Middleware Solutions for Self-organizing Multi-hop Multi-path Internet Connectivity Based on Bluetooth

Paolo Bellavista and Carlo Giannelli

Dept. Electronics Computer Science and Systems (DEIS), University of Bologna
Viale Risorgimento, 2 – 40136 Bologna, Italy
{paolo.bellavista,carlo.giannelli}@unibo.it

Abstract. The availability of heterogeneous wireless interfaces and of growing computing resources on widespread portable devices pushes for enabling innovative deployment scenarios where mobile nodes dynamically self-organize to offer Internet connectivity to their peers via dynamically established multi-hop multi-path opportunities. We claim the suitability of novel, mobility-aware, and application-layer middleware based on lightweight evaluation indicators to support the complexity of that scenario, involving heterogeneous wireless technologies over differentiated and statically unpredictable execution environments. To validate these claims, we have implemented an innovative middleware that manages the durability/throughput-aware formation and selection of different multi-hop paths simultaneously. This paper specifically focuses on how our middleware effectively exploits Bluetooth for multi-hop multi-path networking, by pointing out the crucial role of i) compliance with standard solutions to favor rapid deployment over off-the-shelf equipment and ii) the reduction of the usual overhead associated with some expensive Bluetooth operations, e.g., device inquiry. In particular, the paper shows how it is possible, on the one hand, to extend JSR-82 to portably access monitoring indicators for lightweight mobility/throughput estimations and, on the other hand, to reduce the time needed to update the set of available Bluetooth-based connectivity opportunities via approximated and lightweight forms of discovery.

Keywords: Mobile Computing, Middleware, Bluetooth, Always Best Served Networks, Multi-hop Multi-path Connectivity, Collaborative Connectivity.

1 Introduction

Nowadays mobile devices, usually equipped with multiple wireless interfaces, can get connectivity to the traditional wired Internet by taking advantage of multiple connectivity opportunities provided by many infrastructure-based components, which tend to be ubiquitously available, e.g., IEEE 802.11 Access Points (APs) or UMTS Base Stations (BSs). In the following, we will call these connectivity components as *infrastructure connectors*. In addition, the increasing and increasing resources of mobile terminals potentially enable novel and more complex scenarios where client nodes can also help other clients to achieve Internet connectivity in a peer-to-peer way, e.g., via Bluetooth Personal Area Network (PAN) or IEEE 802.11 Independent Basic

C. Giannelli (Ed.): Mobilware, LNICST 0007, pp. 58–71, 2009.

Service Set (IBSS) connections. In other words, peer nodes can offer their resources by acting as intermediate entities in a multi-hop (possibly heterogeneous) path towards the Internet. We use the term *peer connectors* to indicate these novel connectivity opportunities. In these envisioned scenarios peer connectors are in charge of creating and properly managing a simple and small Mobile Ad-hoc NETwork (MANET) with the peers in proximity and of correctly routing packets between their MANET and the Internet by exploiting the near infrastructure connectors.

The increased complexity of this scenario, enabled by the concurrent exploitation of infrastructure/peer connectors, is widely counterbalanced by its potential benefits. In fact, benefits include not only the possibility to exploit a significantly wider set of connectivity opportunities, but also to dynamically select which opportunity to use at any time depending on system/user/node/application-specific requirements. For instance, a dynamic selection mechanism could be useful for connector load balancing, for always privileging connectivity opportunities that are for free, for preserving client/global node battery, or for respecting bandwidth requirements (see Section 2). However, the dynamic selection of the wireless interface for a client to exploit with its connectors is not trivial. Only to mention an example of such complexity from the beginning, let us consider that, nowadays, most mobile phones integrate GPRS/UMTS and Bluetooth interfaces (and several of them are equipped also with IEEE 802.11). In addition, many laptops and PDAs already combine medium/short-range IEEE 802.11 and Bluetooth wireless technologies with wide-range GPRS/UMTS/HSDPA cellular communication interfaces. We claim that it is inappropriate to leave to client application designers the whole burden of properly managing the wide set of Multi-hop Multi-path Heterogeneous Connectivity (MMHC) opportunities that are dynamically available. Therefore, in our opinion there is the need for novel, suitable, and lightweight middleware solutions for effective MMHC management, which should mainly work at the client side in a decentralized way to minimize management overhead and generated network traffic.

These middlewares should have visibility of different kinds of innovative context data to take proper MMHC decisions, especially to ensure the usability of enabled MMHC opportunities by selecting the ones expected to be more reliable during the service session that is going to be established. In particular, lightweight estimations about client mobility (with regards to both fixed infrastructure and mobile peer connectors) could allow to exclude the connectors that are probably going out of the coverage area of the considered client soon. In that way it is possible to reduce the search space of potential connectivity opportunities to take into account. Similarly, context data about estimated throughput (achievable by a single wireless hop and by the multi-hop paths including that hop) can help filtering out connectors that do not comply with session quality requirements. Finally, context data about connector residual energy could help in balancing energy consumption and in taking proactive re-configuration of exploited paths if some composing hops are expected to fail soon due to power exhaustion.

