The Real Ad-hoc Multi-hop Peer-to-peer (RAMP) Middleware: an Easy-to-use Support for Spontaneous Networking

Paolo Bellavista, Antonio Corradi, and Carlo Giannelli
Dip. Elettronica Informatica e Sistemistica (DEIS) – University of Bologna - ITALY
{paolo.bellavista, antonio.corradi, carlo.giannelli}@unibo.it

Abstract

Spontaneous (or opportunistic) networks are multi-hop ad-hoc networks where nodes opportunistically exploit peer-to-peer contacts to share content and available resources in an impromptu way. Even if spontaneous networking has recently received growing interest, there is still the lack of impactful and wide-scale applications fully exploiting its potential. We claim that this is due to the intrinsic complexity of spontaneous network management, unsuitable to be directly handled by application developers. Therefore, this paper proposes a novel easy-to-use middleware, called RAMP, for the autonomic, cross-, and application-layer management of spontaneous networks. RAMP enables the dynamic sharing of all resources available via multiple, heterogeneous, intermittent, infrastructure-based, and ad-hoc links, which are orchestrated in a lightweight way to compose the multi-hop paths needed by sharing applications at runtime. The RAMP prototype is a useful tool for the community of researchers in the field and can be rapidly deployed over real execution environments. The reported experimental results demonstrate the feasibility of our approach and the limited RAMP overhead over common deployment scenarios.

1. Introduction

In the last couple of years spontaneous networking has received growing and growing attention from the community of academic/industrial researchers for its promising aspects of better exploitation of available wireless connectivity, resource connectivity sharing, and immediate connectivity offer in regions with difficult coverage [1-3]. Spontaneous networks stem from the impromptu interaction of mobile and (most usually) fixed devices, which do not statically know each other. Specific and still open challenging aspects of spontaneous networks are:

- concurrent exploitation of heterogeneous wireless technologies. This potentially allows to build multi-hop paths made up by multiple heterogeneous links (rarely exploited in current solutions for spontaneous networking);
- concurrent exploitation of multiple wireless technologies and interfaces, also over the same node. This potentially allows to build different multi-hop paths traversing a single node (rarely exploited in current solutions);
- concurrent exploitation of infrastructure-based and ad-hoc links (rarely exploited in current solutions);
- sporadic and intermittent access to the traditional wired Internet infrastructure (e.g., via UMTS and IEEE 802.11 access points).

To practically understand the potential benefits of spontaneous networks, let us rapidly sketch a simple application scenario. Suppose that a nodeA has sufficient local resources (e.g., under-utilized processing power and bandwidth) to decide offering some contents in its local storage and that a nodeB is looking for some of these contents. Then, a spontaneous network could be dynamically set up for the time of application provisioning, by using a WiFi ad-hoc single-hop link between nodeA and nodeC, a Bluetooth single-hop link between nodeC and nodeD, and finally another WiFi infrastructure-based link between nodeD and nodeB. All nodes could move at runtime, exit from reciprocal visibility, or revoke their resource/connectivity offer, with obvious impact on path availability; offered contents could be transmitted over multiple paths; management decisions should be taken locally given the costs of an overall updated view of the status of sporadic and mission-oriented spontaneous networks.

It is obvious that the concurrent exploitation of both ad-hoc and infrastructure-based connectivity in intermittent and heterogeneous multi-hop paths make complex to manage spontaneous networks directly into the interested applications. We claim the need for novel easy-to-use middleware capable of properly addressing the hard technical challenge of dynamically exploiting all the resources available in spontaneous networks,

with no need of complete, global, and strictly updated knowledge about network topology and characteristics. To this purpose we propose an original middleware, called Real Ad-hoc Multi-hop Peer-to-peer (RAMP), for the transparent management of multi-hop heterogeneous spontaneous networks. RAMP adopts an application-layer approach and exploits cross-layering for the dynamic and application-specific (*mission-oriented*) configuration of spontaneous networks.

