Available online at www.sciencedirect.com
ScienceDirect
Ad Hoc Networks 5 (2007) 680–703
Ad Hoc Networks
www.elsevier.com/locate/adhoc

Mitigating the gateway bottleneck via transparent cooperative caching in wireless mesh networks

Saumitra M. Das, Himabindu Pucha, Y. Charlie Hu *

Center for Wireless Systems and Applications, School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907, United States

Received 4 April 2006; received in revised form 13 October 2006; accepted 17 November 2006
Available online 20 December 2006

Abstract

Wireless mesh networks (WMNs) have been proposed to provide cheap, easily deployable and robust Internet access. The dominant Internet-access traffic from clients causes a congestion bottleneck around the gateway, which can significantly limit the throughput of the WMN clients in accessing the Internet. In this paper, we present MeshCache, a transparent caching system for WMNs that exploits the locality in client Internet-access traffic to mitigate the bottleneck effect at the gateway, thereby improving client-perceived performance. MeshCache leverages the fact that a WMN typically spans a small geographic area and hence mesh routers are easily over-provisioned with CPU, memory, and disk storage, and extends the individual wireless mesh routers in a WMN with built-in content caching functionality. It then performs cooperative caching among the wireless mesh routers.

We explore two architecture designs for MeshCache: (1) caching at every client access mesh router upon file download, and (2) caching at each mesh router along the route the Internet-access traffic travels, which requires breaking a single end-to-end transport connection into multiple single-hop transport connections along the route. We also leverage the abundant research results from cooperative web caching in the Internet in designing cache selection protocols for efficiently locating caches containing data objects for these two architectures. We further compare these two MeshCache designs with caching at the gateway router only.

Through extensive simulations and evaluations using a prototype implementation on a testbed, we find that MeshCache can significantly improve the performance of client nodes in WMNs. In particular, our experiments with a Squid-based MeshCache implementation deployed on the MAP mesh network testbed with 15 routers show that compared to caching at the gateway only, the MeshCache architecture with hop-by-hop caching reduces the load at the gateway by 38%, improves the average client throughput by 170%, and increases the number of transfers that achieve a throughput greater than 1 Mbps by a factor of 3.

Keywords: Wireless mesh networks; Cooperative caching; System design and implementation; Cross-layering; Internet access; Multi-hop wireless networks

* Corresponding author. Tel.: +1 765 494 9143; fax: +1 765 494 3376.
E-mail addresses: smdas@purdue.edu (S.M. Das), hpucha@purdue.edu (H. Pucha), ychu@purdue.edu (Y.C. Hu).

1570-8705/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.adhoc.2006.11.004

1. Introduction

Wireless mesh networks [2] are characterized by mesh routers connected by wireless links to each other and to a few gateway nodes. Recently, the deployment and use of WMNs has increased significantly and several cities have planned and/or deployed WMNs [43,49,48,42,45,47]. Thus, improving WMN performance will have a direct impact on a growing population of users. The most significant application of such networks is to provide broadband Internet access to static or mobile hosts in areas where wired infrastructure is difficult or economically infeasible to deploy. Since all Internet-access traffic flows through one or a limited few gateway nodes, it can cause significant congestion around the gateway.

Previous studies have shown that significant locality exists in Internet accesses from a given population of clients. Web caching has been proposed and extensively studied to exploit such locality in reducing the Internet-access traffic and the client-perceived access latency. We anticipate that similar locality will exist in the Internet-access traffic in WMNs once they become widely deployed. In this paper, we exploit such locality among the client Internet-access traffic in a WMN and explore content caching to mitigate the congestion bottleneck at the gateway nodes of a WMN.

One way of exploiting locality in client Internet accesses in a WMN is to have the client nodes in the WMN form a peer-to-peer network and perform cooperative caching directly with each other [18,40]. However, we argue that this approach has several disadvantages: (1) It does not leverage the available infrastructure in a WMN, i.e. the mesh routers; (2) It faces deployment challenges as it is non-transparent to clients and requires clients to contribute resources; (3) It requires the cooperative caching protocols to deal with churn and mobility of clients, e.g., clients could leave in the middle of a download; (4) It needs to deal with security and privacy issues in the presence of malicious clients which is not an issue if the infrastructure is deployed by a single service provider (e.g. Google WiFi). To avoid these disadvantages, in this paper, we focus on transparent cooperative caching at the mesh routers.

In this paper, we propose *MeshCache*, a transparent cooperative caching system that exploits locality in Internet-access traffic in a WMN to mitigate the gateway bottleneck effect. Unlike transparent web caching in the Internet where a web cache is attached to the gateway of an organization, MeshCache leverages the fact that a WMN typically spans a small geographic area and hence mesh routers are easily over-provisioned with CPU, memory, and disk storage, and extends the individual wireless mesh routers in WMN with built-in content caching functionality. It then performs *cooperative caching* among the wireless mesh routers. Currently available mesh routers (summarized in Table 1) have good processing speeds and can be expanded easily with USB-based or microdrive storage to also perform caching.

Table 1
Mesh router hardware specifications popular in WMNs

Mesh router device	Processor speed (MHz)	Memory	Storage
MeshCube	400	64 MB	USB expandable
LocustWorld MeshBox	500	128 MB	USB expandable
Soekris net4801	266	128 MB	USB expandable
Netgear WGT634U	200	32 MB	USB expandable
iBox Slim Mini-ITX PC	533	Up to 1 GB	USB expandable
Metrix Mark II	233	64 MB	USB expandable

Cooperative caching among mesh routers allows clients to fetch cached data from routers within the WMN. This spreads the load in the network and hence alleviates the congestion around the gateway. In addition, cooperative caching among mesh routers in a WMN has two other performance benefits: (1) Cooperative caching allows clients to potentially obtain content from nodes closer (in network hops) than the gateway. This improves the client throughput since the throughput falls rapidly with the increased hop count in multi-hop wireless networks. Further, such local communication improves the capacity of WMNs as they scale in size [25]. (2) When there are multiple cached copies of the requested content in the network, cooperative caching enables clients to choose the *best* cached copy based on high throughput link-quality routing metrics (e.g. [9]), thereby further improving client throughput.

In this paper, we explore two architectural design choices for MeshCache (A2 and A3 below), and compare it to a third one (A1), which is similar to the typical way that a web cache is deployed in the Internet:

- A1: A web proxy cache is connected to the WMN gateway node similar to how a web cache is attached to a gateway router in the Internet. This proxy cache transparently hijacks clients' content requests to exploit the locality within the client population of a WMN.
- A2: Each mesh router acts as a cache using expandable storage devices. When a client issues a request for a data object, its access mesh router transparently hijacks the request and searches for the object in its local cache. On a hit, the object is simply served from the access mesh router. On a miss, the access mesh router searches for a cached copy of the data object in other mesh routers including the gateway. If a copy is found, the access mesh router fetches the copy, otherwise it obtains the object from the origin server via a proxy cache at the gateway (as in A1).
- A3: In addition to A2, when the data object is fetched either from a gateway or another mesh router along a multi-hop route, the object is cached at each mesh router along the route. This increases the availability of the data object for future requests without explicit replication. To enable such *hop-by-hop caching*, we use *per-hop transport* that breaks a single end-to-end transport connection, e.g. S to D using route S–A–B–D, into multiple single-hop transport connections along the route, e.g. S–A, A–B, B–D, and pipelines data over these sub-connections. This enables the data object to be cached at A and B in addition to S.

The architectures A2 and A3 require a cache selection protocol to locate a cached copy of content and/or choose among multiple cached copies. To this end, we design and compare three cache selection protocols for MeshCache. Our designs leverage the abundant research results from cooperative web caching in the Internet.

We vigorously evaluate the performance of MeshCache using simulations and testbed experiments. Our evaluation results shows that A3 (cooperative caching with hop-by-hop caching) outperforms the other architectures in reducing the gateway load and improving the client throughput, irrespective of the cache selection protocol used. Additionally, we found that the per-hop transport for hop-by-hop caching in A3 provides increased content availability without adversely affecting transport throughput or network overhead compared to end-to-end transport. Further, for the best performing architecture A3, we found that the cache selection based on *limited broadcast flooding* is the best strategy to alleviate the gateway bottleneck and obtain throughput improvement. Particularly, our simulations showed that: (1) A2 increased client throughput by up to 50% and reduced gateway load by up to 35% compared to A1, (2) A3 increased throughput by up to 52% and reduced gateway load by up to 66% compared to A2, and (3) the best cache selection strategy for A3 is based on limited broadcast flooding and provides a reduction in gateway load by up to 20% compared to a strategy with no search delay or overhead. Our smaller scale testbed experiments also show that MeshCache can improve the overall throughput and reduce the network load significantly. Specifically, measurement results from a deployed implementation of MeshCache on the 15-router MAP mesh network testbed [28] show that the number of transfers that achieve a throughput greater than 1 Mbps is increased from 20% in A1 to 60% in A3 while the load at the gateway is reduced by 38% in A3 compared to A1.

The contributions of this paper are summarized as follows: (1) We propose to alleviate the bottleneck at the WMN gateway that commonly arises in WMNs by exploiting locality in client Internet accesses. (2) We present a practical cooperative caching system, *MeshCache*, and explore the design space of the MeshCache architecture and the associated cache selection protocols via extensive simulations. (3) We design and implement the MeshCache system by modifying an open-source caching proxy, *Squid*, developed for the Internet, and demonstrate the benefit of MeshCache using an implementation deployed over a mesh network testbed of 15 mesh routers.

The remainder of the paper is structured as follows. Section 2 presents a feasibility study of MeshCache by analyzing the potential locality in WMN traffic. Section 3 presents various architectural design choices for MeshCache and Section 4 presents various cache selection algorithms for MeshCache. Section 5 presents our simulation methodology and Section 6 presents detailed simulation results comparing different MeshCache architectures and cache selection algorithms. Section 7 presents the system design and implementation of MeshCache in a WMN testbed and Section 8 presents the measurement results from the testbed. Finally, Section 9 discusses related work and Section 10 concludes the paper.

2. Motivation

In this section, we motivate the cooperative caching approach of MeshCache by assessing the extent of locality in the Internet-access traffic expected to be carried by a WMN.

It has been shown that Internet-access traffic has substantial locality [7], i.e., multiple users are likely to request some common data objects from the Internet. Since WMNs primarily carry Internet-access traffic, significant locality is also expected to exist in the traffic of a WMN. However, the extent of locality in the traffic is dependent on the size of the client population. A fundamental question that determines the potential performance benefits of MeshCache is whether there exists significant locality in Internet-access traffic in a WMN, given the small client population served by each gateway router.

While no measurement study has been performed specifically for Internet client traffic in WMNs, we can approximate the client population served by a gateway node of a WMN with that seen by the gateway proxy cache of a small organization. For such an approximation, we analyzed real web proxy traces collected in the week of October 19, 2005 by www.ircache.net. The collective trace of 1 day's traffic from 10 proxies contains 2.7 million requests originating from 1151 unique clients to 81,289 servers. The number of clients and requests in each proxy's trace are depicted in Fig. 1a and b, respectively. The number of clients in each trace ranges from 80 to 160 nodes which is a potential target size for a WMN with a single gateway. For example, a recent work [14] shows that 114 users can potentially be supported with a 21 mesh router WMN.

We studied the locality and working set size of the cacheable content in each trace.[1] The locality in the access patterns of such a small set of clients is encouraging as the average hit rate is around 37% with a maximum of 46% (Fig. 1c). This suggests that a significant fraction of requests can be fetched from peer mesh routers if caching is enabled. The hit rate also tends to increase with the client population. We note these observed hit rates are consistent with other studies [18,38] on web caching using different proxy traces for client populations of similar sizes. To study how the client population affects the hit rate, we combined all the traces into one trace and simulated different client population sizes ranging from 50 to 500. For each population size k, we selected all the requests from k randomly chosen clients and ran a simulation to find the hit rate and the working set size. For each k, we tried 25 random trials to find average behavior. The results in Fig. 1e show that the hit rate increases gradually from 35% to 45% as the client population grows from 50 to 500. In summary, there is significant locality to exploit in smaller client populations such as those in WMNs.