Our previous work has already demonstrated the crucial role of context to dynamically evaluate networking opportunities in MMHC scenarios [1] and presented the architecture and the primary design guidelines of our MMHC middleware [2, 3]. Our MMHC prototype is able to self-organize client nodes, by dynamically retrieving the set of available infrastructure and peer connectors. In addition, MMHC client nodes

self-hail multi-hop paths to the Internet, by possibly modifying routing rules at runtime in case of intermediate peer failure. Here, the paper originally focuses on how our novel middleware supports the exploitation of Bluetooth to get and provide Internet connectivity via peer-to-peer multi-hop paths. In general, let us notice that Bluetooth is widely considered as a complementary connectivity technology to IEEE 802.11 and cell-based communications: in fact, on the one hand, it exhibits significantly lower power consumption, thus maximizing node battery life, and, on the other hand, its limited throughput (less than 1Mbps for Bluetooth1.2, about 3Mbps for Bluetooth2.0EDR) is sufficient for many commercial applications, e.g., periodic download of email messages. Also note that only a subset of Bluetooth capabilities are currently exploited by a large public of users: Bluetooth is mainly used only to connect mobile clients with remote devices, e.g., a wireless mouse/keyboard or a printer, with a very few industrially-relevant support solutions for exploiting it to get/provide Internet connectivity, as MMHC can enable.

In particular, this paper originally addresses (and presents related design/implementation solutions) two key aspects for the support of Bluetooth-based multi-hop multi-path networking, i.e., the adoption of *standard* solutions capable to be rapidly deployed over off-the-shelf equipment and the achievement of *performance efficiency* by limiting the usual overhead associated with some expensive Bluetooth operations. Standardization is crucial to provide a solution that can run on different mobile clients, equipped with heterogeneous resources in terms of operating systems, wireless card interfaces, and related drivers, provided by different manufactures. That is particularly relevant given the high heterogeneity of current wireless devices, from PDAs to smart phones. In addition, the efficiency improvement of some Bluetooth mechanisms, such as device discovery via the Bluetooth inquiry procedure, is of primary importance to quickly determine new paths at runtime. In fact, node mobility may delve into frequent abrupt disconnections, periodically requiring discovery and reconnection phases which call for fast, effective, and possibly approximated lightweight solutions for the dynamic update of available connectivity opportunities.

The remainder of the paper is organized as follows. Section 2 presents our target deployment scenario, by pointing out the main characteristics of Bluetooth-based multi-hop multi-path networking and the need for improved efficiency in Bluetooth connector discovery and connectivity establishment. Section 3 sketches the architecture and the primary components of our MMHC middleware, while Section 4 details the MMHC tasks to support multi-hop paths, by specifically showing how Bluetooth devices may represent a performance bottleneck for path management. Section 5 proposes our original extension of the JSR-82 Java APIs for Bluetooth [4], integrated in the MMHC middleware to easily achieve a portable and efficient solution for Bluetooth-based MMHC networks. Conclusive remarks and directions of current research end the paper.

2 Deployment Scenario, Motivations, and Solution Guidelines

The MMHC scenario relevantly improves the traditional networking capabilities of wireless environments. First of all, it *extends connectivity opportunities* via multi-hop

ad-hoc paths, thus allowing the Internet access of nodes not directly in the coverage area of infrastructure connectors. Second, it enables the *exploitation of multiple paths simultaneously*, e.g., to improve the overall throughput available at clients. Third, it permits to *increase connectivity availability*, e.g., by enabling the rapid rerouting of traffic flows to alternative paths when the exploited ones become unavailable.

To better and practically point out these advantages, let us rapidly sketch an example of a possible MMHC deployment scenario, involving different wireless interfaces, Bluetooth included. Consider the realistic case of a group of tourists moving together and sharing pictures via Bluetooth single-hop links. Due to their limited coverage range, there could be the need for multi-hop paths to reach target friends who are currently lingering in a shop; that is enabled by collaborating tourist devices that, for instance, can transparently exploit Bluetooth to receive and forward packets along the right direction, e.g., node C in Fig. 1. In addition, some tourists may be willing to periodically publish their pictures on their Web blogs even if they have no direct UMTS connectivity, e.g., they do not want to subscribe to a local UMTS provider while visiting Italy. These tourists can benefit from Bluetooth multi-hop ad-hoc connectivity toward the devices of friends with flat-rate UMTS subscription, who offer them free Internet connectivity, e.g., the E-C-A node chain toward BS_1. In that way client nodes greatly benefit from the widening of connectivity opportunities: thanks to MMHC, they can reach other peers and infrastructure connectors even if they are not directly available via single-hop links. Note that tourists' mobility may reduce the reliability of MMHC opportunities; usually there is the need to favor the selection of MMHC opportunities with application-compatible reliability (especially in terms of expected durability).

