disjoint trees one after the other in a serial fashion. Source triggers tree formation and trees are formed one after the other. However, this approach might not always ensure receiver connectivity. The authors of MDTMR themselves show that unless the node density is high, source cannot connect to all the receivers. Also the frequent flood of the network leads to a tremendously high overhead. Our protocol is aimed at constructing multiple maximal node-disjoint multicast trees in a distributed fashion with lesser overhead in ensuring connectivity to all the receivers. A preliminary version of *K*-Tree protocol was appeared in [29]. In this work, we thoroughly extended that one adding theoretical analysis, additional performance results on its scalability, and comparison results with MDTMR protocol and a well known mesh-based multicast routing scheme (ODMRP [9]).

Network coding techniques can [26] complement multiple tree multicast schemes and help to further improve multicast efficiency, end-to-end latency, and network throughput. In network coding, a common intermediate node of multiple unicast sessions sends out a coded packet that is a linear combination of two or more packets it received from its neighbors (i.e., each packet belongs to a different session and comes from a distinct neighbor). When a receiver gets this coded packet and one or more original packets (either it already has that original packet as it being the source of that packet or received it via another path from the source), it decodes it for obtaining original packets sent by the source node. Network coding helps to increase network throughput by reducing number of channel contentions and packet transmissions. We could employ network coding in multiple tree multicast schemes at shared nodes (forwarding nodes in more than one tree) to further improve multicast efficiency and network throughput. However, network coding requires that each shared node should not be part of more than one path for any receiver [27]. In other words, multiple tree based multicast session can have shared nodes, but all receivers should have node-disjoint paths from the multicast source. Otherwise, receiver node gets only coded packets and will not be able to decode them. This kind of approach to network coding can be termed as network coding with coding-oblivious multicast routing [28] as we first construct multiple trees and then look for coding opportunities. The most systematic approach to network coding is coding-aware multicast routing where in we select overlapped paths to different receivers (which has no effect on robustness as these paths carry different sub-streams for different receivers) for increasing coding opportunities.

However, it is very challenging to design such coding-aware multicast routing schemes due to the additional constraint on nature of shared nodes compared to our *K*-Tree protocol. To the best of our knowledge, there exists no coding-aware multicast routing protocols for Ad hoc wireless networks.

3. Graph theoretic approaches

As already pointed out, an efficient approach to robust multicast communication is constructing multiple multicast trees and sending one MDC description on each tree. The trees used in multicast session must be as disjoint as possible to obtain reliability in case of breakdown of one or more trees. Hence we have to find out node-disjoint (meaning forwarding nodes in each tree are distinct) trees for multicast communication. Theoretically we can have as many node-disjoint trees as the network allows. But resource constraints at nodes impose some upper bounds on the number of such trees the network can afford. Hence there is a trade-off between video quality and resource usage. In such a case we need to construct and maintain a given number of node-disjoint multicast trees in the network. Also based on the video encoding and streaming options available at the multicast source, not all receivers might want to connect to all the sub-streams from the source. Hence each receiver has a list of trees that it wants to participate in. These trees have to be node-disjoint for robustness. For example if the source is using MDC for video encoding and it is streaming different descriptions through different trees, say 3 trees, and due to receiver heterogeneity, all receivers need not connect to all 3 trees. Hence based on the video encoding and the receiver heterogeneity, the set of trees that a receiver wants to participate in changes. Hence we need to maintain maximal node-disjoint multicast trees where receivers need not be a part of all the trees. A node that is receiving in some subset of trees can participate in other trees as forwarding node. Further, a node that is receiver in one multicast session can participate in other multicast sessions as an intermediate node.

Feasibility of constructing a given number of node-disjoint multicast trees may not be always possible. Maximum number of node-disjoint trees possible in a network would give us an idea on the feasibility of finding a given number of node-disjoint trees. A simplification to the problem is to look at the case when there is only one receiver. A reliable video multicast (in this case a unicast) would require the search and maintenance of a given number of node-disjoint paths between the source and the receiver. A simple extension to the problem is finding the maximum number of node-disjoint paths possible. If we can answer this question we can throw light on the answer to the previous question on finding maximum number of node-disjoint trees. This is the well explored disjoint path problem or the 0–1 maximum-flow problem [30–34]. It has been proved in [34] that this problem of finding the maximum number of node-disjoint paths between two given nodes can be solved in polynomial time. Our objective is to find out node-disjoint trees between the source and receiver nodes such that each tree contains at least all those nodes which wish to participate as receivers (leaves) in that tree. That is each receiver has a list of trees that it wants to participate in. The list of trees that a receiver wants to participate in is represented through a bit vector. If the *i*th bit in the bit vector is set then the receiver wants to participate in the *i*th tree. The next section formally describes the problem that has to be solved to get the trees as required by the receivers. We introduce it as a packing problem as we are trying to pack multiple trees into the network.

3.1. K-Tree Packing Problem

In this section we introduce *K*-Tree Packing Problem (KTP) [35]. Here we assume that a multicast session contains only one source node and formulate KTP problem. An instance of KTP consists of a graph $G(V,E)$, where V is the set of nodes and E is the set of edges in the network; a set of nodes, $T = \{s : s \epsilon V\} \cup \{r_i : r_i \epsilon V, 1 \leqslant i \leqslant l\}$, where s is the source, the r_is are receivers, l is the number of receivers, and each receiver has a K-bit vector, B_i. The objective is to find K intermediate-node (non leaves) disjoint trees of G, say $X_j,\ 1 \leqslant j \leqslant K$, such that receiver r_i is a leaf node in tree X_j if the jth bit in the K-bit vector of receiver r_i is set to one. We now prove that KTP is NP-Hard when we need to find the maximum number of intermediate-node-disjoint trees out of these K trees that are needed to be found. We also prove that KTP is fundamentally harder for directed graphs. In case of directed graphs we prove that KTP is NP-Hard even for $K = 2$. We prove our hardness result via a reduction from Edge Disjoint Path Problem (EDP). EDP is considered one of the "classic" NP-Hard problems. An instance of EDP has a graph $G(V,E)$, and a multi-set $T = \{(s_i,t_i) : s_i, t_i \epsilon V\}$, of source ($s_i$s) sink ($t_i$s) pairs. The objective is to connect as many of these pairs as possible using edge disjoint paths. Past work on EDP includes [36–40].

3.1.1. Overview of proof

We prove our hardness result via a reduction from EDP. EDP is NP-Hard and also hard to approximate as shown in [39,40]. In Section 3.1.2 we show how to translate an instance F of EDP into a simple instance H of KTP. In Section 3.1.3 we show how to take solution to KTP in H back into a solution to EDP in F. In Section 3.1.4 we tie all the analysis together. In Section 3.1.5 we show how our analysis can be extended to give hardness result for directed graphs via a reduction from directed subgraph homeomorphism [41].

3.1.2. Construction of a simple KTP instance

Consider an undirected graph G with vertex (node) set V and edge set E, and a set $T = \{(s_i,t_i) : s_i, t_i \epsilon V, 1 \leqslant i \leqslant k\}$ of source–sink pairs. This constitutes an instance of EDP, F. We convert it to an instance of KTP as follows. For each source node s_i introduce a new dummy source node s'_i. Similarly for each sink node t_i introduce a dummy sink node t'_i. Now add edges between corresponding dummy

sources and the real sources and similarly between corresponding dummy sinks and real sinks. Now call the pairs (s_i', t_i') as the revised source–sink pairs. It should be noted that s_is and t_is need not be distinct. In such a case we handle in the following way. For each non-unique source or sink, replicate the node and add edges from this replicated node to all the nodes that the original node is connected to, for example if $s_a = s_b$ then replicate s_a and then add edges from this duplicate node to all those nodes to which s_a was connected. Now construct a new graph G' with vertices as the edges of G (with revised source–sink pairs). Add edges between vertices of G' if the corresponding edges in G are incident upon a vertex in G. Each vertex $n_{i,j}$ in G' therefore represents an edge joining vertices i and j in G. Let us denote the edges in G connecting the dummy sources and sinks to real sources and sinks as s_i''s and t_i''s in G'. Introduce a new node s into the graph G'. Add edges joining s to each of the nodes s_i''. Hence the source s is now connected to exactly k vertices. Consider the set $T' = \{s\} \cup \{s_i'' : 1 \leqslant i \leqslant k\} \cup \{t_i'' : 1 \leqslant i \leqslant k\}$, call s as the source and the set $\{s_i'' : 1 \leqslant i \leqslant k\} \cup \{t_i'' : 1 \leqslant i \leqslant k\}$ as the set of receivers $\{r_i : 1 \leqslant i \leqslant 2k\}$. Here we assume that $r_i = s_i'', 1 \leqslant i \leqslant k$ and $r_i = t_i'', k+1 \leqslant i \leqslant 2k$. Define for each receiver r_i, a k-bit vector, B_i, as follows. $B_i, 1 \leqslant i \leqslant k$ is a k-bit vector with all 1's except for the ith bit and $B_i, k+1 \leqslant i \leqslant 2k$ is a k-bit vector with all 0's except for the $(i-k)$th bit. Now the graph G' with all the new nodes, new edges together with (V, E) and the set T' and k-bit vectors B_is constitutes an instance of KTP, H, with $K = k$. The objective is to find out intermediate-node-disjoint trees of G' which have the leaves as r_is, corresponding to the k-bit vectors B_i. Fig. 3 shows an instance of EDP in graph G. Fig. 4 shows dummy sources and dummy sinks added to G. Fig. 5 shows G' with source s constituting an instance of KTP (we renamed the vertices in G to illustrate the construction of G').

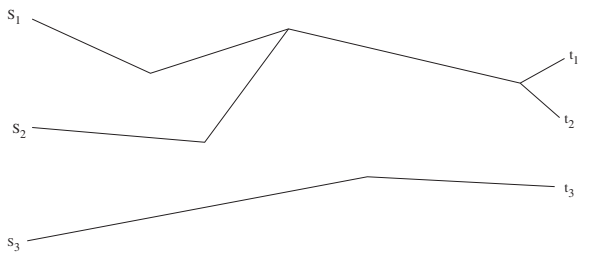

Fig. 3. Graph G with an EDP with $k = 3$.

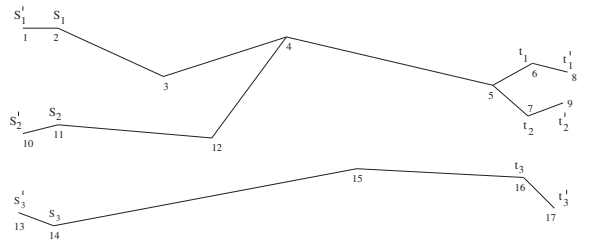

Fig. 4. Graph G with dummy sources and sinks.

