Per-application Mobility Management: Performance Evaluation of the UPMT Solution

Marco Bonola
University of Rome "Tor Vergata"
Rome, Italy
marco.bonola@uniroma2.it

Stefano Salsano
University of Rome "Tor Vergata"
Rome, Italy
stefano.salsano@uniroma2.it

Abstract— **In this paper, we provide the performance evaluation of the UPMT (Universal Per-application Mobility management using Tunnels) solution. UPMT offers per-application mobility management, i.e. the capability of separately taking handover decisions for each application. UPMT supports legacy applications, private IP addressing/NATs and it is an overlay solution that does not require the access network to offer any specific support. We have implemented UPMT under Linux OS and made it available under the GPL Open Source license.**

Keywords: Mobility management, per-application mobility, vertical handovers, performance evaluation

I. INTRODUCTION[1]

Current notebooks, netbooks, and handheld devices are able to connect to several wireless (and wired) access technologies. The capability to move from one access technology to another one, switching also the active connections, is typically referred to as "Vertical Handover". The capability of roaming across different network access technologies automatically using the most suitable ones has been called "Always Best Connected" (ABC) service in [1]. In order to support Vertical Handover and ABC services in IP networks, countless solutions have been proposed in the literature and several have been considered in the IETF, see [2] for a survey.

Recently, the need to transport different applications on different access technologies at the same time, and take separate handover decisions for each application has been identified and addressed [3][4][5]. The motivation is that different applications have different requirements that can be mapped into the different characteristic of the access technologies (e.g. cost, QoS, security). For example, when moving from an office WiFi access to a public 3G networks it could be sensible to handover an ongoing voice call, but one could want to suspend connectivity to a background bulk file transfer.

In [4] we have proposed an application level solution called UPMT ("Universal Per-application Mobility management using Tunnels"). The solution was extended in [6] in order to address scalability issues. UPMT main features are:

- per-application independent vertical handovers: UPMT is able to independently direct the traffic of each application on the "best" network interface (even different flows of the same application can be controlled independently if needed)
- support of legacy applications, i.e. existing applications that are not aware of UPMT can use it
- support of legacy correspondent hosts, i.e. only one end of the communication needs to implement UPMT
- support of mobile terminals behind NAT (Network Address Translation) devices, i.e. using private IP addresses
- full compatibility with existing network infrastructure, UPMT does not require the access network to offer any support but plain IPv4 connectivity
- for scalability, UPMT supports a multiple anchor nodes scenario without requiring coordination between "Anchor Nodes" (AN)

Let us focus on the first two features, namely "per-application independent handovers" and "support of legacy applications". Combining these two features is really challenging as it means that the application cannot be upgraded and become "UPMT aware". If this were possible, an application could perform some actions to move the flows over the different interfaces or to react to the switch of a flow from an interface to another. On the other hand in our solution the applications do not need to be aware of UPMT and are able to work without even noticing that handovers are being performed. Of course our intent is not to rule out the possibility that future applications will become UPMT aware, integrating some mobility management features in their logic. We just want to guarantee that in the short/medium term all existing applications are able to exploit our mobility management solution without any change.

From the above considerations, it is clear that to combine "per-application independent handovers" and "support of legacy applications" the UPMT modules need to be able to intercept and properly manipulate the traffic directed to and coming from a legacy application on the mobile device, implementing the needed mobility management decisions. In this light, the UPMT modules that can perform flow manipulation need to be Operating System (OS) specific, while the "networking" aspects of UMPT (e.g. how packets are tunneled, how handovers are signaled) are independent from the OS.

[1] This work was supported in part by the EU under the project FP7 – 224024 "PERIMETER"

978-1-4244-9538-2/11/$26.00 ©2011 IEEE

II. UPMT BASICS

UPMT is a solution for mobility management over heterogeneous networks based on IP in UDP tunneling. The basic idea can be summarized as follows. A multi-homed MH establishes an IP in UDP tunnel for each active network interface with its UPMT "correspondent peer". This correspondent peer can be an "Anchor node", a correspondent Fixed Host or a correspondent Mobile Host and plays the role of "Tunnel server" as it provides the second end of the tunnels established by the MH.

