Evaluating Load Balancing in Peer-to-Peer Resource Sharing Algorithms for Wireless Mesh Networks

Claudia Canali* IIT-CNR Pisa, ITALY

M. Elena Renda* IIT-CNR Pisa, ITALY

Paolo Santi* IIT-CNR Pisa, ITALY

Abstract

Wireless mesh networks are a promising area for the deployment of new wireless communication and networking technologies. In this paper, we address the problem of enabling effective peer-to-peer resource sharing in this type of networks. In particular, we consider the well-known Chord protocol for resource sharing in wired networks and the recently proposed MeshChord specialization for wireless mesh networks, and compare their performance under various network settings for what concerns total generated traffic and load balancing. Both iterative and recursive key lookup implementation in Chord/MeshChord are considered in our extensive performance evaluation. The results confirm superiority of MeshChord with respect to Chord, and show that recursive key lookup is to be preferred when considering communication overhead, while similar degree of load unbalancing is observed. However, recursive lookup implementation reduces the efficacy of MeshChord cross-layer design with respect to the original Chord algorithm. MeshChord has also the advantage of reducing load unbalancing with respect to Chord, although a moderate degree of load unbalancing is still observed, leaving room for further improvement of the MeshChord design.

1 Introduction

Wireless mesh networks are a promising technology for providing low-cost Internet access to wide areas (entire cities or rural areas), and to enable the creation of new type of applications and services for clients accessing the network. Among innovative applications enabled by mesh networking, we mention wireless community networks (see, e.g., the Seattle Wireless initiative [14]), in which users in a community (neighborhood, city, rural area, etc.) spontaneously decide to share their communication facilities (wireless access points) and form a wireless multi-hop network to be used by community members. Wireless community networks can be used to share the cost of broadband Internet access, but also to realize innovative services for the community, such as sharing of community-related resources, live broadcast of local events, distributed backup systems, and so on.

As the above mentioned innovative applications suggest, peer-to-peer resource sharing is expected to play an important role in forthcoming wireless networks based on the mesh technology. Hence, designing an efficient overlay network for enabling effective resource sharing can be considered as a fundamental middleware technology for mesh networks.

With respect to other types of wireless networks (e.g., MANETs), the availability of a wireless backbone of stationary routers in mesh networks lessen the well-known difficulties of enabling large-scale, peer-to-peer resource sharing. Recently, we have proposed MESHCHORD [3], a specialization for mesh networks of the well-known Chord algorithm [15], which exploits i) the availability of a wireless infrastructure and ii) the 1-hop broadcast nature of wireless communications, to realize a) location-aware ID assignment to nodes, and b) implementation of a cross-layering technique that bridges the MAC to the middleware layer.

MESHCHORD has been shown to have superior performance with respect to Chord, achieving a reduction of the total number of network-level packets exchanged to maintain the overlay structure and resolve queries in the order of as much as 40%. Yet, important issues are left unaddressed in [3], such as whether the load induced by Chord/MESHCHORD is evenly distributed among the wireless routers composing the mesh network. Load unbalancing is a major cause of performance degradation in mesh networks, as clients connected to relatively highly loaded routers experience a much lower QoS than those connected to relatively lightly loaded ones. Hence, evaluating the total number of exchanged packets gives only a partial view of the expected Chord/MESHCHORD performance, view which should be complemented with a careful evaluation of load distribution. Such an evaluation is the main contribution of this paper.

The second contribution of this paper is considering another method for resolving queries (key lookup operation)

*This work has been partially supported by IIT-CNR under Project P056: "Networks for Critical Infrastructures and Web Analysis".

978-1-4244-2575-4/08/$20.00 ©2008 IEEE

with respect to the one considered in [3]. More specifically, while in [3] we considered *iterative* query resolution, in which the intermediate results of a query are returned to the node which issued the query, in this paper we consider also the potentially more efficient *recursive* query resolution, in which intermediate results of a query are directly forwarded to the next node in charge of query resolution (details in Section 3). Since the query resolution primitive is invoked not only to resolve client-generated queries, but also to proactively maintain the overlay structure, a considerable reduction in packet overhead with respect to the case of iterative query resolution is expected.