3.1.3. From KTP to EDP

Here we show how the solution to H is also a solution to F. Consider a solution to H. The solution to H has as many trees of the given K as possible with those nodes as the leaves whose bits in B_i are set. Consider the solution to be satisfying only a few of the K required ones. We claim that this set of trees that are satisfied also encompasses a solution to the EDP, F. Consider the nodes s_i''s, each of these wants to be a leaf in all the trees except the ith tree (k-bit vector B_i for these defines it). Contrary to this, each of the nodes t_i'' wants to be a leaf only in the ith tree. Hence the only connectivity possible between the source node s and the node t_i'' must be through the node s_i''. Hence the solution to H has a certain number of node-disjoint paths between the pairs (s_i'', t_i''). Now consider a solution to F. It has the maximum number of source–sink pairs, (s_i, t_i)s, connected via edge disjoint paths. These paths represent maximum number of edge disjoint paths also between the source–sink pairs (s_i', t_i'). Hence they represent maximum number of node-disjoint paths in G' for the source–sink pairs (s_i'', t_i''). Let the number of trees in the solution to H be k_h and the number of paths in the solution to F be k_f. Solution to F is trivially a solution to KTP in H. This is because each of the nodes s_i'' can directly connect to s for connectivity in trees except the ith tree

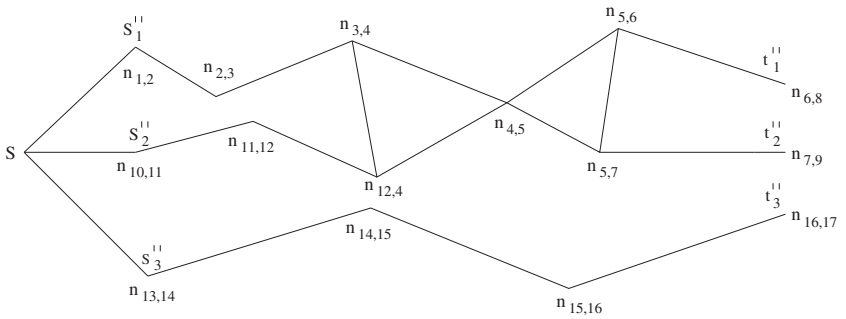

Fig. 5. Graph G' with source s forming a KTP.

and act as an intermediate node for the ith tree, t_i'' reaches source through the node s_i'', which is the only node which can connect in X_i to the source. Hence the solution to F satisfies k_f trees of the wanted K trees. Since k_h is the maximum number of trees possible we have that $k_h \geqslant k_f$. Also since solution to H represents a certain number of node-disjoint paths between source–sink pairs (s_i'', t_i'') (as t_i'' can reach the source s only through s_i''), k_h can be at most the maximum number of pairs, (s_i'', t_i'')s, we can connect with node-disjoint paths, which is in fact k_f. Hence $k_h \leqslant k_f$. Hence a solution to H maps directly to a solution to F.

3.1.4. Hardness of KTP

We here connect our analysis in the previous sections and prove that KTP is at least as hard as EDP which is NP-Hard. Given an algorithm to solve KTP, and an instance of EDP, convert the EDP instance into a simple instance of KTP using the method in Section 3.1.2 and because of the mapping possible given in Section 3.1.3, solution to KTP is same as the solution to EDP. Hence KTP is at least as hard as EDP which is NP-Hard. Hence the problem of finding maximum number of node-disjoint multicast trees according to a given configuration, where receivers are leaves in the trees they want to participate in, is NP-Hard.

3.1.5. KTP for directed graphs

In this section we use the result that EDP is fundamentally hard in directed graphs to prove that KTP is fundamentally hard in directed graphs. In directed graphs, it is established that given two source–sink pairs (s_1, t_1) and (s_2, t_2), it is NP-Hard to determine node-disjoint paths between the sources and corresponding sinks [41]. Scaling the analysis done in Sections 3.1.2, 3.1.3, 3.1.4 we get that KTP is NP-Hard even for $K = 2$ in directed graphs.

3.2. Online heuristic

In this section we look into an online heuristic solution to find a given number of node-disjoint multicast trees. We seek to find out maximally node-disjoint multicast trees (allowing some amount of overlap) in the network and maintain them for transporting video substreams given by MDC coder. This overlap leads to reduction in robustness but as proven even with the complete knowledge of network topology we may have no polynomial time solution to our problem of finding node-disjoint trees. Moreover, it is well known that maintaining complete network topology in Ad hoc networks involves very high overhead as the topology keeps continuously changing due to unrestricted mobility of the nodes. Hence we need a distributed approach to construct multicast trees and not a centralized one. Also in a realistic multicast scenario receivers keep joining and leaving the multicast group as time progresses. Hence a centralized heuristic approach would not work efficiently as the heuristic has to applied again and again to account for receivers joining and leaving. The heuristic has to be online to accept receiver requests to join and to leave on the fly. These two requests are sufficient to model such a network where there is no mobility or mobility is so low that it would never affect the topology of the network. Mobility is a characteristic feature of Ad hoc networks and hence the topology of the network keeps changing. Hence to model this kind of dynamic system, we also need to include a request called the movement request. This request changes the network topology, in a certain way. The change might be reflected in the multiple multicast tree system, that is, it might lead to tree break or it might create possibilities for a better system (lesser overlap). Hence each movement request changes the adjacency matrix and creates possibility of maintenance or betterment of the multiple tree system.

Heuristic approaches can help to reduce the overlap by minimizing the number of common nodes among different trees. Hence we propose a distributed online heuristic solution to minimize the number of intersections, while serving the requests online, among the trees it has to maintain. Let us henceforth call such a system as K-Tree system where we are trying to maintain K maximally node-disjoint multicast trees. A join request would involve passing some control messages in the network and getting the new K-Tree system. A leave request would involve percolating up the trees a delete message until a node with out-degree more than one (which indicates minimum number of receivers under its branch of tree) in that tree is met.

A movement request would involve changing a node position according to the request. For simplicity let us assume movements occur in bursts. The reason for this assumption is that it would be difficult to model a continually moving node theoretically. Hence we assume that a node moves in bursts of distance and these movement bursts are instantaneous. These bursts partially capture what can happen in a real network. A network system does not wait until a node has become stationary. Even mobile nodes participate in tree formations and repairs. Hence a good model should consider this and make necessary assumptions to consider participations during the node movement. Thus we assume all movement requests are in bursts of small distances.

It is already easy to see that, that online algorithm is the best one which serves the request with minimum message passing, adding the least possible number of forwarding nodes, and making the least increase in the number of intersections among the trees in the system. But some thought would directly prompt us to the fact that there is mobility. Minimization of forwarding nodes can lead to a problem. When a node moves from one place to another, in the network, it might lead to a substantial number of receivers dependent on this to go orphan for the time until the system takes care of them. Hence it is in the best interest not to go for trees trying to minimize the number of forwarding nodes. At the same time it would not be appropriate to include many forwarding nodes. This is because a tree system with more forwarding nodes leads to more movement requests leading to maintenance requests. This observation comes from the fact that not all movement requests can lead to

maintenance requests. But as number of nodes increases, the fraction of movement requests leading to maintenance requests also increases.

Hence the problem now boils down to finding an algorithm which serves the requests by minimizing the number of forwarding nodes and minimizing the control overhead and also minimizing the fraction of movement requests leading to maintenance requests and also the number of intersections among the trees. We try to solve this problem by a weight (a positive quantity used to judge a node) model.

Weight model: To attain node-disjointedness with a weight model we incorporate a penalty system. We penalize a node for participating in more than one tree. The penalty inculcated increases with the number of trees the node participates in for a given multicast session. We associate a weight with each node. The penalty increases the weight of the node as the number of trees it participates in increases. The system, for each request of multicast session, has to reduce the total increase in weight of the nodes in the network.

In assigning weights to the nodes it must be noted that a single node serving two trees must weigh more than two nodes, which are each serving a tree. In a similar way, it has to be noted that choosing a single node for l trees should increase the weight more than the case when l different nodes are chosen for the l trees. Hence the minimum weight of a node participating in l trees should be at least l times the weight of a node participating in $l-1$ trees. The following cost model satisfies these conditions.

- The weight of a node in the graph is zero if it participates in no trees for a given multicast session.
- The weight of a node in the graph which participates in one tree is x, for some $x > 0$.
- The weight of a node in the graph which participates in l trees, w_l, $l \leqslant k$ is $l * w_{l-1} + y$ for some $y > 0$.
- The cost of a path is the sum of the weights of the nodes in the path.
- The cost of an operation on the graph is the positive change in the total weight of the graph.

The term y can be used to quantify the number of intersections among the trees while node participations can be captured by the term x. If x is large compared to y then, in the total weight of a path, the contribution made by node participations will dominate the contribution made by the number of intersections and vice-versa.

Bringing in other parameters like local contention, node bandwidth, remaining battery lifetime, link latency, number of multicast sessions a node participate in, etc into weight calculation of forwarding nodes could help us in finding more efficient trees. But obtaining values for some of these additional parameters at the routing layer requires cross-layer solutions, which is beyond the scope of our work. However, weight function proposed in this work provides a general framework for bringing in other relevant parameters easily into node's weight calculation.

4. The *K*-Tree protocol

In this section we present our *K*-Tree protocol which is modeled as an online algorithm based on the weight model discussed in the previous section. The protocol defines how each of the requests (Join, Leave, Movement) is handled while trying to minimize the total weight of the nodes in the network. Minimizing the weight is the heuristic idea behind reducing the number of shared nodes among the trees. A scenario for the protocol would be to maintain maximal node-disjoint multicast trees while processing a request sequence. Each request in the sequence triggers an operation on the network which determines the changes that need to be done to the network in order to serve the request and maintain the multiple tree system. The changes are usually accomplished by passing control messages in the network. These messages contribute to the control overhead of the protocol. In order to reduce this overhead we relax our condition that each receiver should be a leaf in the tree that it wants to participate in. By allowing a receiver to be a forwarder, overhead can be reduced to an extent in realistic scenarios. For example, a forwarder who wants to later become a receiver can directly become so without any message passing.

We elucidate the protocol by partitioning it into two phases. The first phase describes how a receiver node joins the multiple tree system. The next phase describes how the maintenance of the multiple tree system is done. Both the phases illustrate how the weight model is used to minimize the number of shared nodes in a greedy fashion. Here we assume that multicast source advertises by other means, such as periodic advertisements and pre-sharing using public key cryptography [42], to all nodes in the network it's multicast group address and number of trees (i.e., MDC descriptions) in which it is streaming the multimedia content.

4.1. Multiple tree initialization phase

The tree initialization phase is initiated by the receivers. Each node in the network can participate in the *K*-Tree system either as a receiver or a forwarder. Hence each node represents its participation in the *K*-Tree system using a *K*-bit vector, *kvp*. If the *j*th bit in *kvp* is set then the node is either a receiver or a forwarder for the *j*th tree, X_j. When some node wishes to become a receiver, it uses a *K*-bit vector, *kvj*, to represent the trees that it wishes to join. If the *j*th bit is set in *kvj* then the node wishes to be a receiver in tree X_j. A node which wishes to become a receiver in X_j can trivially become so, if it is already a forwarder in X_j. Thus the receiver needs to join only those trees where it is not currently participating as a forwarder. Hence it forms a *K*-bit vector, *kvs*, which represents the trees for which it is not a forwarder but it wants to join. If *j*th bit is set in *kvs*, then the node is not a forwarder in tree X_j but it wants to join in that tree now. It then floods the network with a *Join Request* control packet expressing its wish to join in the trees represented by *kvs*. Hence a node wanting to become a receiver follows the Algorithm 1, Join Group.

