Exploiting Graph Partitioning for Hierarchical Replica Placement in WMNs

Zakwan Al-Arnaout, Qiang Fu, Marcus Frean
School of Engineering and Computer Science
Victoria University of Wellington
Wellington, New Zealand
{zakwan.arnaout, qiang.fu, marcus.frean}@ecs.vuw.ac.nz

ABSTRACT

Content replication has gained much popularity in recent years both in the wired and wireless infrastructures. A key challenge faced by Wireless Mesh Networks (WMNs) is to determine the number and locations of content replicas (e.g. video clip) such that the mesh clients access cost is minimized. Furthermore, the dynamic nature of the wireless environment favors a distributed and adaptive solution to this problem. In this paper, we present an efficient, lightweight and scalable object replication and placement scheme for WMNs. Since the placement problem is NP-Complete, the scheme decomposes the problem into smaller sub-problems to facilitate the distributed approach in a P2P fashion. Moreover, it exploits the long-term link-quality routing metrics to augment the replica placement decision and the instantaneous link-quality metrics for replica server selection. The effectiveness of our scheme is evaluated through extensive simulation studies.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Wireless communication; G.1.2 [**Approximation**]: Minimax approximation and algorithms

General Terms

Algorithms, Performance, Design

Keywords

Replica Placement; Distributed Algorithm; Wireless Mesh Network; Peer-to-peer Networking

1. INTRODUCTION

Wireless Mesh Networks (WMNs) are a promising solution to extend Internet access to wider areas with high bandwidth [1]. They are generally composed of three types of nodes: Mesh Clients (MCs), Mesh Routers (MRs), and Gateways (GWs). WMNs are deployed in areas where the wired infrastructure is not available or feasible such as rural areas, enterprise and university campuses. MRs compose the backbone layer through multi-hop wireless communications. A MC connects to other MCs or the Internet through its access MR. With their extended coverage and low cost, WMNs have attracted the attention of both the research community and the industry. However, similar to other wireless networks, WMNs also face challenges arising from the resource constraints and the resource demanding multimedia content retrieval [2], such as: (*i*) scalability of cache schemes with content size, network size and user population; (*ii*) poor wireless bandwidth; (*iii*) mobility of MCs; (*iv*) limited battery power of mobile clients; (*v*) poor channel quality; and (*vi*) the bottleneck effect around the GWs.

One of the key challenges for WMNs development is improving the data access efficiency and Quality of Experience (QoE), which determines the satisfaction degree of service users. Furthermore, as the Internet traffic flows through a limited number of GWs, heavy congestion around these GWs presents a serious problem. WMNs have the potential to increase network capacity by adding resource-sharing services such as content caching and replication. It has been observed [3] that for a given client population, a significant workload *locality* exists in WMNs content retrieval. Locality means that multiple users request the same content over time (and possibly at the same time). Web content caching and replication are two techniques that exploit locality to reduce the Internet traffic and access latency.

The problem we aim to address in this paper can be described as follows. We consider an infrastructure WMN where nodes (MRs) are deployed in static locations. In such networks, nodes do not join/leave the network dynamically. However, the wireless channel state between two neighbors may oscillate frequently due to various reasons. Since object replica placement is a costly operation, a replica must remain for a reasonable period of time. Most of the works in the wired/wireless literature consider minimizing the number of hops or the Euclidean distance in the placement decision. However, we argue that this approach might not yield to efficient placement since a shortest distance can lead to lossy/congested links. Other link-quality routing metrics prove to be more efficient than the simple Hop-Count such as Expected Transmission Count (ETX) [4], Expected Transmission Time (ETT) [5] and Minimum Loss (ML) [6]. In a prior work [7], we proposed a replication scheme for WMNs that builds an overlay P2P network formed by MRs. The scheme dynamically adapts the number of object replicas

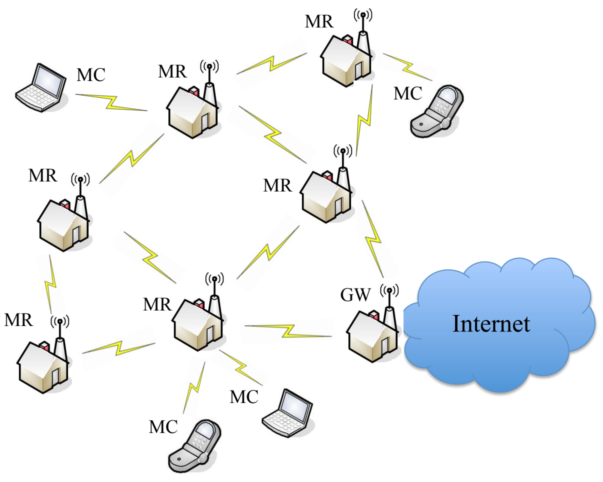


Figure 1: Network model considered in our proposed scheme.

over time based on popularity in a distributed and scalable fashion that exploits workload locality in WMNs by finding the number of replicas needed per object (e.g. video file for a Video-on-Demand service) in a time window.

We motivate our work by showing the performance gain when the replica placement considers the long-term link cost that represents its throughput or channel capacity. The short-term link cost may fluctuate frequently, whereas its long-term cost varies much slowly. In WMNs, the contention between neighboring mesh nodes for the wireless channel, together with the interference from the adjacent wireless links, results in a significant reduction in throughput over a long lossy path. Therefore, MRs that are far from the GWs suffer from long access latency and low throughput. We consider minimizing the demand-weighted distance between the requesting node and the replica server. Our contribution can be summarized as follows: (*i*) using advanced link-quality metrics to accurately gauge the client's object access cost using the long-term link-quality values for the replica placement decision, which is performed periodically. Furthermore, for replica server selection, we use the instantaneous link-quality values to satisfy a client's request; (*ii*) providing a fully-fledged replication and placement strategy for WMNs that optimally utilizes the storage capacity of MRs—by distinguishing variable sizes of objects—and avoids the creation of hotspot zones; and (*iii*) implement relevant schemes and conduct extensive simulation tests to compare the performance of our scheme with the most relevant ones. The simulation results show that our scheme *SP-DNA* (Single Partition per Delegate Node Assignment) can significantly improve the system performance.

The rest of the paper is organized as follows. Section 2 presents the network model and the proposed object replication and placement scheme is described in Section 3. In Section 4, we describe the evaluation methodology, while Section 5 discusses the results. Section 6 reviews the related work and finally, Section 7 concludes the paper.