Similarly, when moving from city to city by train, tourists should be able to exploit MMHC opportunities offered by other passengers, possibly in other cars, reachable via multi-hop heterogeneous paths, and connected to the Internet via Wi-Fi/WiMAX APs, such as node B. In this case the nodes tend to move together (joint mobility) and MMHC opportunities have similar expected durability. Therefore, MMHC selection should not only be mobility-aware, but also consider application-specific quality

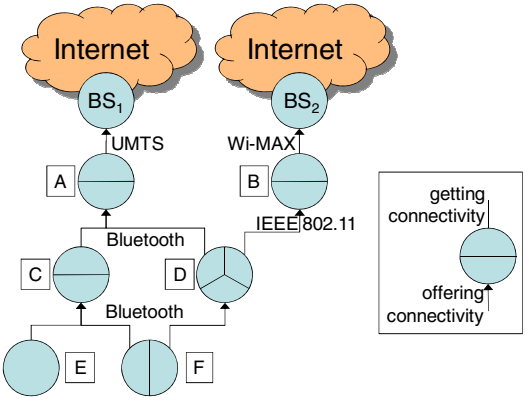

Fig. 1. An example of MMHC scenario

requirements, e.g., expected throughput. MMHC enables to fully take advantage of all the paths that are simultaneously available, e.g., by dynamically switching to a path with estimated larger bandwidth. Moreover, if node A leaves the network, e.g., to limit its battery consumption, node D can reroute its active connections from node A to B, thus minimizing user-perceived service disruption. However, in that case, node C would have no access to the Internet anymore, since A was its only connector. In that simple MANET, MMHC self-organizes to provide new Internet access opportunities, e.g., with node F starting to serve as connector, thus providing C with connectivity towards BS_2. Thus, MMHC contributes to increase connectivity availability, by self-healing paths via dynamic reconfiguration of network topology.

To clearly position our approach with regards to very recent related literature, our novel MMHC middleware solution enables multi-hop multi-path networking in a similar way, to some extent, to what the Extended Service Set mesh network intends to do in the upcoming IEEE 802.11s standard [5]. MMHC and IEEE 802.11s scenarios have some similarities but also relevant differences in terms of node roles, multi-hop multi-path connectivity establishment, and management metrics for routing rules.

First of all, both MMHC and IEEE 802.11s define different possible roles for participating nodes: the nodes getting and providing connectivity to others (i.e., MMHC peer connectors and IEEE 802.11s Mesh Points) and the ones only getting connectivity (client nodes in MMHC, client stations or STAs in the traditional IEEE 802.11 terminology). In addition, IEEE 802.11s supports the exploitation of Mesh Access Points, to provide also mesh-external nodes with connectivity, and of Mesh Portals, to interconnect the supported mesh network with other external networks. Our MMHC solution does not remark the static distinction among Mesh Points, Mesh Access Points, and Mesh Portals; any (peer) connector can dynamically decide which of the three roles to play depending on the execution environment and also different roles simultaneously. Moreover, MMHC connectors offer connectivity opportunities without requiring a global notion of the mesh network, thus promoting local and decentralized management decisions.

Furthermore, both MMHC and IEEE 802.11s provide multi-hop multi-path access to the Internet, also in order to facilitate and improve the effectiveness of connectivity self-healing in the case of intermediate node failure. However, for routing purposes IEEE 802.11s can only exploit Mesh Nodes, Mesh Access Points, and Mesh Portals based on IEEE 802.11s. On the contrary, MMHC enables the exploitation of multiple wireless technologies simultaneously, integrated into the MMHC solution with an application-layer approach, thus allowing also the exploitation of paths made up by heterogeneous single-hop links. Finally, IEEE 802.11s exploits a low-level radio-aware link metric, mainly based on bit error rate. On the contrary, to evaluate link and multi-hop paths, MMHC can exploit more expressive context information: node mobility estimations for reliability purposes, path throughput estimations to maximize quality, and energy availability estimations to enhance long-term availability [1].

In our opinion, there is wide space for proposing innovative solutions for evolving towards more powerful and dynamic heterogeneous scenarios where nodes can collaborate by exploiting different interfaces simultaneously to receive and send data in multi-hop paths. For instance, [6] proposes a two-hop-relay architecture, based on Relay Gateway (RG) nodes that can behave both as usual nodes and as cellular gateways. They can seamlessly switch interfaces depending on network availability. In

addition, they can improve WLAN coverage by exploiting cellular interfaces where WLAN connectivity is not available. Mobile nodes have to explicitly request for RG-based connectivity in a non-transparent way. In [7] mobile nodes, namely the Proxy Client, can interwork with both cellular and IEEE 802.11 ad-hoc networks. Differently from previous examples, in [7] mobile nodes can interact with Proxy Clients not only directly, but also via intermediate mobile nodes in a multi-hop ad-hoc way. With an additional degree of mobile node involvement, there are a very few proposals aiming at the coordination of a set of mobile nodes to create MANET connectivity opportunities. Cooperating ad Hoc networking to sUpport Messaging (CHUM) dynamically elects one node to play the role of gateway between the MANET and the fixed network infrastructure [8]. In particular, CHUM exploits WLAN connectivity on the MANET side and 3G on the infrastructure side. [9] provides a similar example of MANET-3G integration, by exploiting SIP as the signaling protocol between nodes and the gateway.