Independently from the number of established tunnels, a virtual interface (viface) is brought up and configured with a virtual IP address (VIpA) in the MH. MH IP stack is configured in such a way that IP packets locally generated by applications "under UPMT control" are routed through the virtual interface and handled by UPMT protocol layer (e.g. by using "policy routing" features). The output tunnel is chosen by UPMT according to a "Per-Application Forwarding Table" (PAFT), a table that binds an application flow identified by the 5-tuple (protocol, IP source address, IP destination address, source port, destination port) with an output tunnel. The PAFT associates an application level "socket" to the tunnel used to send packets from the MH to the AN and from AN to MH. For each application packet sent over the viface, a PAFT look-up is performed. The packet is then encapsulated in the resulting IP in UDP tunnel and sent over the real network interface. With this approach, both IP readdressing of the underlying network interfaces and application flows handovers from one tunnel to another are hidden to the virtual interface's IP stack and application sockets. In the tunnel server, for each incoming packet received on a registered IP in UDP tunnel the PAFT is updated (if necessary) and reply packets are sent over the same tunnel. In case of downstream traffic, MH can update the PAFT of its tunnel server also by sending an explicit request.

UPMT design takes into account a multi AN scenario in which the MH is allowed to concurrently use more then one AN and independently choose the AN for any application flow under UPMT control. The virtual addresses management is fully decentralized and no cooperation between ANs is required. In [6] we proposed a solution in which each AN assigns a "locally unique" VIpA to MH. According to the AN to which a given tunnel is established, the MH performs local NAT so that still only one virtual viface is required regardless of the actual number of ANs concurrently used.

The AN provides the Mobile Host with a second level NAT service on the "Anchor-NAT", which is a key element in our mobility architecture (Figure 1). In fact, the idea is that the Mobile Host will access the Internet through the NAT in the Anchor Node, using a public IP address provided by the AN. Since a CH never sees the VIpA used by MH, communication with legacy CH is provided.

The MH connects to the AN using IP in UDP tunnels, one tunnel per each access network. The UPMT mobility management procedures allow the MH to select the tunnel towards the Anchor Node, if needed in a separate way for each application/flow to be supported.

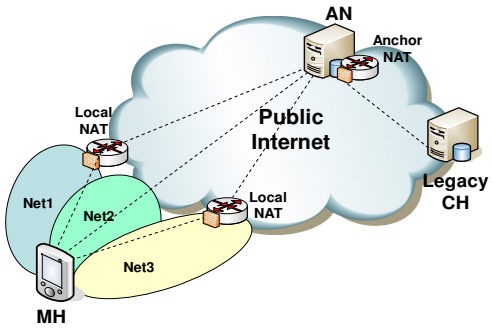

Figure 1. UPMT reference scenario

It is worth spending some words about the choice of IP/UDP encapsulation as tunneling method in UPMT, as opposed to other candidates like for example Generic Routing Encapsulation (GRE), L2TP, IP in IP. The pros of IP/UDP encapsulation are: (i) NAT traversal of UPMT data flows does not require any additional mechanism since UDP traverses all existing NAT implementations. (ii) There is no need to register a new IP protocol number to handle tunneled data. (iii) IP/UDP encapsulation requires less overhead[2] with respect to GRE.

The networking and signalling aspects of UMPT have been presented in [4]. The full details can be found in the technical report [7], including the procedures to establish the tunnels, to manage the PAFTs, to support the NAT functionality and to drive the handover of application flows across the tunnels.

III. RELATED WORK

MIPv4 [8] is the networking level mobility solution for IPv4 standardized within IETF. MIPv4 requires support by the Foreign Network (FH) access router (Foreign Agent - FA). This is a major limitation for the deployment of ubiquitous ABC services in the short/ medium term, as all access networks visited by the MH must support MIPv4. Moreover, support for "IP flow binding" [8] (required to build a per-application mobility management infrastructure) is still at early stage of standardization process.

MIPv6 [9] benefits from both the previous work on MIPv4 and from some features offered by IPv6 itself. Even though MIPv6 shares many features with UPMT and fulfills a number of requirements (e.g. multiple CoAs [10], flow binding update [11]), IPv6 connectivity cannot be given as granted for most Internet Service Providers. Again, in the short/ medium term MIPv6 mobility solutions for ABC are limited by the access and transport network infrastructure.

HIP [12] seems to be more suited for ABC services. HIP is totally end-to-end but also defines mechanisms for rendezvous services and host discovery. HIP works with NAT and with all existing network infrastructure. Even though HIP

[2] IP/GRE encapsulation with the KEY option requires 20 + 16 B against 20 + 8 B of IP/UDP. (The key option is required to multiplex tunnels on a single address without inspecting the inner packet).

specifications do not support a per-application mobility management, quite recently a per-application mobility management solution based on HIP has been proposed in [5]. This solution has only been simulated using a network simulator, and no real implementation has been developed.

In [3] a per-application mobility management platform was first proposed. The solution is based on address translations on the two endpoints, and it requires that both endpoints are "mobility-aware". The interoperability with NATs is not discussed and the solution is evaluated by simulation, no real implementation is reported.