Another property of the Internet-access traffic which determines the feasibility of caching is the working set size of the traffic. Fig. 1d shows the working set of a day's worth of traffic to be 1.3 GB on average with a maximum of 2.6 GB. We also measured the working set size as a function of the client population. Fig. 1f shows that as the client population grows from 50 to 500, the working set size grows from 500 MB to 5.2 GB for a day's worth of traffic. These sizes can be accommodated in current mesh routers throughput expandable storage devices.

3. MeshCache architecture

Before discussing the architectural design choices for MeshCache, we present our basic network model. Without loss of generality, this paper assumes that MeshCache is deployed in a WMN similar to RoofNet [43], consisting of single-channel, single-interface 802.11 mesh routers with omnidirectional antennas and sparsely deployed gateways for Internet access. The gateways are not widely deployed due to cost and uplink constraints. A similar architecture can also be used for corporate networks to replace wired access-point-based WLAN systems [14]. Fig. 2a depicts a typical network setup.

Such a WMN provides Internet access as follows: Each client's packets are first received by the client's *access mesh routers*, i.e., a mesh router the client's interface is associated with (e.g., MR 1 and MR 6 are access mesh routers for their clients). These mesh routers then forward the packets to the *gateway mesh router* (GMR) using other mesh routers. The GMR provides Internet connectivity through a high bandwidth wired/WiMax interface. The gateway may perform other functions such as IP address assignment or NAT. All the MRs use a routing

[1] Similar to in [18], we consider requests with SSL and dynamic content as not cacheable and always resulting a miss. We also limit the maximum size of any single cacheable object to 16 MB similar to many deployed web caching systems.

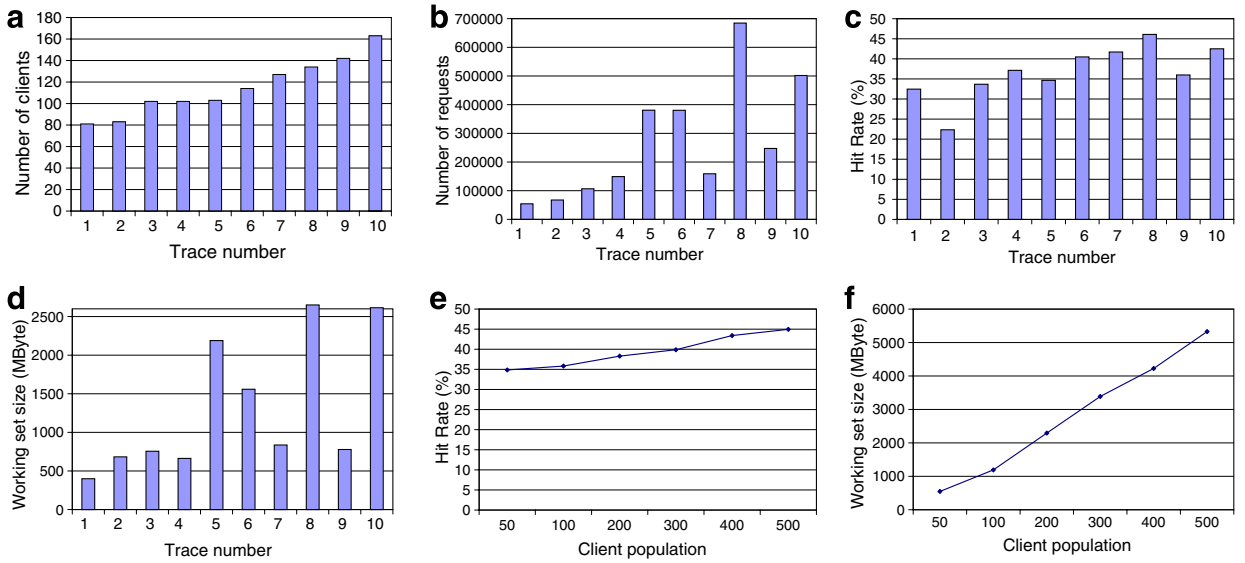


Fig. 1. Study of locality in small client populations. (a) Clients in each trace, (b) requests in each trace, (c) hit rate for each trace, (d) working set for each trace, (e) hit rate v/s population and (f) working set v/s population.

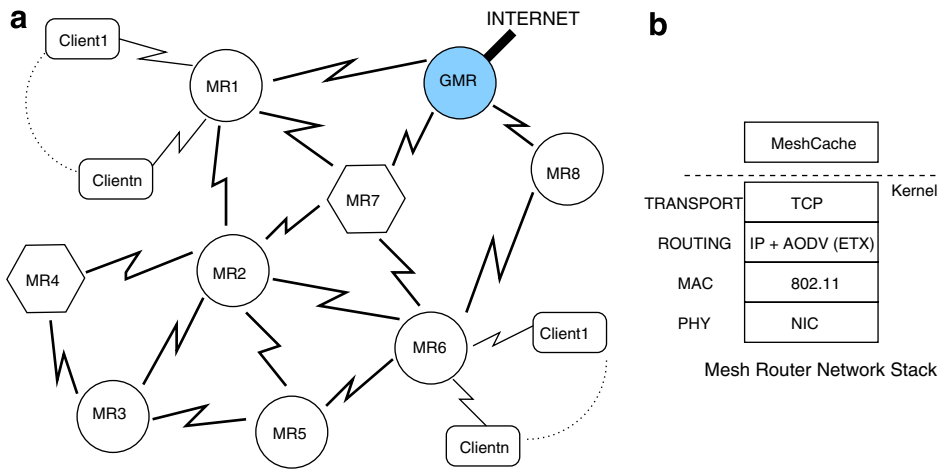


Fig. 2. MeshCache architecture. (a) Basic network model and (b) operational layer.

protocol (e.g. AODV) with metrics such as ETX [9], ETT [13] to find routes to each other and to the gateway. We assume that clients use TCP at the transport layer as it is widely used for Internet access. Further, we assume that the WMN employs a 802.11 MAC layer. As shown in Fig. 2b, the MeshCache system is implemented over this underlying WMN by a user-level MeshCache daemon.

3.1. Architectural design choices

We now discuss the architectural design choices for MeshCache that differ in where caching is performed to exploit locality and the implications of such choices.

(a) Architecture 1 (A1): In A1, a caching proxy is connected to the WMN gateway node similar to how a web cache is attached to a gateway router of an organization in the Internet. Client requests are transparently hijacked at the GMR and redirected to the caching proxy. As the results in Section 2 showed, the hit rates in the attached caching proxy are expected to be in the range of 30–40%, resulting in significant bandwidth savings between

the GMR and the origin server. The requests with hits in the caching proxy could also have higher TCP throughput as they will not be affected by TCP's inefficiencies in traversing wireless-wired links [39]. However, the bottleneck at the gateway still remains since the medium access load around the gateway is not reduced in this architecture.

(b) Architecture 2 (A2): A key observation in WMNs is that a WMN typically spans a small geographic area and hence mesh routers are easily over-provisioned with CPU, memory, and disk storage, and the MRs themselves can be potentially leveraged to cache content and provide further performance improvements. A survey of hardware specifications (Table 1) of some popular choices for implementing MR devices shows that these devices already have adequate processing power to implement a caching proxy. Unlike high-end web proxies and routers which service thousands of connections simultaneously, the smaller scale of a WMN reduces the computational load on the MRs and makes implementing a caching proxy on each of them computationally feasible. In addition, all of the devices can be expanded to include USB flash drives (which can store 4–8 GB currently) and some can even use USB hard drives with capacities of up to 20 GB. The small form factor and declining cost of the USB flash drives makes them a suitable candidate for addressing the storage needs of mesh routers. Moreover, Section 2 shows that the working set of a small client population is small enough to be accommodated by these USB devices. Cooperative caching at all the MRs further reduces the individual cache sizes required to accommodate the working set.
To exploit the above observation to enhance MeshCache's performance, in architecture A2, all MRs performs caching in addition to routing. When a client requests content, its request is transparently hijacked by the client's access MR and the content is searched for in the access MR's cache. On a local hit, the content is simply served from the access MR. However, on a local miss the access MR searches for a cached copy of the content in other MRs (including GMR). If a copy is found, the access MR fetches the copy, locally caches the content and returns it to the client. Otherwise, the access MR fetches the content from the origin server via a proxy cache at the gateway (as in A2), locally caches the content, and returns it to the client. Thus, the main difference between A2 and A1 is that *all* MRs can cache data if they are required to. They could cache content simply because they are the access MR or for example, if the content is hashed to them.

(c) Architecture 3 (A3): Architecture A3 builds on A2 and pushes the cooperative caching technique to the extreme. Note that when content is fetched either from the origin server (via GMR) or another mesh router along a multi-hop route, the MRs along the path over which data flows can also cache the data. This increases the availability of content for future requests without explicit replication. Increased availability of data improves the cache hit rate and reduces the path length of future requests.

Ideally, this increased availability of cached content can be achieved without extra overhead if each node along the path can snoop network layer packets to assemble an entire file. However, this is difficult to achieve in practice because: (1) Routing paths may fluctuate during a transfer and the bytes of a file may travel different paths. (2) Routing paths in forward and reverse directions may be different in many cases, e.g., due to unidirectional links. We found this to occur in our testbed (Section 8). (3) Even if all bytes of a file pass through each MR in a path during its download, it is difficult to decipher the transport and application protocol states to assemble the packets into the application file.

To enable hop-by-hop caching, we resort to *per-hop transport*. Per-hop transport breaks a single end-to-end transport connection, for example, from S to D using route S–A–B–D, into multiple single-hop transport connections along the route, e.g., S–A, A–B, B–D, and pipelines data on these sub-connections. This allows the data object to be cached at A and B in addition to S while it flows through the route. Thus, content is transferred over an N-hop path through N pipelined transport connections, resulting in the file being cached at each intermediate node in its entirety. This per-hop transport mechanism essentially *fixes* the route for the duration of a file transfer in order to cache it along the path. We argue that this is viable since most content requests (i.e. HTTP) are small in size, and hence the

underlying routing path is unlikely to deteriorate in the time the route is fixed. Note that per-hop transport precludes unidirectional links from being used for transferring content since a bi-directional TCP handshake is required at each hop. In summary, A3 is similar in operation to A2, except that when the a data object is fetched, the object is cached at every hop along the route by using per-hop transport.

In hop-by-hop transport, all the mesh routers along a path (including the gateway) initiate persistent connections, which simply replay client/server behavior. Only when the client or server explicitly closes the connection, all the hop-by-hop connections will close. Thus, if the client's browser uses a persistent HTTP connection, the persistent connection semantics is preserved from the client's point of view.

While A3 maximizes the opportunities for caching among the wireless mesh routers, the per-hop transport requires data packets to traverse up the TCP/IP stack at every hop which incurs extra overhead and delay compared to the end-to-end transport. We study these issues thoroughly in Section 6.1. In addition, per-hop transport requires per-connection state to be set up on all nodes along the path. While this may not be a good idea in mobile ad hoc networks which typically consist of resource constrained devices, it is less of a concern for the mesh routers which are less resource constrained.

An important component in both architecture designs A2 and A3 is an efficient cache selection protocol to locate a cached copy of the content. We next discuss the design choices of a cache selection protocol.

4. Cache selection protocols

The design of cache selection protocols has been widely studied in the context of Internet web caching systems. Thus, we first provide a brief background of the approaches used in the Internet and draw inspiration from them in designing cooperative cache selection protocols for WMNs.