Algorithm 1. Join Group

```
node ← node requesting to join;
kvp ← K-bit vector representing the participation of
   node in the tree system as a forwarder;
kvj ← K-bit vector representing the trees that node
   seeks to join;
kvs ← K-bit vector representing the trees that node
   has to send with Join Request control packet;
needed ← 0;
for i ← 1 to K do
  kvs[i] ← 0;
  if kvj[i] = 1 and kvp[i] = 1 then
    Become a receiver in that tree using the existing
   path as a forwarder;
  end if
  if kvj[i] = 1 and kvp[i] = 0 then
    kvs[i] ← 1;
needed ← 1;
  end if
end for
if needed = 1 then
  Send Join Request control packet with bit vector kvs;
  Set a timer with value MAX_REPLIES_TIMER;
end if
```

The flooded *Join Request* control packet reaches different nodes in the network. Any flooding technique needs a control mechanism to avoid the situation where nodes continuously keep exchanging the same packet over and over again. Hence a node when receives a *Join Request* control packet has to check if the packet has been already processed by the node before. To make this identification, each *Join Request* control packet carries the full path that it has traversed. If the node is in the path traversed by the packet, then the packet is dropped. Otherwise the node has to check if it can reply to the sender of this *Join Request* control packet. A reply can be sent only in the case when the node receiving the packet participates in any of the trees sought by the *Join Request* control packet. The receiving node has to check its *kvp* to find out to how many trees it can send a reply. The node then replies using a *K*-bit vector, *kvr* to the sender of *Join Request* control packet using the *Join Reply* control packet. If *j*th bit in *kvr* is set then the node is replying for the tree X_j, meaning the node participates in X_j and also that the *Join Request* control packet sender wants to join X_j. The *Join Reply* control packet contains the complete path information to the source, that is, the nodes and their participation vectors in the path to the source for each of the trees represented by *kvr*.

As long as there are trees for which the node cannot reply (as it may not be participating in them at the moment), the *Join Request* control packet has to be forwarded to other nodes. That is, even if there is one tree for which a reply has not been sent by this node then the node needs to forward the *Join Request* control packet with the vector *kvf*. If the *j*th bit in *kvf* is set then it means the node cannot reply to the *Join* control packet sender for tree X_j as it does not participate in it. Algorithm 2, Process *Join Request* control packet, explains how a node that receives a *Join* control packet processes it.

Algorithm 2. Process *Join Request* Control Packet

```
node ← node that received the Join Request control
   packet;
kvp ← K-bit vector representing the participation of
   node in the tree system as a receiver/forwarder;
kvj ← K-bit vector representing the trees that Join
   Request seeks;
kvf ← K-bit vector representing the trees that the
   Join Request has to be forwarded with;
kvr ← K-bit vector representing the trees that node
   can send Join Reply for;
path ← path traversed by Join Request control packet;
repneeded ← 0;
fwdneeded ← 0;
if node is in path do
  exit;
end if
for i ← 1 to K do
  kvf[i] ← 0;
  kvr[i] ← 0;
  if kvj[i] = 1 and kvp[i] = 1 then
    kvr[i] ← 1;
    repneeded ← 1;
  end if
  if kvj[i] = 1 and kvp[i] = 0 then
    kvf[i] ← 1;
    fwdneeded ← 1;
  end if
end for
if repneeded = 1 then
  Form Join Reply with the complete path information
   to source for the trees represented
by kvr;
  Send Join Reply control packet back in the path with
   kvr as the vector;
end if
if fwdneeded = 1 then
  Append node to path;
  Forward Join Request control packet with kvf as the
   vector;
end if
```

An example of tree initialization is shown in Figs. 6 and 7. Fig. 6 shows a multicast session in which 4 receivers (nodes 3, 8, 12, and 18) have already joined along the two ($K = 2$) trees. It also shows weights of the nodes, determined based on nodes' current participation vectors. Node 5 is a new receiver who wants to join into the session. As shown in Fig. 7, node 5 broadcasts a combined *Join Request* packet which is in turn forwarded by non-participating nodes 4, 6, and 7 for both trees and a current forwarding node 2 for one of the trees. Since node 2 is already a forwarder for Tree 2, it can directly reply for that tree while it has to forward *Join Request* for Tree 1.

The receiver when receives a *Join Reply* packet has to collect the reply and store all the paths that are obtained due to this reply. For each tree X_j, for which a *Join Request* control packet was sent, the receiver stores all paths that

are obtained. It uses caches to store the paths obtained through *Join Reply* control packets. Each receiver that has an outstanding *Join* control packet, meaning there is *Join Request* control packet which has been sent whose timer has not expired yet, maintains a cache called ReplyBuf which stores a certain number of replies per tree.

After the *MAX_REPLIES_TIMER* timer expires, the receiver has to choose those paths to each of the trees that it sought, which add the least amount of weight to the multiple tree system, as this would be intuitively minimizing the number of intersections among the trees. It evaluates all possible path combinations that can be chosen to reach each of the trees. But it chooses those paths which add the least amount of weight to the system. Hence a receiver upon the expiry of the timer follows the Algorithm 3, Paths Selection.

Algorithm 3. Paths Selection

```
kvj ← K-bit vector representing the trees that the
   Join Request sought;
ReplyBuf ← Array which can store MAX_REPS number
   of replies per tree;
combo ← Array representing all possible combinations of paths in
   K trees;
/* combo[i][j] stands for the path to source in jth tree of
   ith combination of paths */
numCombos ← Number of combinations possible;
minAddedWeight ← INTEGER_MAX;
for i ← 1 to numCombos do
   nlist ← array of distinct nodes in the paths of
   combo[i];
   /* nlist now has the nodes in combo[i] with their
   present participation vectors */
   oldWeight ← calcWeight(nlist); /* refer Algorithm 4
   */
   size ← size of nList;
   for j ← 1 to size do
      for k ← 1 to K do
         if nList[j] is in path combo[i][k] then
            Set kth bit in the participation vector of node
      nList[j];
         end if
      end for
   end for
   /* nlist now has the nodes with their updated
   participation vectors according to paths of combo[i]
   */
   newWeight ← calcWeight(nlist);
   addedWeight ← newWeight-oldWeight;
   if addedWeight < minAddedWeight then
      minAddedWeight ← addedWeight;
      minCombo ← combo[i];
   else if addedWeight = minAddedWeight then
      Out of minCombo and combo[i], select the
      combination that gives maximal node-disjoint trees
      (i.e., the combination having minimum number of y
      terms in the added weight);
   end if
end for
Send Join Ack control packets along the chosen
   combination of paths, minCombo;
```

In continuation to our example tree initialization, as shown in Fig. 8, new receiver node 5 gets *Join Reply* packets from current tree nodes: node 3 replies for both trees and node 2 replies for tree 2 and node 1 replies for tree 1 and node 8 replies for both trees. Each *Join Reply* packet carries the complete path information from the source *S* to receiver node 5. Let paths retrieved from reply packets be $p_{11}, p_{31}, p_{81}, p_{22}, p_{32}, p_{82}$, where p_{ij} represents the path from the source to receiver node 5 via node i for tree j. Hence, the new receiver node 5 has 9 combinations (i.e., 3 replies for tree 1 multiplied by 3 replies for tree 2) of paths to choose from.

The x and y values of the weight model affect the nature of trees constructed by Algorithm 3, Paths Selection. If x is large compared to y, Algorithm 3 selects paths that add least number of new forwarders into the *K*-Tree system. In other words, it tries to construct trees with more number of intersections among them. But such shared trees lack robustness against path breaks. If y is large compared to x, Algorithm 3 leads to construction of maximal node-disjoint trees. Eventhough node-disjoint trees are more robust against path breaks, they increase average hop length to the receivers participating in the *K*-Tree system.

Algorithm 4. calcWeight(*nList*)

```
/* nList is an array of nodes along with their
   participation vectors */
size ← size of nList;
weight ← 0;
for i ← 1 to size do
   weight ← weight + weight of the node nList[i];
   /* Weight of a node participating in l trees is w_l */
   /* w_l = l × w_{l-1} + y;   1 < l ⩽ K */
   /* w_1 = x ; Refer Section 3.2 for more details on the
   weight model */
end for
return weight;
```

The receiver now finally has to send the *Join Ack* control packets to acknowledge the nodes in the paths chosen. It simply unicasts *Ack* control packets to its immediate parents in each of the trees and they in turn percolate it up. The nodes receiving *Join Ack* packets establish forwarding states and initialize timers for tree maintenance and tree tear down. Tree maintenance is triggered when either a data packet or *KeepAlive* control packet does not arrive from the parent in time *KEEP_ALIVE_TIME*. The multicast source starts sending *KeepAlive* control packets, passively by piggybacking in data packets or explicitly when the source is not sending data temporarily. Forwarders in turn keep forwarding these packets down the trees. The *KeepAlive* also has a *K*-bit vector representing the trees it wishes to refresh, it carries along with it the path information for these trees, that is, the nodes and their participation vectors. *KeepAlive* packets are also used to update the path information by the nodes in the trees.

In continuation to our example tree initialization, new receiver node 5 follows the Algorithm 3, Paths Selection

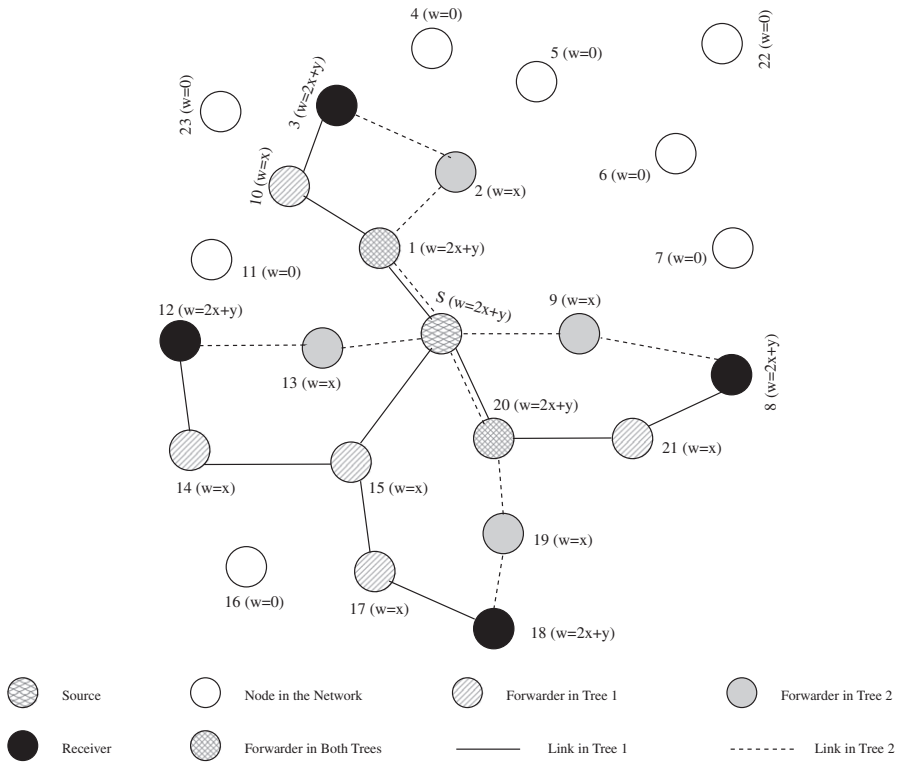

Fig. 6. *K*-Tree initialization: *K*-Tree depicting a multicast session with two trees. Nodes 3, 8, 12, and 18 have already joined along the two trees.