IV. UPMT IMPLEMENTATION DESIGN

In this section we focus on some architectural aspects of our UMPT implementation running in the Mobile Host and in the Anchor Node. In particular, we will consider two aspects: i) the implementation of tunnelling mechanisms (encapsulation and de-capsulation of packets); ii) the handling of the flows without active cooperation of the applications and without linking ad-hoc modified socket libraries. The former aspect (tunnelling) is common to both MH and AN, the latter (flows handling) specifically concerns the MH. Figure 2 and Figure 3 show the architecture of our Linux based UPMT implementation respectively for the Anchor Node and for the Mobile Host.

As shown in the figures, the realized Linux implementation is composed by user space and kernel space components. The UPMT implementation architecture in the Anchor Node is a subset of the one in the Mobile Host. The common modules are the UPMT tunnelling, the PAFT (per Application Forwarding Table) and the UPMT Configu-ration Tool.

The PAFT works at the level of the single flow, i.e. identified by the 5-tuple (protocol, IP src, IP dst, src port, dst port), storing the correspondence between flows and tunnels, both in the MH and in AN.

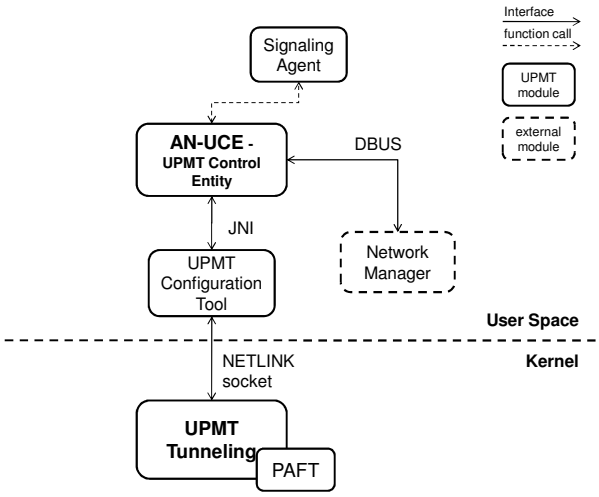

Figure 2. Linux UPMT modules on Anchor Node

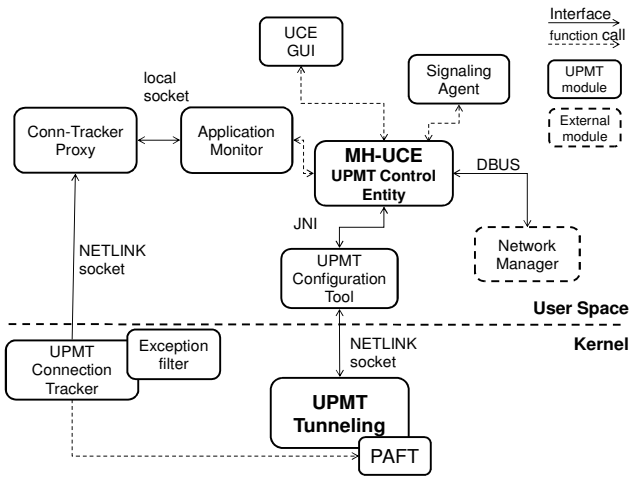

Figure 3. Linux UMPT modules on Mobile Host

The UPMT tunnelling component and the PAFT are implemented in kernel space for maximum performance. This means that the UDP flows used for tunnelling data between the AN and the MH are handled within the kernel: the UDP packets received from these tunnels are not transferred to a user space application as a "normal" UDP socket would do. Likewise, the packets outgoing on these UDP flows are not received from a user space application through a socket, but are generated within the kernel that "intercepts" the proper IP packets in a layer 2 virtual interface and encapsulate them into the UDP tunnel. As shown in Figure 4, this is different from other existing IP in UDP tunnelling solution (such as OpenVPN [13]) which operates the encapsulation and de-capsulation in user space, by creating "normal" UDP sockets. In these user space solutions, an outgoing packet is transferred by an application to the Kernel, then the kernel intercepts it and sends it to the user space tunnelling, which encapsulate it and sends it again to the kernel using a UDP socket. This approach is clearly un-efficient, as we will be able to show with performance measurements.

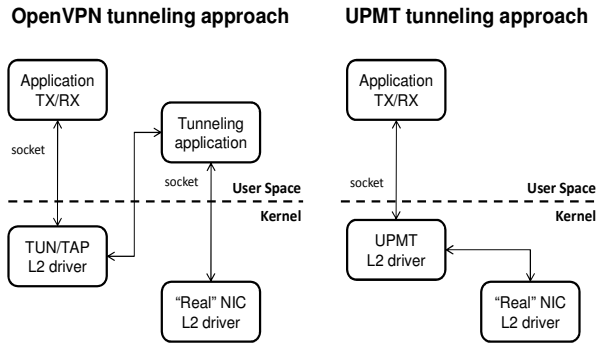

Figure 4. Tunneling approach comparision

The advantage of a user space solution is that it does not require kernel modifications. Actually, we implemented a first prototype of UPMT using a user space tunnelling module. We discarded it, when we noticed the performance limitations and we realized that we needed in any case modification to the

kernel in order to provide per-application handover supporting legacy applications.