In particular, RAMP reduces the complexity of application development over spontaneous networks by transparently managing the technical challenges related to i) IP addressing, ii) global decisions based on limited local visibility, and iii) erratic behaviors of mobile peers sharing resources in an impromptu way. First, it handles heterogeneous physical links, effectively manages potentially conflicting IP address spaces, and enables application-layer routing (without the need of layer-3 support), which has demonstrated good flexibility and limited overhead for spontaneous networks. Second, RAMP only exploits a localized and scarcely coordinated vision of the network status (*lightweight autonomic management*): one of the primary and original guideline is to effectively and locally manage the mission-oriented needed path at the application layer, depending on dynamically determined application-specific requirements. Third, it addresses the issues of working with statically unknown network topologies and of handling heterogeneous wireless nodes that can join/exit/move dynamically; the hard technical challenge is to achieve these goals by limiting node overhead, so to encourage peer-based resource sharing.

Differently from many related proposals in the literature, the RAMP API is easy to use (to leverage the realization of sharing applications by a large community of developers, also not experts of spontaneous networking at all). Most important, the RAMP prototype is available for download [4] and easy to deploy, with the intent of becoming a useful and practical contribution for the research community and of encouraging application development and experimentation over existing wireless technologies.

2. Middleware Solution Guidelines for Spontaneous Networking

To better point out the challenging environments targeted by RAMP, let us rapidly sketch a practical example of spontaneous network. Consider the realistic case of a group of students in a lecture hall carrying on mobile clients equipped with multiple heterogeneous communication interfaces (see Fig. 1), e.g., laptops with IEEE 802.11 and Bluetooth, cell phones with UMTS and Bluetooth, and smart phones with UMTS, IEEE 802.11, and Bluetooth. Students can interact to share lesson notes via subgroups created in an impromptu way, by possibly participating to multiple subnets simultaneously. The dynamicity of such a scenario pushes for distributed management; nodes should administrate their just created and mission-oriented subnet(s), by providing addresses with local scope and possibly exploiting the same address ranges in different subnets. Therefore, only nodes in the same subnets can directly identify and communicate each other. In addition, nodes can abruptly revoke shared resources with relatively high frequency, e.g., because of mobility.

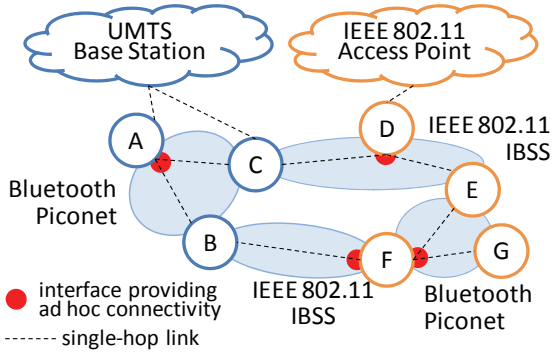

Figure 1. A simple example of spontaneous network.

Considered the above scenario, we claim the need of adopting the following solution guidelines for spontaneous networking middleware:

- ***application-layer management***. For instance, traditional layer-3 routing is unsuitable for spontaneous networking given the mission-oriented, heterogeneous, and intermittent nature of the exploited multi-hop paths. Routing management at the application layer could enable application-specific and provision-time decisions with limited overhead (see Section 5);
- ***cross-layering management***. Mainly for the performance sake, applications can relevantly benefit from the mediated awareness of some low-layer details, e.g., number of hops between clients and servers, and vice versa. Similarly, management decisions traditionally taken at lower layers can fruitfully exploit application-layer visibility (see *bufferSize* in Section 4);
- ***limited local visibility and local management decisions***. For instance, the middleware components at a node should be able to locally reconfigure a supported path at runtime by exploiting only partial localized visibility and by involving only a very limited set of other nodes, either in the vicinity or along the path;
- ***reactive and mission-oriented approach*** to find

resources exactly and only when needed by the specifically supported applications. Given the relatively high dynamicity of spontaneous networks, proactive general-purpose approaches and distributed caching of monitoring status are usually expensive and inefficient from the point of view of performance improvements;

- ***stateless communications***. Considered the dynamicity and heterogeneity of spontaneous networks, it is either unfeasible or unsuitable to explicitly create an end-to-end client-to-server channel. Paths should be dynamically handled as the application-layer combination of independent single hops, with no state kept at low layers and with proper process&forward techniques at each node.