2. NETWORK MODEL

In this section, we describe the network model of our object replica placement scheme over infrastructure WMNs. The system employs Mesh Routers (MRs) to act as content replica servers in a peer-to-peer fashion. We assume that MRs use IEEE 802.11 radios to build the wireless mesh backhaul infrastructure. MRs have replica server capabilities such as processing and storage, in other words, a MR acts as a relaying node and as a server. Fig. 1 illustrates the model considered in the paper scope. It consists of mesh access points/MRs and Mesh Clients (MCs). The MRs are interconnected by wireless links to form a multi-hop backhaul infrastructure. One or more MRs are connected to the wired Internet and are referred to as gateways (GWs). MRs are used to access and relay packets, therefore, MRs support two types of interfaces for the wireless medium. The access interface offers network access for MCs, while the relay interface is used to relay client's traffic to its destination. Typically, the two interfaces work on non-overlapping channels to prevent interference with each other.

A mesh client (e.g. laptop or smart phone) is associated with a nearby MR to access the mesh network, but it does not participate in packet relaying. To fetch an object (e.g. video file for a video on demand service), one of the structured P2P network directory services is employed (e.g. Chord [8]). A requested object is fetched first from within the mesh network. Upon a fail, the request will be forwarded to the mesh backhaul to one of the GWs and on to the origin server. We also assume that the MRs are aware of the network topology and employ one of the widely deployed routing protocols such as OLSR [9]. OLSR is defined by the IEEE 802.11s standard as the basis for future routing protocol implementations [10]. It also scales well for few hundreds of nodes. We assume that the MRs use TCP at the transport layer since it is broadly deployed for Internet access and we assume that the exchanged messages between nodes are delivered in a reliable fashion and follow their transmission order. Our scheme is implemented over the underlying TCP protocol by a user space *ReplicaDaemon*.

3. THE PROPOSED SCHEME

In this section, we present our proposed scheme. In a wireless environment, having a centralized entity to perform the placement is not suitable as it presents a bottleneck at the central entity increasing computation and communication cost resulting from acquiring popularity information about the objects. This is challenging and costly for limited resource networks such as WMNs.

Our approach is focused on decomposing the replica placement problem by using *graph partitioning* techniques. The main goal of graph partitioning is to divide a graph into a set of sub-graphs such that each sub-graph has roughly the same number of nodes and the sum of all edges that connect different sub-graphs is minimized. Therefore, graph partitioning is useful to distribute the problem into a set of partially independent sub-problems especially when dealing with large problems. The search space is split according to the computed number of replicas per object. In each partition, a predetermined *Delegate Node* (DN) solves part of the complete search space. To this end, the replica placement problem is divided into a set of smaller loosely connected ones. Thus, we can take advantage of graph partitioning techniques to solve the problem considered in a divide-and-conquer approach.

The replica placement problem can be formulated as a *p-median* [11] problem which is simply stated as: Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, find $\mathcal{V}_p \subseteq \mathcal{V}$ such that $|\mathcal{V}_p| = p$, where p may

Table 1: Notation used in the proposed scheme.

Symbol	Meaning
$\mathcal{N}$	set of demand nodes indexed by i, and potential replica servers indexed by j.
$\mathcal{N}'$	set of demand nodes within a partition indexed by i', and the set of potential replica servers indexed by j'.
$\mathcal{M}$	number of distinct objects within the WMN indexed by m.
$\mathcal{K}$	total number of the generated graph partitions indexed by k.
$\mathcal{DN}_k$	delegate node of partition k.
$\mathcal{O}_m$	identifier of object m.
$\|\mathcal{O}_m\|$	size of object m in bytes.
$\overline{X}_t$	trimmed mean of the link-quality metric (e.g. ETX, ETT and ML).
$\Gamma(\overline{X}_t)_{i'}^{j'}$	distance between demand node i' and potential replica server j' within $\mathcal{N}'$ as a function of $\overline{X}_t$.
τ	time interval window during which object requests are observed.
$\lambda_{mi}(\tau)$	number of requests for $\mathcal{O}_m$ from node i during τ.
$\lambda_m(\tau)$	total requests for $\mathcal{O}_m$ from all nodes during τ.
$\mathcal{P}r_m(\tau)$	global popularity of $\mathcal{O}_m$ during τ.
$\mathcal{P}r_{mk}(\tau)$	popularity of $\mathcal{O}_m$ during τ within partition k.
$\mathcal{P}r_{mi}(\tau)$	popularity of $\mathcal{O}_m$ at node i during τ.
$p_m(\tau+1)$	no. of replicas of $\mathcal{O}_m$ needed for $\tau+1$.
$\mathcal{SC}$	storage capacity of a node.

either be variable or fixed, and that the sum of the shortest distances from the vertices in $\{\mathcal{V} \backslash \mathcal{V}_p\}$ to their nearest vertex in $\mathcal{V}_p$ is minimized. This problem is NP-Complete for variable values of p. To distribute the replica placement problem, we simplify the *p-median* problem by partitioning the network graph into p sub-graphs, where p represents a potential number of replicas. Then we select for each partition a DN, which will be responsible for placing an object replica within its partition. The partitioning algorithm we use is proposed in [12]. In our scheme, we recursively bipartite the graph until the partition size ≤ 2. Since the total number of partitions $= \mathcal{N} - 1$, each DN will be assigned at most a single partition, hence, we call it *SP-DNA*. Our scheme involves two phases, the *Network Setup Phase* and the *Content Replication and Placement Phase*. We use the notation in Table 1 to describe our scheme.

3.1 Network Setup Phase

In this phase, the aim is to build a hierarchy of partitions and assign a DN for each partition to lookup for the placement within the partition member nodes. We assume that all nodes[1] are bootstrapped with Alg. 1. To trigger the network setup phase, an application-level process called the *Content Manager* (CM)—which is a component of the *ReplicaDaemon*—starts this phase. To reduce the message overhead, the CM can be hosted by a centroid node. The CM plays a role in collecting statistical information about different objects' requests, computing the number of replicas per object for the next τ period, and assigning the placement job to the corresponding DNs. The CM selects itself to act