The results presented in this paper show that MESHCHORD with recursive lookup implementation is the best solution in terms of total network-level traffic, and that MESHCHORD in both iterative and recursive case achieves also a better load balancing with respect to the basic Chord design. Nevertheless, a moderate degree of load unbalancing is still observed, especially in case of random router distribution, which might lead to the creation of hot-spots in large-scale mesh deployments. This calls for further improvements of the MESHCHORD design.

2 Related work and contribution

Recent papers have addressed the problem of enabling P2P resource sharing in mobile ad hoc networks (MANETs) [5, 8, 11, 13, 16]. A standard technique used to improve performance of P2P algorithms when used in wireless networks is cross-layering, i.e., taking advantage of information delivered from lower layer protocols (typically, the network layer) when constructing the logical links between peers. Approaches based on this idea are [4, 9, 10]. Although a careful design of the overlay improves the efficiency of P2P systems for MANETs, the combination of node mobility, lack of infrastructure, and unreliable communication medium has hindered the application of P2P approaches in medium to large size ad hoc networks.

Only a few recent papers have explored how P2P approaches can be applied to wireless mesh networks. In [1], the authors evaluate the gain that can be obtained from using network coding when running file sharing applications in wireless mesh networks, and conclude that some gain can actually be achieved, although not as much as in a wired network. In [6], some of the authors of this paper introduced a two-tier architecture for file sharing in wireless mesh networks: the lower tier is composed of mobile mesh clients, which provide the content (files/resources to share) to the P2P overlay; the upper tier is composed of stationary mesh routers, and implements a distributed hash table (DHT) for locating resources within the network.

The work that is more closely related to this paper is [3], in which we investigated the performance of the Chord P2P algorithm in typical wireless mesh network scenarios through extensive, packet-level simulations. Furthermore, we proposed the MESHCHORD specialization of the basic Chord design, which, by explicitly accounting for peculiar features of mesh networks, reduces total network-level traffic of as much as 40% with respect to Chord.

The investigation presented in this paper complements our previous work [3] in many respects. While the emphasis in [3] was on evaluating the *total* network-level traffic under varying network conditions, in this paper we consider *how the total traffic* generated by Chord/MESHCHORD *is distributed among the wireless routers*. The motivation for this study is that load unbalancing is a major cause of performance degradation in mesh networks and, say, a protocol generating a perfectly balanced load might be preferable to another protocol which generates a relatively lower total, but highly unbalanced, traffic load. Although Chord has been designed to balance the load (with high probability) among nodes *at the overlay level*, whether the load is also well balanced when *overlay links are mapped to (possibly several) physical links* of a wireless mesh network is an important open issue. Also, the effects of the specialized Chord design for mesh networks (MESHCHORD) on load balancing are still unexplored. To the best of our knowledge, the one presented in this paper is the first study of load balancing in peer-to-peer resource sharing algorithms for wireless mesh networks (and for wireless multi-hop networks in general).

Another major difference between this paper and our previous work [3] is the implementation of two alternative techniques (both part of the original Chord design) for key lookup implementation: *iterative* (the technique used in [3]) and *recursive*. Since the number of overlay-level packets generated to recursively resolve a query is at most as high as in the case of iterative query resolution in the basic Chord design, we expect a reduction also in terms of network-level packets. However, recursive query resolution impairs efficiency of MESHCHORD cross-layer mechanism (see Section 3 for details), and whether recursive query resolution is advantageous in combination with MESHCHORD is not clear.

3 Chord and MESHCHORD

3.1 Network architecture

Similarly to [3, 6], we assume a two-tier architecture for file/resource sharing: the lower tier of the architecture is composed of (possibly) mobile mesh clients (clients for short), which provide the content to be shared in the P2P system; the upper tier of the architecture is composed of stationary mesh routers (routers for short), which implement a DHT used to locate file/resources within the network. Unless otherwise stated, in the following we use the term *peer* to refer to a router forming the DHT at the upper tier of the architecture.