In short, as a final comparative consideration, we claim that our MMHC middleware can relevantly extend the potential benefits targeted by IEEE 802.11s Extended Service Set mesh networks via the interconnection and exploitation of heterogeneous wireless technologies. In addition, MMHC can easily integrate with IEEE 802.11s and other wireless technologies, e.g., IEEE 802.11abg and Bluetooth, by exploiting them as possible wireless interfaces for its connectors.

By focusing on Bluetooth-enabled connectivity, which is the specific original topic of this paper, we claim that, nowadays, the primary issues limiting the adoption of Bluetooth in multi-hop multi-path scenarios are i) the lack of a portable and standard solution to enable multi-hop paths on heterogeneous clients and ii) the poor efficiency of currently adopted Bluetooth discovery/connection procedures.

First, Bluetooth is supported in a differentiated way, e.g., with differentiated APIs with heterogeneous functions and capabilities, over different operating systems and even depending on implementor-specific drivers. That calls for support solutions that can identify the execution context at runtime, in order to load and use specific modules to access the dynamically available drivers by exploiting different facilities (or sequence of facilities) to achieve the same goal in different environments. To this purpose, JSR-82 Java APIs for Bluetooth represents a notable example of industrial standardization effort. The JSR-82 APIs provide a uniform interface to Bluetooth capabilities, already available on many execution environments; however, they do not provide some features that are crucial for MMHC, such as gathering Received Signal Strength Indication (RSSI) values and establishing Bluetooth Network Encapsulation Protocol (BNEP) connections, as better detailed in Section 5.

Secondly, despite Bluetooth is primarily designed for mobile environments, it does not manage dynamically available connections promptly, requiring a long time period for pairing (i.e., connecting, according to Bluetooth terminology) two devices. For instance, it requires 10.24s for a complete inquiry procedure (i.e., remote device discovery in the Bluetooth terminology) and more than 6s for PAN connection plus DHCP configuration [3]. Of course, in mobile environments where clients can freely and abruptly move, there is the central need of efficient mechanisms for node discovery and connection creation. For instance, just to give a rough, preliminary, but practical idea, considering that the Bluetooth coverage range of most devices is around 10m, two mobile nodes should have a relative speed of less than 2m/s just to

have enough time to perform mutual discovery; to enable real data transferring, additional time for pairing and setup network configuration is required.

Starting from the above considerations, we have worked to include in our MMHC middleware novel facilities to access Bluetooth-related low-level details in a portable and effective way. On the one hand, our MMHC prototype exploits the JSR-82 APIs to minimize connection activation time portably. On the other hand, we have significantly extended the JSR-82 APIs to fully support the monitoring features required by our advanced MMHC estimation functions (portable and effective visibility of RSSI values for node mobility evaluation) and the efficient creation of BNEP connections.

3 The MMHC Architecture

By following the above design guidelines and considerations, we have developed and experimentally validated our MMHC solution, an open-source middleware prototype for the wireless management research community, available for download at our MMHC-companion Web site, in different distributions for the most spread operating systems (Linux, MS Windows XP/Vista, and MacOSX[1]).

Fig. 2 gives a high-level overview of our MMHC middleware architecture. Its main components are Network Interface Provider (NIP), which provides homogeneous management access to heterogeneous interfaces, Connection Manager (CM), which operates to establish and manage single-hop links, and Routing Manager (RM), which creates and dynamically handles multi-hop paths.

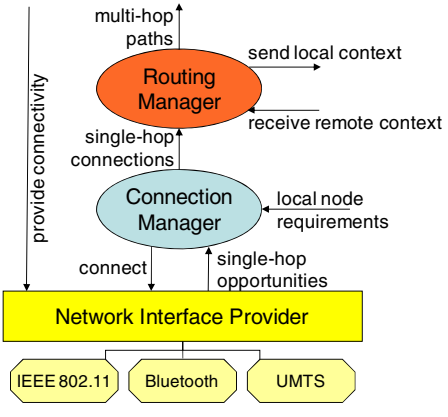

Fig. 2. Our MMHC middleware architecture

3.1 Network Interface Provider (NIP)

NIP interacts with network interfaces and provides upper layers with a transparent access to interface capabilities, by hiding low-level details of heterogeneous interface drivers and operating systems. NIP is organized into two layers: features and

[1] http://lia.deis.unibo.it/Research/MMHC/

wrappers. At middleware initiation time the feature layer considers the underlying execution environment and loads the right wrappers to communicate with the available interface drivers. In addition, it exposes an API to upper layers in order to enable the access to wireless interfaces even without any knowledge of low-level and interface-specific implementation details. The wrapper layer is in charge of directly interacting with interface drivers to perform the commanded operations, possibly in a system-dependent way. Note that the upper layer is developed once for all interfaces, while the lower layer has been implemented in different versions for each supported operating system. In that way, NIP also facilitates the introduction and exploitation of new interfaces over different operating systems by simply extending the only lower layer.