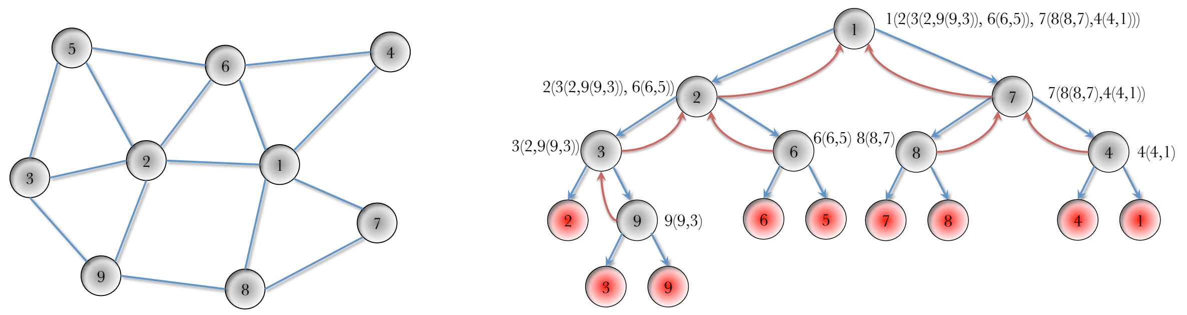
Algorithm 1: This function builds the DBT during the network setup phase.

```
1  Function BTree($Graph\ g, String\ s$)
2  Graph $g_{\text{Left}}, g_{\text{Right}}$
3  Node $dn_{\text{Left}}, dn_{\text{Right}}$
4  String $s_{\text{Left}}, s_{\text{Right}}$
5  $s_{\text{Left}} \leftarrow s_{\text{Right}} \leftarrow s$
6  if $|g| = 2$ then
7      $dn_{\text{Left}} \leftarrow this$
8      $dn_{\text{Right}} \leftarrow \{g\} \setminus dn_{\text{Left}}$
9      return $dn_{\text{Left}} \mid\mid \{g\}$
10 else if $|g| = 3$ then
11     $dn_{\text{Left}} \leftarrow rand(g) \notin s$
12     $g_{\text{Right}} \leftarrow \{g\} \setminus dn_{\text{Left}}$
13     $dn_{\text{Right}} \leftarrow rand(g_{\text{Right}}) \notin s$
14     forward $(g_{\text{Right}}, s) \rightarrow dn_{\text{Right}}$
15     wait $(dn_{\text{Right}})$
16     return $dn_{\text{Left}} \mid\mid dn_{\text{Right}}$
17 else
18     Bipartite $(g, g_{\text{Left}}, g_{\text{Right}})$
19     $dn_{\text{Left}} \leftarrow rand(g_{\text{Left}}) \notin s_{\text{Left}}$
20     $dn_{\text{Right}} \leftarrow rand(g_{\text{Right}}) \notin s_{\text{Right}}$
21     $s_{\text{Left}} \leftarrow s_{\text{Left}} \mid\mid dn_{\text{Left}}$
22     $s_{\text{Right}} \leftarrow s_{\text{Right}} \mid\mid dn_{\text{Right}}$
23     forward $(g_{\text{Left}}, s_{\text{Left}}) \rightarrow dn_{\text{Left}}$
24     forward $(g_{\text{Left}}, s_{\text{Right}}) \rightarrow dn_{\text{Right}}$
25     wait$(dn_{\text{Left}}, dn_{\text{Right}})$
26     return $this \mid\mid dn_{\text{Left}} \mid\mid dn_{\text{Right}}$
```

as a DN for the network graph $\mathcal{G}$ and runs Alg. 1 to bipartite $\mathcal{G}$, select a DN for each partition, and then forward each partition (sub-graph) to its corresponding DN. The process is recursively repeated until the partitions are fine-grained. When the base case is reached (line 6), the node running the algorithm will select itself as the DN (the term *this* refers to the node itself (line 7)). Then, the DNs in the lowest level partitions reply to their callers (parent node) with a list representing the DN and its partition members. When the caller receives the lists from its child DNs, it merges the received lists, appends its node ID and forwards the resulting list to its caller. Eventually, the CM (root node) will receive a list of DNs in the form of a balanced binary tree that we call the *Delegates Binary Tree* (DBT) as depicted in Fig. 2. Notice (Fig. 2b) that a node appears twice. Once as a parent node (i.e. partition-hosting) responsible for the descendant members (and itself) and another as a leaf node (i.e. self-hosting). We underline that this phase needs to be performed once. However, if the topology changes permanently (e.g. relocation of MRs), then we need to perform this phase again. We do not consider a temporary topology change due to temporary link variations as content replication is required for long periods.

Following the DBT creation, the CM maps the possible numbers of replicas to their DNs in the DBT, creating a MapList by running Alg. 2. It recursively divides the possible number of replicas (p) by 2 starting from the CM (root node).

[1]In order to avoid confusion throughout this paper, the terms MRs, nodes and $\mathcal{N}$ refer to the same meaning since our system is implemented by MRs decoupling it from MCs.



(a) Graph $\mathcal{G}$ representing mesh nodes. (b) The generated Delegates Binary Tree.

Figure 2: This figure illustrates the conversion of the network graph $\mathcal{G}$ (a) into a balanced binary tree DBT (b), where each node represents a DN for the underlying partition.

Algorithm 2: Function to map the possible number of replicas (p) to their DNs.

```
1 Function Map($Node\ n,\ int\ p$)
2 if $p = 1$ then
3     if $n\ is\ leaf$ then
4         return $n||S_{host}$
5     else
6         return $n||P_{host}$
7 else
8     return $map(n.left, \lfloor \frac{p}{2} \rfloor)\ +\ map(n.right,\ \lceil \frac{p}{2} \rceil)$
```

When the base case is reached (i.e. $p = 1$), it checks whether the node n is a leaf node in the DBT or not. If it is a leaf node, then it means that node n has to *self-host* the object for p replicas and returns the node ID concatenated with a flag (S_{host}) to indicate *self-hosting*. Otherwise, it returns the node ID along with a flag (P_{host}) to indicate *partition-hosting* (i.e. the DN will need to find the optimal location for the object within its partition). For every possible p, the CM creates an entry in the form of $\langle p$, set of DNs$\rangle$ and inserts it in the Map-List structure. Upon completion, the CM broadcasts the Map-List to all the nodes. This will facilitate for both the CM and the DNs to know *a priori* that for a specific p, which DNs the CM shall communicate with to assign the placement job and the DN knows from the Map-List its role whether self-hosting or partition-hosting.

3.2 Content Replication and Placement Phase

This phase consists of three steps to be performed aiming to collect statistical information about the objects, compute the number of replicas for each object and then find the placement for the replicas. These steps are described as follows:

3.2.1 During the τ period

Every node maintains a list containing an entry in the form $\langle \mathcal{N}_i, \mathcal{O}_m, \lambda_{mi}(\tau) \rangle$ that represents the request count λ_{mi} for every object $\mathcal{O}_m$ during τ at node i. An object request is counted when a MC initiates one to its access MR regardless of being served by the access MR, any other MR or even the origin server. Moreover, it will also observe the link-quality (e.g. ETX, ETT and ML) values with its neighboring nodes by logging the values of the used metric.