The UMPT tunnelling module communicates with the user space UPMT Configuration module using the so-called "Netlink Sockets" of the Linux OS. The UPMT Configuration module is used by the coordinating entities of UPMT, called UPTM Control Entities (MH-UCE and AN-UCE for Mobile Host and Anchor Node respectively). The UPMT Control Entities run in user space, they sends commands to the UPMT tunnelling kernel module as needed to configure the PAFT (Per-Application Forwarding Table) and to drive the handover process.

In the Mobile Host, the UCE module is complemented with a Graphical User Interface that can be used for user interaction, especially useful for testbed and demo purposes. The UCE and the associated GUI are written in Java. On the MH, the UCE GUI presents the user the list of available network interfaces, the list of active application and the list of the sockets open by each applications. The user is able to decide the interface (i.e. the tunnel) to be used for an application or even for each single socket of each application, "manually" controlling the handovers. The UCE is also able to work autonomously according to a set of configurable policies. A configuration file is now used to configure the per-application policies. Current policies are simple (e.g. a priority list of interfaces to be used when available), it is for further study to define and implement more complex policies, e.g. based on performance measurements, performance estimates, costs and so on. On the MH, the UCE receives notifications about the presence and the status of network interfaces from the Network Manager [15] and it communicates with remote UPMT entities (e.g. the Anchor Node) using the UPMT Signalling Agent.

It is worth to discuss how it is possible to populate the PAFT on the Mobile Host, adding the sockets that are used by the applications. Assume for example that at a given time there is the policy to send all sockets of application X through tunnel M. It is relatively easy to retrieve the list of all the active sockets for the application X, and it would be easy to add a set of PAFT entries pointing each socket into tunnel M (and this could be repeated periodically with a sort of polling mechanism). The problem is that moving the sockets into the tunnels would break the sockets itself as the packets would be NATed at the exit of the tunnels in the Anchor Node. Therefore these sockets must be inserted in the tunnel M from the very first packet of the flow. The solution is to use the conntrack [16] facilities in the Linux Netfilter framework [17] which allows to "capture" the first packets of each socket. Once a "first packet of a socket" is captured, our module checks the application to which the socket belongs, decides[3] if the packet has to be handled by UPMT and, if so, it decides which tunnel must be used and sets the PAFT accordingly. Then the packet continues its "regular" processing, so that the PAFT entry will be used to decide on which tunnel to send it.

[3]Once the first packet is captured, policies implementation is straightforward, and can be based for example on application name, protocol, destination address etc.

Strange as it may sound, there is no simple facility in the Linux kernel to understand which application (e.g. which Process ID or "PID") has produced a packet from the data structure (called skbuf) that contains the packet itself, which only carries a socket identifier or "file descriptor". Associating a file descriptor with a Process ID is possible in user space by browsing the information contained in the /proc file system (recursively reading all process folders in /proc and looking for the file descriptor that is associated to the packet in the skbuf). It is not possible to perform the same interrogation from kernel space, so one could think to (1) "stop" the packet; (2) ask some software module responsible for reading the /proc to find the process that opened a given socket; (3) buffering all related packets till the response is received (because sleeping is not allowed in this particular kernel context). This is clearly not optimal, therefore we decided to extend the skbuf structure in the Linux kernel to transport the application PID. This way, when processing a packet in the networking stack of the kernel, it is easy to get the PID of the owner application, at the prize of the need to recompile the kernel to extend the skbuf structure.

V. TESTBED AND PERFORMANCE EVALUATION

We have implemented the proposed UPMT solution under Linux OS. The implementation is available as open source at [18]. In the following subsections we provide a performance evaluation of the UPMT implementation. In Section V.A we deal with the impact of handovers on "user level" performances like throughput. In Section V.B we assess the performances of the realized implementation of the Anchor Node and of the Mobile Host with respect to the processing load. In both cases we followed an experimental approach by performing measurements on the actual implementation.