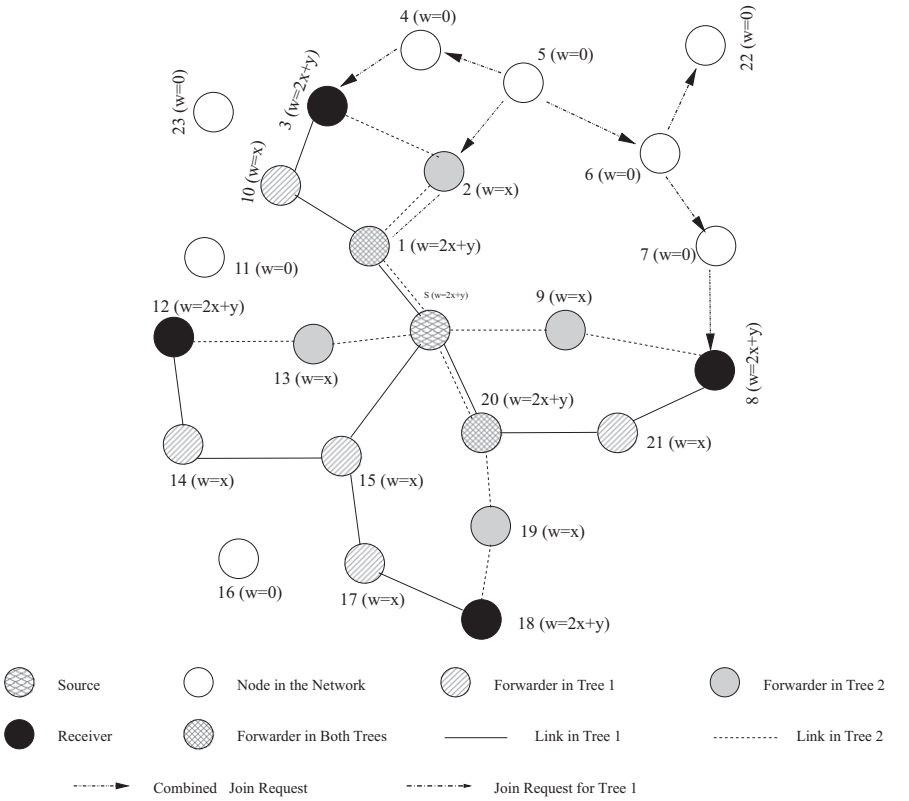

Fig. 7. *K*-Tree initialization: *K*-Tree depicting a multicast session with two trees. Node 5 broadcasts a combined *Join Request* packet into the network.

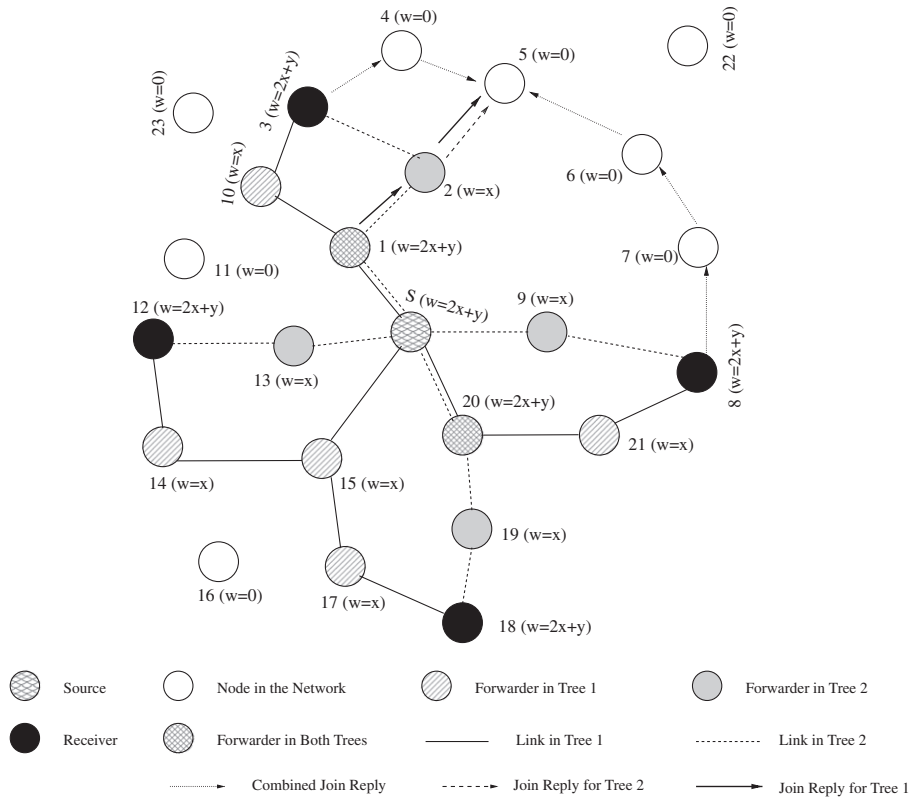

Fig. 8. *K*-Tree initialization: *K*-Tree depicting a multicast session with two trees. Node 5 gets six *Join Reply* packets (three per each tree).

Table 1
Comparison of path combinations.

Combination	Added weight
p_{11} and p_{22}	$3x+2y$
p_{11} and p_{32}	$4x+2y$
p_{11} and p_{82}	$5x+2y$
p_{31} and p_{22}	$3x+y$
p_{31} and p_{32}	$4x+2y$
p_{31} and p_{82}	$5x+y$
p_{81} and p_{22}	$4x+y$
p_{81} and p_{32}	$5x+y$
p_{81} and p_{82}	$6x+3y$

and decides that p_{31} and p_{22} is the best combination as shown in Table 1. Then node 5 unicasts *Join Ack* packets along paths p_{31} $(5 \rightarrow 4 \rightarrow 3)$ and p_{22} $(5 \rightarrow 2)$ for trees 1 and 2, respectively (refer Fig. 9). Fig. 9 also shows that the weights of nodes 4 and 5 have increased by x (from 0 to x) and $2x+y$ (from 0 to $2x+y$), respectively due to joining of node 5 into the multicast session. However, there is no increase in the weight of node 2 (forwarder in tree 2) as the new receiver node 5 is joining via node 2 only for the tree in which node 2 is already part of (i.e., tree 2). Similarly, there is no increase in the weight of node 3 (receiver) as it is already part of both trees. Finally, Fig. 10 shows revised *K*-Tree system after new receiver node 5 joined into the multicast session.

4.2. Multiple tree maintenance phase

The multiple tree maintenance is done in a soft state manner. Whenever, a receiver or a forwarder gets a data packet or explicit *KeepAlive* control packet in a tree then it refreshes its timers for that tree. When the timer expires, that is if there is no data packet or a *KeepAlive* control packet for a *KEEP_ALIVE_TIME* time then the node initiates this process. The node initiating this process sends out a *Join Request* control packet for those trees alone which are broken. It may happen that a few trees are broken simultaneously. Hence whenever there is a tree where there is no packet since *KEEP_ALIVE_TIME* time, all the other trees are tested for timeouts, and all the trees that are broken are found out and a combined join request for those trees is sent. This is done using the *Join Request* packet by setting only those bits in the *K*-bit vector. Because it is a maintenance operation, the tree might just be nearby, so the time out for starting the processing of the received *Join Reply* packets is maintained much lesser than the one corresponding to the tree initialization process. Also since the tree is expected to be nearby, the *Join Request* packet will initially be sent out with a time-to-live (*ttl*) of 3. If the node initiating the tree maintenance fails to get any *Join Reply* packets within a specific amount of time, then it sends a *ttl* free *Join Request* packet. An example of tree maintenance is shown in Fig. 11. When the node 1 moves away, the link in tree 1 between node 10 and node 1 is broken.

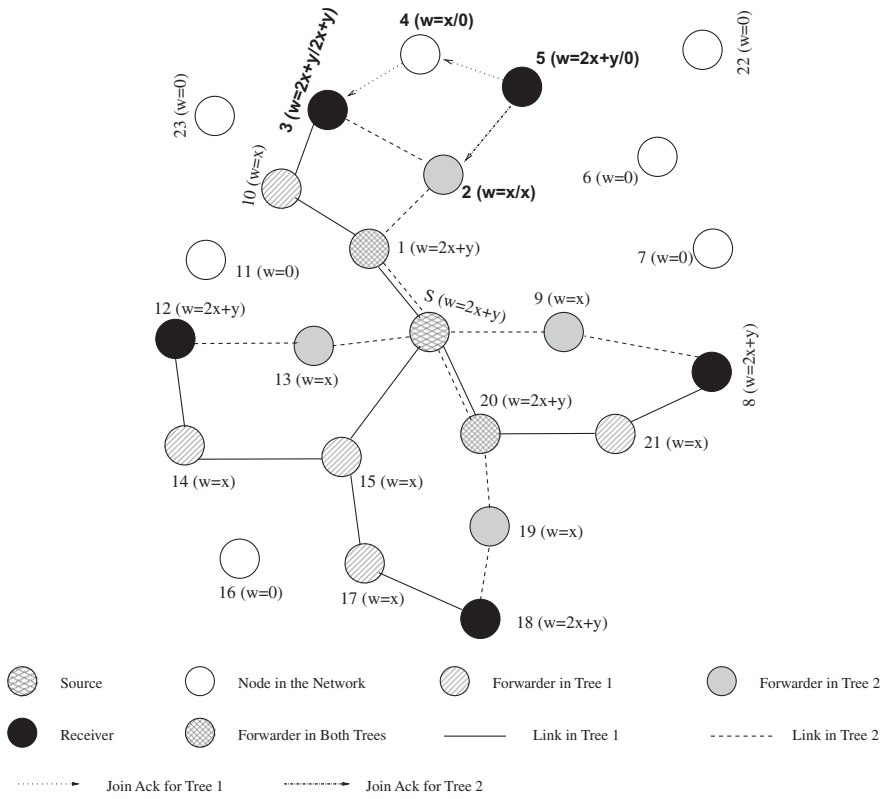

Fig. 9. *K*-Tree initialization: *K*-Tree depicting a multicast session with two trees. Node 5 unicasts *Join Ack* packets along paths p_{31} ($5 \rightarrow 4 \rightarrow 3$) and p_{22} ($5 \rightarrow 2$).