By delving into finer details, the feature component of NIP provides a set of capabilities common to any interface:

- *perform as peer connector*, to start offering connectivity with a specific interface in a peer-to-peer way;
- *connect to a connector*, to require the connection of an interface with a given connector and consequently establishing the associated channel, thus enabling inter-node communication;
- *get available connectors*, to obtain the list of connectors and their related information (e.g., the RSSI value) that an interface can currently access.

Additional details on how MMHC provides these features with Bluetooth devices on heterogeneous client nodes in the Section 5.

3.2 Connector Manager (CM)

CM gathers RSSI sequences from wireless interfaces in order to estimate node mobility for any single-hop MMHC opportunity. On this basis, it takes local decisions on the sub-set of single-hop paths to activate. CM is a crucial component of the MMHC middleware because it has a direct and relevant impact on client channel decisions. It interacts with the underlying interfaces to change their configuration. Due to the criticality of the actions it performs, CM cannot be directly configured by a single application: in fact, one application may be selfish and require always the maximum connectivity performance at the expense of other applications running at the same node. For these reasons, CM provides RM and any application with a limited set of channel possibilities, i.e., only with the channels that are considerable feasible for the whole client node, with “no risks” for other running applications. In order to correctly estimate whether a connector is suitable for establishing a channel, CM has to gather and consider many client-related context data, since channel realization may affect the capabilities of the whole mobile client. In first approximation, CM determines the set of single-hop paths to activate based on durability estimation inferred by mutual distance monitoring: if several one-hop-distant devices have durability estimation values compliant with application requirements, CM prioritizes APs and BSs (see [1] for additional details on MMHC connector evaluation and selection schemas).

By delving into finer details, CM is in charge of:

1) discovering new connectors for all the supported wireless interfaces;
2) evaluating their suitability degree;
3) performing layer2 connections with the subset of selected connectors;
4) configuring layer3 parameters of established connections via DHCP.

The implementation of steps 1 and 3 strictly depends on the adopted wireless technology, e.g., inquiry procedure and PAN connection for Bluetooth and AP scan and association for IEEE 802.11, while steps 2 and 4 are independent from it.

3.3 Routing Manager (RM)

RM is in charge of establishing, controlling, and updating heterogeneous multi-hop channels by properly managing routing rules at runtime. It works to send/collect information/requirements on path suitability to/from collaborating nodes with the goal of estimating path durability (based on both node mobility end energy availability) and throughput. It interacts with CM to get the set of activated single-hop connections. When notified of a single-hop path disruption, it autonomously changes routing rules. In addition, routing rules are updated in an on-demand way anytime a new device becomes available or there is the need for a path renegotiation, e.g., because a path goes below the negotiated thresholds for expected throughput.

Since the main goal of RM is to modify paths based on local information provided by CM and remote information from collaborating nodes, the RM implementation does not depend on any particular wireless technology. In fact, RM:

1) interacts with one-hop distant nodes to send/receive context information;
2) evaluates the suitability of available single-hop links to form activated multi-hop and possibly heterogeneous channels;
3) changes routing rules to force the creation/update of valuable multi-hop paths.

By specifically focusing on Bluetooth exploitation in our MMHC middleware, let us stress that RM can interconnect multiple piconets together to form the dynamically needed multi-hop path and does not work to create a single multi-hop scatternet. In fact, while piconet-based multi-hop connectivity may introduce slightly additional overhead, that is greatly counterbalanced by a notable improvement in terms of proper dynamic selection of the most suitable connectivity opportunity depending on high-level context information. In addition, in this way MMHC can manage Bluetooth and IEEE 802.11 single-hop links in a homogeneous manner, with relevant advantages in terms of middleware/application development.

4 Implementation Insights and Performance Considerations

Based on our in-the-field experience with the creation and management of Bluetooth-based multi-hop paths, we have observed that the complexity of the middleware tasks needed for MMHC usually varies if they have to exploit basic mechanisms offered by either the operating system or the wireless technology.

First of all, middleware operations that do not depend on the local operating system and on the exploited wireless interface have minimum impact on total overhead and on the MMHC performance. For instance, the dynamic evaluation of remote Bluetooth connectors simply depends on quantitative data related to context information: CM spends about 120ms to evaluate a set of 5 connectors (further details on the adopted quantitative indicators are in [1]).