A. User Level Performance Analysis

We setup a test-bed where we could show separate handovers of different applications over three access networks active at the same time (Ethernet, WLAN and 3G access provided by a commercial operator). To the best of our knowledge, our implementation is the only one performing per-application independent handover in a real test bed. We monitored the execution time of the handover by capturing packet traces with the tcpdump tool. The results show that there is virtually no delay in the handover execution, the difference in the time stamps of the received packets is only due to the differential network delay among the interfaces and that there is no packet loss. The test-bed consists of the following components:

1. an anchor node *AN* at public IP address 160.80.103.66. *AN* is a PC with Pentium M 1.2 GHz processor, 1.256 GB RAM and Linux kernel 2.6.35.4-upmt.
2. a mobile host *MH* with three network interfaces. *MH* is a laptop with Linux kernel 2.6.35.4-upmt, Core 2 Duo 1.83 GHz, 2 GB RAM. *MH* is equipped with 3 NICs: (i) *wlan0,* a wireless 802.11g NIC connected to a AP on the LAN 192.168.100.0/24; (ii) *eth0,* a Ethernet 100Mbs NIC on the LAN 192.168.100.0/24; (iii) *ppp0,* a HSPA USB card connected on a PPP link and IP address 95.75.196.58. In

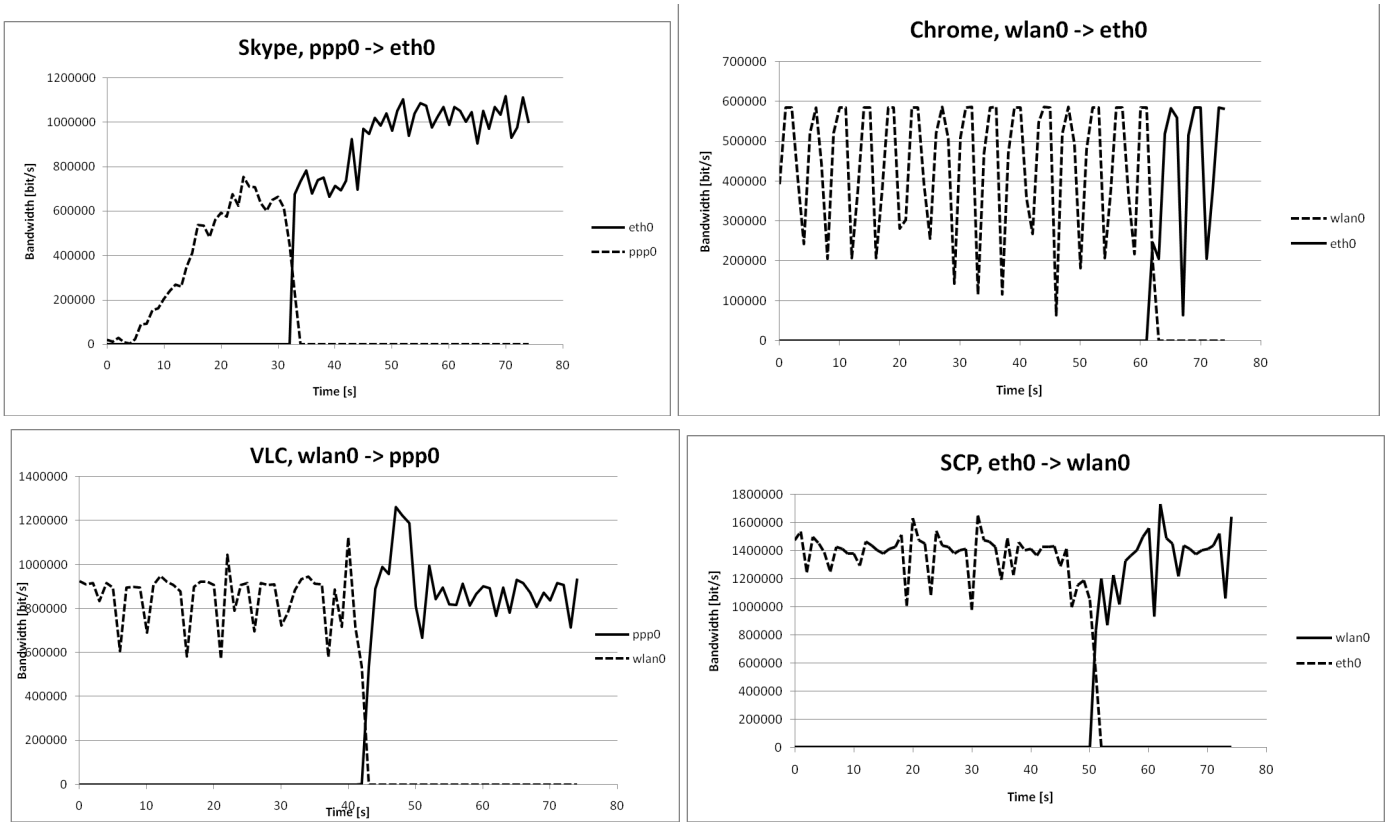

Figure 5 User level performance evaluation

addition *MH* has a UPMT virtual interface has a fixed virtual IP address *VIpA_fix* 5.6.7.8.