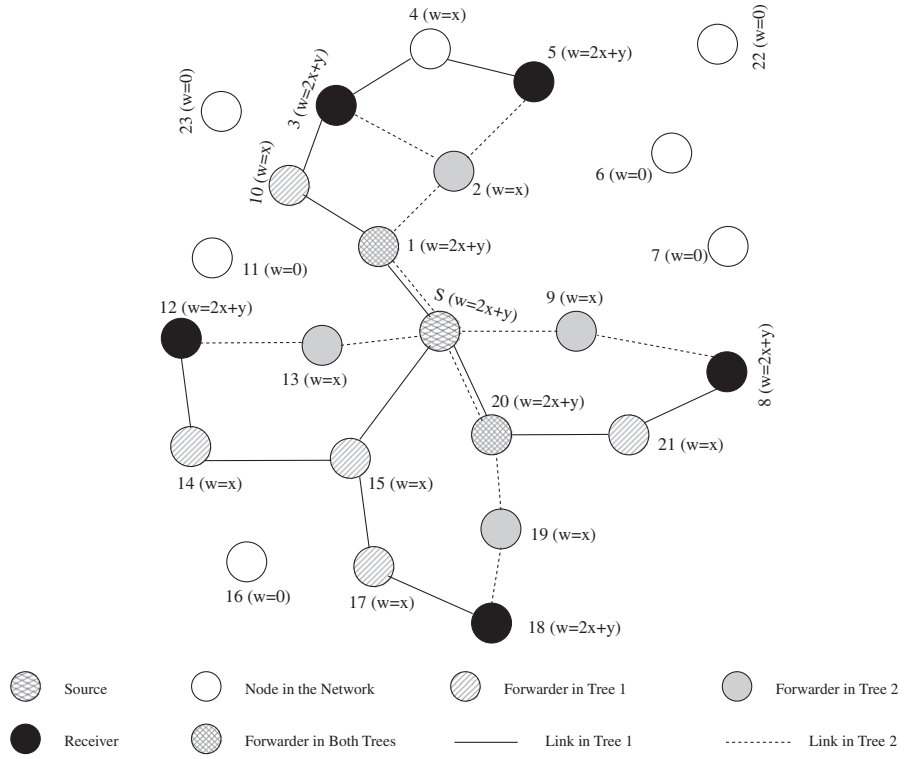

Fig. 10. *K*-Tree initialization: *K*-Tree depicting a multicast session with two trees. New receiver node 5 has joined into multicast session.

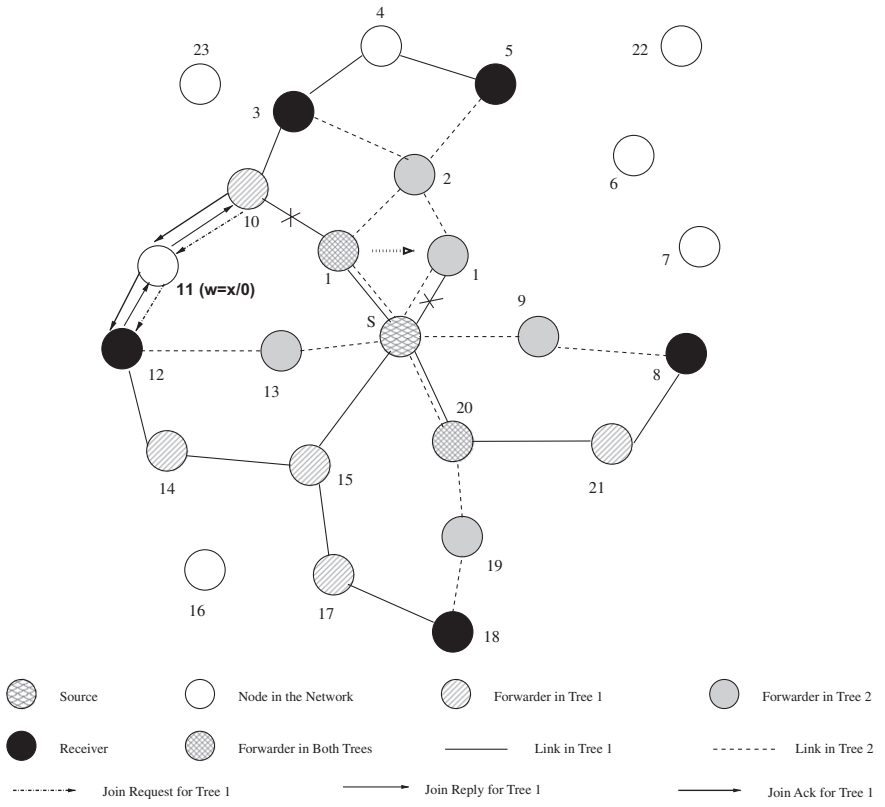

Fig. 11. *K*-Tree maintenance: *K*-Tree depicting a multicast session with two trees. Link between node 10 and node 1 is broken in tree 1. The downstream node 10 triggers the maintenance process.

The downstream node 10 detects the breakup and triggers the maintenance process. It broadcasts a *Join Request* control packet with a *ttl* of 3. This control packet is replied by node 12. Then an exchange of *Join Reply* and *Join Ack* packets reconnects the node 10 to tree 1. Fig. 12 shows reconstructed tree 1 after the maintenance process.

K-Tree protocol does not have explicit *leave* messages, instead forwarders prune themselves away if they detect that they do not have any downstream children. Each multicasted data packet carries a field containing the parent of the node that forwarded it. This field serves to passively acknowledge the parent of that node to continue forwarding data. Forwarder discards the forwarding state when it does not overhear passive *KeepAlive* control packets from the downstream nodes for *TREE_TEAR_DOWN_TIME*. However, since pure receivers do not forward data packets, they need to explicitly acknowledge their parents by periodically unicasting an explicit *KeepAlive* control packet. This way a forwarder not having any downstream receivers prunes itself away from the *K*-Tree system.

5. Performance results

We use the simulation model based on NS-2 [43] to evaluate the performance of proposed *K*-Tree protocol. The Monarch research group in CMU has extended the NS-2 network simulator to include physical layer, link layer, and MAC layer models to support multi-hop wireless network simulations. The IEEE 802.11 DCF is used as the underlying MAC layer protocol. The radio model is based on the Lucent/AgereWaveLAN/OriNOCO IEEE 802.11 product, which is a shared-media radio with a transmission rate of 11 Mbps, and a radio range of 250 m. The random waypoint model is used to model mobility of the nodes. Each node moves with some constant speed (i.e., minimum speed is equal to maximum speed) with zero pause time. To change the mobility level of the network we change the speed from 3 m/s to 18 m/s in steps of 3 m/s. The main purpose of studying high node speeds is to evaluate the dynamics of mobile Ad hoc network and demonstrate that multiple tree structure is constructed and maintained efficiently by our *K*-Tree protocol. We can think of some futuristic applications in network-centric military warfare scenarios in which high-moving war plane jets forming a mobile Ad hoc network for streaming multimedia content. We may not have people moving at this high speeds in any civilian applications. In each run, we simulate a 65 node Ad hoc wireless network within a $1350 \times 1200\ m^2$ area. Each simulation is 900 s long and the results are averaged over 30 runs. We randomly choose one sender and 10 receivers in each simulation who join and leave the multicast trees randomly during the 900 s interval.

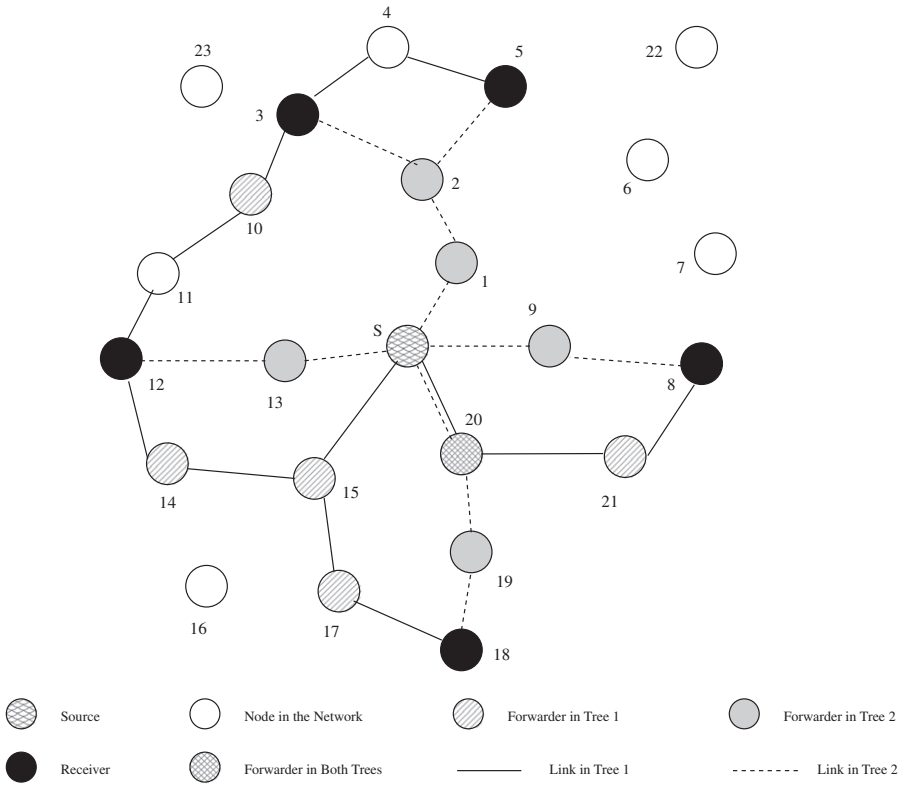

Fig. 12. *K*-Tree maintenance: *K*-Tree depicting a multicast session with two trees. It shows reconstructed tree 1 after the maintenance process.

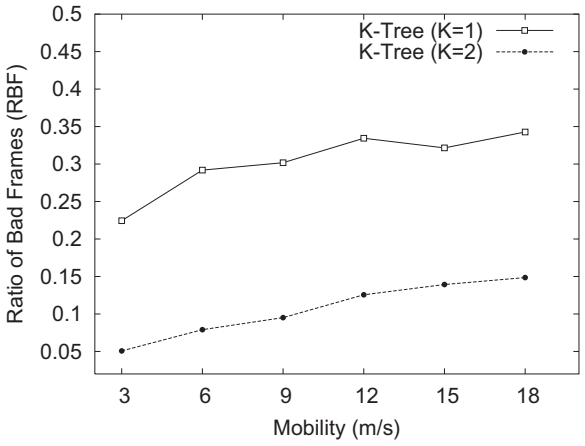

Fig. 13. Variation of RBF vs. mobility for *K*-Tree ($K = 1$) and *K*-Tree ($K = 2$).

5.1. Metrics

We use the following metrics to evaluate and compare different settings of our protocol with an existing multiple tree based multicast protocol and a mesh-based multicast protocol.