In addition, to leverage its adoption, middleware for spontaneous networking should support both unicast and broadcast communication abstractions, in order to provide a simple and wide-accepted basis for any general-purpose application need. Moreover, the middleware should easily enable the registration/discovery/invocation of services dynamically and temporarily offered by spontaneous network peers. Finally, to facilitate use and leverage adoption, the middleware facilities for communication/service management should be transparent (independent) with regard to low-level implementation details about i) how the spontaneous network has been created (single-hop instantiation and control), ii) which specific wireless technologies are employed (e.g., Bluetooth, WiFi in ad-hoc mode, and WiFi in infrastructured mode), and iii) which operating system runs at each participating node.

3. The RAMP Middleware

The architecture of the RAMP middleware consists of two main layers: the Service Layer and the Core Layer. The former supports high-level features for peer-to-peer service offering and discovery, while the latter provides low-level primitives for end-to-end communication (unicast and broadcast communication abstractions). These layers work with another cross-layer middleware component, i.e., Social Enforcer, which can perform traffic monitoring/shaping in a per-node and per-packet way, thus encouraging social sharing of resources by awarding collaborative nodes and penalizing selfish ones. Details about Social Enforcer are out of the scope of this paper and can be found in [5].

In addition, RAMP exploits the mechanisms developed within the Multi-hop Multi-path Heterogeneous Connectivity (MMHC) project, already presented elsewhere [6], for the dynamic setting of ad-hoc subnets (layer-2 link creation and layer-3 network configuration). The MMHC goal is to provide the best multi-hop Internet connectivity via proper local configuration based on innovative context indicators (e.g., probability of joint mobility of peer nodes) to maximize connectivity reliability, throughput, and availability. In particular, RAMP takes advantage of MMHC to create/manage heterogeneous single-hop links and to identify nodes in distant subnets. However, the former MMHC rerouting mechanism manages multi-hop paths only based on default gateway modifications, e.g., to forward connections via the wireless interface with minimum energy consumption or maximum throughput [6]. Instead, RAMP more flexibly dispatches packets at the application layer by exploiting every available single-hop link enabled by the underlying MMHC, thus with the valuable additional advantage of supporting simultaneous exploitation of any available path, despite operating system-level configuration of routing tables.

By delving into finer details, RAMP identifies a remote node by exploiting the IP addresses assigned by MMHC to the intermediary nodes composing the path to that node. For instance, in Fig. 2 NodeA identifies NodeB via the [2, 4, 6] sequence while NodeB identifies NodeA as [5, 3, 1] (sequences differ depending on path directions because different wireless interfaces are exploited in the two ways). Let us rapidly note that RAMP can be used on top not only of MMHC but of any available solution for dynamic path determination and re-qualification over spontaneous networks.

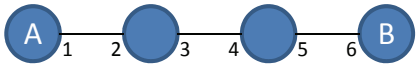

Figure 2. Node identification depending on traversed wireless (possibly heterogeneous) interfaces.

3.1. The RAMP Service Layer

The RAMP Service Layer supports the registration and discovery of peer-to-peer services that can exploit the Core Layer communication abstractions described in the following sub-section. Service layer consists of two primary components:

- **Service Manager**, which handles the registration, advertising, and discovery of services offered by RAMP nodes in a peer-to-peer way;
- **Application**, a generic component to be refined and registered to offer real services, e.g., audio/video on-demand streaming or file sharing (see the related details in Section 5).