3. a number of legacy correspondent hosts placed over the Internet. In particular, along the demonstration time, MH will connect to a WEB server *CH1* at *www.torvergata.tv*, a backup file server *CH2* at *ubfsrl.ath.cx*; a streaming server *CH3* at *vipnrj.yacast.net;* a FTP server *CH4* at *ftp.archlinux.org* and a laptop *CH5* placed in another LAN running a skype client.

The demonstration described in the reminder of this section is intended to show UPMT capability of performing per-application independent handovers. After initial association and tunnel establishment, five applications are started in the same sequence as they are listed in Table 1. Four applications are put under UPMT control; one application (firefox) is not handled by UPMT and is routed through eth0 for the whole demo time. The initial interface for each application is the first interface in the "Handover" column. The applications under UPMT control are handed over as described in Table 1, in the same sequence as they are listed.

We monitored the execution time of the handover by capturing packet traces with the tcpdump tool. The results are reported in Figure 5. Each sub-figure shows for each application the sum of received and transmitted bits-per-second on the first and the second network interfaces used over the demo time (the initial period in which the application are sequentially launched has been cut). The trace related to firefox traffic (no UPMT) has been omitted.

Table 1 Application list in the reported experiment

App	Activity	Traffic	Handover	CH
skype	video conference	RT (UDP) video/audio + control	ppp0→eth0	CH5 + al.
vlc	streaming	MMS (TCP)	wlan0→ppp0	CH3
scp	backup	SSH (TCP)	eth0→wlan0	CH2
chrome	ftp download	FTP (TCP)	wlan0→eth0	CH4
firefox	streaming	HTTP (TCP)	eth0 (no UPMT)	CH1

Further details of the measurements are available at [7]. We just note that the results are fully satisfactory: (i) there is virtually no delay in the handover execution, the difference in the time stamps of the received packets is only due to the differential network delay among the interfaces; (ii) we have no packet loss and indeed, the sum of the bit-rate for the two interfaces around the handover instant keeps the aggregate bandwidth in trend; (iii) the oscillation visible on the traces are not caused by UPMT tunnelling, but they are caused by network loss and TCP congestion control, as we observed identical behaviour without UPMT. In conclusion we want to underline that the current UPMT implementation already provides "break-before-make" handovers and per-socket handovers, which was not possible to show here.

B. System Level Performance Analysis

In this section we want to analyze the processing load performances of the proposed solution, both on the anchor nodes and on the mobile devices. For different reasons, it is important that the processing burden is as limited as possible in both cases. On the anchor nodes, the reason is that the scalability of the solution (i.e. how many concurrent flows and concurrent users can be supported) depends on the processing load.

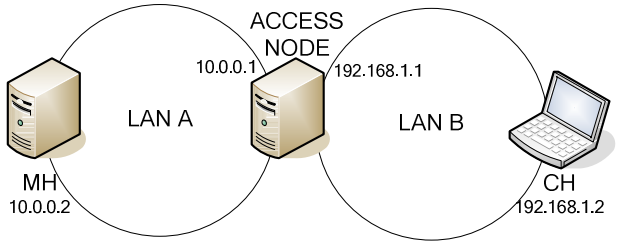

Figure 6 Test-bed for System Level performance analysis

On the mobile devices an excessive processing load due to the mobility management can limit the performance of the applications, considering the limited processor capacity. Moreover the processing load has an impact on the duration of the batteries. We show that the proposed tunneling mechanism implemented in kernel space is very efficient with respect to the processing load.

We used an experimental methodology, both for the analysis of Anchor Node processing load and for the Mobile Host processing load. Figure 6 and Table 2 respectively show the networking architecture and the hardware used in the test-bed. We compared three solutions, as shown in Figure 7: i) sending packets without UMPT tunneling (i.e. only using NAT functionality in the AN), which is the "reference" solution (denoted as *plain-NAT*); ii) sending packets using UMPT tunneling (denoted as *UPMT*); iii) sending packets using a user-level tunneling mechanism, in particular OpenVPN [13] with null ciphering.