3.2.2 At the end of the τ period

In this step, every node finds the *trimmed mean* value of the link-quality metric with the neighboring nodes by excluding *outlier* values. Since the replica placement is intended for long periods, using the current value of the link-quality metric will not reflect the actual distance cost. Therefore, we believe that using the trimmed mean can give a precise estimation of fluctuating wireless links that captures the long-term link cost instead of using the mean or instantaneous link cost. Anomalous outlier values can completely change the mean, thus putting the link cost at big risk of errors. Therefore, it is important to identify and eliminate them as follows: We let x_1, x_2, ..., x_s be a sample of size s on measurement of a particular link-quality metric. Then the $100\alpha\%$ trimmed mean is defined as: $\overline{X}_t = \frac{\sum_{i=t+1}^{s-t} x_i}{(s(1-2\alpha))}$, where $\alpha \in (0,1)$; $t = [s\alpha + .4]$. To simplify, we take t to be the floor of $(s\alpha + .4)$ as an approximation [13]. Every node computes $\overline{X}_t$ with its neighboring nodes to represent the range of sample points unaffected by outliers. We simply ignore t of the lowest and t of the highest sample points. Then the following sub-steps are performed:

(a) The partition members of the lowest level DNs forward their object frequency list and the computed link-quality truncated mean with the member's neighbors to the parent DN. Each DN aggregates the received lists from its children along with its own list, stores the resulting list and then forwards it to its parent node. The process of aggregate, store and forward is repeated until the CM (root node) receives the full list from all the nodes. The usefulness of this hierarchical approach is: (i) reducing communication overhead; and (ii) fusing popularity information at different levels of the DBT helps distributing this information instead of collecting it by a central node.

(b) For every $\mathcal{O}_m$, the CM computes the global popularity $\mathcal{P}r_m(\tau)$ and the number of replicas per object $p_m(\tau+1)$ according to Eq (1) and Eq (2) respectively.

$$\mathcal{P}r_m(\tau) = \frac{\sum_{i=1}^{N} \lambda_{mi(\tau)}}{\sum_{i=1}^{N} \sum_{m=1}^{M} \lambda_{mi(\tau)}} \qquad (1)$$

$$p_m(\tau+1) = \frac{SC \times N \times \mathcal{P}r_m(\tau)}{|\mathcal{O}_m|} \qquad (2)$$

(c) The CM creates a *Replica-List* (RL) in the form of a 4-tuple $\langle p_m(\tau+1), \mathcal{O}_m, |\mathcal{O}_m|, \lambda_m(\tau) \rangle$ grouped by $p_m(\tau+1)$ in a decreasing order to prioritize the most popular objects.

Within each group, the objects are sorted by the objects size $|\mathcal{O}_m|$ in a decreasing order. The reason behind this way of sorting is that for a group of objects of semi-equal popularity, prioritizing large objects would minimize the cost weighted by the object size given the storage constraint. However, our scheme is fair with small objects in terms of the computed number of replicas, since it divides the popularity of an object by its size (see Eq (2)).

Algorithm 3: The Distributed Replica Placement Heuristic *SP-DNA*.

```
1  foreach group ∈ $\mathcal{RL}$ do
2      multicast a message containing the group of object
       replicas to the corresponding list of DNs in Map-List
3  receiving $\mathcal{DN}_k$ performs the following:
4  lookup the Map-List for the given $p_m(\tau+1)$
5  forall the $\mathcal{O}_m$ in the received group do
6      if $\epsilon_{mk} > \eta_m$ then
7          forward $\langle p_m(\tau+1), \mathcal{O}_m, |\mathcal{O}_m|, \lambda_m(\tau)\rangle \rightarrow parent(\mathcal{DN}_k)$
8      else
9          if $\mathcal{DN}_k$ is flagged $\mathcal{S}_{host}$ then
10             Fetch($\mathcal{O}_m$)
11         else
12             pick a node that minimizes the cost:
               $Min \sum_{\substack{1\le i'\le N' \\ 1\le j'\le N'}} \mathcal{P}r_{mi'}(\tau)\Gamma(\overline{X}_t)_{i'}^{j'} s.t \exists SC$
13             assign $\mathcal{O}_m$ to the selected node
14             Fetch($\mathcal{O}_m$)
```

3.2.3 Replica placement

In this step, the *SP-DNA* heuristic (Alg. 3) is executed. The CM multicasts a message that contains the 4-tuple $\langle p_m(\tau+1), \mathcal{O}_m, |\mathcal{O}_m|, \lambda_m(\tau)\rangle$ to the corresponding DNs obtained from the function in Alg. 2. When each DN receives the message, a decision has to be made (line 6) to find out whether it is feasible to place $\mathcal{O}_m$ in $\mathcal{DN}_k$ partition or forward it to the parent of $\mathcal{DN}_k$ in the DBT. The corresponding $\mathcal{DN}_k$ compares the ϵ_{mk} value from Eq (3) with a threshold value η_m from Eq (4). ϵ_{mk} acts as a *feasibility index* to decide upon the feasibility of placing $\mathcal{O}_m$ within the DN's partition or forwarding it to the parent DN. η_m is inversely proportional to the number of replicas $p_m(\tau+1)$, serving as an error margin. This means that, for a given network size, a smaller $p_m(\tau+1)$ or in other words larger partitions yields a larger error margin (η_m). A larger η_m means reduced computation cost at the expense of placement accuracy. Note that, for large partitions, the computation cost is high. This is a tradeoff between computation cost and placement accuracy. The threshold η_m can be tuned using the coefficient θ, to trade off between placement accuracy and computation cost for a given partition size. If ϵ_{mk} exceeds η_m, then $\mathcal{DN}_k$ will handoff $\mathcal{O}_m$ to its parent DN, which is responsible for a larger partition where a better placement can take place. The parent DN on its turn will take the decision of placement or handoff.

$$\epsilon_{mk} = \frac{1}{p_m(\tau+1)\times \mathcal{P}r_{mk}(\tau)} - 1 \tag{3}$$

Table 2: Numerical example showing how a DN decides to *place* an object in its partition or *forward* it to its parent DN given $p_m(\tau+1)=3$, $\theta=1$ and $\eta_m=0.33$.

$\mathcal{DN}_k$	$\mathcal{P}r_{mk}(\tau)$	ϵ_{mk}	Decision
3	5%	5.67	Handoff
6	30%	0.11	Place
7	40%	-0.17	Place

$$\eta_m = \frac{1}{p_m(\tau+1)\times\theta} \tag{4}$$

$$\mathcal{P}r_{mk}(\tau) = \frac{\sum_{i'=1}^{N'}\lambda_{mi'}(\tau)}{\lambda_m(\tau)} \tag{5}$$

If ϵ_{mk} is not larger than η_m (line 8), it looks up the Map-List for its role. If it has to *self-host* the replica $\mathcal{O}_m$, then it only needs to fetch it. Otherwise (i.e. *partition-host*), it computes $\mathcal{P}r_{mk}(\tau)$ using Eq (5), which represents the popularity percentage of $\mathcal{O}_m$ within partition k with respect to the whole network. $\mathcal{DN}_k$ will compute the demand-weighted total cost for every partition member and assigns $\mathcal{O}_m$ to the node that minimizes the total cost.