We assume routers are stationary, but they can be switched on/off during network lifetime. When a client u

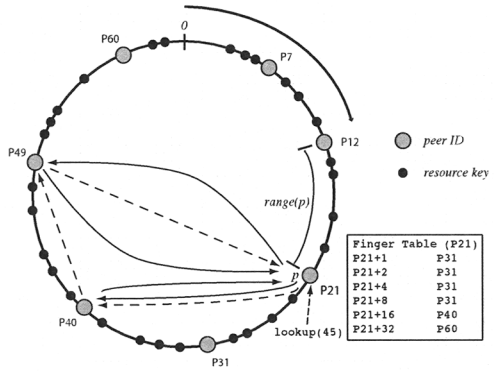

Figure 1. Basic Chord operations ($m = 6$). Iterative (solid) and recursive (dashed) *lookup* implementation.

wants to find a certain resource, it sends to its responsible router a (a mesh router within its transmission range) a *FindKey* packet, containing the key (unique ID) of the resource to find (see next section for details on key assignment to router/resources). The responsible router forwards the resource request in the DHT overlay according to the rules specified by the Chord protocol (see below), until the resource query can be answered. In case of successful query resolution, a packet containing the IP address of the client holding the requested file/resource is returned to client u through its responsible router a. For details on the rules for responsible router selection, on the procedures needed to deal with client mobility, and to add/remove resources from the distributed index, see [6].

3.2 Basic Chord operations

Chord [15] is based on the idea of mapping both peer (mesh router) IDs and resource IDs (keys) into the same ID space, namely the unit ring $[0, 1]$. Each key resides on the peer with the smallest ID larger than the key (see Figure 1), i.e., peer p manages keys comprised between its own ID and the ID of the predecessor of p in the unit ring (denoted $range(p)$). Chord maps peer and resource IDs into the unit ring using a hashing function, named $Sha1$, which has the property of uniformly distributing IDs in $[0, 1]$. Indeed, IDs in Chord are represented through m-bit numbers, i.e., at most 2^m distinct (peer or resource) IDs are present in the Chord system (we set $m = 24$ in this study).

The main operation implemented by Chord is the $lookup(x)$ operation, which can be invoked at any peer to find the IP address of the peer with ID$= x$ if x is a peer ID, or the IP address of the peer responsible of key x in case x is a resource ID. *lookup* operations are used both for query resolution and for overlay maintenance.

To speed up *lookup* operations, every peer maintains a table of up to m distinct peers (fingers). The i-th finger of peer j, with $1 \leq i \leq m$, is the peer which has the smaller ID larger than $j + 2^{i-1}$. Note that some of the fingers (especially for low values of i) can actually coincide (see Figure 1). In order to facilitate join/leave operations, each peer maintains also the ID of its predecessor in the Chord ring (peer $P12$ for peer $P21$ in Figure 1).

The $lookup(x)$ operation can be implemented in an iterative or recursive way. In the iterative implementation, when a $lookup(x)$ operation is invoked at peer p and the operation cannot be resolved locally (because x is not within $range(p)$), a packet is sent to the peer p' with largest ID $< x$ in the finger table of node p. If p' cannot resolve the *lookup* operation, it replies to peer p with a packet containing the ID of the peer p'' with largest ID $< x$ in its own finger table. Peer p then forwards the request to peer p'', and so on, until the *lookup* operation can be resolved (in at most m steps). Referring to Figure 1, a lookup operation for key 45 issued at node P21 is first forwarded to node P40, and then to node P49, which is responsible for the key and can resolve the *lookup*. In the recursive implementation, the intermediate response to a *lookup* operation (e.g., the ID of the peer p'' with largest ID $< x$ in p''s finger table) is not sent to the peer p which first invoked the $lookup(x)$ operation, but it is directly forwarded to the next node in charge of $lookup(x)$ resolution. Referring again to Figure 1, peer P40 does not return the ID of the next peer to interrogate (P49) to peer P21, but it directly forwards the $lookup(x)$ operation to peer P49. As the example of Figure 1 shows, recursive *lookup* implementation is more efficient in terms of overlay-level packet overhead with respect to the iterative implementation.

Chord also includes procedures to deal with dynamic join/leaves of peers in the systems, and to periodically updates peer finger tables. However, in this study we have considered only static network conditions, in which peers initially join the network, and remain active throughout the entire simulation time. This choice has been done to study load balancing induced by Chord/MESHCHORD in isolation, excluding possible effects due to dynamic join/leave of peers. For details on procedures for dealing with dynamic join/leave of peers, and their effects on total packet overhead, the reader is referred to [3]. Details on how to deal with client mobility at the lower tier of the architecture are given in [6].

3.3 MESHCHORD

Two modifications are implemented in MESHCHORD with respect to the basic Chord design: a) location-aware peer-ID assignment, and b) MAC-middleware cross-layering.