4.1. Cooperative caching in the Internet

Several previous works have proposed cooperative cache selection protocols for the Internet. The works in [10,8] introduced *hierarchy-based selection*, i.e., using a hierarchy of caches that resolve MISSES from lower levels of the hierarchy until the root is reached which fetches content from the origin server. Another technique of *reactive query-based selection* is exemplified by ICP [36] which on a MISS, queries its peer caches for the content item and chooses one of them to forward the request to. To remove the delay and processing overhead of query-based selection, *hash-based selection* (e.g. CARP [35]) has been proposed in which on a MISS, a cache fetches the content item from a cache selected by hashing the content item's URI (Uniform Resource Identifier). Finally, in *proactive dissemination-based selection* [15,32], caches proactively distribute summarized information about their cached content items (using Bloom filters) to each other to remove the delay penalty from queries.

However, *the solutions proposed for cooperative cache selection in the Internet domain are not directly applicable to MeshCache* for the following reasons: (1) They do not optimize the network overhead in terms of the number of packets in the system which is important in WMNs. (2) They fail to incorporate advantages possible from the broadcast nature of wireless communication. (3) Since the mesh router itself is a cache in a WMN, unique cross-layer approaches can be designed to improve the cache-selection schemes (e.g. consulting the underlying routing protocol to choose from multiple cached copies based on routing metrics such as ETX [9]). Solutions developed in the Internet can only exploit the caches themselves and not the routers in between to perform cache-selection decisions. Thus, they are restricted to using application-specific metrics such as end-to-end latency.

In the next section, we explore the design space of cooperative cache selection protocols for the MeshCache system. The design for these cache selection protocols is inspired by their counterparts in the Internet, but have been adapted to operate efficiently in the wireless environment. All the protocols are implemented at the application layer to allow them to be useful for a multitude of WMN architectures. For example, they should be able to operate on top of source routing as well as hop-by-hop routing protocols. Additionally, they should operate over and be able to exploit protocols implemented with new routing metrics or multi-channel multi-radio devices [12,30]. They should also be operable with any new transport protocol that may be invented for WMNs. At the same time, these protocols should leverage the information exposed by the underlying layers using a cross-layer

approach to enhance their performance. To satisfy both these requirements, MeshCache adopts a loose coupling principle whereby MeshCache cache selection protocols interact with underlying layers via a set of predetermined APIs. This enables portability to any underlying layer that provides the APIs as well as performance benefit arising from the interaction with lower layers.

4.2. Cache selection protocols for architecture A2

The cache selection protocol in MeshCache should select a suitable MR for each content item, and retrieve the content from the chosen MR. In the following, we present a set of design choices for the cache selection protocol.

1. *Tree-based hierarchy cache selection protocol (THCP):* This is a basic scheme in which the access MR simply routes the content request to the GMR (gateway) in case of a local miss. The access MR selects the current best gateway[2] by querying the routing protocol (through an API *BestGateway()*) and forwards the request to the corresponding GMR. This is a two-level hierarchy-based cache selection approach with the GMRs being the parent caches for other MRs. We do not perform gateway selection at the application layer in any cache selection protocol because the routing protocol has more fine-grained information about the *best* gateway, through probing of link metrics such as loss rates [9], latency [1] or complex metrics that take into account link bandwidths and loss rates in the presence of multiple radios and multiple channels [13].

Following the selection of a GMR, an end-to-end transport connection is established to the selected GMR via a multi-hop network layer path and the content transferred.

2. *Broadcast cache selection protocol (BCP):* While THCP does not cause search overhead or search delay, it can only exploit local hits at each access MR and hits at the selected GMR, and not at the caches of other MRs in the vicinity. In contrast, the broadcast-flooding search-based protocol (BCP) can locate content items in other MR nodes.

In this protocol, on a local miss, the access MR first queries the routing protocol for the path metric, say ETT (expected transmission time) [13] X, to the closest gateway (through an API *BestGatewayMetric()*). The access MR then initiates a UDP broadcast of a *content locate* message by inserting the metric X, a search path metric $Y = 0$ and a locally unique sequence number. As the *content locate* message is propagated, each node rebroadcasting the message exactly once based on the sequence number. In doing so, each node also adds the ETT to the node it got the packet from into the value Y when rebroadcasting the message. If at some node the value Y exceeds X, there is no need to search further since the originating node already has a better path, i.e., to the gateway, and the broadcast is terminated at the node. In this manner, the broadcast search is limited to paths better than the one to the gateway.

Each node that has a hit in its local cache for the content item replies with a *content found* message and does not rebroadcast the *content locate* message. The access MR then queries its routing protocol to find a node with the best path metric from among those that had hits (through an API *Best Node*(⟨*node IP list*⟩)) and selects the MR returned by the API to fetch the content. Note that if all nodes with hits have a worse routing metric than the gateway or no other nodes have a cached copy of the content, the gateway itself is chosen (same as THCP). The data is once again transferred over a multi-hop network layer path.

Another important parameter in BCP is search timeout (ST) which is the amount of time a node should wait to receive replies from a broadcast search before reverting to the default behavior (THCP). BCP uses the following technique to estimate ST: Whenever a new closest gateway is discovered by the routing protocol, a node pings that gateway once to get an approximation for the round trip latency from the node to the gateway. This ping time is then used by BCP as the search timeout. Effectively this method adapts the search timeout at each node with respect to its closest gateway.

BCP draws its inspiration from the ICP protocol for the Internet. However, there are some important differences in the operation of the two protocols. First, BCP exploits wireless multicast advantage (WMA), i.e. the ability to deliver multiple query packets with a single transmission [37]. Second, the scope of the BCP queries is limited by exploiting the proximity of the gateway. Both these differences reduce the overhead and the delay of BCP compared to ICP. Thus we take an approach of loose coupling between the two layers: Search at the application layer and determine the best cache by

[2] There may be multiple gateways advertising Internet connectivity in a WMN.

querying an API that can be provided by any routing protocol with a few modifications.

3. *Geographic hash cache selection protocol (GHCP):* In GHCP, on a local miss, the access MR uses a well-known hash function to hash the content item's URL and maps it to a MR whose nodeID (hash of IP) is numerically closest to the hash of the URL.[3] The request is then forwarded to the hashed MR with the expectation that a hit will occur since all requests for that particular URL are redirected to the hashed MR. If the hashed MR has a miss, it uses THCP to fetch the content.

Unlike Internet hash-based cache selection protocols, GHCP controls the proximity of MRs to which the content items are hashed. The hash is not done globally by considering all the MRs in the WMN (in which case the path lengths to the hashed MR could increase dramatically). Instead, the entire area is divided geographically into virtual squares (grids), and each node hashes the URL by considering *only other MRs contained in its own virtual grid*. The grid size is a tradeoff between exploiting locality and the path length. If it is too small, there likely will be no locality and no hits; if it is too large the hash point will be far away and thus the throughput will either be bad or the gateway will always be chosen. To balance the tradeoff, we chose a grid size equal to $500\,\mathrm{m} \times 500\,\mathrm{m}$ in our evaluations.

In GHCP, if the hashed MR is still determined (by querying the routing protocol) to be worse in routing metrics than the best gateway, the request is simply redirected to the best gateway. Note that this scoped hashing requires knowledge of the global virtual grid boundaries and locations of the MRs to determine the MRs in each node's virtual grid. In a static WMN network, locations of mesh routers and grid boundaries can be encoded during deployment. We use this one-time encoding method in our testbed. Other deployed testbeds such as RoofNet [43] also maintain the GPS coordinates of their MRs.

4.3. Cache selection protocols for architecture A3

The difference between the architectures A2 and A3 is that when content flows back to the access MR in A3, it is cached at nodes along the way. This caching is enabled by per-hop transport. Per-hop transport naturally forms a chained (hierarchical) organization of the caches along the way which can be leveraged to further optimize the cache selection protocols in A3. Both THCP and GHCP benefit from this optimization by searching for the request data object in this chain of caches in addition to the caches selected by their cache selection algorithms. The cache selection protocols for A3 are described below.

1. *Per-hop tree-based hierarchy cache-selection protocol (PH-THCP):* PH-THCP is an optimization of THCP when architecture A3 is used. In this scheme, the access MR, on a local miss, selects a best gateway G as before and then finds the next hop node for G by querying the routing protocol (through the API *GetNextHopForNode*(G)) and establishes a transport connection to that next hop node. For example, in Fig. 3 MR 4 has a miss in its local cache and consequently contacts MR 2. MR 2 repeats this process and on a miss forwards the request to the next hop towards G. If any intermediate node has a hit, it sends the content back to MR 4 and does not search further. Each node on the path then automatically caches the content as it is downloaded through hop-by-hop transport connections. Subsequently, when MR 1 has a miss and contacts MR 2, a hit occurs due to hop-by-hop caching and the content is fetched directly from MR 2.

2. *Per-hop broadcast cache-selection protocol (PH-BCP):* PH-BCP is a per-hop variant of the BCP protocol for architecture A3. In this scheme, the method to select the MR with the requested content is same as in BCP. The only difference is that the content is transferred via per-hop transport from the selected MR. Also, if no hits occur or the metrics for all the hits are worse than the gateway, the access MR reverts to using PH-THCP to forward the request to the gateway. For example, in Fig. 3, MR 7 has a miss in its local cache and no hits from the broadcast search. It then uses PH-THCP to fetch the content from the gateway through MR 6. Subsequently, when MR 1 performs a broadcast search for the same content item, it receives a hit from MR 6 from where it receives the content, again with per-hop caching. Thus, MR 1 achieves better performance due to broadcast search and hop-by-hop caching compared to using end-to-end transport over the default path of 1–2–3–GW (shown with dotted line). PH-BCP has a delay penalty associated with it, but can be better

[3] This is consistent hashing, same as that used in DHTs in the Internet.

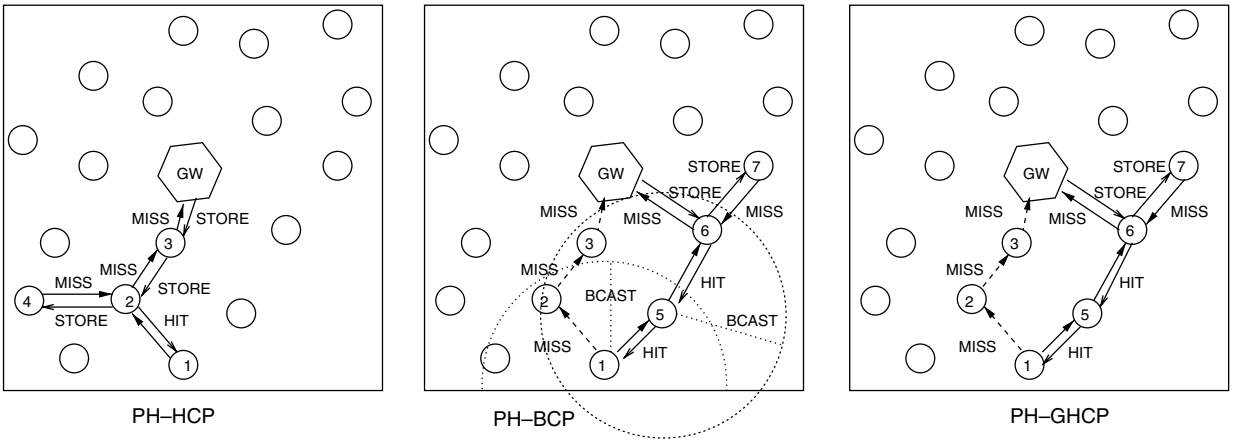


Fig. 3. Design choices for cache selection protocols.

at spreading load, as the content may be fetched from a node not on the path towards the gateway.