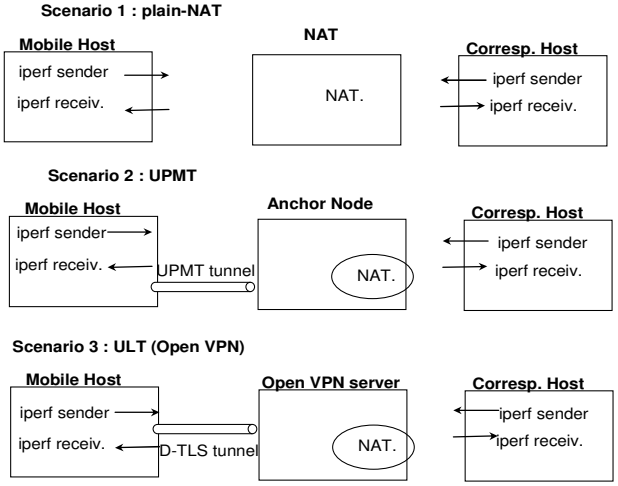

Figure 7:. System Level performance evaluation scenarios

As for the Anchor Node, we produced synthetic flows with the iperf traffic generation tool using two external hosts and analyzed the behavior of the AN, in terms of packet loss and CPU processing load (measured using the mpstat Linux tool). We were able to estimate the CPU saturation load of the AN by increasing the generated traffic. Figure 8 shows the packet loss due to CPU overload in the Anchor Node in the three scenarios. On the x axis the input packet rate (packets/s) is reported. For each scenario it is possible to identify a "saturation load": when the input traffic is lower than the saturation load there is no packet loss, when the input traffic is higher than the saturation load, the excess traffic is dropped. The saturation load for UPMT (around 60.000 pkt/s) is not much lower that the one for plain-NAT (70.000 pkt/s). This shows that the additional processing load due to encapsulation/de-capsulation has limited impact on the scalability of the Anchor Node. On the other hand, the saturation load in the user level tunneling (OpenVPN) is much lower (around 20.000 pkt/s). This is due to the inefficiency of performing tunneling in the user space. The result is confirmed by the analysis of CPU load for the same range of input traffic (Figure 9). When the input traffic rate grows and approaches the saturation load, the CPU load grows and approaches 100%. From the saturation load onward, the CPU load remains at 100%.

Table 2 – Hardware used in the experiments

Mobile Host (iperf source and receiver)	PC Intel Core 2 Quad 2.66 GHz, NIC Ethernet 100 Mb/s
Access Node	PC VIA C7-D Processor 1.5 GHz, 2 NICs Ethernet 100 Mb/s
Correspondent Host (iperf source and receiver)	PC Intel Core 2 Duo 1.83 GHz, NIC Ethernet 100 Mb/s

As for the Mobile Device processing load, we considered the iperf application generating packets at different rates. It should represent a generic application that is producing a packet stream. We measured the overall processing load of the device under the three different networking scenarios (plain-NAT, UPMT and Open-vpn) in order to evaluate the impact of our tunneling mechanism and to compare it with that of a user-space tunneling mechanism. Figure 10 shows that, as in the Access Node experiments, the difference between the plain-NAT (no tunneling) and the UPMT (kernel-based tunneling) is limited. In this case the relative difference is even smaller, as the CPU load is dominated by the user-space operations performed by the application and in particular by system-calls needed to relay the data from user-space to kernel-space (transmitting side) and vice-versa (receiving side). On the other hand, the difference between the plain-NAT and the Open-vpn processing load is relatively high.

VI. CONCLUSIONS

We remark that to the best of our knowledge, we provided the first implementation of per-application mobility management in real devices. We have performed some user level and system level performance analysis on the real implementation. As for user level performance, we measured the throughput of

different applications across the UPMT handover procedures and found that the impairment is negligible. As for the system level performances, the analysis shows that the processing penalty incurred by the tunneling operation is limited. This is achieved thanks to the kernel based implementation, as a comparable user space tunneling solution shows much higher processing requirements.