The distance $\Gamma(\overline{X}_t)_{i'}^{j'}$ represents the sum of the computed long-term link-quality between i' and j'. Note that if there is no sufficient space to accommodate $\mathcal{O}_m$, $\mathcal{DN}_k$ selects the second best node and so forth, even if all partition members' storage is full, the object will be forwarded to the parent DN. We note here that a parent DN waits for an acknowledgement from its child DNs when they complete the placement of the *feasible* objects so that it can start placing the forwarded objects or forward up the hierarchy. Simultaneously, it can undertake the placement of its own *feasible* objects. This rule is to make sure that when it has to place a forwarded object, the placement should consider other replicas (if any) in the underlying partitions. We give a numerical example (see Table 2) to describe how the placement decision is made using Fig. 2. Suppose an object requires 3 replicas ($p_m(\tau+1)=3$). For instance, the DN set will be nodes 3, 6 and 7. Each DN decides based on the local popularity $\mathcal{P}r_{mk}(\tau)$ within its partition. Therefore, node 3 will forward the placement job to node 2, while both nodes 6 and 7 will find the optimal placement in each partition.

Algorithm 4: Function to fetch an object performed by the replica server node.

```
1  Function Fetch($\mathcal{O}_m$)
2  if $\mathcal{O}_m$ is in the node's storage then
3      goto line 10
4  else if lookup($\mathcal{O}_m$) → the Directory Service then
5      receive the list of nodes hosting $\mathcal{O}_m$
6      select the nearest hosting node for $\mathcal{O}_m$
7      fetch $\mathcal{O}_m$ from the selected replica node
8  else
9      fetch $\mathcal{O}_m$ from the origin server
10 update $\mathcal{O}_m(\tau+1)$
11 update the Directory Service with $\mathcal{O}_m(\tau+1)$
```

Upon successful assignment, the selected node fetches the object replica using Alg. 4 by trying to find it locally (line 2).

Upon a miss, it consults the Directory Service (e.g. Chord [8]) by sending the object's key to obtain the list of nodes hosting it. If the object is found, the replica server requests it from the closest node. Upon a miss, it requests $\mathcal{O}_m$ from the origin server. We note here that lines 10 and 11 refer to two separate updates. The first (line 10) is for the replica server to differentiate objects of the next period ($\tau + 1$) from objects of the current period (τ). The second (line 11) is used to update the Directory Service with the node ID hosting $\mathcal{O}_m$ for the next period ($\tau + 1$). After fetching $\mathcal{O}_m$, the node evicts an object(s) from previous τ and inserts $\mathcal{O}_m$ for ($\tau + 1$). It can be observed that the *SP-DNA* heuristic is not fully distributed as it depends slightly on the CM that acts as a Super-Peer [14]. However, the major burden is on the replica placement which is performed by the DNs. This avoids the fully distributed approach that incurs excessive message overhead resulting from the exchange of popularity statistics between all the participating nodes.

Table 3: Default simulation parameters

Parameter	Default Value
Simulation area	3000m x 1000m
Number of MRs ($\mathcal{N}$)	150
Number of MCs	900
Radio interface	IEEE 802.11g
Link rate	54 Mbps
Transmission range	200m
Zipf-like parameter α	0.95
Threshold coefficient θ	1
Carrier frequency	2.4 GHz
Max. Tx power	2.0 mW
Noise level	-110 dBm
Channel model	Rayleigh
Percent of trimmed values t	10%
$\mathcal{M}$	3000
$\lvert\mathcal{O}_m\rvert$ range	1 -> 4 MB
τ	90 minutes
$\mathcal{SC}$	256 MB

4. EVALUATION METHODOLOGY

To evaluate our scheme, we used OMNeT++ simulator [15] with Inetmanet [16] and OverSim [17] implementations. Inetmanet provides an implementation for different MANET routing protocols with the support for multiple link-quality metrics including hop-count, ETX, ETT and ML. OverSim is a flexible overlay network simulation framework based on OMNeT++. It includes several structured and unstructured peer-to-peer protocols. In our implementation, we use Chord [8], which is a well-known distributed, scalable and DHT based content lookup protocol that is designed for structured P2P networks. We simulated a stationary WMN for different scenarios in different mesh topologies. Objects are looked-up using Chord. We used the common OLSR routing protocol augmented with one of the link-quality metrics (i.e. hop, ETX, ETT and ML). Although the schemes we are comparing with aim to minimize the Hop-Count distance, we find it unfair to compare our scheme using the other advanced metrics and use the simple Hop-Count for the other schemes. Therefore, we use the same metric for all the schemes in every simulation run. The request pattern follows a *Zipf-like* distribution.

The browsing behavior of the clients is simulated using a random think time period that represents the time elapsed between successive web page downloads by a MC. This period ranges between 5 and 10 minutes on average since the requested object represents a multimedia Web content and not the ordinary Web page browsing. The simulation results were averaged over multiple random scenarios. The application traffic is created using FTP traffic generator. In each simulation test, the baseline *Random* heuristic was run first, followed by one of the other heuristics or *SP-DNA*. For all the heuristics, requests are served from the closest replica server depending on the instantaneous value of the link-quality metric used. The simulation tests are carried out in online fashion such that object replica placement is performed whilst clients' requests are generated. Table 3 summarizes the default parameters' values used. Now, we give a brief description of the heuristics we are comparing with:

1. ***Random*** [18]: Objects are assigned to nodes randomly subject to the storage constraints. Both objects and replica servers are selected in a uniform probability until each node's storage is full. If the replica server already stores the object, a new object and a new server are selected.

2. ***Lat-CDN*** [19]: Initially, objects are stored at the origin servers and all the replica servers are empty. The heuristic selects the *object-server* pair that produces the largest network latency, and thus assign the object to the server. The cost matrix is updated following the new placement. This process is iterated until all the servers become full.

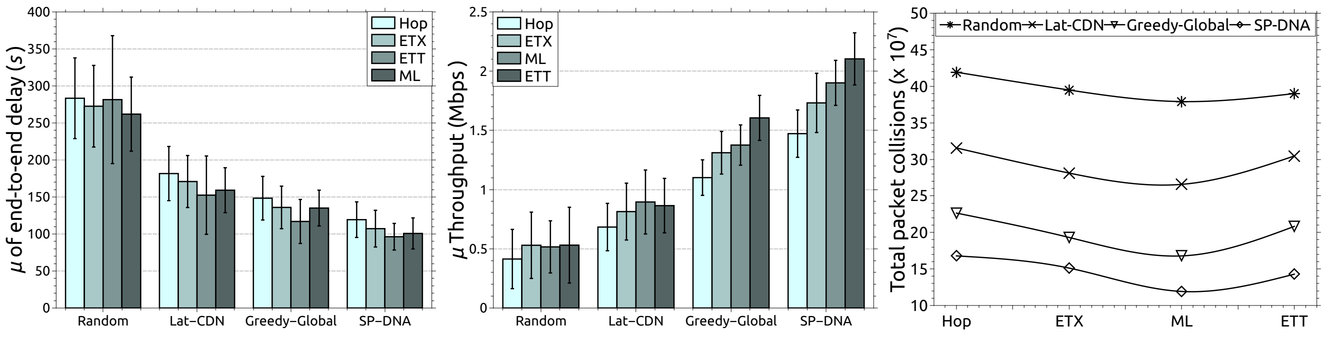
3. ***Greedy-Global*** [18]: For every node, the heuristic finds the object that yields the highest demand-weighted cost. It then picks the *object-server* pair that has the highest cost and stores the object in that node. This results in a new placement. Then it re-calculates the costs and picks the *object-server* pair that yields the highest cost. The process is repeated until all nodes' storage is full.