3. *Per-hop geographic hash cache selection protocol (PH-GHCP):* PH-GHCP is a per-hop variant of the GHCP protocol for architecture A3. In this scheme, the access MR upon a local miss hashes the URL of the content item to a MR in its virtual grid (say B), similar to in GHCP. If B is closer than GMR, it then finds the next hop node towards B by querying the routing protocol (through the API *GetNextHopForNode(⟨nodeIP⟩)*) and the request is forwarded in a hop-by-hop fashion to B. If an intermediate node has the content, it does not forward the request further and replies to the access MR. Note that it is possible B itself does not have the content, e.g., if this is the first request for a content item. In this case, B reverts to PH-THCP to retrieve the content from the gateway and forward it to the access MR. For example, in Fig. 3, MR 7 hashes the URL to MR 6 which on a miss fetches the content from the gateway. Subsequently, MR 1 also needs to retrieve the same content item and thus hashes the URL to node 6 from which the content item is retrieved in a hop-by-hop fashion improving performance from its default route of 1–2–3–GW. Note that although PH-GHCP has no query delay, it can cause detours in routing paths without any benefit for unpopular content items since they result in a miss at the hashed MR.

Finally, we argue that the approach of proactive dissemination of content indices used in Internet cooperative web caching is not suitable for WMNs (both A2 and A3) as proactive message exchange can cause high overhead to the wireless network. Nodes would have to proactively flood the network whenever they cache a new object, a cached object expires, or a cached object is evicted.

Note that in all protocols, we implement the routing protocol API to only provide next hop nodes that are connected bi-directionally. In the absence of this feature, hop-by-hop transport connections cannot be established.

5. Methodology

In the next section, we examine the feasibility of per-hop transport by comparing it to end-to-end transport, and compare the A1, A2 and A3 Mesh Cache architectures and the associated cache-selection protocols through detailed simulations. We describe our methodology for those experiments in this section.

We use the Glomosim simulator [41] to evaluate MeshCache. Glomosim has been widely used to study multi-hop wireless networks.

(a) *Network model:* We simulate a static mesh network of 50 mesh routers placed randomly in an area of $1000\,\text{m} \times 1000\,\text{m}$. Each node is assumed to have one interface equipped with an omnidirectional antenna. All the sources communicate with the gateway node to simulate an Internet access pattern. The two-ray path loss propagation model is used. We also evaluate the performance under fading (Rayleigh) and lossy conditions.

(b) *Client behavior model:* Each mesh router aggregates queries from five clients. For each individual client, a successive request arrives after the current request has been served.

However, the mesh router may service requests from different clients concurrently. The file request model is similar to previous caching studies [40].[4] The file request pattern of each client is based on the Zipf-like distribution that has been found to model web traces. The value of the Zipf parameter θ was chosen to be 0.8 based on measurements on web traces and similar to in previous significant studies in this area [40]. Similar to in [40], we used a request pattern in which nearby nodes have similar, although not the same request popularity distribution. The user browsing behavior is simulated using the model developed in [26]. This model also allows us to simulate the size of HTTP items retrieved, number of items per "Web page", and think time (the time elapsed between successive web page downloads from each user). We assume that 25 mesh routers have active clients.

(c) *Content model:* We assume a set of 1000 files with file sizes between 1 and 100 KB from which each client makes a request based on the Zipf distribution. Similar to in [40], content items have a TTL of 5000 s. The mesh router cache size is set to 2 MB due to the working set size and length of the experiment. Note that a production system will use much larger caches using expandable flash storage and main memory. Finally, we use the LRU cache replacement policy used in the extended Squid [46] caching proxy evaluated in our testbed evaluation in Section 8.

(d) *Routing, transport and MAC:* We currently use TCP as the reliable transport protocol in our study since it is widely used and available as a standard part of operating systems. The routing protocol used is AODV, same as that used in our testbed evaluation in Section 8. We used IEEE 802.11b as the MAC/PHY layer which was verified to produce close to theoretically maximum throughput [19]. All simulation results were averaged over multiple random scenarios and each simulation modeled a 1-h period of client activity.

(e) *Metrics:* The metrics used in our evaluation are: (1) Network load: The average total number of packets transmitted by the network layer for a single file download. This accounts for all control as well as data packets, and is averaged across all the transfers initiated in the network. (2) Throughput: Throughput of any transfer is the ratio of the size of the transfer to the time taken to complete the transfer. (3) Average aggregate throughput (AAT): Aggregate throughput is the ratio of the total number of bytes downloaded by a single client to the total time spent by that client downloading those bytes. AAT is the average of the aggregate throughput of all the clients. (4) Hit rate: A cache hit occurs in the MeshCache system when a client has a hit in its access mesh router's cache. We refer to this as a *local* hit. Local hit could be due to another client connected to the same access MR requesting the same object. Another possibility is that when per-hop transport is enabled, an MR is an intermediate hop for a transfer to a different MR and hence has cached the content item. Now, when the intermediate MR's client requests the same object, a cache hit occurs. We log the latter type of cache hits and refer to them as *ph-local* hits. Both local and ph-local hits result in the content being fetched directly from the access MR without any communication with peer MRs or the gateway. A cache hit also occurs when an access MR obtains the content item from its peer MRs instead of the gateway. We refer to this as a *remote* hit. Once again, remote hits could be due to locality in the access pattern of two routers or due to per-hop transport. We log all remote hits.

6. Performance evaluation of MeshCache architectures

In this section, we first evaluate the feasibility of architecture A3 by studying the performance of per-hop transport mechanism with respect to the download size, hop length and wireless environment.

6.1. Impact of per-hop transport

Per-hop transport is an elegant method to achieve hop-by-hop caching in A3. However, it is

[4] We did not use the traces in Section 2 for the simulations as we wanted to experiment with different locality parameters and use similar workloads as existing work on caching for mobile networks [40].

essential to understand the impact of such a per-hop mechanism on the throughput and the overhead of the transport connection. Note that per-hop transport requires establishing and growing multiple TCP connections and may require application layer data buffering due to the mismatch between upstream and downstream throughput. However, in order to conserve memory at the intermediate hops, this study uses the default 16K in-kernel socket buffers for each TCP connection and a constant application-level buffer of 8K. We first evaluate the impact of the number of hops on the per-hop transport connection.

1. *Performance with varying path length:* In this experiment, we take a chain of nodes of increasing path length and compare the throughput and the network load incurred in the presence of per-hop transport versus direct end-to-end transport. The inter-node distance in the chain is set to 200 m. We vary the path length from a single hop to up to nine hops. The choice of the maximum path length of 9 is based on observations in [6] on the path lengths in large scale WMN deployments.

Fig. 4 demonstrates that the throughput obtained by per-hop transport is similar to that on a direct end-to-end transport connection across all values of the path length. Similarly, the network load on both kinds of transport connections are identical. Thus, using per-hop architecture does not adversely impact the throughput or the incurred network load of a transport connection and hence hop-by-hop caching is a viable design choice for the A3 Mesh-Cache architecture. Further, in the presence of fading, which results in errors or loss of packets, the per-hop architecture has slightly better throughput than the direct transport connection.

The improvement from per-hop transport is explained as follows: Consider a chain of nodes A–B-C–D where A is the source and D is the destination. Suppose a packet is lost on the link C–D. In case of direct TCP connection, A will be notified when the packet is dropped on the link C–D and A will retransmit the packet. On the other hand when per-hop transport is employed, the packet loss on link C–D will be recovered by the TCP connection C–D. Thus, compared to the direct case, in per-hop transport, the detection of packet loss and the corresponding recovery is localized (between C and D) and does not affect the remaining TCP sub-connections. Also, the TCP window grows back faster after a loss on C–D than on the direct connection since the ACKs travel a single hop. Further, per-hop transport does not increase the number of packets transmitted. For example, in the direct connection, an ACK packet travels three hops from D to A while in per-hop transport three separate ACK packets are transmitted, one on each one hop sub-connection.

2. *Performance with varying loss rates:* We now study the behavior of the two transport connections by varying the loss experienced by the links at each hop. Such loss of packets could be due to fading, congestion, interference, etc. We impose a loss rate at the radio layer. The loss rate is varied between 5% and 30% based on the observations in [6] that such loss rates are highly probable even on routes selected by link-quality routing protocols. Note that not all loss events at the radio layer translate to loss events at the transport layer since the MAC protocol already makes an attempt to recover from losses by retransmitting packets. Each link has a loss rate randomly chosen between 0 to the value on the *X*-axis in Fig. 5 which depicts the performance of the per-hop and direct transport connections as the loss rate is varied. The per-hop transport connection outperforms the direct transport connection across all the values of the loss rate for both the 6-hop and the 4-hop chain. Also, the network load in the per-hop connection is slightly lower than that of the direct connection. This can be attributed to the quick localized detection and recovery from loss events in the per-hop transport connection.

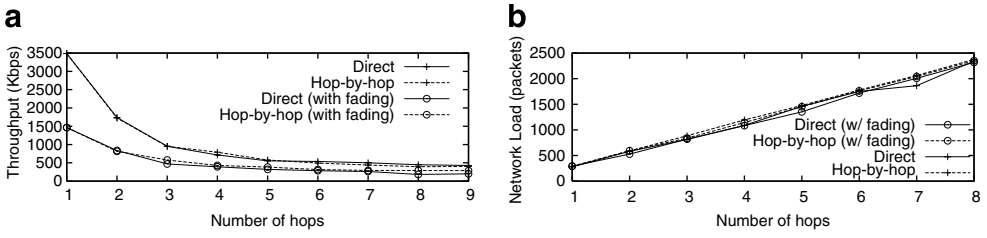


Fig. 4. Performance of per-hop transport with varying path length. (a) Throughput and (b) network load.

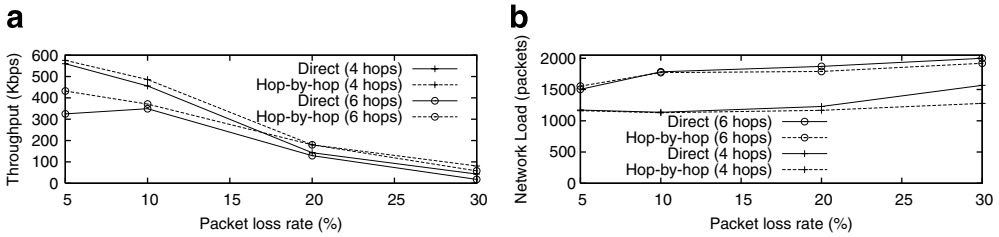


Fig. 5. Performance of per-hop transport as packet loss rate is varied. (a) Throughput and (b) network load.

3. *Performance with varying download size:* Finally, we study the impact of the size of the file download on the transport connections by fixing the path length to be four hops and the loss rate at 10%. Fig. 6 illustrates that as the download size increases, the per-hop connections outperform the direct connections by an increasing margin. This is because, as the download size increases, the duration of the connection increases, thereby increasing the number of chances presented to perform quick detection and recovery from loss. Similarly, the network load in the per-hop connection is lower than that of the direct connection.

In summary, we conclude that the per-hop transport required to support hop-by-hop caching in architecture A3 does not adversely affect performance in comparison to end-to-end transport. In fact, the throughput of per-hop transport is better than end-to-end transport under fading channels with lossy links. Since these conditions are typically true in wireless networks, we expect per-hop transport to generally perform better than or comparable to end-to-end transport, thereby *improving caching without a performance penalty*. Note that the above simulations capture the delay of traversing up and down the TCP/IP stack when simulating per-hop transport. Hence, the performance benefit of per-hop transport holds despite this overhead. The performance benefit of per-hop transport is also confirmed by the experimental measurements in our testbed (Section 8).