The Service Manager API includes two main methods: `registerService` and `findServices`. `registerService` allows to register a local service in a local repository directly handled by the Service Manager. For each service, there is the need to specify service name, local port where the service waits for requests, and protocol.

```
void registerService(
   String serviceName, int servicePort, int
   protocol );
```

The `findServices` method permits to local applications to discover the location of remote services based on UDP `sendBroadcast` (see Section 3.2). The discoverer application has to specify service name, `TTL` of the discovery process, a timeout to avoid indefinite blocking, and maximum number of service replicas the application is willing to wait for. Note that the method returns either after `timeout` ms or after that `responseAmount` services have been found. Then, the application can select the preferred one among the multiple replicas, e.g., by invoking the closest printing service. Let us note that working at the application layer allows RAMP to easily include additional features to discovery. For instance, we are currently working to support the indication of service names with wildcard characters and to recommend the selection of the available replica based on path throughput estimation [6].

Service Manager actively listens to a well-known port to receive broadcast packets looking for remote services. Once received a discovery request, it looks for the requested service in the local repository: if locally available, Service Manager sends (`sendUnicast` – see Section 3.2) a response specifying the port and protocol that the discovered service exploits. These data, coupled with Core Layer addressing information, allow the discoverer to identify and invoke the service.

```
Vector<ServiceResponse> findServices (
   String serviceName, int TTL, int timeout,
   int responseAmount );
```

3.2. The RAMP Core Layer

The RAMP Core Layer is composed by two primary components (see Fig. 3):

- End-to-End (E2E), offering low-level unicast/broadcast communication abstractions to send/receive packets;
- Inter-node, really exchanging packets between RAMP nodes to support E2E high-level primitives.

E2E component. E2E offers three main methods: `sendUnicast`, `sendBroadcast`, and `receive`. `sendUnicast` sends a payload to a destination node. The `dest` parameter identifies the destination via the ordered set of intermediary nodes (and in particular their MMHC addresses) composing the multi-hop path between sender and receiver. `destPort` and `protocol` specify the port and protocol (either UDP or TCP) at which the receiving application is waiting for the packet. If `ack` is false, `sendUnicast` immediately returns true. If `ack` is true, the sender waits at most `timeoutAck` ms for an explicit acknowledgment from the receiver; `sendUnicast` returns true if the acknowledgment packet reaches the sender before `timeoutAck` ms, false otherwise.

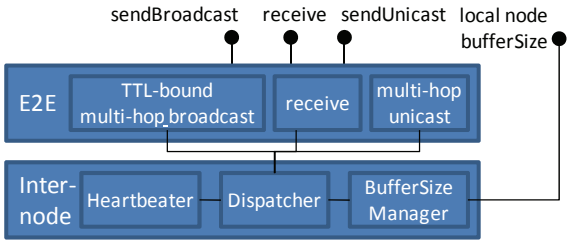

Figure 3. The RAMP Core Layer.

In the case of UDP protocol, the RAMP payload size is kept under 60KB, not to exceed the maximum underlying UDP size. Instead, in the case of TCP, the RAMP payload has not a maximum size; for instance, a single RAMP-layer packet can include even a whole large-size file. In this case, the `bufferSize` parameter specifies the fragmentation granularity to be used by our middleware, which significantly affects the achievable performance, as detailed in Section 5.

```
boolean sendUnicast (
   Vector<InetAddress> dest, int destPort, int
   protocol, boolean ack, int timeoutAck,
   int bufferSize, Object payload );
```

The `sendBroadcast` method sends a RAMP packet with the desired payload to every node whose distance in hops is equal to or less than `TTL` from the sender. Every time the packet reaches a node, RAMP forwards it to the specified local `destPort` using `protocol`, either UDP or TCP. In order to limit the impact of broadcast communications on network performance, the size of broadcast packets is kept under 60 KB.

```
void sendBroadcast (
   int TTL, int destPort, int protocol,
   Object payload );
```

Finally, `receive` waits for unicast and broadcast packets at the specified `localPort` using `protocol`. The method waits at most `timeout` ms or until a packet is received if `timeout` value is set to 0.

```
Packet receive (
   int localPort, int protocol, int timeout );
```

Inter-node component. The Inter-node component manages the packets provided by E2E and routes them to destination. Its primary sub-component is the Dispatcher, which performs communication operations between nodes at single-hop distance. Whenever Dispatcher sends a packet to a remote node, it opens a new socket over the single-hop link, via either TCP or UDP. Dispatcher exploits different sockets to send different packets, with one packet entirely via the same socket. In this way, the concurrent transmission of different packets (possibly of large size for TCP) can achieve

good performance, as detailed in Section 5. Dispatcher manages unicast packets as follows:

a) if the local node is the destination, it sends the packet to the localhost at `destPort`;
b) otherwise, it sends the packet to the Dispatcher of the following node specified in `dest`.