For what concerns a), the idea is to exploit locality, and to assign peers which are close in the physical network with close-by IDs in the unit ring. This choice is motivated by the observation that, according to Chord specifications, most of the packets are exchanged between a peer and its succes-

sor/predecessor in the unit ring.

More specifically, location-awareness is implemented by assigning IDs to peers according to the following function (see [6]):

$$ID(x,y) = \begin{cases} \frac{x\Delta}{s^2} + \lfloor \frac{y}{\Delta} \rfloor \cdot \frac{\Delta}{s} & \text{if} \lfloor \frac{y}{\Delta} \rfloor \text{ is even} \\ \frac{(s-x)\Delta}{s^2} + \lfloor \frac{y}{\Delta} \rfloor \cdot \frac{\Delta}{s} & \text{if} \lfloor \frac{y}{\Delta} \rfloor \text{ is odd} \end{cases},$$

where $ID(x,y)$ is the ID of a peer with coordinates $(x,y) \in [0,s]^2$, s is the side of the deployment region, and Δ is a parameter which defines the 'granularity' of location-awareness: the lower the value of Δ, the closer the peers must be in the physical network in order to be mapped into close-by regions of the unit ring[1].

For what concerns b), the idea is to use 1-hop broadcast nature of wireless communications to speed up *lookup* operations. More specifically, whenever a peer u receives a packet at the MAC layer, u sends it up to the middleware layer for further processing, even if the packet was not destined to u. If the packet does not contain a *lookup* request, it is discarded. Otherwise, u checks whether it may resolve the $lookup(x)$ operation. This occurs if x is comprised between u's ID and the ID of the predecessor of u in the unit ring. In this case, u sends a packet containing its own ID to the peer that originally invoked the $lookup(x)$ operation (called *source peer* in the following). It is important to note that, since the *lookup* primitive is invoked for both query resolution and overlay maintenance, cross-layering may improve the performance of both these operations.

It is important to observe that recursive *lookup* implementation challenges the above described cross-layering technique, due to the following reason. Suppose peer p is the source peer, and that resolving a $lookup(x)$ operation initiated by p involves forwarding the $lookup(x)$ primitive to peers $p_1, p_2, \ldots, p_k$, for some $k \leq m$. Suppose that, on the way to, say, peer p_1, the overlay packet is intercepted by peer p', which is the responsible peer of key x. Then, our cross-layering technique ensures that peer p' sends a packet with the result of the $lookup(x)$ operation directly to the source peer p, with the advantage of considerably speeding up the *lookup* operation. In case of iterative $lookup(x)$ implementation, when the source peer receives the first intermediate *lookup* result from peer p_1 (indicating that peer p_2 is the next peer to contact), it realizes that it already knows the result of the $lookup(x)$ operation. So, the source peer p can abort the rest of the $lookup(x)$ operation, and no further overlay-level packets are sent. However, if *lookup* is implemented recursively, the source peer p does not receive intermediate *lookup* results, and aborting the ongoing $lookup(x)$ operation is not immediate. Two possible choices can be considered here: i) *lookup* chasing, i.e., the source peer sends an $abort(x)$ packet on the overlay, whose purpose is intercepting the ongoing $lookup(x)$ operation and killing it; and ii) leaving the $lookup(x)$ operation alive, and ignore the final result when eventually received at the source peer (which already received the correct result from the intercepting peer p'). Note that which one of i) and ii) is more efficient in terms of exchanged overlay-level packets is not clear, since the number of packets exchanged to chase and abort an ongoing *lookup* operation might actually be higher than the useless packets exchanged to finish an already resolved *lookup*. In view of the above, and with the goal of keeping MESHCHORD design reasonably simple, we have opted for option ii).

[1]Note that the above location-aware ID assignment function requires that peers are aware of their location, which can be easily accomplished in wireless mesh networks through, e.g., the use of GPS receivers.

4 Performance evaluation

We have evaluated the performance of Chord and MESHCHORD on mesh networks using GTNetS, a packet-level network simulator developed at Georgia Institute of Technology [12].

4.1 Simulation setup

We considered two network topologies in simulations:
– *grid*: n peers are located in a square, equally-spaced grid; peer separation is $100m$;
– *random uniform*: n peers are distributed uniformly at random in a square area of side s, where $s = \sqrt{n} \cdot 100m$.

Grid deployments are a representative case of planned mesh deployment, while random uniform distribution accounts for unplanned mesh deployments (e.g., spontaneously created wireless community networks).