On the contrary, the operating-system-dependent middleware tasks are generally more complex to design and implement because they should perform equivalent but different actions on different execution environments. For instance, to create a new multi-hop Bluetooth path on Linux nodes, RM should perform the commands reported in Fig. 3, where `interfNameInt` is the name of the local interface offering connectivity and `interfNameExt` is the name of the local interface connected to the next hop of the path (an infrastructure connector in case of single-hop paths, a peer connector in case of multi-hop paths). In particular, lines 1-6 enable incoming, outgoing, and traversing packets to exploit `interfNameInt` and `interfNameExt` interfaces. It is the last line that actually creates the multi-hop path, by changing the routing rules to forward packets coming from the remote client with `clientIP` address connected via the `interfNameInt` interface to the external `interfNameExt` one.

```
iptables -A INPUT -i interfNameInt -j ACCEPT
iptables -A INPUT -i interfNameExt -j ACCEPT
iptables -A FORWARD -i interfNameInt -o interfNameExt -j ACCEPT
iptables -A FORWARD -o interfNameInt -i interfNameExt -j ACCEPT")
iptables -A OUTPUT -o interfNameInt -j ACCEPT
iptables -A OUTPUT -o interfNameExt -j ACCEPT"

iptables -t nat -A POSTROUTING -o interfNameExt -s clientIP -j MASQUERADE
```

Fig. 3. Example of RM commands on Linux to create a new Bluetooth-based multi-hop path

On the opposite, on MSWindows nodes, RM should perform the command

```
ROUTE ADD destination_ip MASK 255.255.255.0 gateway IF interface_number
```

where `destination_ip` is the destination network address, `gateway` the peer connector address, `interface_number` the identifier of the interface according to the MS Windows interface ordering. Anyway, despite the heterogeneity-related complexity and the need to perform different actions on different operating system, multi-hop path creation is not the most relevant source of overhead in MMHC: for instance, on Linux nodes, RM takes less than 300ms to create a new multi-hop path [3] by exploiting the commands in Fig. 3.

Instead, the MMHC tasks that significantly depend on specific features of the exploited wireless technologies may impose the most relevant delays. In particular, we have already shown in a previous paper [3] that the performance results achieved by MMHC when establishing new Bluetooth single-hop links are mainly the result of overheads due to the discovery of remote Bluetooth connectors and to PAN connection. The former's latency is about 11s (10.24s for the inquiry procedure, the remainder because of process initialization and result parsing), the latter more than 3s. It is worth noting that this relevant delay greatly limits (indeed, it is the primary bottleneck) the MMHC capability to quickly react to connectivity changes in Bluetooth-based deployment environments. This is particularly true at system startup and

whenever in-use connectors become abruptly unavailable, because there is the need to wait for a whole inquiry procedure before evaluating and connecting to new connectors. In other words, this heavy overhead significantly limits the degree of node mobility/dynamicity that MMHC can support when exploiting Bluetooth.

While it is impossible to shorten the PAN connection phase without modifying the Bluetooth standard, the latency of the inquiry procedure can be relevantly reduced at the potential cost of not discovering all available connectors. Peterson et al. [10] have demonstrated that it is possible to shorten the Bluetooth inquiry of 50%, by assuming the risk of not discovering, on the average, less than 1% of the totally available Bluetooth devices. We have followed a similar approach and designed CM to minimize the latency in creating single-hop Bluetooth connections: our CM implementation adopts a quick discovery procedure (with halved inquiry time according to Peterson's guidelines) at system startup and whenever there are no Bluetooth connectors available; otherwise, when latency requirements are less stringent, a "traditional" discovery procedure is exploited to ensure the completeness of the set of discovered connectors.

NIP exploits the JSR-82 APIs standard to access Bluetooth devices and, through the invocation of these APIs, CM implements the optimized inquiry in a portable way. However, as better detailed in the following section, even if the JSR-82 adoption permits to achieve relevant advantages in terms of portability, the JSR-82 APIs are insufficient to support some crucial features of the MMHC middleware, such as BNEP networking and RSSI monitoring, thus calling for a portable extension of both the JSR-82 specification and its reference implementation. Finally, let us observe that the JSR-82 APIs are a relevant support for NIP and for its rapid deployment over off-the-shelf equipment, but cannot be in place of it. In fact, JSR-82 provides a homogeneous access only to Bluetooth devices (operating system and driver independency), while NIP supports homogeneous access to a set of heterogeneous wireless technologies, by supporting common features independently of the exploited wireless interface. In other words, in the MMHC architecture, NIP exploits JSR-82 as much as possible to access Bluetooth devices portably, while CM exploits NIP to access any heterogeneous interface available on MMHC nodes in an interface-independent way.

5 The MMHC Extensions to the JSR-82 APIs

The JSR-82 Java APIs for Bluetooth has the purpose of providing Java developers with homogeneous access to Bluetooth features, transparently with regards to underlying operating systems, Bluetooth drivers, and card implementors. For instance, JSR-82 portably supports the discovery of remote devices, the browsing of services offered by discovered devices, and the creation of RFCOMM/L2CAP/OBEX connections. In addition, by exploiting the mechanisms provided by JSR-82, it is possible to reduce the latency of the inquiry procedure (see the `DiscoveryAgent` class and its `startInquiry()` and `cancelInquiry()` methods).