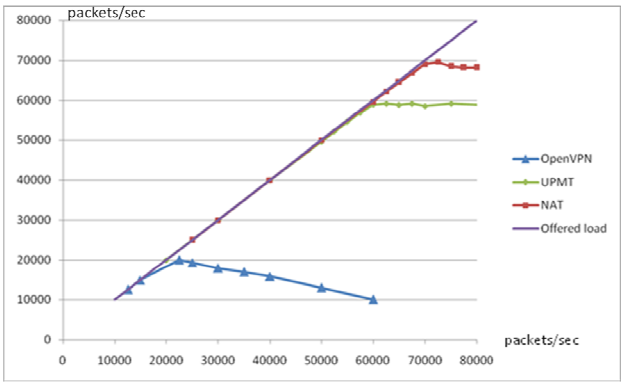

Figure 8. Anchor Node packet loss comparison

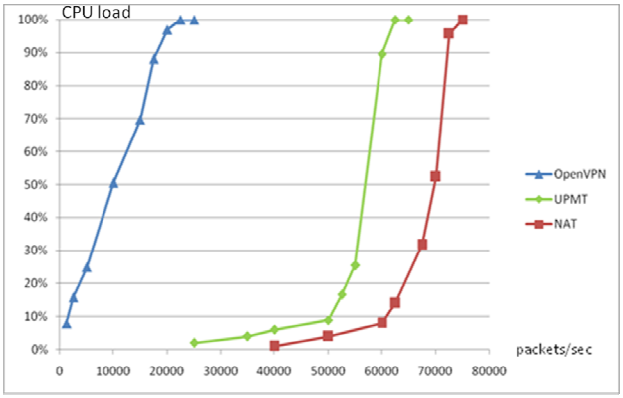

Figure 9. Anchor node CPU load comparison

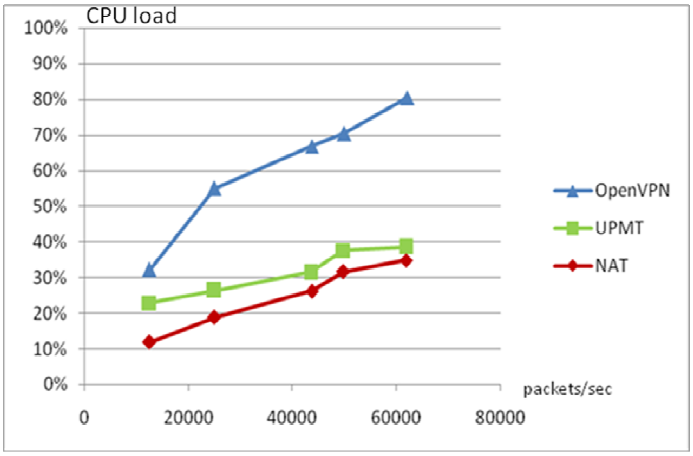

Figure 10. Mobile Device CPU load comparison

ACKNOWLEDGMENT

We would like to thank Alessio Bianchi, Andrea Gambitta, Fabio Patriarca, Enrico Gagliano, Andrea Capitani, Fabio Ludovici, for their contribution to UPMT development.

VII. REFERENCES

[1] E. Gustafsson et al. “Always Best Connected”, IEEE Wireless Communications, Feb 2003.

[2] Deguang Le, Xiaoming Fu, Dieter Hogrere, “A Review of Mobility Support Paradigms for the Internet”, IEEE Communications surveys, 1s t quarter 2006, Volume 8, No. 1

[3] M. Chang, H. Lee, M. Lee, "A per-application mobility management platform for application-specific handover decision in overlay networks", Computer Networks, Volume 53, Issue 11, July 2009

[4] M. Bonola, S. Salsano, A. Polidoro, “UPMT: Universal Per-Application Mobility Management using Tunnels”, IEEE GLOBECOM 2009

[5] L. Bokor, L. T. Zeke, S. Nováczki. G. Jeney, “Protocol design and analysis of a HIP-based per-application mobility management platform”, Proceedings of the 7th ACM international symposium on Mobility management and wireless access, 2009, Tenerife, Spain

[6] M. Bonola, S. Salsano, “Achieving Scalability in the UPMT Mobility Management Solution”, Future Internet and Mobile Summit 2010, 16 - 18 June 2010, Florence, Italy

[7] S. Salsano, M. Bonola et al., “The UPMT solution (Universal Per-application Mobility Management using Tunnels)”, technical report available at http://netgroup.uniroma2.it/TR/UPMT.pdf

[8] C. Perkins, “IP Mobility Support for IPv4”, IETF RFC 3344, (Aug. 2002) S. Gundavelli, et al., ”Flow Binding Support for Mobile IPv4” - draft-ietf-mip4-multiple-tunnel-support-00, (Aug. 2010)

[9] D. Johnson, C. Perkins, J. Arkko, “Mobility Support in IPv6”, IETF RFC 3775, (Jun. 2004)

[10] R. Wakikawa et al, “Multiple Care-of Addresses Registration”, IETF RFC 5648 (Oct 2009)

[11] G. Tsirtsis et al., “Flow Bindings in Mobile IPv6 and NEMO Basic Support”, draft-ietf-mext-flow-binding-11

[12] R. Moskowitz, P. Nikander, P. Jokela, T. Henderson, “Host Identity Protocol”, IETF RFC 5201, (Apr. 2008)

[13] OpenVPN, http://openvpn.net/index.php/open-source/overview.html

[14] T. Henderson, P. Nikander, M. Komu, “Using the Host Identity Protocol with Legacy Applications”, IETF RFC 5338, (Sept. 2008)

[15] Network Manager home page http://projects.gnome.org/NetworkManager/

[16] P. Ayuso. Netfilter’s connection tracking system, ;LOGIN: The USENIX magazine, 32(3):34–39, 2006

[17] The netfilter project, http://www.netfilter.org/

[18] UPMT homepage: http://netgroup.uniroma2.it/UPMT