5. RESULTS AND DISCUSSIONS

In this section, we present the simulation results and discuss them according to the performance metrics described in each sub-section.

5.1 Mean end-to-end delay

In this subsection, the comparison is with respect to the mean end-to-end delay observed by mesh clients. This refers to the system-wide mean time taken since the initiation of a request until the completion of object's transmission. We ran multiple simulation tests for each scenario for different topologies. Fig. 3a depicts a comparison for each scenario. It can be noticed that the worst performance is obtained using the simple Hop-Count, which does not reflect the channel quality leading to lossy/congested links and as a result, yields to packet delay, loss, and retransmission due to channel congestion and degradation in signal quality. On the other extreme, it clearly shows that all the heuristics perform best using the ETT metric, which employs the link-rate information and the packet delivery ratio to represent the wireless link-quality more precisely than both ETX and ML. We can note that *SP-DNA* outperforms the other heuristics.



(a) Mean end-to-end delay (b) Mean throughput (c) Packet collisions

Figure 3: Performance comparison between different schemes using different link-quality metrics.

The performance gain is caused by: (a) given the same popularity, our scheme favors large objects over small ones when a DN makes the placement decision and as a result, the number of bytes traversing is reduced; (b) the shorter convergence time (as we show in subsection 5.5.2) for *SP-DNA* compared to *Lat-CDN* and *Greedy-Global* that require longer time to converge. Recall that the heuristics are evaluated in online fashion, clients' requests cannot be satisfied from nearest replica servers until the replica placement is complete; and (c) *SP-DNA* gives a more accurate placement than *Greedy-Global* since it considers the local popularity within the partition. We can notice that using the ETT metric yields the lowest end-to-end delay. This is because ETT improves the performance of ETX by probing the link bandwidth using fixed-size data packets periodically using two packets, a small one followed by a large one. The bandwidth is the size of the large packet divided by the minimum delay received for that link.

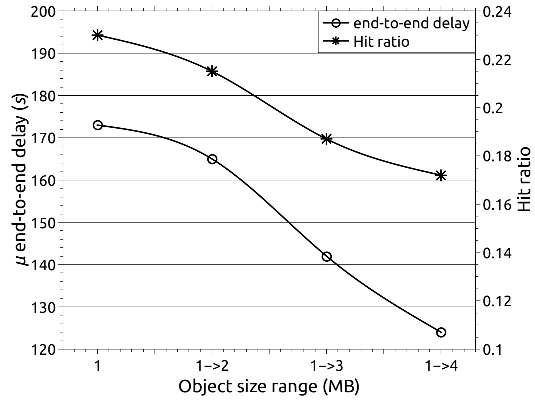
5.2 Mean throughput

In this subsection, we compare *SP-DNA* with the other heuristics with respect to the mean throughput in the network. This is the summation of all the sizes of served objects divided by the total time taken to complete the transmission of all the packets. The results in Fig. 3b represent the mean throughput obtained using different heuristics. It can be observed that *Lat-CDN* has a better throughput than the *Random* heuristic since servers cooperate to place objects with the high latency. However, it does not perform better than *Greedy-Global* and *SP-DNA* because it leads to improper replica placement as it considers latency cost without distinguishing the variable demand (popularity) among replica servers. We can notice that *SP-DNA* outperforms the *Greedy-Global*. This is because *Greedy-Global* does not distinguish different sizes of objects as it may not be an issue for high-performance CDN servers. However, *SP-DNA* considers this crucial factor in finding the density share for each object, which yields to a significant performance improvement. To compare with *Lat-CDN*, it clearly shows that *SP-DNA* has a significant performance gain over *Lat-CDN*. This gain is due to the same reason as in *Greedy-Global* (i.e. does not consider different sizes of object). Furthermore, it assumes similar popularity for an object among replica servers.

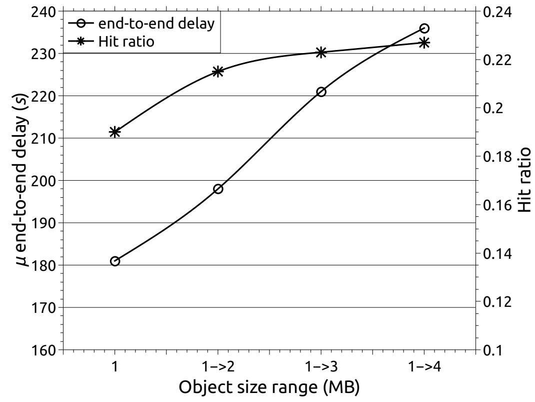
Although the *Greedy-Global* might lead to a better placement than *SP-DNA* in extreme situations such as when the popularity is concentrated in one partition and comparatively low in the neighboring partitions. In such a situation, *Greedy-Global* might place multiple replicas in one partition; while *SP-DNA* places a replica in each partition. However, the traffic in such situation will be concentrated in that partition; therefore, two problems will arise. The first problem is the immense traffic within that partition, which will lead to problems like the contention to access the wireless medium, packet loss, congestion and degrading the link-quality. The second problem is the load imbalance between replica servers. The servers in the hotspot partition will be overloaded and the servers in the neighboring partitions are less loaded. Our scheme can achieve a better load balance than *Greedy-Global*, since requests can be forwarded to replica servers in the neighboring partitions to avoid the aforementioned problems. For *Greedy-Global*, this might not be a problem as long as there is sufficient bandwidth and high capacity CDN servers, but in the wireless environment these resources are scarce. We can also infer that *SP-DNA* performs best using the ETT metric, which selects paths with high link rates.