To the purpose of easy and runtime portability, our NIP wrapper exploits the JSR-82 APIs to access Bluetooth devices as much as possible. In particular, it exploits the JSR-82 implementation by BlueCove [11], an open-source project for the Mac OS X, Linux, and MS Windows operating systems; over MS Windows, BlueCove supports multiple drivers, such as Widcomm, BlueSoleil, and the ones compliant with the MS Bluetooth stack available starting from Windows XP SP2. However, as already

pointed out, JSR-82 cannot provide some features that are central for MMHC working and, for that reason, we have decided to significantly extend its capabilities as detailed in the following.

First of all, JSR-82 does not provide any support for the BNEP protocol, which is extremely suitable to setup an IP-based network via DHCP in an Ethernet-like style. In fact, the JSR-82 specification is mainly oriented on the simple provisioning of communication channels for data exchange. It does not support, instead, the possibility of setting up networking capabilities via the discovery/pairing of remote devices and the explicit successive establishment of higher-layer channels, e.g., standard TCP/IP sockets exploited by application-level clients and servers. Secondly, JSR-82 does not provide any facility to gather RSSI values describing the quality of active connections. Instead, the periodic monitoring of RSSI values is crucial for CM to correctly evaluate the mobility degree of candidate connectors [12].

Given the above limitations, we have worked to extend the JSR-82 APIs and reference implementation (on both Linux and MS Windows XP/Vista[2]) primarily to provide BNEP and RSSI features. We have implemented our extensions by properly modifying the Bluecove project source code; on Linux we have exploited BlueZ, its official Bluetooth protocol stack; on MS Windows we have focused on Widcomm drivers for Bluetooth.

In particular, by referring to the functionality supported by NIP, the *Perform as peer connector* feature requires setting up a Group ad-hoc Network (PAN GN) [13] and activating a DHCP server to automatically configure client-side network parameters. Analogously, *Connect to a connector* requires accessing to a remote PAN GN network and then activating the DHCP client.

To that purpose, we have extended the JSR-82 APIs by adding a novel `BNEPConnector` class: it supports both server- and client-side BNEP capabilities via the `server()` and `client(String remote_addr)` methods, respectively exploited by *Perform as peer connector* and *Connect to a connector*. The `server()` method is provided on Linux nodes based on the exploitation of the BlueZ `pand` command:

```
pand -i hciX --listen --role GN --devup ./devup.sh --master
```

where `hciX` is the identifier of the local Bluetooth device and `devup.sh` is a script that activates a DHCP server whenever a new remote device connects. Note that it is impossible to activate the DHCP server before the execution of the BlueZ `pand` command because BlueZ creates a new `bnepX` virtual interface only after actual connection establishment. For this reason, in this case the DHCP server is started only after connection establishment in MMHC, by introducing an additional delay when performing new Bluetooth connections. The `server()` method is not currently implemented on MS Windows because the Widcomm driver does not support the setup of a PAN server. In addition, MS Windows XP/Vista does not provide a standardized command to configure and start a DHCP server from MMHC, forcing to rely on additional manual configurations (a command-line and standardized DHCP server is available only on MS Windows Server 2008). Therefore, the current implementation of MMHC supports *Perform as peer connector* only for Linux machines, while MS Windows nodes can only behave as clients.

[2] We are currently working on the prototype of the extended JSR-82 implementation for MacOSX; that version is the only one not available yet on the MMHC Web site.

On Linux nodes, also the `client(String remote_addr)` method is provided by exploiting the BlueZ `pand` command but in this case with PAN User role:

```
pand -i hciX --connect remote_addr --role PANU --service GN
```

where `remote_addr` is the Bluetooth address of the remote device offering the PAN GN network. Once connected, NIP activates the DHCP client via `dhclient bnepX` where `bnepX` is the virtual interface created at PAN connection establishment. On MS Windows the `client(String remote_addr)` method, instead, exploits the Widcomm function:

```
LAP_RETURN_CODE CreateConnection(
     BD_ADDR bda,
     GUID guid,
     CSdpDiscoberyRec &sdp_rec
)
```

where `bda` represents the Bluetooth address of the remote device and `guid` the PAN type (set to `CBtlf::guid_SERVCLASS_GN` for PAN GN networks). Once connected, NIP performs `ipconfig /renew *` to properly update the related IP parameters via DHCP.