(1) *Ratio of bad frames (RBF)*: It is the ratio of number of bad frames (totaled for all receivers) vs. the number of frames that have to be received in all, by all the receivers. We define a frame as bad if none of the multiple descriptions of a frame are received at the receiver before the playback deadline of that frame. This is a better metric than packet delivery ratio as this captures the time constraint characteristic of the video stream.

(2) *Normalized packet overhead (NPO)*: It is the ratio of the total number of control and data packets exchanged in the network over the total number of data packets received by the receivers. This is used to illustrate the forwarding efficiency and also maintenance ability.

(3) *Normalized control overhead (NCO)*: It is the ratio of the total number of control packets exchanged in the network over the total number of data packets received by the receivers. This is used to illustrate the overhead of the protocol and also the maintenance ability.

5.2. Simulation results

In our simulations we have chosen the following values for the parameters of *K*-Tree protocol. $x = 1$, $y = 3$, *KEEP_ALIVE_TIME* = 250 ms, *MAX_REPLIES_TIMER* = 200 ms, and *MAX_REPS* = 4.

5.2.1. Effect of increasing K

Here we show how the protocol seamlessly scales from one tree to two trees and three trees without doubling and

tripling the overhead and at the same time improving the video quality at the receivers. We compare the results for $K = 1$ against $K = 2$. In this case we use a two description coding of video. The total video transmission rate is fixed at 48 Kbps with 8 frames per second. MDC coder generates 2 descriptions with 24 Kbps in each description. We have kept packet rate as 8 packets per second with each packet having a size of 3 kilo bits in each description. When $K = 1$ we send both the descriptions on the same tree and when $K = 2$ we use one tree per description. Similarly we compare $K = 1$ and $K = 3$. Here we split the video into three descriptions each of 8 packets per second with packet size of 2 kilo bits. The playback deadline for a frame is 120 ms after it was created. We use the same deadline for all our simulations.

- Comparison of K-Tree ($K = 1$) and K-Tree ($K = 2$): Fig. 13 shows the expected decrease in the RBF when K is increased for different mobility values. RBF has almost fallen down by 60% when moving from $K = 1$ to $K = 2$. As expected RBF values for both the cases decrease with increasing mobility. Fig. 14 shows the expected increase in the overhead due to increase in the number of forwarders. But it has to be noted that the NPO has not doubled, yet a significant improvement has been achieved in RBF. This directly follows from the fact that the protocol, K-Tree, uses bit vectors in common join and maintenance control packets to reduce overhead.
- Comparison of K-Tree ($K = 1$) and K-Tree ($K = 3$): Fig. 15 shows the expected decrease in the RBF when K is increased. RBF has almost fallen by 65% when moving from $K = 1$ to $K = 3$. Fig. 16 shows the expected increase in the overhead due to increase in the number of forwarders. NPO has not tripled, yet there is a substantial decrease in RBF. This again shows importance of the bit vectors in control packets for reducing the overhead.

It is to be noticed that in Figs. 13–16, there is a marginal decrease in RBF/NPO values at 15 m/s compared to 12 m/s. It could be explained as follows. When a receiver detects that its upstream forwarding node moves away and the path to multicast source is broken, it initiates route maintenance by sending a Join Request packet with ttl of three. Due to the random movement of nodes in the network terrain (we use Random Way Point mobility model in our simulation studies and consider 1350 × 1200 rectangular terrain with boundaries) and high mobility of nodes, sometimes broken paths get repaired automatically when forwarders bounce back after hitting terrain boundary. This automatic repair can happen quite quickly when nodes move at high speeds and hence can lead to slight decrease in RBF/NPO at high speeds as observed in above plots.

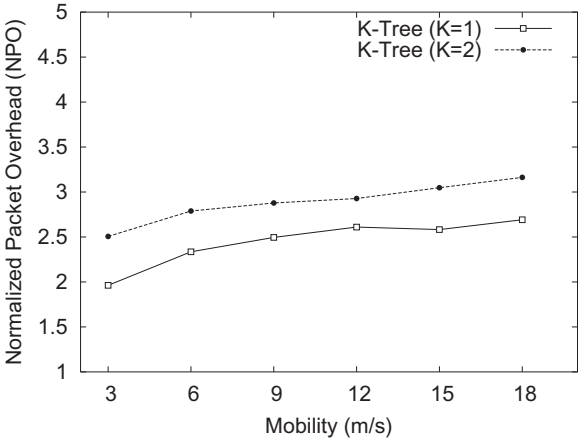

Fig. 14. Variation of NPO vs. mobility for K-Tree ($K = 1$) and K-Tree ($K = 2$).

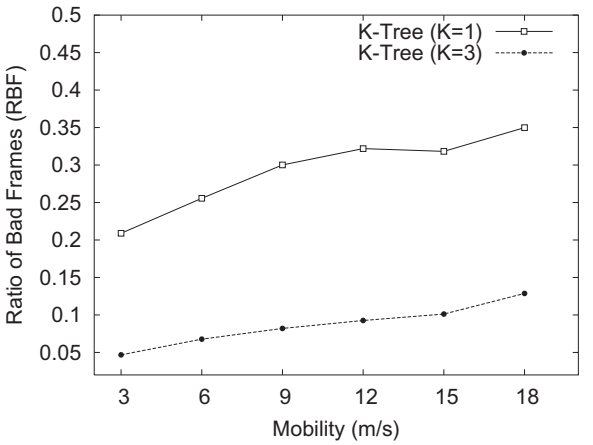

Fig. 15. Variation of RBF vs. mobility for K-Tree ($K = 1$) and K-Tree ($K = 3$).

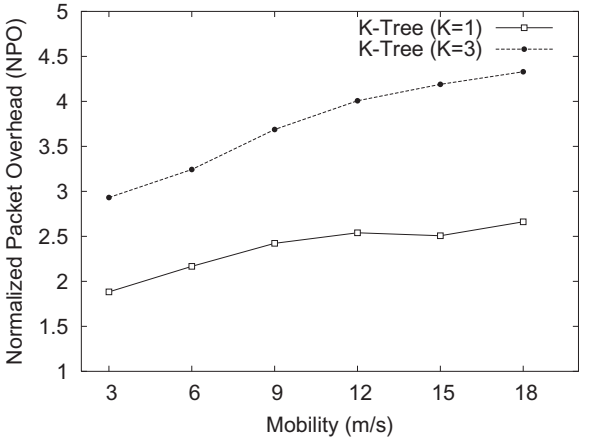

Fig. 16. Variation of NPO vs. mobility for K-Tree ($K = 1$) and K-Tree ($K = 3$).

5.2.2. Saturation of K-Tree protocol

Here we show beyond what value of K the performance of k-Tree protocol saturates. To find out its saturation point, we conducted experiments by varying K from one to four. Similar to our previous experiments, here also we set the total video transmission rate fixed at 48 Kbps with 8 *fps*. But we use different MDC coders for different number of trees such that number of MDC descriptions sent in the multicast session matches with number of trees

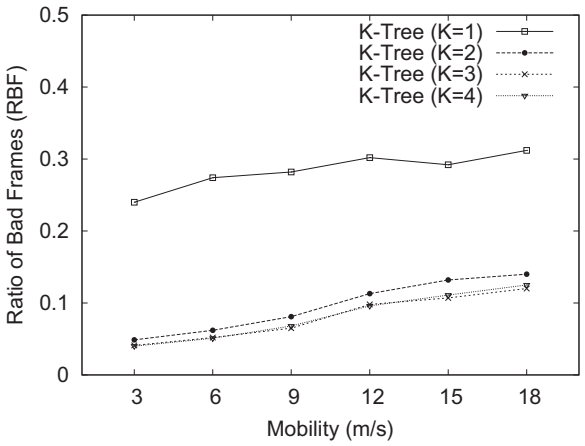

Fig. 17. Variation of RBF vs. mobility for *K*-Tree (*K* = 1–4).

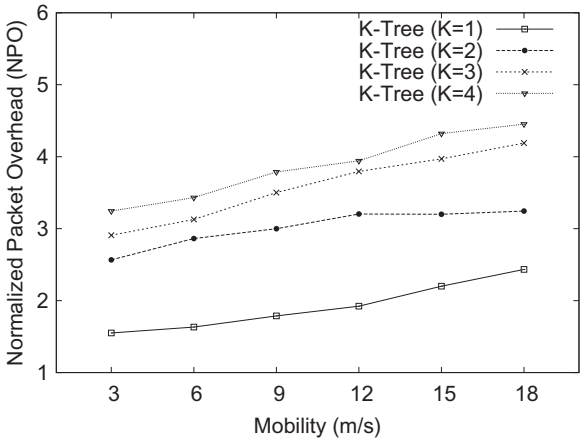

Fig. 18. Variation of NPO vs. mobility for *K*-Tree (*K* = 1–4).

constructed. In other words, we send only one MDC description per tree. In case of *K*-Tree ($K = 1$), we use SDC (single description coding) coder which basically generates single description stream at 48 Kbps (8 packets per second with packet size of 6 Kb). For all other *K*-Tree schemes ($K = 2$–4), we use MDC coders which, respectively, generate K descriptions at $\frac{48}{K}$ Kbps in each description ($8 \times K$ packets per second with packet size of $\frac{6}{K}$ Kb).

Figs. 17 and 18 show variation in RBF and NPO vs. mobility for different *K*-Tree schemes, respectively. As we have seen earlier, here also we observe that RBF reduces significantly only when moving from $K = 1$ to $K = 2$ without doubling NPO. If we go for $K = 3$, RBF reduces further to some extent by incurring some additional control overhead, but as we see from the plot it saturates at $K = 3$. This happens because for higher values of *K*, due to lack of node-disjoint trees in the network, the resulting trees contain a lot of shared forwarding nodes. The shared nodes face higher contention and have to handle higher traffic load for higher values of *K* (packet rate increases linearly with *K*). Further, if a forwarding node moves away, it could result in multiple tree breaks. Hence we conclude that increasing *K* beyond three does not lead to any performance benefits in our *K*-Tree protocol.

5.2.3. Comparison of K-Tree ($K = 2$) with ODMRP and MDTMR

Here we compare *K*-Tree with a well known mesh-based multicast routing, ODMRP [9] and an existing multiple tree protocol, MDTMR [25]. ODMRP builds a multicast mesh by periodically flooding the network with control packets to create and maintain the forwarding state of each node, when the source node has packets to send. It uses a forwarding group concept where in only a subset of nodes forwards the multicast packets via scoped flooding. MDTMR constructs two node-disjoint trees one after the other. It then uses a two description coding to use the two trees for sending the descriptions. It maintains the trees by periodically flooding a *joinReq* control packet. Receivers reply with *joinAck* control packets. Disjointedness is maintained when a node forwarding a *joinReq* for one tree does not forward *joinReq* for the other tree. Since there is no explicit repair involved, frame rate might be affected and also due to the periodic flood there might be high overhead in the system. Also the authors of [25] themselves point out that unless the network is dense all receivers might not participate in the trees leading to a drastic increase in average frame loss rate.