5.3 Total number of packet collisions

In this subsection, the comparison is conducted with respect to the number of packet collisions. This refers to the total number of colliding packets resulting from the contention to access the wireless medium during a simulation run. Since MRs are typically stationary, link failure due to mobility is rare. However, transmission fails due to packet collisions, interference, or inadequate link rate selection. We can notice from Fig. 3c that the packet collision increases drastically for the baseline *Random* heuristic as a result of the long distance traveled by the packets, which degrades the network capacity. It can be noticed that *SP-DNA* can significantly reduce the number of packet collisions compared to the other heuristics. This performance gain is because *Greedy-Global* might lead to the creation of hotspot zones, where replicas of an object are concentrated yielding to problems such as contention and congestion in these hotspot zones that increases packet collision. The increased packet loss in *Lat-CDN* comes as a result from the assumption of similar object popularity among replica servers. This assumption is not practical in WMNs that do not have sufficient bandwidth as in CDN servers yielding improper and inefficient placement of unpopular objects in nodes that can benefit from hosting popular objects with respect to their MCs demands. As a result, the distance traveled is increased and a node's $\mathcal{SC}$ is inefficiently utilized. However, in our scheme, the traffic is distributed among the partitions to avoid the aforementioned problems. We note that in terms of packet collision, ML performs best since it is based on using the path that introduces the minimum loss rate by multiplying the success probability of each link along the path. This avoids lossy links and therefore, it reduces packet collisions.



(a) Decreasing order

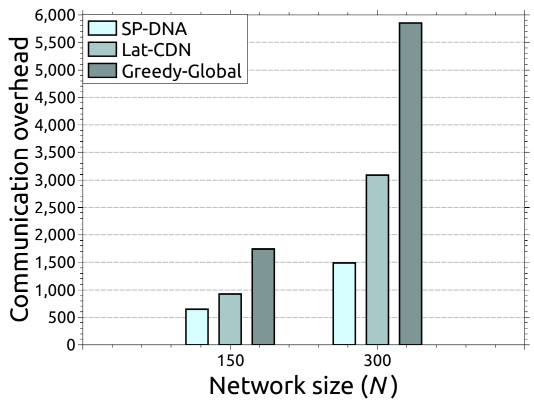


(b) Increasing order

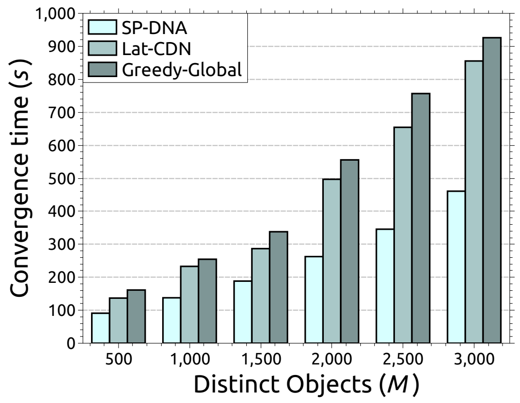
Figure 4: The tradeoff between the mean end-to-end delay and the hit ratio vs. variable ranges of object size.

5.4 Mean end-to-end delay vs. hit ratio

In these simulation tests, we investigate the effectiveness of sorting objects of semi-equal popularity based on objects' size both in decreasing and increasing orders. This is a tradeoff between latency and hit ratio as can be noticed in both Figs. 4a and 4b. Prioritizing large objects (see Fig. 4a) shows that as the range increases, the latency improves as the placement optimality of large objects converges, which reduces the amount of packets traversing. However, the hit ratio drops slightly as the range increases. On the contrary, Fig. 4b shows a latency increment as the size gap increases and an improvement in the hit ratio since replicating smaller objects usually results in higher hit ratios [20]. However, beyond a limit (1->3) this improvement becomes slight, hence, we favor the decremental sort as the benefit in access cost is more than the benefit in the hit ratio.



(a)



(b)

Figure 5: Comparison between different heuristics in terms of communication overhead (a) and convergence time (b).

5.5 Communication / convergence overhead

5.5.1 Communication overhead

This is the total number of control messages (wireless transmissions) generated from collecting popularity information and disseminating the placement decision on each node. Two network sizes ($\mathcal{N} = 150$ and 300) were considered to quantify the scalability of each scheme. In the simulation runs, we considered placing the replica placement entity for both *Greedy-Global* and *Lat-CDN* in a centroid node to reduce the wireless transmissions. The results in Fig. 5a reveal that *Greedy-Global* incurs the highest overhead. This is caused by the hop-by-hop forwarding of popularity messages to the central entity, which finds the set of object replicas for each node and then forwards to each node its own replica set. *Lat-CDN* works similar to *Greedy-Global* except that it does not require the collection of popularity statistics, therefore, the overhead is reduced by approximately half. The agglomeration approach used by *SP-DNA* significantly reduces the message overhead since the partition members are in the proximity of their DN, the wireless transmissions traverse a short distance. As we go up the hierarchy, the agglomerated message might travel a longer distance, however, it combines multiple popularity messages from its descendants. On the other direction, when the CM assigns the placement job, it batches multiple object IDs in a multicast message (Alg. 3) that is forwarded to the corresponding DNs. Reducing communication overhead is a very important aspect in a wireless environment.

5.5.2 Convergence Time

This is the total time taken to decide on allocating and fetching all the object replicas on all participating nodes for the next τ period measured in seconds. We compare the heuristics for $\mathcal{N} = 300$ and for variable $\mathcal{M}$. Fig. 5b clearly shows that *Greedy-Global* and *Lat-CDN* demonstrate long time requirements to converge. This is because both of them are centralized and require heavy sorting operations since both use a two dimensional cost matrix composed of $\mathcal{N}$ and $\mathcal{M}$. Therefore, both of them require recalculating the cost function for every *Object-Server* assignment after every replica placement. As $\mathcal{N}$ increases, the growth of the convergence time for both increases dramatically. This indicates that both do not scale well with respect to $\mathcal{N}$. The performance gain for *Lat-CDN* over *Greedy-Global* is because the latter requires collecting popularity statistics about objects, which is not required by the former algorithm. In contrast, *SP-DNA* is distributed and therefore, the burden of the placement decision is performed in parallel by the DNs and runs in groups of objects. As the number of replicas $p_m(\tau + 1)$ increases, the partition size ($\mathcal{N}'$) decreases yielding a decrement in convergence time. Therefore, *SP-DNA* depends on $p_m(\tau + 1)$ rather than $\mathcal{N}$. However, the other heuristics depend on $\mathcal{N}$, $\mathcal{M}$, $\mathcal{SC}$ and the granularity of object size $|\mathcal{O}_m|$. Overall, the long convergence time for all heuristics is because it includes both the placement decision and the time to fetch the objects' replicas.

6. RELATED WORK

Many caching and replication algorithms have been proposed for the Internet and ad hoc networks, but there has been much less effort devoted to schemes tailored for WMNs. On one hand, many replica placement algorithms designed for the Internet are centralized and incur a high computational cost. On the other hand, caching and replication schemes in ad hoc networks focus on issues such as low bandwidth and energy constraints. In [21], the goal is to find a placement strategy that minimizes the distance that needs to be searched to find an arbitrarily chosen piece of content. However, it does not consider content popularity at different nodes since they formulate the problem as the *k-center* problem. The work in [22] proposes a strategy to determine the optimal number of replicas to minimize object access cost (defined as the Euclidean distance from the requester to the nearest replica) in WMNs when the prior knowledge on the global popularity of objects is available. *P2PMesh* was proposed in [23] for P2P file sharing system over WMNs. It aims to reduce both the number of failed lookups and the file lookup latency. However, replica placement was not considered. The authors in [24] propose *H2-VIP* to compute the optimal number of replicas of video blocks so that the overall system failure rate can be minimized. However, we consider the content placement in mesh routers storage instead of home devices. Hence, the reliability is high.