6.2. Comparison study of MeshCache architectures

This section presents the performance comparison of the three MeshCache architectures, A1, A2 and A3, using different cache selection protocols.

6.2.1. THCP performance

We compare the performance of A2 and A3 using the THCP protocol. THCP represents architecture A2 while PH-THCP represents A3. We also include the results from A1 which is referred to as *Default*. For this experiment, we consider three different placements (Fig. 7) for the gateway node where each scenario presents a different topological balance. The comparison between Default, THCP and PH-THCP is presented in Table 2.

In Scenario 1, using THCP and PH-THCP, the gateway serves 13.4% and 31.6% lesser bytes respectively compared to Default. The reduction in

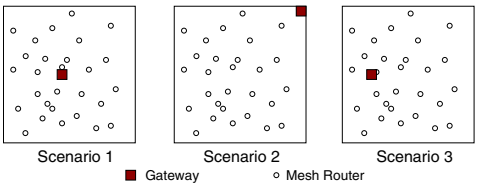


Fig. 7. Three different scenarios used for comparison of cache selection schemes.

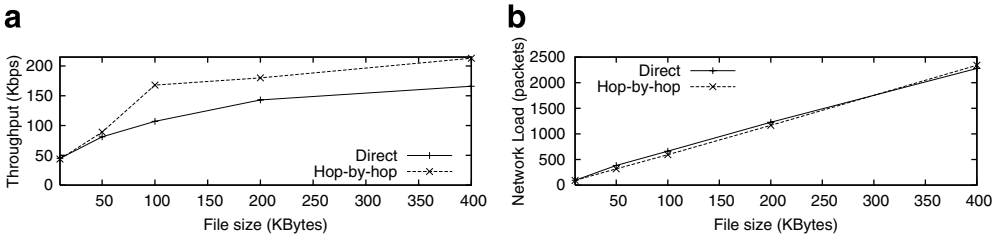


Fig. 6. Performance of per-hop transport with varying transfer sizes. (a) Throughput and (b) network load.

Table 2
Performance of the architectures under THCP scheme

Metric	AAT (Kbps)			Gateway load (KB)			Network load (Packets/file)		
Scheme	Default	THCP	PH-THCP	Default	THCP	PH-THCP	Default	THCP	PH-THCP
S1	1382.8	1479.2	1715.1	60,727	52,579	41,530	144.4	124.1	114.6
S2	467.5	561.9	853.3	60,224	52,140	17,680	279.9	242.6	171.8
S3	1312.2	1509.6	1612.7	58,128	50,215	41,011	145.8	126.4	117.4

gateway load is accompanied by an increase in AAT by 7% and 24% for THCP and PH-THCP, respectively when compared to Default. Also, PH-THCP has a gateway load reduction of 21% and increase in AAT by 16% with respect to THCP. THCP improves the performance when compared to Default by exploiting the local hits (hit rate of 14.4%) at each access MR. These hits are served directly by the access MR to the client and the client experiences higher throughput. In fact, the reduction in the gateway load is very close to the hit rate. PH-THCP, in addition to exploiting local hits at access MR as in THCP, has a *ph-local* hit rate of 4% and a *remote* hit rate of 13.6% due to hop-by-hop caching. Thus, all the hits (a total hit rate of 32%) in PH-THCP avoid the gateway resulting in a corresponding reduction in gateway load when compared to Default.

Similarly, the gateway load reduces by 13.4% and 70.6% in Scenario 2 and by 13.6% and 29.4% in Scenario 3 for THCP and PH-THCP respectively. Note that these reductions correspond to the total hit rate observed in these experiments. Also, since THCP exploits only local hits, its hit rate and hence reduction in gateway load remain similar across all scenarios. However, the placement of gateway affects the extent of ph-local and remote hits in PH-THCP. For instance, the reduction for PH-THCP in Scenario 2 is larger compared to that in Scenarios 1 and 3. This is due to the imbalance in the topology of Scenario 2. The imbalance causes the gateway to be accessible from only a few nodes and thus all the content obtained from the gateway is concentrated in these nodes. When other nodes request the same content, the content is more readily located in these nodes and hence results in an improved hit rate for PH-THCP.

The imbalance in Scenario 2 also causes the routes from some nodes to the gateway to be much longer than other nodes. Thus, when such nodes experience a reduction in path length due to PH-THCP as compared to THCP, there is a significant gain in AAT (82% compared to Default and 51% compared to THCP). In contrast, in Scenarios 1 and 3, the path lengths between nodes and the gateway are typically smaller. Thus most nodes already achieve good throughput that cannot be drastically improved from per-hop transport. A reduction in network load is also observed in both THCP and PH-THCP across all scenarios. A nice feature of PH-THCP is that data replication increases proportionally with the distance to the gateway, which can increase the hit rate. Thus PH-THCP is more resilient to bad gateway placement.

6.2.2. BCP performance

In this section, we investigate the performance of the MeshCache architectures using the BCP scheme.

Here, BCP represents architecture A2 while PH-BCP represents A3. We also include the results from A1 which is referred to as *Default*. The results are presented in Table 3. In Scenario 1, the gateway serves 36% and 46% less bytes using BCP and PH-BCP, respectively, compared to Default. The AAT is increased by 20% for BCP and 23% for PH-BCP when compared to Default. Also, PH-BCP achieves 16% lower gateway load and 3% higher AAT than BCP.

BCP achieves better performance than default by exploiting the local hits (hit rate of 14.4%) at each

Table 3
Performance of the architectures under BCP scheme

Metric	AAT (Kbps)			Gateway load (KB)			Network load (Packets/file)		
Scheme	Default	BCP	PH-BCP	Default	BCP	PH-BCP	Default	BCP	PH-BCP
S1	1382.8	1656.2	1705.5	60,727	38,761	32,597	144.5	110.0	107.3
S2	467.5	713.7	940.5	60,224	39,684	17,680	279.9	198.1	159.5
S3	1312.2	1546.9	1623.8	58,128	39,613	33,719	145.8	114.1	110.8

access MR as well as the remote hits from nearby MRs (hit rate of 23%). These hits are served by MRs with higher throughput than the gateway, resulting in an increase in AAT by 20%. In fact, the reduction in the gateway load in BCP is similar to its total hit rate (38%). PH-BCP further improves the availability of content and hence the hit rate (apart from the 14.4% local hit rate, *ph-local* hit rate of 4% and *remote* hit rate of 28%) due to hop-by-hop caching. Together, all the hits (total hit rate of 47%) in PH-BCP avoid the gateway resulting in a corresponding reduction in gateway load when compared to Default.

Further, the gateway load reduces by 34% and 70% in Scenario 2 and by 32% and 42% in Scenario 3 for BCP and PH-BCP, respectively. Note that BCP's hit rate and hence reduction in gateway load remain similar across all scenarios since the availability of content is independent of the position of the gateway in A2. However, the placement of gateway affects the extent of ph-local and remote hits in PH-BCP similarly as in PH-THCP. This enhanced locality in Scenario 2 also results in 101% higher AAT in PH-BCP compared to Default. Unlike in THCP, when a cache miss occurs at the access MR, BCP looks for other MRs that have requested the same content item, resulting in remote hit rate. This also explains why the gain of PH-BCP over BCP is smaller than that of PH-TCP over THCP.

Similarly, we also found that PH-GHCP benefits from per-hop transport connections and outperforms GHCP which in turn outperforms the Default scenario. Therefore, we conclude that architecture A3 outperforms architecture A2 irrespective of the cache selection algorithm used. Further, both architectures A3 and A2 outperform A1. Next, we evaluate which cache selection algorithm should be used for the best architecture A3.

6.3. Comparison of cache selection protocols

In this section, we compare all the cache selection protocols for architecture A3. The simulation set up is similar to the previous sections. Table 4 depicts the performance of the three per-hop cache selection protocols.

PH-BCP imposes lower load at the gateway than PH-THCP and PH-GHCP. For example, in Scenarios 1 and 3, it incurs 21.5% and 17.8% lower load in bytes than PH-THCP, respectively, and 17% and 10% lower load in bytes than PH-GHCP, respectively. This is because while PH-BCP chooses nodes anywhere in the network that improve its performance, PH-THCP restricts itself to nodes only en route to the gateway and thus suffers from load imbalance. PH-GHCP's higher gateway load can be attributed to the following: Although we use a localized hash function, a detour always occurs when using PH-GHCP. Unlike in PH-BCP where the detour is only taken when a hit is assured, PH-GHCP has many detours that result in misses. This is due to the heavy tail of the request popularity distribution. Unpopular content is typically universally unpopular [38]. Thus a significant number of requests take a detour followed by a miss, resulting in contacting the gateway and hence higher gateway load. Interestingly, in the scenario with severe imbalance (Scenario 2), all three schemes exhibit similar load at the gateway. In this case, since the gateway is placed in a corner, all the files obtained from the gateway will also be cached around the gateway due to hop-by-hop caching. Hence, any request that is already served by the gateway can be obtained enroute to the gateway before reaching the gateway itself. Hence, PH-THCP and PH-GHCP have similar gateway load as in PH-BCP.

Further, the results show that for all scenarios considered, PH-BCP provides the best throughput performance. PH-BCP also has the lowest network load across all three scenarios. Even in Scenario 2, although the gateway load is similar, the AAT of PH-THCP and PH-GHCP is lower than that of PH-BCP. This is because PH-BCP spreads more load away from the gateway when compared to the other two schemes. In summary, PH-GHCP achieves lower throughput than PH-THCP and PH-BCP from taking detours. While PH-THCP incurs no search delay or search overhead,

Table 4
Performance of per-hop cache selection protocols

Metric	AAT (Kbps)			Gateway load (KB)			Packet transmissions per file		
Scheme	PH-THCP	PH-BCP	PH-GHCP	PH-THCP	PH-BCP	PH-GHCP	PH-THCP	PH-BCP	PH-GHCP
S1	1715.1	1705.5	1545.6	41,530	32,597	39,275	114.6	107.3	127.4
S2	853.3	940.5	567.4	17,680	17,680	17,812	171.8	159.5	193.6
S3	1612.7	1623.8	1478.2	41,011	33,719	37,462	117.4	110.8	131.3

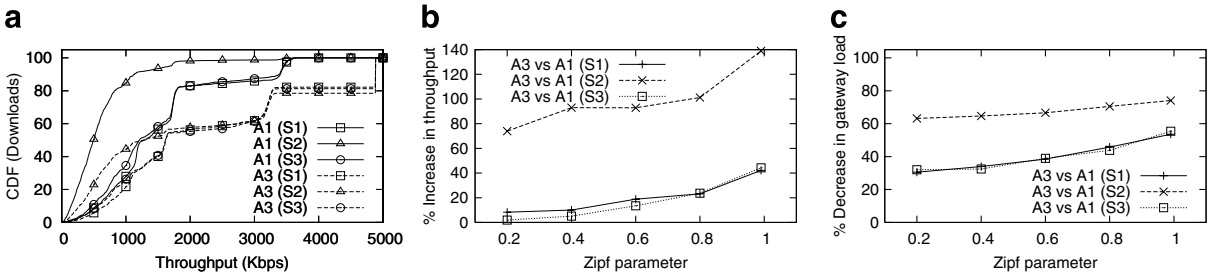


Fig. 8. Performance comparison of A3 and A1. PH-BCP is used for cache selection.

PH-BCP can alleviate congestion better by sending requests anywhere in the network and achieves better throughput and lower network and gateway load. We conclude PH-BCP is the best cache selection protocol in A3.