We have kept the *joinReq* flood interval for MDTMR as 6 s. In this case we use a two description coding of the video. The total video transmission rate is fixed at 62.4 Kbps with 8 frames per second. MDC coder generates 2 descriptions with 31.2 Kbps in each description. We have kept packet rate as 8 packets per second with each packet having a size of 3.9 kilo bits in each description. We send one description per tree in case of MDTMR and our *K*-Tree ($K = 2$). In case of ODMRP, we send both descriptions on the same mesh structure.

Fig. 19 shows how our protocol with two trees fares against ODMRP and MDTMR protocols. MDTMR performs marginally better compared our protocol for slower speed of 3 m/s. But if we look at Figs. 20 and 21, this marginal improvement was achieved at the cost of very high control overhead (more than three times compared to our proto-

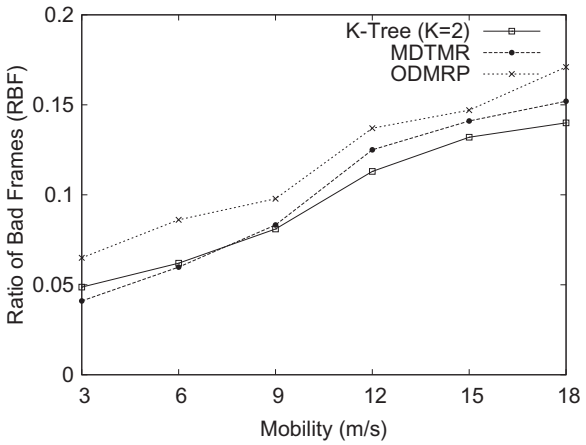

Fig. 19. Variation of RBF vs. mobility.

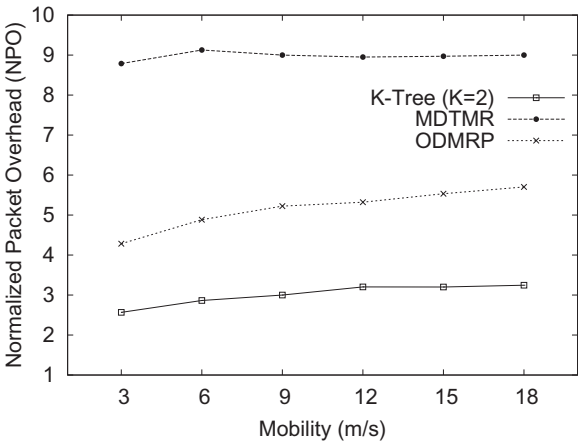

Fig. 20. Variation of NPO vs. mobility.

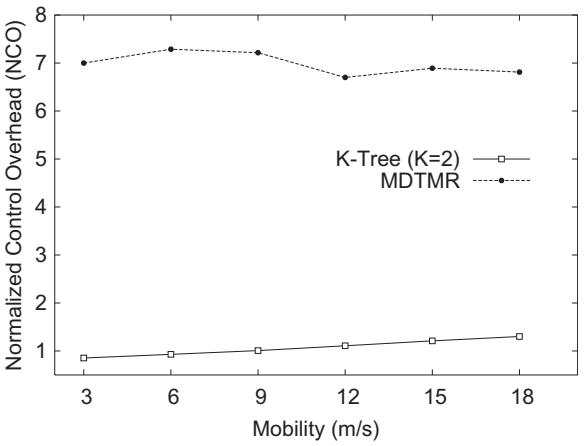

Fig. 21. Variation of NCO vs. mobility.

col). The control overheads in ODMRP and MDTMR are tremendous. MDTMR has the highest overhead due to the frequent flooding of the network to refresh the forwarding state. This is because there is no exclusive maintenance in MDTMR. The overhead would only increase with the decrease in the *joinReq* flooding interval. The authors of MDTMR protocol [25] set the *joinReq* flood interval as 3 s. Control overhead in our protocol is low because of the piggybacked *KeepAlive* packets. As the mobility increases, *K*-Tree proves to be a better protocol as the maintenance is done whenever there is a path break. This is not the case with MDTMR, in case of a path break, the receivers affected have to wait until the next *joinReq* flood to reconnect to the source. Similar to MDTMR, ODMRP also does periodic control packet flood to refresh the forwarding state. It results in higher control overhead compared to our protocol. Hence *K*-Tree outperforms ODMRP and MDTMR as the mobility increases with much lesser overhead as shown in the simulations.

Scalability of K-Tree protocol: We now illustrate scalability of our *K*-Tree protocol against MDTMR protocol which constructs trees sequentially. We measure scalability in terms of receiver join delay. The receiver join delay is defined as average time taken by a receiver to join into a multicast session or refresh it's group membership status. Fig. 22 shows join delay times of *K*-Tree protocol ($K = 1$–3) and MDTMR. The join delay times shown in this plot are obtained when nodes

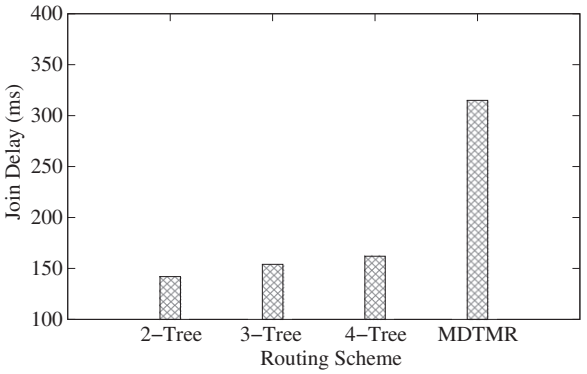

Fig. 22. Scalability of *K*-Tree protocol.

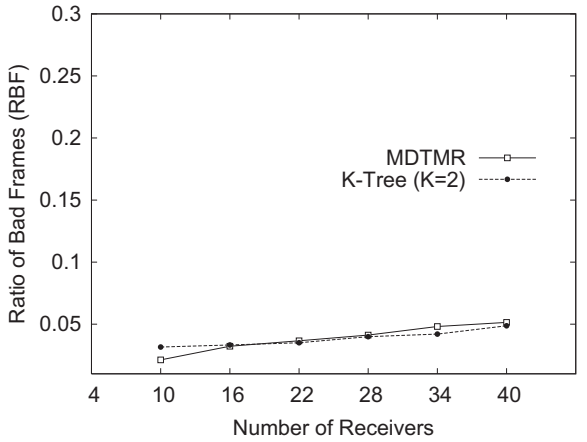

Fig. 23. Variation of RBF vs. number of receivers (mobility = 0 m/s).

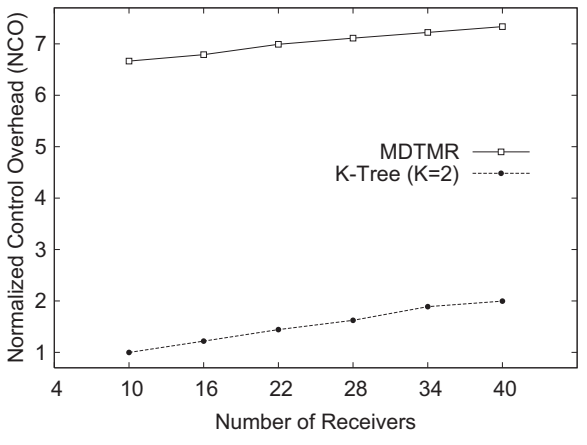

Fig. 24. Variation of NCO vs. number of receivers (mobility = 0 m/s).

move at 3 m/s. Since receivers use combined *Join Request* packets in our *K*-Tree protocol to join into multiple trees parallelly, join delay does not increase linearly with *K*. But serial MDTMR constructs two trees in sequential fashion and hence receivers face higher join delays.

Effect of receiver density: Figs. 23 and 24 show the comparison of MDTMR vs. *K*-Tree for different number of receivers in a static scenario. As the number of receivers increases, the control overhead (NCO) in both the protocols steadily increases but the overhead in MDTMR is higher than that of *K*-Tree. The RBF of both the protocols is almost the same, *K*-Tree being slightly better as number of receivers increases. This is due to the earlier mentioned fact that MDTMR fails to connect all the receivers to the source when the network is not very dense.

Figs. 25 and 26 compare MDTMR and *K*-Tree at a low mobility of 6 m/s. They show that RBF and NCO increase with the number of receivers in the presence of mobility. Unlike in *K*-Tree protocol, overhead in MDTMR is affected by only the number of receivers and not the mobility as

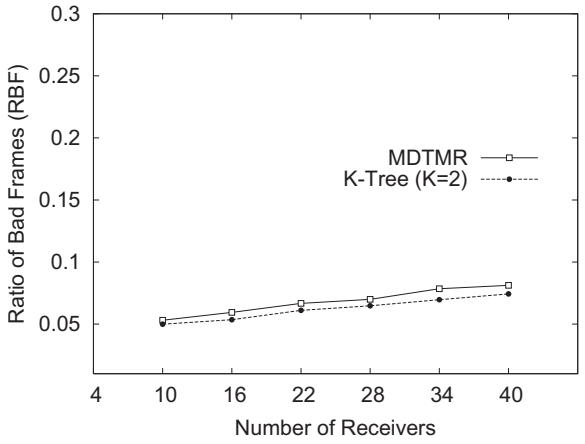

Fig. 25. Variation of RBF vs. number of receivers (mobility = 6 m/s).

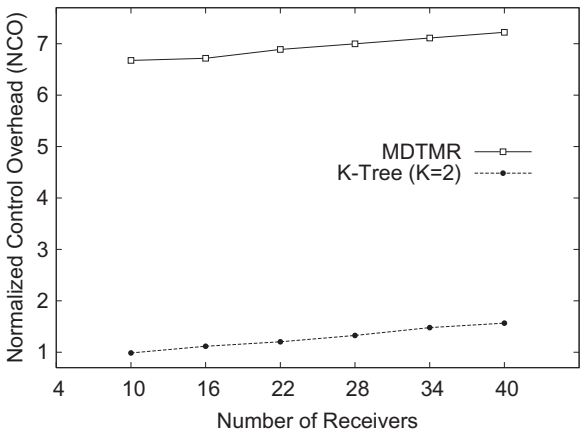

Fig. 26. Variation of NCO vs. number of receivers (mobility = 6 m/s).

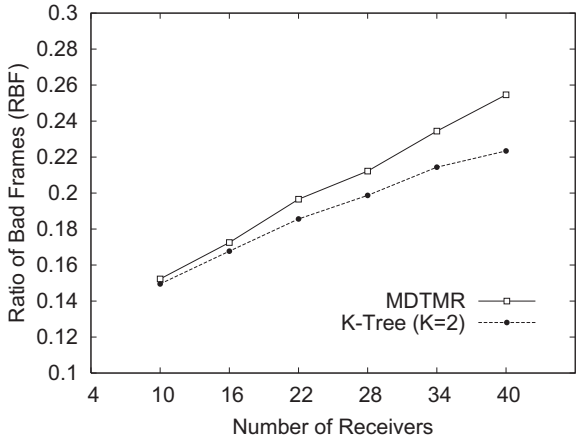

Fig. 27. Variation of RBF vs. number of receivers (mobility 18 m/s).