To the best of our knowledge, the closest works are given in [18, 19]. In [18] the authors proposed four heuristics: *Random*, *Popularity*, *Greedy-Single* and *Greedy-Global*. They found that *Greedy-Global* outperforms the other three heuristics. In [19] *Lat-CDN* was proposed that aims to allocate objects to replica servers with respect to the total network's latency. However, both [18, 19] are centralized and do not take into account the special features of wireless networks.

7. CONCLUSIONS

In this paper, we have proposed an efficient and scalable object replication and placement scheme for WMNs. To make our scheme distributed and hierarchical, we firstly build a balanced binary tree of multi-level partitions of the network. This is then used to facilitate replica placement and reduce communication and computation cost. The scheme makes the placement decision in a hierarchical way. A delegate node decides to place a replica in its partition when it finds that it is feasible. If not, it forwards the placement job to a larger partition up the hierarchy. As a result, the scheme is capable of making accurate placement without incurring high overhead cost. The scheme takes into account the factors of long-term link cost, object popularity and size to compute the number of replicas per object. To improve latency, the scheme favors large objects over small ones for the similar popularity. Our simulation results show that the proposed scheme can significantly improve network performance with respect to latency, throughput, packet collision and communication/computation cost.

8. REFERENCES

[1] I. F. Akyildiz, X. Wang, and W. Wang, "Wireless mesh networks: a survey," *Comput. Netw.*, vol. 47, pp. 445–487, March 2005.

[2] H. Chen and Y. Xiao, "Cache access and replacement for future wireless internet," *Comm. Mag.*, vol. 44, pp. 113–123, May 2006.

[3] S. M. Das, H. Pucha, and Y. C. Hu, "Mitigating the gateway bottleneck via transparent cooperative caching in wireless mesh networks," *Ad Hoc Netw.*, vol. 5, pp. 680–703, August 2007.

[4] D. S. J. De Couto, D. Aguayo, J. Bicket, and R. Morris, "A high-throughput path metric for multi-hop wireless routing," *Wirel. Netw.*, vol. 11, pp. 419–434, July 2005.

[5] R. Draves, J. Padhye, and B. Zill, "Routing in multi-radio, multi-hop wireless mesh networks," in *Proceedings of the 10th annual international conference on Mobile computing and networking*, MobiCom '04, (New York, NY, USA), pp. 114–128, ACM, 2004.

[6] D. Passos, C. de Albuquerque, M. Campista, L. Costa, and O. Duarte, "Minimum loss multiplicative routing metrics for wireless mesh networks," *Journal of Internet Services and Applications*, vol. 1, pp. 201–214, 2011. 10.1007/s13174-010-0015-6.

[7] Z. Al-Arnaout, Q. Fu, and M. Frean, "A content replication scheme for wireless mesh networks," in *Proceedings of the 22nd international workshop on Network and operating systems support for digital audio and video*, NOSSDAV '12, 2012.

[8] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: a scalable peer-to-peer lookup protocol for internet applications," *IEEE/ACM Trans. Netw.*, vol. 11, pp. 17–32, Feb. 2003.

[9] T. Clausen, P. Jacquet, C. Adjih, A. Laouiti, P. Minet, P. Muhlethaler, A. Qayyum, and L. Viennot,

"Optimized Link State Routing Protocol (OLSR)," 2003. Network Working Group Network Working Group.

[10] M. Campista, P. Esposito, I. Moraes, L. Costa, O. Duarte, D. Passos, C. de Albuquerque, D. Saade, and M. Rubinstein, "Routing metrics and protocols for wireless mesh networks," *Network, IEEE*, vol. 22, pp. 6 –12, jan.-feb. 2008.

[11] J. Current, M. Daskin, and D. Schilling, "Discrete network location models," *Industrial Engineering*, pp. 81–118, 2002.

[12] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell System Technical Journal*, vol. 49, no. 2, pp. 291–307, 1970.

[13] A. Welsh, "The trimmed mean in the linear model," *The Annals of Statistics*, vol. 15, no. 1, pp. 20–36, 1987.

[14] B. Yang and H. Garcia-Molina, "Designing a super-peer network," *Data Engineering, International Conference on*, vol. 0, p. 49, 2003.

[15] A. Varga and R. Hornig, "An overview of the omnet++ simulation environment," in *Simutools '08: Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, (ICST, Brussels, Belgium, Belgium), pp. 1–10, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[16] A. A. Quintana, E. Casilari, and A. Triviño, "Implementation of manet routing protocols on omnet++," 2008.

[17] I. Baumgart, B. Heep, and S. Krause, "Oversim: A flexible overlay network simulation framework," in *IEEE Global Internet Symposium, 2007*, pp. 79–84, 2007.

[18] J. Kangasharju, J. Roberts, and K. W. Ross, "Object replication strategies in content distribution networks," *Computer Communications*, vol. 25, no. 4, pp. 376 – 383, 2002.

[19] G. Pallis, A. Vakali, K. Stamos, A. Sidiropoulos, D. Katsaros, and Y. Manolopoulos, "A latency-based object placement approach in content distribution networks," in *Proceedings of the Third Latin American Web Congress*, (Washington, DC, USA), pp. 140–, IEEE Computer Society, 2005.

[20] C. Cunha, A. Bestavros, and M. Crovella, "Characteristics of www client-based traces," tech. rep., Boston, MA, USA, 1995.

[21] B.-J. Ko and D. Rubenstein, "Distributed self-stabilizing placement of replicated resources in emerging networks," *IEEE/ACM Trans. Netw.*, vol. 13, pp. 476–487, June 2005.

[22] S. Jin and L. Wang, "Content and service replication strategies in multi-hop wireless mesh networks," in *Proceedings of the 8th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*, MSWiM '05, (New York, NY, USA), pp. 79–86, ACM, 2005.

[23] A. Al Asaad, S. Gopalakrishnan, and V. Leung, "Peer-to-peer file sharing over wireless mesh networks," in *Communications, Computers and Signal Processing, 2009. PacRim 2009. IEEE Pacific Rim Conference on*, pp. 697 –702, aug. 2009.

[24] J.-W. Ding, W.-T. Wang, and C.-F. Wang, "An efficient data replication scheme for peer-to-peer video streaming over wireless-mesh community networks," in *Intelligent Information Hiding and Multimedia Signal Processing, 2008. IIHMSP '08 International Conference on*, pp. 767 –770, aug. 2008.