6.4. Summary

In summary, our extensive simulations have demonstrated that per-hop transport is a viable mechanism to enable hop-by-hop caching in A3. Architecture A3 using PH-BCP outperforms A2 by increasing the client AAT by up to 100% and reducing gateway load by up to 70%. In fact, Fig. 8a demonstrates the CDF of per file throughput in A3 and A1 for the three scenarios. The CDF confirms the gain in throughput observed in A3 when compared to A1. For instance, in Scenario 2, while in A1 approximately 20% of the file downloads achieve throughput higher than 1 Mbps, in A3 50% of file downloads achieve throughput greater than 1 Mbps.

Note, however, that these results are dependent on the locality in the access pattern of the clients. As seen from the simulation parameters, the Zipf parameter used in the above study is 0.8 and is based on previous studies of caching in wireless networks [40] and measurements from web traces [7]. To study the effect of varying locality on the MeshCache performance, we vary the Zipf parameter from 0.2 to 1.[5] As seen in Fig. 8b and c, as the Zipf parameter is increased, the percentage increase in AAT in A3 vs. A1 goes up and so does the percentage reduction in gateway load. Similar to the observations above, the gain is highest in Scenario 2 due to its significant topological imbalance. Specifically, the work in [7] studied web request traces from a fixed group of users and found that the Zipf parameter varied between 0.64 and 0.83. For this range of Zipf parameter values, MeshCache exhibits significant improvement in throughput and reduction in gateway load.

To validate the performance and usability of MeshCache, we implemented and deployed MeshCache on our wireless mesh network testbed [28] and evaluated it extensively.

7. MeshCache implementation

In this section, we describe the implementation of the MeshCache system. The MeshCache system is implemented by a user-level daemon – `MeshCache D`. The functions of MeshCacheD are: (1) Transparently hijack web requests initiated at the clients and serve these requests from either its own cached content in case of a hit. (2) Locate the appropriate parent cache to fetch the data from in case of a cache miss. (3) When data is fetched, enable hop-by-hop transport of the data to facilitate caching at every hop. (4) Cache the content fetched for a client and maintain its freshness. We leverage Squid [46], an open-source proxy cache software developed for Internet web caching, to implement the MeshCacheD. The MeshCacheD consists of three modules: (1) The *MSquid* module (MSM) is responsible for transparently hijacking client requests and serving them from its cache or the appropriate parent cache, caching and validating fetched data, and performing hop-by-hop transport. Squid software is modified to obtain the MSM. We disabled Squid's cache selection protocols and instead interfaced it with an implementation of our cache selection protocols. Further, we exploited Squid's functionality to deal with a hierarchy of caches to perform hop-by-hop caching. (2) The *Cache Selection* module (CSM) implements the cache selection protocols as described in Section 4 to locate a suitable parent cache given a URI. The CSM is also responsible

[5] The larger the value of Zipf parameter, the higher the locality.

for enabling hop-by-hop caching via the MSM. To enable both the above functions, the CSM exports **FindParentCache (givenURI)** interface to the MSM and also interacts with the underlying routing protocol via the *Cross-Layer* module (CLM). (3) The CLM is responsible for communicating with the underlying routing protocol to obtain information and export this information via known APIs to the other modules in MeshCacheD. Our current prototype implements these APIs as functions calls that operate on routing table information continuously updated to the /proc virtual file system by the routing protocol in the OS kernel. The virtual file system provides a shared memory interface between MeshCacheD CLM and the underlying routing protocol in the kernel. The information shared contains the current best next hops for known destinations, path metric to a destination node (e.g. accumulated ETX of all links) and the best known gateway with path metric. The CLM exports the following APIs on behalf of the routing protocol to other MeshCache modules: *BestGateway()*, *BestGatewayMetric()*, *BestNode*(⟨*node IP list*⟩) and *GetNextHopForNode*(⟨*node IP*⟩).

In the following sections, we describe how the MeshCacheD implements architecture A2 using THCP and A3 using PH-THCP and PH-BCP. A2 is implemented to obtain a comparison point.

7.1. THCP implementation

Each mesh router runs the MeshCacheD. Referring to Fig. 9, THCP works as follows: (1) An unmodified client applications in the WMN generates a web request for a URI *X*. (2) The request for *X* is routed through the access mesh router (AMR) for this client. (3) At AMR, the client request is transparently hijacked by the MSM using the standard Linux Netfilter [44], and queuing it up for MeshCache processing using *libipq*. (4) The MSM now captures all further communication from this client. This enables the MSM to parse the client request to locate the data item requested. (5) The MSM now checks for *X* in its cache. If there is a cache hit, *X* is served to the client from the AMR cache. (6) In case of a cache miss, the MSM issues a **FindParentCache(X)** call to the CSM. (7) Since THCP does not exploit cached objects at other mesh routers, the CSM simply returns the best gateway obtained from the CLM. (8) Upon returning from the FindParentCache() call, the MSM now contacts the gateway to fetch *X*. (9) The gateway either serves the file from its cache or retrieves it from the origin server. The data is now returned to the AMR that serves the data back to the requesting client. (10) AMR also caches *X* in its cache and maintains its freshness information for future requests.

7.2. PH-THCP Implementation

Referring to Fig. 9, the PH-THCP scheme works as follows: (1) Steps 1–6 for the PH-THCP scheme are identical to that in the THCP scheme described above. (7) To implement PH-THCP, the CSM extracts the origin server's IP address (*SIP*) from the URI. It then issues a **GetNextHopForNode(SIP)** call to the CLM. (8) The CLM communicates with the routing protocol to obtain the best next hop for SIP based on the metric currently employed by the routing protocol. If the SIP is an IP address outside of the current WMN, all the messages to the SIP need to be routed via the gateway (GMR). Thus, essentially, the next hop returned to the CSM in this case will be the best next hop for GMR. (9) When the CSM receives the best next hop for SIP, say MR 1, it returns it as the best parent cache to the MSM. (10) The MSM now contacts the MSM at MR 1 requesting *X*. Note that the choice of MR 1 is **not** made after ensuring that MR 1 has *X*. (11) Thus, when the MSM at MR 1 receives the request for *X*, it may have a cache hit or a miss. In the event of hit, *X* is returned to AMR, which in turn serves

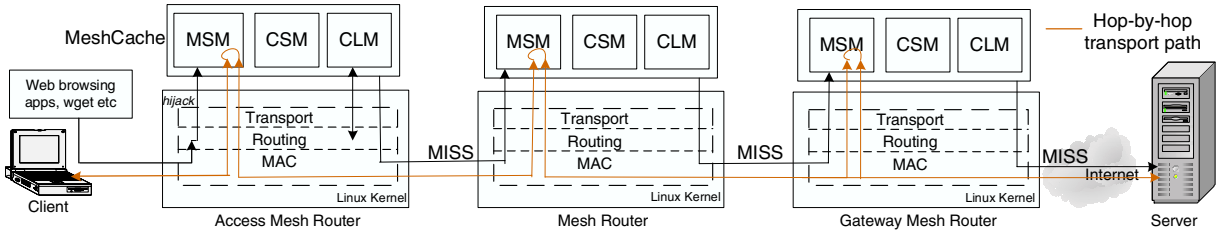


Fig. 9. MeshCache implementation.

the data to the client and also caches *X* for future use. (12) In case of a miss, MR 1 repeats the process from steps 6 to 11 described above. Let us assume that the next hop for *SIP* from MR 1 is MR 2. If MR 2 has a cache hit, the data will now be served from MR 2 to MR 1, which caches the data and further serves it to AMR, which once again caches the data and serves it to the client requesting *X*. Thus, the CSM exploits Squid's behavior when contacting parent caches to perform hop-by-hop caching. (13) In case of a miss at MR 2, the process repeats itself until the request is received by the gateway itself. (14) The gateway then either serves the file from its cache or retrieves it from the origin server and returns it to the client, and each router along the way caches the file.

7.3. PH-BCP implementation

Referring to Fig. 9, the PH-BCP scheme works as follows: (1) Steps 1–6 for the PH-BCP scheme are identical to that in the THCP scheme described above. (7) The CSM maintains a one-to-one mapping between a URI and the corresponding IP address of the mesh router that has a cached copy of the content. This mapping is maintained with a refresh timeout. (8) When the CSM receives the **FindParentCache(X)** call, it consults its mapping table to determine if a corresponding IP address is found. (9) When the CSM finds no such mapping, it begins the PH-BCP cache selection protocol. For example, when AMR receives the request for *X* for the first time and experiences a cache miss, its corresponding CSM will also fail to locate a mapping for *X*. (10) To implement PH-BCP, the CSM extracts the data item *X* from the URI. It then sends a UDP *content locate* message to the broadcast IP address along with a gateway path metric calling the **BestGatewayMetric()** CLM function. (11) Any mesh router receiving the *content locate* message checks its cache to see if it contains *X*. In case of a hit, the mesh router unicasts a *content found* message back to AMR. All mesh routers also rebroadcast the *content locate* message to their neighbors until the metric in the message exceeds the path metric to the gateway. (12) AMR collects all the *content found* messages received in a time period of *timeout* seconds. It then picks the mesh router that maximizes the throughput out of all routers with a cache hit. Let us denote this node as *FIP*. Note that if AMR does not obtain any *content found* messages, it picks GMR as the node containing *X*. (13) It then calls **GetNextHopForNode(FIP)** and obtains the next hop for *FIP*, say MR 1. It returns MR 1 to the MSM. (14) The CSM also makes a note of mapping between *X* and *FIP* in its mapping table. It also send a *setup* message to MR 1 with the mapping information. (15) When MR 1 receives the setup message, it finds the best next hop for *FIP* and forwards the message to that mesh router. It also makes a note of this mapping in its mapping table. This process is repeated till it reaches a node such that the next hop for a message reaching *FIP* is *FIP* itself. (16) Now, the MSM of AMR contacts the MSM of MR 1 for *X*. MR 1 gets a cache miss and hence contacts its CSM. The CSM will now find a mapping for *X* created by the setup message sent by AMR. It then gets the best next hop for *FIP* and returns it to the MSM. (17) This process continues till *FIP* is contacted and a cache hit is obtained. *FIP* then transfers *X* which is pipelined till the client. Effectively, all the intermediary mesh routers cache the content thereby enabling hop-by-hop caching. (18) When *X* is not found in the WMN, the request reaches GMR and GMR contacts the origin server.

7.4. Cache consistency

Since MeshCache uses the Squid code base, we rely on the inbuilt mechanisms of Squid to ensure object consistency. Objects downloaded in HTTP use HTTP headers (e.g., in HTTP/1.1) to specify an expiry time or other types of freshness information which is used by MeshCache on each router to determine when objects are stale. Since HTTP is likely to be the most widely used object retrieval protocol used over MeshCache, we focus on it in this paper. If some proprietary protocol (e.g., p2p sharing) is used fetch objects then that protocol will need to specify expiry information for data items to work with MeshCache, or some default timeouts will be required.

8. MeshCache performance

In this section, we evaluate the performance of MeshCache over a deployed wireless mesh network testbed.