In both cases, we assume peers are equipped with $802.11b$ radios, the link data rate is $11Mbs$ with a typical $250m$ transmission range, and radio signal obeys free space propagation. For routing packets between far-away peers, we used DSR [7].

A certain number of queries is generated during Chord/MESHCHORD lifetime. Queries are generated uniformly over time (every t_{query} seconds); when a new query is generated, we uniform randomly choose the peer which issues the query on behalf of the client[2], and the ID of the key k to be searched is chosen uniformly at random in [0,1] (expressed as an m-bits binary number).

In order to better understand load balancing behavior, we have run separate set of experiments with and without client-generated queries. In case of no client-generated queries, what is evaluated is the packet overhead and load distribution for building and maintaing (e.g., updating finger tables) the overlay network. In both sets of experiments, the simulated time interval was $1200sec$, where the first $200sec$ were used to incrementally add peers to

[2]This corresponds to the situation in which clients are associated uniformly at random with mesh routers.

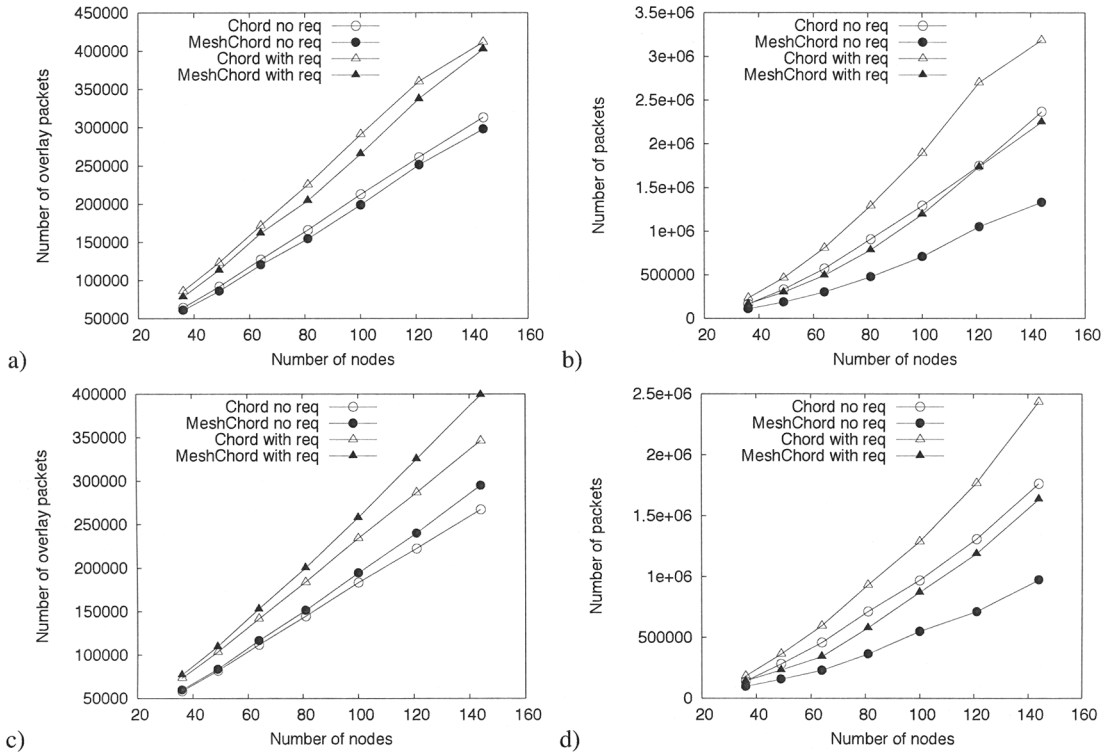

Figure 2. Total number of exchanged overlay-level and network-level packets for different values of n and grid topology: iterative (a)-b)) and recursive (c)-d)) $lookup$ implementation.

Chord/MESHCHORD, and to stabilize the overlay. In case of simulation with client-generated queries, these are generated after this initial $200sec$ stabilization period, with an average rate of n queries every $30sec$. The simulation results presented in the following are averaged over 10 runs.