Dispatcher manages broadcast packets as follows:

a) if the local node is not the sender (first hop), it decrements `TTL` and sends the packet to the localhost at `destPort`;
b) if `TTL`>0, it gathers the set of neighbors from Heartbeater (see the following) and then sends the packet to any neighbor, except for nodes in the same subnet of the sender.

It is worth noting that the set of MMHC addresses exploited to send a packet from a sender to a receiver depends on the direction (Fig. 2). Also for this reason, when Dispatcher sends a packet to a remote node (either unicast or broadcast, either TCP or UDP), it adds the address of the node it has received the packet from (to the `source` packet header field). So, the destination also receives the already determined set of nodes to possibly send back data to the sender.

The Heartbeater sub-component works to keep track of the set of single-hop RAMP neighbors. To discover these nodes, Heartbeater periodically sends a heartbeater request via UDP to 255.255.255.255 (every 60s). Any RAMP-enabled node in the same subnet replies with a heartbeater response to the sender. Given its specific relevance for the presented application scenario, the implementation of BufferSize Manager is presented in detail in Section 4.

4. RAMP Implementation Insights

To achieve reasonable performance in challenging spontaneous networks, there is the need to properly and effectively manage RAMP packets, in particular when they are of large size. In fact, just to make an easily understandable basic example, if RAMP intermediary nodes managed packets in the trivial *receive&forward* way, transmission time would increase linearly with path length: a node should complete the reception of a packet before passing it to the following one. However, also because Dispatcher exploits different sockets for different packets, it can split packets in data chunks and perform their pipeline management, i.e., passing the first chunk of a packet to the following node while still waiting for receiving other chunks of the same packet.

The size of each data chunk is defined by `bufferSize`. Whenever the size of a TCP unicast packet is greater than `bufferSize`, the sender Dispatcher splits it and sends the resulting chunks one after the other (broadcast and UDP unicast packets have limited size, making unnecessary to split them). First of all, Dispatcher sends the packet header, containing `dest`, `destPort`, `protocol`, `bufferSize` and `source`; then, it sends the payload in chunks of at most `bufferSize` bytes. Via the header fields the receiving Dispatcher knows the identity of the next node prior to receiving the whole packet, thus enabling chunk forwarding. The value of `bufferSize` is a key parameter for RAMP middleware performance:

- a low `bufferSize` value, on the one hand, triggers very frequent read/write operations, thus increasing communication overhead. On the other hand, it shortens the read phase before forwarding and limits its memory usage of forwarding nodes, thus permitting to exploit only small local buffers;
- a high `bufferSize` value, on the one hand, reduces the communication overhead because it limits the number of open sockets and read/write operations. On the other hand, it increases packet delivery time because nodes have to receive relatively large chunks of data before forwarding them and requires large buffers for temporary data storage.

We have performed extensive in-the-field experiments to characterize the performance of our middleware depending on packet size, path length, and `bufferSize`. Based on those results we can propose setting/deployment values that lead to reasonable performance in a wide range of situations (see Section 5). However, selecting a proper `bufferSize` value is challenging because its optimal value depends on different factors. For instance, in a specific application domain, high `bufferSize` values could be preferable to increase reliability (lower number of chunk transmissions). In addition, a traversing node may call for reducing `bufferSize` to limit local memory usage, or for increasing it to limit the number of read/write operations. Therefore, we claim the suitability for the RAMP middleware to enable the dynamic setting of `bufferSize`, if needed, either by applications in a cross-layer way (via `sendUnicast`) or by intermediate nodes.