The *Get available connectors* function implemented in our NIP prototype has two operating modes: a basic one performing an entire inquiry procedure and another one reducing the discovery latency by properly invoking `startInquiry()` and `cancelInquiry()` methods of the JSR-82 `DiscoveryAgent` class according to Peterson's proposal. Moreover, *Get available connectors* requires gathering not only the list of available remote Bluetooth devices, but also their RSSI values. To this purpose we provide an implementation of the JSR-82 `DiscoveryListener` listener which, whenever a new remote device is discovered, i) performs a Baseband connection, ii) gathers the corresponding RSSI value and iii) provides it to the upper layers (to the requesting middleware and/or applications). Note that the establishment of the Baseband connection is necessary because in Bluetooth there is visibility of RSSI indications only after device connection. In particular, our MMHC prototype exploits, on Linux nodes:

```
hcitool -i hciX cc remote_addr
hcitool -i hciX rssi remote_addr
```

to respectively get a Baseband connection (not requiring PIN authentication) and to gather RSSI values. Instead, the implementation for MS Windows nodes exploits:

```
BOND_RETURN_CODE Bond(BD_ADDR bda, BT_CHAR *pin_code);
BOOL GetConnectionStats(BD_ADDR bda, tBT_CONN_STATS *p_conn_stats);
```

where `pin_code` is a PIN code of at most 16 characters and `p_conn_stats` includes various data about the connections, such as their RSSI. Note that, differently from BlueZ, Widcomm does not currently provide the capability to perform Baseband connections; instead it supports the pairing of remote nodes via the `Bond` function, which provides Baseband-like connections with the additional requirement of typing PIN codes.

6 Conclusions

Our research efforts for the design/implementation of the MMHC prototype and our practical experience on its in-the-field deployment over multi-hop networks with off-the-shelf equipment demonstrate the feasibility of middleware solutions for

self-organizing MMHC solutions. By specifically focusing on Bluetooth multi-hop networking, this paper contributes to show the crucial importance of considering both client heterogeneity and wireless technology efficiency to support MMHC challenging scenarios in a feasible, effective, and easily deployable way. Our novel middleware solution exploits the JSR-82 APIs to access Bluetooth devices in an efficient and portable manner by realizing an enhanced JSR-82 version to fully support BNEP networking and RSSI gathering. The current prototype fully supports Linux nodes, while implementation limitations of Widcomm drivers allow MS Windows nodes to behave only as MMHC clients.

The encouraging results that we have already obtained are stimulating our further research work in the field. In particular, we are working on lightweight and completely decentralized models for trust management and incentives to encourage the collaborative peer offering of MMHC opportunities in a completely open and dynamic execution environment.

References

1. Bellavista, P., Corradi, A., Giannelli, C.: Context-aware Middleware for Reliable Multi-hop Multi-path Connectivity. In: 6th IFIP Work. on Software Technologies for Future Embedded & Ubiquitous Systems (SEUS 2008), Anacapri, Italy (October 2008)
2. Bellavista, P., Corradi, A., Giannelli, C.: A Layered Infrastructure for Mobility-Aware Best Connectivity in the Heterogeneous Wireless Internet. In: 1st Int. Conf. on MOBILe Wireless MiddleWARE, Operating Systems, and Applications (Mobilware 2008), Innsbruck, Austria (February 2008)
3. Bellavista, P., Corradi, A., Giannelli, C.: Mobility-aware Middleware for Self-Organizing Heterogeneous Networks with Multi-hop Multi-path Connectivity. IEEE Wireless Communications Magazine 15(6), 22–30 (2008)
4. JSR-82 Bluetooth API, `http://java.sun.com/javame/reference/apis/jsr082/`
5. Camp, J., Knightly, E.: The IEEE 802.11s Extended Service Set Mesh Networking Standard. IEEE Communications Magazine 46(8), 120–126 (2008)
6. Wei, H.-Y., Gitlin, R.D.: Two-hop-relay Architecture for Next-Generation WWAN/WLAN Integration. IEEE Wireless Communications 11(2), 24–30 (2004)
7. Luo, H., Ramjee, R., Sinha, P., Li, L.E., Lu, S.: UCAN: a Unified Cellular and Ad-hoc Network Architecture. In: 9th Int. Conf. Mobile Computing and Networking, San Diego, CA, September 2003, pp. 353–367 (2003)
8. Kang, S.-S., Mutka, M.W.: A Mobile Peer-to-peer Approach for Multimedia Content Sharing using 3G/WLAN Dual Mode Channels. Wiley Journal on Wireless Communications and Mobile Computing 5(6), 633–645 (2005)
9. Fu, C., Khendek, F., Glitho, R.: Signaling for Multi-media Conferencing in 4G: the Case of Integrated 3G/MANETs. IEEE Communications Magazine 44(8), 90–99 (2006)
10. Peterson, B.S., Baldwin, R.O., Kharoufeh, J.P.: Bluetooth Inquiry Time Characterization and Selection. IEEE Trans. on Mobile Computing 5(9), 1173–1187 (2006)
11. BlueCove JSR-82 Project, `http://www.bluecove.org/`
12. Bellavista, P., Corradi, A., Giannelli, C.: Mobility-Aware Connectivity for Seamless Multimedia Delivery in the Heterogeneous Wireless Internet. In: 2nd Work. on multiMedia Applications over Wireless Networks (MediaWiN 2007), Aveiro, Portugal (July 2007)
13. Bluetooth Profile Specifications, Personal Area Networking Profile (February 2003), `http://www.bluetooth.com/`