The performance of the iterative and recursive version of Chord/MESHCHORD considered in our simulations is expressed in terms of:

1) *packet overhead*: total number of *network-level* packets exchanged by Chord/MESHCHORD to maintain the overlay, and to resolve client-generated queries;

2) *overlay packet overhead*: total number of *overlay-level* packets exchanged by Chord/MESHCHORD to maintain the overlay, and to resolve client-generated queries;

3) *load balancing*: we use two metrics to evaluate load balancing. The first metric is the well-known *balance index* (see, e.g., [2]), which is formally defined as follows: let L_i denote the load observed at the i-th peer, $1 \leq i \leq n$. The balance index β is defined as

$$\beta = \frac{\left(\sum_{i=1}^{n} L_i\right)^2}{n \cdot \sum_{i=1}^{n} L_i^2}.$$

The balance index is defined so that if the load is equally divided amongst the peers, we have $\beta = 1$; conversely, with highly unbalanced network load we have $\beta \approx 1/n$. Since it is known that relatively low values of β (say, below 0.8) occurs only if the load is extremely unbalanced, we have used also a more intuitive index of load balancing, namely the ratio max/min between the load observed at the maximally loaded peer over that observed at the minimally loaded one.

Note that metrics 1) and 3) are of fundamental importance in wireless mesh networks, where communication bandwidth used by the P2P application should be kept as low as possible, and load unbalancing (possibly causing network congestion) should be reduced to the minimum possible extent. Metric 2), on the other hand, is important to understand the efficiency of a design solution at the overlay level.

4.2 Communication overhead evaluation

Let us first consider the total number of overlay- and network-level packets for increasing values of network size, in case of iterative (Figure 2-a),b)) and recursive (Figure 2-c),d)) $lookup$ implementation. In the iterative case, while the number of exchanged overlay-level packets with MESHCHORD is only marginally smaller than with Chord, we have a considerable reduction in number of network-level packets (in the order of 30%). In the recursive case the situation is different: the number of overlay-level packets exchanged with MESHCHORD is *bigger* than with Chord, owing to the inefficiency of the cross-layering mechanism de-

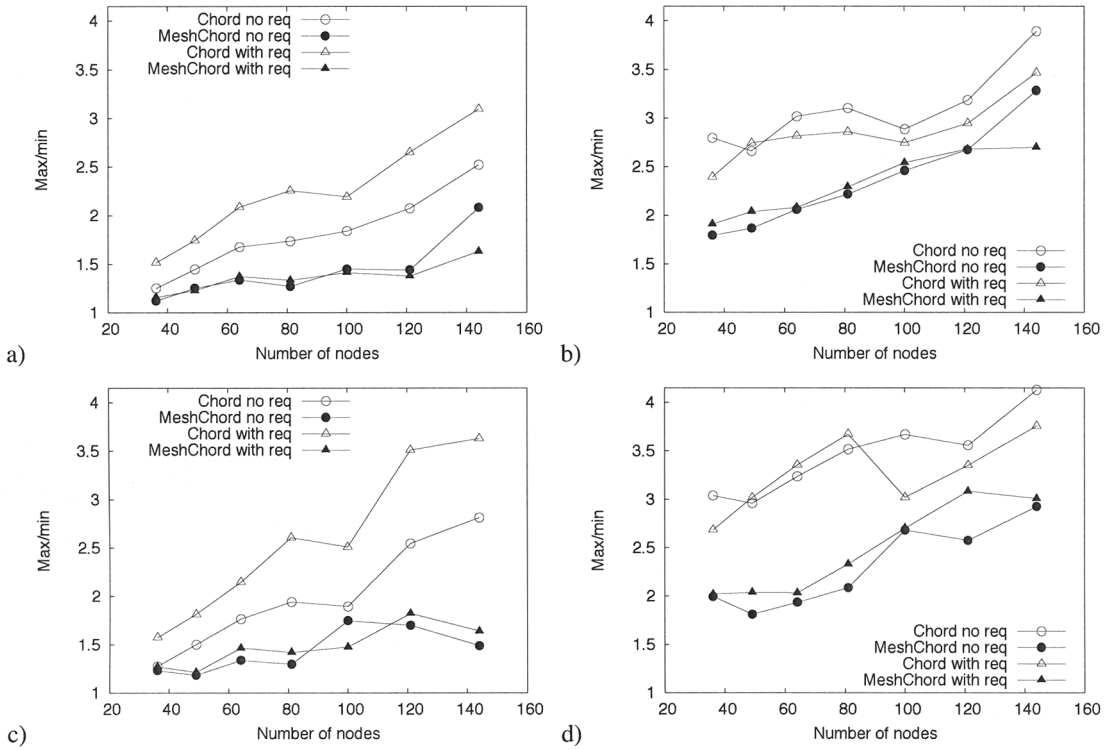

Figure 3. max/min ratio for overlay-level and network-level traffic, for different values of n and grid topology: iterative (e)-f)) and recursive (g)-h)) $lookup$ implementation.