To the purpose of dynamic `bufferSize` configuration and, more generally, of enabling the flexible introduction of any application-layer operation on RAMP transmitted packets at runtime, we have implemented the Dispatcher according to a listener-based architecture, which permits to efficiently and easily modify exchanged packets at any traversing node. Dispatcher permits to add and register listeners interested in monitoring incoming packets (Fig. 4): for instance, the BufferSize Manager listener handles the `bufferSize` value based on the requirements of the local intermediate node.

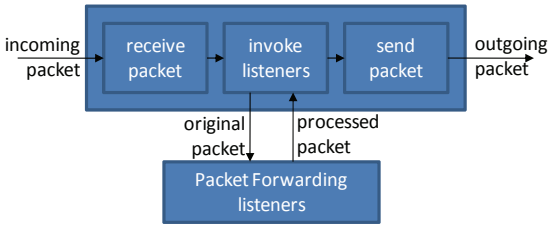

Figure 4. Activity flow in RAMP Dispatcher.

Note that the adoption of a listener-based architecture permits to easily extend RAMP capabilities. On the one hand, the basic Dispatcher component does not perform any additional computation related to `bufferSize`; only nodes interested in controlling `bufferSize` activate and register the BufferSize Manager listener; that permits to limit middleware overhead, by only activating the needed features with a per-node fine granularity. On the other hand, developers can implement and deploy additional listener-based components to support novel features, by extending RAMP capabilities without any modification of the basic Dispatcher.

5. The File Sharing Service and Related Experimental Results

To validate the feasibility of our approach and to evaluate RAMP performance, we have implemented a simple file sharing service. In particular, in this section we show the relevance of exploiting buffers of proper size when transmitting huge packets. This aspect is crucial to allow the efficient transmission of data among multi-hop spontaneous networks.

On the server side (node offering the service), to make the new service available to the other RAMP nodes, there is only the need to register it to the local Service Manager (`registerService`) and then simply wait for requests (`receive` with `timeout` set to 0). In our implementation, a file sharing request may be for either the list of available files or the download of a given file. In both cases File Sharing Service replies directly to the requester, with no necessity to manage the underlying network heterogeneity and complexity (via `sendUnicast`). On the client side, there is only the need for the File Sharing Client to call `findServices` of local Service Manager; once determined the node offering the service, the client can simply require the list of shared files or the content of a given file via `sendUnicast` and `receive`. Note that File Sharing Client and Service exploit both UDP for service discovery and TCP for file content transfer.

We have tested our file sharing service while varying file size ([100KB, 10MB] range), path length (1, 2, or 3 hops), and `bufferSize` (disabled or in the [1KB, 1MB] range). To easily compare performance results, we have limited the bandwidth of each single-hop link to a maximum of 2Mbit/s. To better understand the reported results, we identify a lower bound transfer time, i.e., the time needed for file transfer over a traditional TCP/IP fixed network, experimentally determined via the `iperf` command (no notable differences have been observed while varying the hop number). The distance between RAMP performance and this lower bound also indicates the overhead of routing choices at the application layer. Table 1 summarizes `iperf`-based results and *file_size/bandwidth* ratios.

Table 1. Lower bounds for file transfer time.

File Size	Size/Bandwidth (s)	`iperf`-based (s)
10 MB	40	42.2
5 MB	20	21.4
1 MB	4	4.2
500 KB	1.95	2.0
100 KB	0.39	0.4

Fig. 5 shows the experimental results about file transfer times. When `bufferSize` is disabled (`bufferSize` >= file size), as expected, the transfer time approximately doubles in the case of 2-hop paths and triples in the case of 3-hop paths, given that intermediate nodes have to receive the whole RAMP packet (entire file) before sending it to the next node. `bufferSize` values lower than the file size relevantly reduce the transfer time: for instance, considering the 10MB file, with `bufferSize`=1MB the transfer time passes from 126.8s/85.6s to 63.8s/54.2s in the case of 3/2-hop path, respectively. Performance results further improve when adopting lower `bufferSize`, rapidly becoming very close to `iperf`-based lower bound and clearly showing the very little overhead introduced by the RAMP management at the application layer.