8.1. Testbed setup

Our testbed, MAP (Mesh@Purdue [28]), consists of 15 wireless mesh routers (small form factor

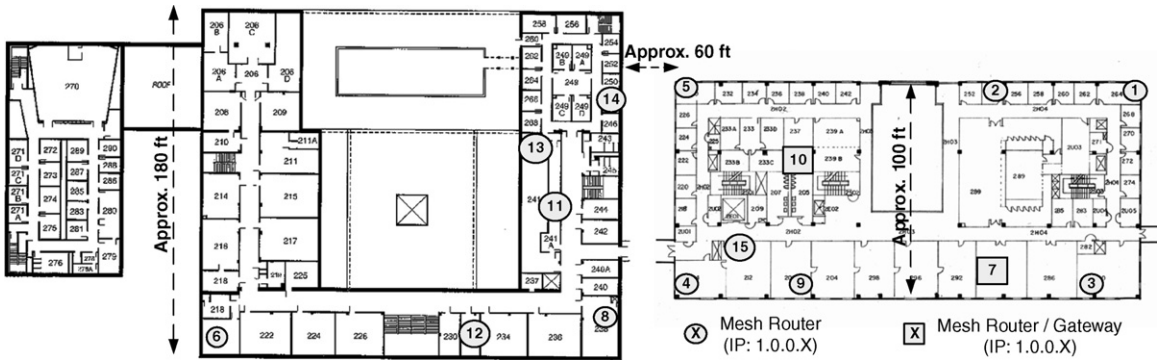


Fig. 10. Deployment of the MAP testbed used for MeshCache evaluation.

desktops) spread out over the second floor of two adjoining academic buildings (MSEE and EE). For the experiments in this paper, each mesh router is equipped with a single wireless card. Nine nodes in the MSEE building are equipped with Atheros 5212 based 802.11a/b/g cards while six nodes in the adjoining EE building are equipped Senao Engenius 2511 802.11b wireless card. We configured the entire network to operate in 802.11b mode. Each radio is attached to a 2dBi rubber duck omnidirectional antenna with a low loss pigtail to provide flexibility in antenna placement. Each mesh router runs Linux kernel 2.4.20-8 and the open-source *hostap* and *madwifi* drivers are used to enable the wireless cards. IP addresses are statically assigned. The wireless cards we use can support a wide range of power settings (up to 200 mW). We used them in their default operational mode.

Fig. 10 shows the layout of nodes in our testbed. The testbed deployment environment is not wireless friendly, having floor-to-ceiling office walls instead of cubicles as well as some laboratories with structures that limit the propagation of wireless signals. Apart from structural impediments, interference exists in our deployment from other 802.11b networks (the Purdue Airlink network). We used channel 11 of 802.11b to operate our network since it was the band furthest away from those being already used in the deployment environment. In summary, the environment is highly variable and many different paths exist between nodes.

The routing protocol used in the testbed is MAP-DV, which we implemented through modifications to the kernel AODV routing protocol implementation [21] to incorporate proactive gateway discovery [11], the ETX metric [9] and ETT metric [13] and the interface to the CLM. We used *netperf* measurements to confirm that MAP-DV provides better throughput than vanilla AODV.

We compare the performance of the three MeshCache architectures using our testbed. Architecture A1 is again referred to as *Default*. Architecture A2 is represented by THCP while both PH-THCP and PH-BCP are included for architecture A3. We choose a deployment where MR 7 is the gateway node with access to the Internet while 10 out of the remaining 14 nodes are chosen as traffic sources. Each source is driven by a synthetic web trace obtained similar to in the simulation. Also, the cache size in each mesh router is set to 2 MB due to the working set size and length of the experiment. Each experiment consists of clients making requests from the trace through an unmodified *wget* client for a period of 1 h. The experiments were repeated once every day for one week and the results averaged. Throughput per file transfer is used as the metric for this study.

8.2. Performance results

We first performed measurements to quantify the interactions between per-hop transport and the underlying routing protocol. Specifically, we want to measure whether fixing of routes in per-hop transport leads to poor adaptation to network conditions by not allowing route changes for the duration of a transfer. We measured if route changes occurred in an interval of 30 s[6] between nodes 5, 1 and 7 using a ping with the record route option every second over a period of 12 h. We found that for node 5, routes remained the same for 92% of

[6] A sample download time for content.

the intervals while 99% of the intervals resulted in same routes in the case of node 1. This shows that it is unlikely that fixing routes in per-hop transport for an interval of 30 s will result in poor link adaptation. The time scales of route changes are longer than the time a route is fixed in MeshCache (i.e. the time to download a typical content item).

We now describe our throughput performance evaluation experiment. In this experiment, the nodes generating traffic can be separated into nodes that are one hop away and those that are more than one hop away from the gateway for ease of explanation. Thus, MRs 2, 3, 4, 9, 10 and 15 are one hop away from MR 7 while MRs 1, 5, 8 and 12 are more than a hop away from the gateway.

The performance of the MRs that are one hop away is depicted in Fig. 11. For each MR, the throughputs obtained per file transfer are sorted and plotted. Due to lack of space, the performance of four such nodes is depicted. When THCP scheme is employed, each MR benefits from the locality in its own access pattern, i.e., when the MR receives a request for a content item that it has fetched in the past and hence cached, the MR can serve the request from the cache and thus obtain a significant improvement in its throughput. For example, for node 15, compared to the default scheme, 20 transfers benefit from caching at the access MR itself. Note that the throughput for a cache hit is very high (about 50 Mbps) and hence is not depicted in the graph. Similar benefit is observed in all the nodes when THCP is employed. For the one-hop nodes, performance using PH-THCP is largely similar to using THCP since when they do not have a local hit, they have to traverse only one hop irrespective of the cache-selection algorithm. However, the one-hop nodes can benefit from PH-THCP when any MR whose route to the gateway passes through them requests for an object that they will request in the future. In this case, when the node whose route passes through them fetches the object from the gateway, the object will be cached at this node. Node 15 shows the presence of gain from PH-THCP as it exploits the locality in the traffic pattern of itself and its neighbors. An additional 10 transfers benefit from PH-THCP as compared to THCP in node 15. Similar benefit is also observed in the other nodes. Thus, with PH-THCP, some transfers that take one hop to be served can now be served from the cache on the MR itself. The performance of PH-BCP is expected to be similar to that of PH-THCP for the one-hop nodes since PH-BCP cannot further reduce the number of hops required to serve the content to less than one. This effect is observed in nodes 3, 4 and 10. Interestingly, node 15 shows significant benefit when using PH-BCP across all transfers. This is because compared to nodes 10 or 3, the connection of node 15 to gateway 7 passes through several walls, resulting in increased attenuation of the signal and hence reduced throughput. PH-BCP enables MR 15 to choose other potential candidates for a content item (say 4, 9 or 10) which provide much higher throughput than the one hop to the gateway. Note that MR 4 does not experience any further improvement with PH-BCP as it is unable to locate a better candidate to obtain the file from.

The performance of the MRs that are more than one hop away is depicted in Fig. 12. Similar to in the 1-hop case, the performance of the MRs using THCP is better than the default case. Once again, the nodes exploit the locality in their own access pattern. For example, THCP improves the performance of 20 transfers in MR 5. Further, PH-THCP has the potential of reducing the number of hops required to obtain a content item up to 1, thereby improving the throughput. Note that similarly as in the one-hop case, these nodes will benefit when the nodes whose routes pass through them request for files that will be requested by these nodes in the future. This effect is observed in all the nodes that are two hops away. For example, MR 5

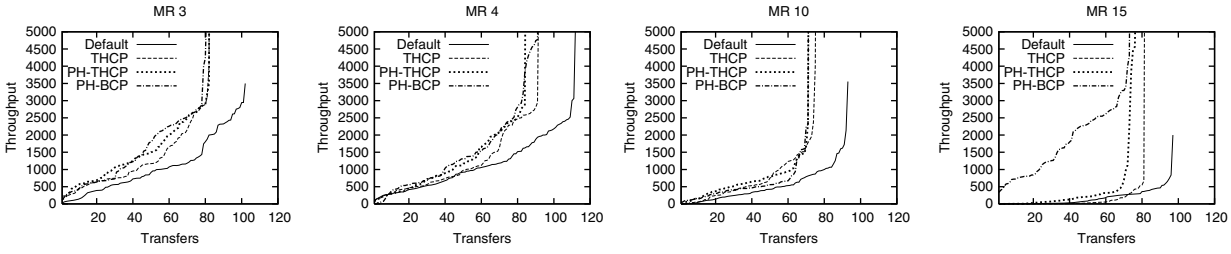


Fig. 11. Download performance of individual mesh routers one hop away. MR 7 is the gateway.

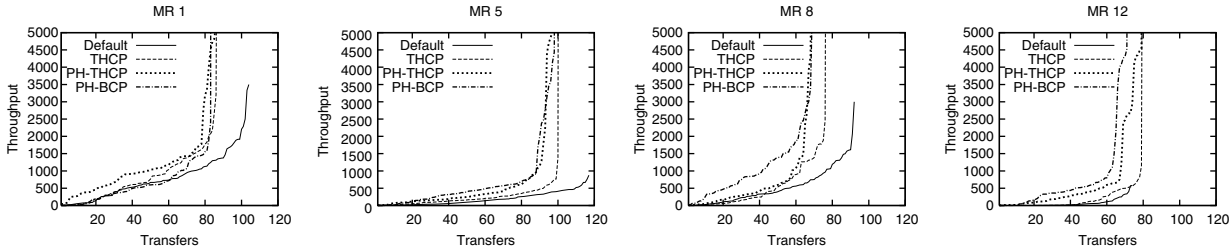


Fig. 12. Individual download performance of mesh routers more than one hop away. MR 7 is the gateway.

improves the performance of 10 transfers using PH-THCP. Finally, using PH-BCP further improves the performance of these nodes. For example, nodes 12 and 8 benefit from using PH-BCP by obtaining content from MRs like 4, 15 or 9 instead of the gateway 7. Also, PH-BCP enables searching all the nodes in the vicinity of a mesh router instead of only those that are en route to the gateway as in PH-THCP. Note that MRs 1 and 5 do not obtain further benefit from PH-BCP as they have limited number of neighbors (nodes 2 or 3 for MR 1, node 10 for MR 5). One among the potential neighbors is already considered by PH-THCP and hence PH-BCP does not have different potential candidates to obtain additional benefit.

The performance of the network using the MeshCache system is summarized in Fig. 13a using a CDF of the total transfers in the network. In summary, in the absence of caching in the network, only 20% of the transfers have a throughput greater than 1 Mbps. When a simple scheme like THCP is used, 40% of the transfers have a throughput greater than 1 Mbps. PH-THCP results in 50% of the transfers having a throughput greater than 1 Mbps. Finally, using PH-BCP results in almost 60% of the total transfers across all the nodes having a throughput larger than 1 Mbps. Fig. 13b also shows that the average load per node as well as the gateway load is reduced by using MeshCache. PH-BCP reduces gateway load the most out of all the schemes.

Finally, we observed that the latency of transfers did not suffer despite the search overhead of PH-BCP. The average and median latency were in fact lower than without MeshCache (by around 5–10%). This is because without cooperative caching, packets suffer queuing delays as they get closer to the gateway which more than even out the extra delay incurred from cooperative caching, except in networks with very low load.

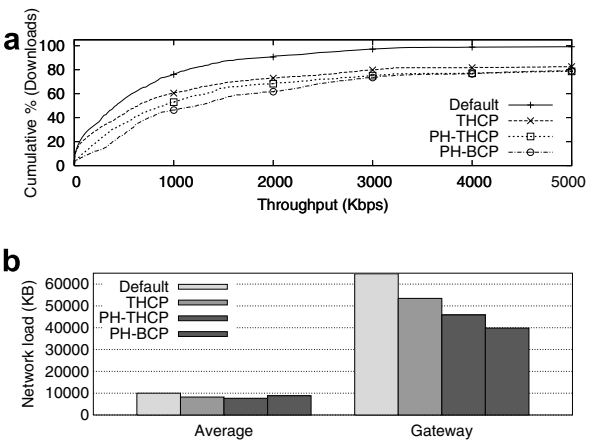


Fig. 13. Overall download performance of mesh routers. (a) Overall throughput and (b) overall network load.

9. Related work

9.1. Wireless mesh networks

Many design issues of WMNs have been recently studied [20,12,31,30,4,24,6] and many companies are offering products for deploying WMNs [27,5]. However, research in WMNs has primarily focused on new routing protocols, improving medium access, as well as new designs for physical layer technology. To the best of our knowledge, no previous work specifically explores incorporating content caching to improve the performance of WMNs.