scribed in Section 3. However, thanks to the better matching between overlay and physical links achieved by location-aware ID assignment, MESHCHORD is considerably superior to Chord in terms of number of exchanged network-level packets (percentage reduction in the order of 35%). When comparing the relative Chord/MESHCHORD performance in the iterative and recursive case, we observe as expected a better efficiency at the overlay level of the recursive technique (only marginal improvements in case of MESHCHORD, owing to inefficiency in cross-layering), and a considerable decrease in number of network-level packets for both Chord (in the order of 24%) and MESHCHORD (in the order of 27%) in case of recursive $lookup$ implementation. Thus, MESHCHORD with recursive $lookup$ implementation, despite some inherent inefficiency in the cross-layering mechanism, is the best solution for what concerns total network-level traffic.

4.3 Load balancing evaluation

Let us now consider how well the overlay-level and, most importantly, network-level traffic is distributed among the network nodes. Figures 3-a)–d) report the max/min ratio of both overlay- and network-level traffic, in case of both iterative and recursive $lookup$ implementation. We observe an increasing trend of load unbalancing with n, which indicates that the considered algorithms might actually lead to

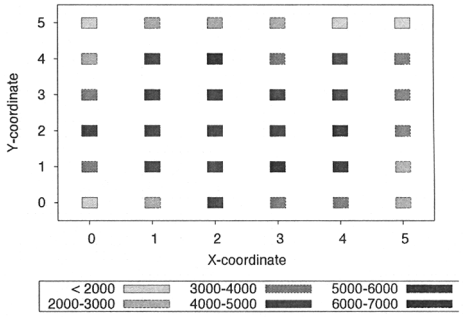

Figure 4. Load distribution of network-level packets in a sample instance with $n = 36$ nodes.

creation of hot-spots in large-scale mesh deployments. We also observe that the difference between iterative and recursive $lookup$ implementation in terms of load balancing is scarcely significant. The better overlay-level load balancing of MESHCHORD w.r.t. Chord is due to the fact that Chord was originally designed to balance overlay traffic *under the assumption that peer IDs are evenly distributed in ring*, which is actually the case with location-aware ID assignment on a grid topology (MESHCHORD). It is also worth observing the relatively worse load balancing which is achieved by both algorithms in terms of network-level

traffic, which is due to the fact that many network-level paths tend to cross the center of the deployment region. This effect can clearly be seen in Figure 4, which refers to a sample scenario with $n = 36$ nodes (Chord with recursive *lookup*). Overall, MESHCHORD provides better performance than Chord also in terms of load balancing, although a relatively moderate degree of load unbalancing is still observed (max/min ratio in the order of 3 for $n > 120$).

The above described trends are confirmed also by the load balance index β (plots not reported due to lack of space), which varied between 0.94 and 0.99 at the overlay- and between 0.95 and 0.97 at the network-level for Chord, and between 0.993 and 0.999 at the overlay- and between 0.965 and 0.98 at the network-level for MESHCHORD.

4.4 Random deployment

The results for the case of random peer deployment, which are not shown due to lack of space, have shown similar trends for what concerns the total network-level traffic. More specifically, MESHCHORD reduced total network-level traffic with respect to Chord in the order of 35% with iterative *lookup* implementation, and in the order of 42% with recursive *lookup* implementation. Recursive *lookup* implementation resulted again the best design choice, allowing a reduction of total network-level traffic w.r.t. iterative implementation in the order of 16% with Chord, and in the order of 45% with MESHCHORD. For what concerns load balancing, simulation results show that load is much less balanced than in the case of grid peer deployment. In particular, the max/min ratio (β) is as high (low) as 5 (0.9) for overlay-level traffic, and as high (low) as 19 (0.67) for network-level traffic. These values refer to Chord with either iterative or recursive lookup implementation. MESHCHORD achieves a better load balancing than Chord, yet the max/min ratio (β) is much higher (lower) than in the case of grid peer deployment: it is as high (low) as 3 (0.95) for overlay-level traffic, and as high (low) as 8 (0.84) for network-level traffic.