However, the file transfer time increases when exploiting very low `bufferSize` due to the number of read/write operations (e.g., see 10MB file, 3-hop path, 5KB and 1KB `bufferSize` values). In addition, for small file sizes the time required to open a new socket becomes not negligible compared with the actual data transfer time; the whole file transfer time shows a linear component depending on the number of hops (see 100KB file, 10KB/5KB/1KB `bufferSize` values).

We define the best `bufferSize` value as the value that minimizes file transfer time while limiting the number of read/write operations; in other words, it is the highest `bufferSize` achieving low transfer time. The performance results presented in Figure 5 show that the best `bufferSize` value varies depending on file size and hop number: for instance, for 10MB files, about 50KB for 3-hop paths and about 100KB for 2-hop paths. We have fixed a sub-optimal default

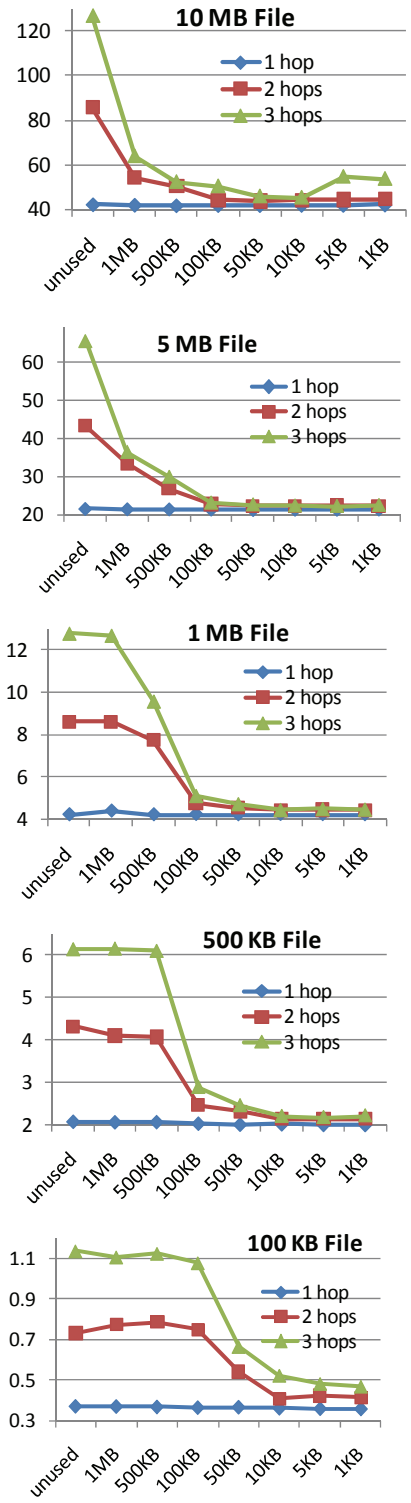

Figure 5. File transfer time (y axis, in s) depending on `bufferSize` (x axis) for files of different size.

`bufferSize` value of 50KB for applications not interested in or not able to explicitly select a suitable `bufferSize`.

The proposed default value achieves good results in a wide range of practical situations, with different file sizes and path lengths. In addition, RAMP provides applications with a method returning the best `bufferSize` depending on packet size and path length; applications can use this mechanism to properly set `bufferSize` in a fine-grained per-packet way, thus showing again the usefulness of a cross- and application-layer approach for maximum flexibility and performance.

6. Related Work

Several proposals have recently investigated some specific partial aspects of multi-hop connectivity. Some work is starting to propose the synergic and simultaneous exploitation of heterogeneous wireless interfaces at mobile terminals. Most have focused on one specific technology, such as IEEE 802.11 or GPRS/UMTS. Their main goal is seamless connectivity in deployment environments where these technologies are integrated. For instance, [7] aims at extending cellular networks via relay stations, primarily to increase coverage. [8-9], instead, specifically address the management of client mobility in heterogeneous multi-hop networks.