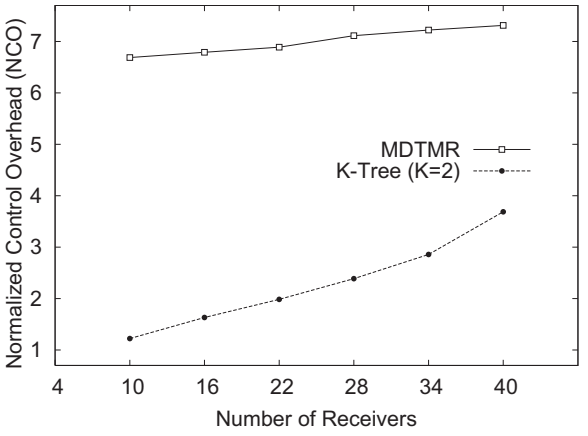

Fig. 28. Variation of NCO vs. number of receivers (mobility = 18 m/s).

there is no explicit maintenance phase. The overhead steadily increases in case of *K*-Tree due to the fact that the size of the tree increases with the number of receivers. More the number of nodes in the tree, more the probability of maintenance. Figs. 27 and 28 compare MDTMR and *K*-Tree for a high mobility of 18 m/s. Here both RBF and overhead increase with the number of receivers as expected. But the increase in RBF in case of *K*-Tree is not as rapid in case of MDTMR as *K*-Tree has an explicit maintenance phase and also *K*-Tree ensures paths to all the receivers whenever possible. The results have shown that *K*-Tree outperforms MDTMR as the number of receivers increases with a lesser overhead.

6. Conclusion and future work

In this paper we proved that finding node-disjoint multicast trees, where receivers are leaves of trees with a particular configuration, is NP-Hard. We then designed a multiple tree based multicast routing scheme, *K*-Tree protocol, which exploits path-diversity for robustness video multicasting. We have shown that the protocol scales to two or three trees without doubling or tripling

the overhead, respectively. But as already discussed, the path-diversity may increase the number of forwarders and hence may increase the data forwarding overhead. Further work might involve finding correlated but yet trustable paths instead of finding disjointed paths. That is to allow correlations but in a way to reduce overhead and also at the same time maintaining the robustness. Also the weights used for correlation might be taken as functions of battery power, number of multicast sessions a node participate in, local contention, and bandwidth available at nodes for choosing paths to trees. This would help bringing in energy, bandwidth, and load awareness into the protocol. We would like to employ network coding techniques at shared nodes of multiple trees constructed by our *K*-Tree protocol and investigate how they could effectively improve the network performance. Developing coding-aware mesh or tree based multicast routing protocols for Ad hoc wireless networks is another future research direction.

References

[1] T. Bheemarjuna Reddy, I. Karthigeyan, B.S. Manoj, C. Siva Ram Murthy, Quality of service provisioning in ad hoc wireless networks: a survey of issues and solutions, Ad Hoc Networks Journal 4 (1) (2006) 83–124.

[2] F.A. Tobagi, L. Kleinrock, Packet switching in radio channels: part II – the hidden terminal problem in carrier sense multiple-access and the busy-tone solution, IEEE Transactions on Communications 23 (12) (1975) 1417–1433.

[3] J. Luo, P.T. Eugster, J.-P. Hubaux, Route driven gossip: probabilistic reliable multicast in ad hoc networks, in: Proceedings of IEEE INFOCOM, vol. 3, 2003, pp. 2229–2239.

[4] J. Chakareski, S. Han, B. Girod, Layered coding vs. multiple descriptions for video streaming over multiple paths, in: Proceedings of ACM Multimedia, 2003, pp. 422–431.

[5] E.M. Royer, C.E. Perkins, Multicast operation of the ad hoc on-demand distance vector routing protocol, in: Proceedings of ACM MOBICOM, 1999, pp. 207–218.

[6] P. Sinha, R. Sivakumar, V. Bharghavan, MCEDAR: Multicast Core Extraction Aistributed Ad Hoc Routing, in: Proceedings of IEEE WCNC, 1999, pp. 1313–1317.

[7] J.J. Garcia-Luna-Aceves, E.L. Madruga, The core-assisted mesh protocol, IEEE Journal on Selected Areas in Communications 17 (8) (1999) 1380–1994.

[8] H. Jiang, S. Cheng, Y. He, B. Sun, Multicast along energy-efficient meshes in mobile ad hoc networks, in: Proceedings of IEEE WCNC, vol. 2, 2002, pp. 807–811.

[9] Sung Ju Lee, William Su, M. Gerla, On-demand multicast routing protocol in multihop wireless mobile networks, Mobile Networks and Applications 7 (6) (2002) 441–453.

[10] S. Mao, S. Lin, S. Panwar, Y. Wang, Reliable transmission of video over ad-hoc networks using automatic repeat request and multi-path transport, in: Proceedings of IEEE VTC, 2001, pp. 615–619.

[11] S. Lin, Y. Wang, S. Panwar, Video transport over ad-hoc networks using multiple paths, in: Proceedings of ISCAS, 2002, pp. 57–60.

[12] S. Mao, S. Lin, S. Panwar, Y. Wang, E. Celebi, Video transport over ad hoc networks: multistream coding with multipath transport, IEEE Journal on Selected Areas in Communications 21 (10) (2003) 1721–1737.

[13] El Al, A.A. Saadawi, T. Myung Lee, Improving interactive video in ad-hoc networks using path diversity, in: Proceedings of IEEE MASS, 2004, pp. 369–378.

[14] J.G. Apostolopoulos, Reliable video communication over lossy packet networks using multiple state encoding and path diversity, in: Proceedings of VCIP, 2001, pp. 392–409.

[15] S. Somsundaram, K.P. Subbalakshmi, R.N. Uma, MDC and path diversity in video streaming, in: Proceedings of IEEE ICIP, vol. 5, 2004, pp. 3153–3156.

[16] Yuan Chen, Shengsheng Yu, Jingli Zhou, Jun Fan, Video transmission over ad hoc networks using multiple description coding and clustering-based multiple paths, in: Proceedings of Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, vol. 3, 2007, pp. 792–797.

[17] S.J. Lee, M. Gerla, Split multipath routing with maximally disjoint paths in ad hoc networks, in: Proceedings of ICC, 2001, pp. 3201–3205.

[18] K. Wu, J. Harms, On-demand multipath routing for mobile ad hoc networks, in: Proceedings of EPMCC, 2001, pp. 1–7.

[19] D.B. Johnson, D.A. Maltz, Y. Hu, The Dynamic Source Routing Protocol for Mobile Ad Hoc Network, Internet-Draft, draft-ietf-manet-dsr-09.txt, 2003.

[20] P. Pham, S. Perreau, Performance analysis of reactive shortest path and multi-path routing mechanism with load balance, in: Proceedings of INFOCOM, 2003, pp. 251–259.

[21] A. Nasipuri, S.R. Das, On-demand multipath routing for mobile ad hoc networks, in: Proceedings of IEEE ICCCN, 1999, pp. 64–70.

[22] Sajama, Zygmunt J. Haas, Independent-tree ad hoc multicast routing (ITAMAR), Mobile Networks and Applications 8 (5) (2003) 551–566.

[23] Meng-Yen Hsieh, Yueh-Min Huang, Tzu-Chinag Chiang, Transmission of layered video streaming via multi-path on ad hoc networks, Multimedia Tools and Applications 34 (2) (2007) 155–177.

[24] W. Wei, A. Zakhor, Multipath unicast and multicast video communication over wireless ad hoc networks, in: Proceedings of BROADNETS, 2004, pp. 496–505.

[25] W. Wei, A. Zakhor, Multiple tree video multicast over wireless ad hoc networks, IEEE Transactions on Circuits and Systems for Video Technology 17 (1) (2007) 2–15.

[26] R. Ahlswede, N. Cai, S.-Y.R. Li, R.W. Yeung, Network information flow, IEEE Transactions on Information Theory 46 (4) (2000) 1204–1216.

[27] Ying Zhu, Baochun Li, Jiang Guo, Multicast with network coding in application-layer overlay networks, IEEE Journal on Selected Areas in Communications 22 (1) (2004) 107–120.

[28] S. Sengupta, S. Rayanchu, S. Banerjee, An analysis of wireless network coding for unicast sessions: the case for coding-aware routing, in: Proceedings of IEEE INFOCOM, 2007, pp. 1028–1036.

[29] B. Anirudh, T. Bheemarjuna Reddy, C. Siva Ram Murthy, K-Tree: a multiple tree video multicast protocol for ad hoc wireless networks, in: Proceedings of HiPC, 2006, pp. 424–435.

[30] L. Ford, D. Fulkerson, Flows in Networks, Princeton University Press, 1962. pp. 297–333.

[31] A.V. Goldberg, R.E. Tarjan, A new approach to maximum flow problem, in: Proceedings of Symposium on Theory of Computing, 1986, pp. 136–146.

[32] G.D. Hachtel, F. Somenzi, A symbolic algorithms for maximum flow in 0–1 networks, Formal Methods in System Design 10 (February) (1997) 207–219.

[33] R.J. Anderson, J.C. Setubal, On the parallel implementation of goldberg's maximum flow algorithm, in: Proceedings of ACM Symposium on Parallel Algorithms and Architectures, 1992, pp. 168–177.

[34] L. Li, T.A. Marsland, A parallel algorithm for finding a maximum flow in 0–1 networks, in: Proceedings of Annual Conference on Computer Science, 1987, pp. 231–234.

[35] J. Cheriyan, M.R. Salavatipour, Packing element-disjoint steiner trees, in: ACM Transactions on Algorithms, vol. 3, no. 4, Article No. 47, 2007.

[36] Y. Aumann, Y. Rabbani, Improved bounds for all optical routing, in: Proceedings of SODA, 1995, pp. 567–576.

[37] A. Baveja, A. Srinivasan, Approximate algorithms for disjoint paths and related routing and packing problems, in: Proceedings of IEEE Symposium on Foundations of Computer Science, 1998, pp. 416–425.

[38] C. Chekuri, S. Khanna, Edge-disjoint paths revisited, in: Proceedings of SODA, 2003, pp. 628–637.

[39] V. Guruswami, S. Khanna, R. Rajaraman, B. Shepherd, M. Yannakakis, Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems, in: Proceedings of Symposium on Theory of Computing, 1999, pp. 19–28.

[40] M. Andrews, L. Zhang, Hardness of undirected edge-disjoint paths problem, in: Proceedings of Symposium on Theory of Computing, 2005, pp. 276–283.

[41] S. Fortune, J. Hopcroft, J. Wyllie, The directed subgraph homeomorphism problem, Theoretical Computer Science 10 (2) (1980) 111–121.

[42] S. Capkun, L. Buttyan, J.-P. Hubaux, Self-organized public-key management for mobile ad hoc networks, IEEE Transactions on Mobile Computing 2 (1) (2003) 52–64.

[43] ns-2: network simulator, <http://www.isi.edu/nsnam/ns/>.