9.2. Caching in wireless networks

Content caching has been previously proposed for improving performance in *mobile* ad hoc networks [33,40,16,17]. However, MeshCache is different from these previous approaches since it is designed for static wireless mesh networks and aims to provide a transparent infrastructure-based solution to exploiting locality. In addition, MeshCache is a complete system design that incorporates

cross-layer communication to enable transparent caching. Previous work has primarily focused on simulations, neglecting the practical considerations in deploying a caching infrastructure. For example, previous work assumes that files can be easily stored as they pass through the routing layer whereas this is not possible without a per-hop transport architecture or major modifications to client applications and network layers of all mesh routers.

9.3. Transport protocol enhancements

There have been numerous enhancements to TCP and even totally new transport protocols proposed for mobile wireless ad hoc networks. In this paper, we evaluated MeshCache using the widely used TCP protocol. However, MeshCache is an application layer system that will work with any enhanced transport protocol as well. The per-hop transport technique used in MeshCache should benefit any new transport protocol by providing them with quick congestion detection and response as well as quick error recovery. The technique of splitting an end-to-end TCP connection into multiple ones (Split TCP) has been previously proposed to improve throughput and fairness in mobile wireless ad hoc networks [22]. However Split TCP is designed to deal with mobility and only splits the connections at a few intermediate nodes. MeshCache splits each hop of the transport connection to enable caching. Split TCP connections also naturally arise in overlay networks in the Internet [23,29,34,3]. However, the overlay split TCP is fundamentally different from the per-hop transport in MeshCache since in the Internet, the original end-to-end path may be totally different from the overlay split transport path. In this case, the overlay path can be better or worse than the end-to-end path. In MeshCache, the per-hop and end-to-end transport session both operate over the exact same underlying network path. Thus, the per-hop transport in MeshCache cannot worsen the network path used. Packet retransmissions in MeshCache also take place from the closest possible point in the path unlike in overlay networks where the per-hop connections span multiple Internet routers.

10. Conclusions

In this paper, we proposed the MeshCache system for exploiting the locality in client request patterns in a wireless mesh network. The MeshCache system alleviates the congestion bottleneck that commonly exists at the gateway node in WMNs while providing better client throughput by enabling content downloads from closer high-throughput mesh routers. MeshCache is loosely coupled with the underlying transport and routing protocols to maximize deployability and exploit cross-layer information awareness. Through simulation and testbed experiments with a deployed implementation, the hop-by-hop caching mechanism coupled with PH-BCP cache selection in MeshCache was shown to be an effective technique to improve WMN performance.

We are currently working on exploiting the MeshCache content cache for current and future applications in WMNs. For example, a community file sharing service could simply query this content cache in peer routers for high throughput content download. Further innovations such as chunk-based parallel download and new WMN services such as downloadable movies and music can also leverage the content cache for improved load balancing and throughput for clients. Finally, the content cache can be extended for delivering large software patches required by many users by allowing such specialized content to be selectively cached despite their large sizes. We also plan to conduct measurement studies to further understand the locality in live WMN traffic as we extend our testbed to cover student dormitories.

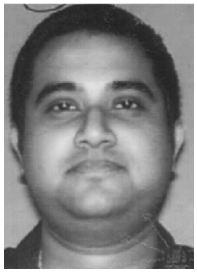
Acknowledgements

We thank the anonymous reviewers for their helpful comments. This work was supported in part by NSF grants CNS-0338856 and CNS-0626703.

References

[1] A. Adya, P. Bahl, J. Padhye, A. Wolman, L. Zhou. A multi-radio unification protocol for IEEE 802.11 wireless networks, in: Proceedings of BroadNets, 2004.
[2] I.F. Akyildiz, X. Wang, W. Wang, Wireless mesh networks: a survey, Comput. Networks 47 (4) (2005) 445–487.
[3] Y. Amir, C. Danilov, Reliable communication in overlay networks, in: Proceedings of DSN, 2003.
[4] P. Bahl, A. Adya, J. Padhye, A. Wolman, Reconsidering wireless systems with multiple radios, ACM MC2R, October 2004.
[5] Bel Air Networks. <http://www.belairnetworks.com>.
[6] J. Bicket, D. Aguayo, S. Biswas, R. Morris, Architecture and evaluation of an unplanned 802.11b mesh network, in: Proceedings of ACM MobiCom, 2005.

[7] L. Breslau, P. Cao, L. Fan, G. Phillips, S. Shenker, Web caching and Zipf-like distributions: evidence and implications, in: Proceedings of IEEE INFOCOM, 1999.
[8] A. Chankhunthod, P. Danzig, C. Neerdaels, M.F. Schwartz, K.J. Worrell, A hierarchical Internet object cache, in: Proceedings of USENIX ATC, 1996.
[9] D.S.J.D. Couto, D. Aguayo, J.C. Bicket, R. Morris, A high-throughput path metric for multi-hop wireless routing, in: Proceedings of ACM MobiCom, 2003.
[10] P.B. Danzig, R.S. Hall, M.F. Schwartz, A case for caching file objects inside internetworks, in: Proceedings of ACM SIGCOMM, 1993.
[11] S.M. Das, H. Pucha, Y.C. Hu, Symmetrical fairness in infrastructure access in multi-hop wireless networks, in: Proceedings of IEEE ICDCS, 2005.
[12] R. Draves, J. Padhye, B. Zill, Routing in multi-radio, multi-hop wireless mesh networks, in: Proceedings of ACM MobiCom, 2004.
[13] R. Draves, J. Padhye, B. Zill, Routing in multi-radio, multi-hop wireless mesh networks, in: Proceedings of ACM MobiCom, September 2004.
[14] J. Eriksson, S. Agarwal, V. Bahl, J. Padhye, A feasibility study of mesh networks for an all-wireless office, in: Proceedings of ACM MobiSys. Also MSR Technical Report MSR-TR-2005-170, June 2006.
[15] L. Fan, P. Cao, J. Almeida, A. Broder, Summary cache: a scalable wide-area Web cache sharing protocol, in: Proceedings of ACM SIGCOMM, 1998.
[16] V.I. Francoise Sailhan, Energy-aware web caching for mobile terminals, in: Proceedings of ICDCS Workshops, 2002.
[17] T. Hara, Effective replica allocation in ad hoc networks for improving data accessibility, in: Proceedings of INFOCOM 2001, 2001.
[18] S. Iyer, A. Rowstron, P. Druschel, Squirrel: a decentralized peer-to-peer web cache, in: Proceedings of ACM PODC, July 2002.
[19] J. Jun, P. Peddabachagari, M. Sichitiu, Theoretical maximum throughput of IEEE 802.11 and its applications, in: Proceedings of NCA, 2003.
[20] R. Karrer, A. Sabharwal, E. Knightly, Enabling large-scale wireless broadband: the case for taps, in: Proceedings of ACM HotNets, 2003.
[21] Kernel AODV. <http://w3.antd.nist.gov/wctg/aodv_kernel/index.html>.
[22] S. Kopparty, S. Krishnamurthy, M. Faloutsos, S. Tripathi, Split-tcp for mobile ad hoc networks, in: Proceedings of GLOBECOM, 2002.
[23] G.-I. Kwon, J.W. Byers, ROMA: reliable overlay multicast with loosely coupled TCP connections, in: Proceedings of IEEE INFOCOM, 2004.
[24] P. Kyasanur, N.H. Vaidya, Routing and interface assignment in multi-channel multi-interface wireless networks, Technical Report UIUC, October 2004.
[25] J. Li, C. Blake, D.S.D. Couto, H.I. Lee, R. Morris, Capacity of ad hoc wireless networks, in: Proceedings of ACM MobiCom, March 2001.
[26] B.A. Mah, An empirical model of http network traffic, in: Proceedings of IEEE INFOCOM, 1997.
[27] Mesh Networks. <http://www.meshnetworks.com>.
[28] Mesh@Purdue. <http://www.engineering.purdue.edu/MESH>.
[29] A. Nakao, L. Wang, L. Peterson, MSB: Media Streaming Booster, Technical Report, Princeton University CS TR-666-02, December 2002.
[30] A. Raniwala, T. Chiueh, Architectures and algorithms for an IEEE 802.11-based multi-channel wireless mesh network, in: Proceedings of IEEE INFOCOM, March 2005.
[31] A. Raniwala, K. Gopalan, T. Chiueh, Centralized channel assignment and routing algorithms for multi-channel wireless mesh networks, ACM MC2R 8 (2) (2004).
[32] A. Rousskov, D. Wessels, Cache digests, in: Proceedings of WCW, 1998.
[33] F. Sailhan, V. Issarny, Cooperative caching in ad hoc networks, in: Proceedings of MDM, 2003.
[34] M. Swany, Improving throughput for grid applications with network logistics, in: Proceedings of SC, 2004.
[35] V. Valloppillil, K.W. Ross, Cache array routing protocol v1.0, Internet draft, 1998.
[36] D. Wessels, K. Claffy, ICP and the squid web cache, IEEE JSAC 16 (3) (1998).
[37] J.E. Wieselthier, G.D. Nguyen, A. Ephremides, Energy-efficient broadcast and multicast trees in wireless networks, Mob. Netw. Appl 7 (6) (2002).
[38] A. Wolman, G. Voelker, N. Sharma, N. Cardwell, A. Karlin, H. Levy, On the scale and performance of cooperative Web proxy caching, in: Proceeding of ACM SOSP, 1999.
[39] L. Yang, W.K. Seah, Q. Yin, Improving fairness among TCP flows crossing wireless ad hoc and wired networks, in: Proceedings of ACM MobiHoc, 2003.
[40] L. Yin, G. Cao, Supporting cooperative caching in ad hoc networks, in: Proceedings of INFOCOM 2004, 2004.
[41] X. Zeng, R. Bagrodia, M. Gerla, Glomosim: a library for parallel simulation of large-scale wireless networks, in: Proceedings of PADS Workshop, May 1998.
[42] Champaign-Urbana community wireless network. <http://www.cuwireless.net>.
[43] MIT Roofnet. <http://www.pdos.lcs.mit.edu/roofnet>.
[44] Netfilter. <http://www.netfilter.org>.
[45] Southampton wireless network. <http://www.sown.org.uk>.
[46] Squid Web Cache. <http://www.squid-cache.org>.
[47] Wireless leiden. <http://www.wirelessleiden.nl>.
[48] Seattle wireless. <http://www.seattlewireless.net>.
[49] Bay area wireless users group. <http://www.bawug.org>.



Saumitra M. Das is currently a Ph.D. candidate in the School of Electrical and Computer Engineering at Purdue University, USA. Previously, he received a MS degree from Carnegie Mellon University, USA and a B.Engg. degree from the University of Bombay, India. His research interests include cross-layer system design for multi-hop wireless networks, scalable routing strategies in wireless ad hoc networks, and mobile robotics.

Himabindu Pucha is currently a Ph.D. candidate in the School of Electrical and Computer Engineering at Purdue University, USA. Previously she received a MSEE degree from Purdue University, USA and a B.Engg. degree from the University of Bombay, India. Her research interests include Internet routing and overlay networks, peer-to-peer systems and applications, and mobile computing.

Y. Charlie Hu is an Assistant Professor of Electrical and Computer Engineering and Computer Science at Purdue University. He received his M.S. and M.Phil. degrees from Yale University in 1992 and his Ph.D. degree in Computer Science from Harvard University in 1997. From 1997 to 2001, he was a research scientist at Rice University. His research interests include operating systems, distributed systems, networking, and parallel computing. He has published over 80 papers in these areas. He received the NSF CAREER Award in 2003. He served as a TPC vice chair for the 2007 IEEE ICDCS and 2004 IEEE International Conference on Parallel Processing, and was a co-founder and TPC co-chair for the International Workshop on Mobile Peer-to-Peer Computing. He is a member of USENIX, ACM, and IEEE.