By focusing on spontaneous networking, some contributions aim at improving connection quality via low-level solutions. For instance, [1] improves wireless medium exploitation, by opportunistically accessing the available spectrum. [10] optimizes bandwidth allocation in multi-hop networks by differently managing real-time and best-effort transmissions. Other solutions determine the best route towards a destination by exploiting evaluation metrics based on low-level context [11]. In short, most recent contributions aim at supporting spontaneous networking in homogeneous networks, by introducing non-standard modifications to layer-2 protocols. In addition, they do not address the heterogeneity issues associated with multiple interfaces and IP addressing as our RAMP middleware does.

Finally, opportunistic networking represents an interesting alternative for connectivity in highly dynamic spontaneous networks [1]. For instance, [3] supports the opportunistic delivery of data in intermittently connected mobile ad hoc networks. However, the proposal in [3] is only based on simulations and only considers homogeneous networks with plain addressing.

7. Conclusions

Recent research is recognizing the suitability of novel middleware to leverage the adoption of multi-hop wire-

less networking, thus fully exploiting the heterogeneous networking opportunities available nowadays. The paper highlights the suitability of adopting an application- and cross-layer middleware for spontaneous networking that concurrently exploits multiple and heterogeneous connectivity opportunities with limited overhead and localized management decisions. The encouraging results obtained by the RAMP prototype are stimulating our on-going research activities. In particular, we are currently evaluating the RAMP performance with continuous services with stricter delay requirements, e.g., real-time multimedia streaming.

References

[1] H.B. Salameh, M. Krunz, “Channel Access Protocols for Multihop Opportunistic Networks: Challenges and Recent Developments”, *IEEE Network*, Vol. 23, No. 4, pp. 14-19, July-Aug. 2009.

[2] M.D. de Amorim, A. Ziviani, Y. Viniotis, L. Tassiulas (eds.), Special Issue on “Practical Aspects of Mobility in Wireless Self-organizing Networks”, *IEEE Wireless Communications*, Vol. 15, No. 6, Dec. 2008.

[3] M. Musolesi, C. Mascolo, “CAR: Context-Aware Adaptive Routing for Delay-Tolerant Mobile Networks”, *IEEE Trans. Mobile Computing*, Vol. 8, No. 2, pp. 246-260, Feb. 2009.

[4] RAMP Web site, lia.deis.unibo.it/Research/RAMP

[5] P. Bellavista, C. Giannelli, "Social Sharing of Connectivity Resources: Control and Encouragement of Unselfishness in Mobile Environments", *5th Int. Mobile Multimedia Comm. Conf. (Mobimedia)*, UK, Sep. 2009.

[6] P. Bellavista, A. Corradi, C. Giannelli, "Mobility-aware Middleware for Self-Organizing Heterogeneous Networks with Multi-hop Multi-path Connectivity", *IEEE Wireless Communications*, Vol. 15, No. 6, pp. 22-30, Dec. 2008.

[7] L. Le, E. Hossain, "Multihop Cellular Networks: Potential Gains, Research Challenges, and a Resource Allocation Framework", *IEEE Communications*, Vol. 45, No. 9, pp.66-73, Sep. 2007.

[8] S. Pack, X. Shen, J.W. Mark, J. Pan, "Mobility Management in Mobile Hotspots with Heterogeneous Multihop Wireless Links", *IEEE Communications*, Vol. 45, No. 9, pp.106-112, Sep. 2007.

[9] P.P. Lam, S.C. Liew, "Nested Network Mobility on the Mul-tihop Cellular Network", *IEEE Communications*, Vol. 45, No. 9, pp.100-104, Sep. 2007.

[10] H. Wu, Y. Liu, Q. Zhang, Z.-L. Zhang, "SoftMAC: Layer 2.5 Collaborative MAC for Multimedia Support in Multihop Wireless Networks", *IEEE Trans. on Mobile Computing*, Vol. 6, No. 1, pp.12-25, Jan. 2007.

[11] M.E.M. Campista, et al., "Routing Metrics and Protocols for Wireless Mesh Networks", *IEEE Network*, Vol. 22, No. 1, pp.6-12, Jan.-Feb. 2008.