5 Conclusions

The main finding of the study reported in this paper is that, despite some inherent inefficiency in MAC cross-layering, MESHCHORD with recursive *lookup* implementation performs best among the considered middleware technologies, in terms of both total generated network-level traffic, and load balancing. However, a moderate degree of load unbalancing can be observed, especially with random router deployment (ratio between maximally and minimally loaded router in the order of 8), as well as a relatively higher degree of unbalancing with increasing network size. This indicates that application of MESHCHORD in large-scale mesh networks might lead to creation of hot-spots, calling for improvements of the MESHCHORD design. Currently, we are considering usage of carefully designed, load-aware hash functions to assign peer/resource IDs, in such a way that, say, relatively less resources are managed by relatively more congested peers. We expect that usage of such hash functions might achieve a better load balancing, while at the same time keeping network-level traffic at a low level.

References

[1] A. Al Hamra, C. Barakat, T. Turletti, "Network Coding for Wireless Mesh Networks: A Case Study", *Proc. IEEE Int. Symposium on a World of Wireless, Mobile and Multimedia (WoWMoM)*, 2006.

[2] A. Balachandran, P. Bahl, G.M. Voelker, "Hot-Spot Congestion Relief and User Service Guarantees in Public-Area Wireless Networks", *Proc. IEEE Workshop on Mobile Computing System and Applications (WMCSA)*, 2002.

[3] S. Burresi, C. Canali, M.E. Renda, P. Santi, "MeshChord: A Location-Aware, Cross-Layer Specialization of Chord for Wireless Mesh Networks", *Proc. IEEE PerCom*, pp. 206–212, 2008.

[4] M. Conti, E. Gregori, G. Turi, "A Cross-Layer Optimization of Gnutella for Mobile Ad Hoc Networks", *Proc. ACM MobiHoc*, May 2005.

[5] M. Denny, M. Franklin, P. Castro, A. Purakayastha, "Mobiscope: A Scalable Spatial Discovery Service for Mobile Network Resources", *Proc. International Conference on Mobile Data Management (MDM)*, 2003.

[6] L. Galluccio, G. Morabito, S. Palazzo, M. Pellegrini, M.E. Renda, P. Santi, "Georoy: A Location-Aware Enhancement to Viceroy Peer-to-Peer Algorithm", *Computer Networks*, Vol. 51, n. 8, pp. 379–398, June 2007.

[7] D.B. Johnson, D.A. Maltz, "Dynamic Source Routing in Ad Hod Wireless Networks", *Mobile Computing*, n. 353, pp. 153–181, 1996.

[8] A. Klemm, C. Lindemann, O.P. Waldhorst, "A Special-Purpose Peer-to-Peer File Sharing System for Mobile Ad Hoc Networks", *Proc. IEEE VTC-Fall*, Oct. 2003.

[9] G. Moro, G. Monti, "W-Grid: a Cross-Layer Infrastructure for Multi-Dimensional Indexing, Querying and Routing in Wireless Ad Hoc and Sensor Networks", *Proc. IEEE Conf. on Peer-to-Peer Computing*, 2006.

[10] A. Passarella, F. Delmastro, M. Conti, "XScribe: a Stateless, Cross-Layer Approach to P2P Multicast in Multi-Hop Ad Hoc Networks", *Proc. ACM MobiShare*, pp. 6–11, 2006.

[11] H. Pucha, S.M. Das, Y.C. Hu, "Ekta: An Efficient DHT Substrate for Distributed Applications in Mobile Ad Hoc Networks", *Proc. IEEE Workshop on Mobile Computing Systems and Applications (WMCSA)*, 2004.

[12] G. Riley, "The Georgia Tech Network Simulator," *ACM SIGCOMM MoMeTools Workshop,* 2003.

[13] F. Sailhan, V. Issarny, "Scalable Service Discovery for MANET", *Proc. IEEE PerCom*, 2005.

[14] `http://www.seattlewireless.net/`

[15] I. Stoica, R. Morris, D. Karger, M. Kaashoek, H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications", *Proc. ACM Sigcomm*, Aug. 2001.

[16] O.Wolfson, B. Xu, H. Yin, H. Cao, "Search-and-Discover in Mobile P2P Network Databases", *Proc. IEEE ICDCS*, 2006.