Efficient Measurements of IP Level Performance to Drive Interface Selection in Heterogeneous Wireless Networks

Fabio Patriarca
Dip. Ingegneria Elettronica
Università di Roma “Tor Vergata”
Roma, Italy
fabio.patriarca.2@uniroma2.it

Stefano Salsano
Dip. Ingegneria Elettronica
Università di Roma “Tor Vergata”
Roma, Italy
stefano.salsano@uniroma2.it

Francesco Fedi
SSI – Sistemi Software Integrati
Taranto, Italy
francesco.fedi@ssi.it

ABSTRACT
Several solutions have been proposed for mobility management in IP based heterogeneous networks, working at different protocol levels, from layer 2 up to application level. In order to take the handover decision, many solutions require to monitor the performance of the heterogeneous networks to which the mobile device is connected. Measuring the physical or link level performance on a given wireless access networks does not provide a reliable indication of the perceived level of service when the application flows are handed over that wireless access network. It is therefore needed to take measurements at IP level, on the (bidirectional) path from the Mobile Host to an intermediate node handling the mobility or even to the remote Correspondent Host. Gathering such measurements in a timely, effective and efficient way is not an easy task. In this paper we show that a naïve approach using application level active measurements is highly CPU intensive. This would severely impact battery usage in Mobile Hosts and does not scale if intermediate mobility management nodes are involved. On the contrary we show that an implementation of active measurements in the Linux kernel has a very low CPU usage. In this approach an efficient use of batteries in Mobile Host can be achieved and intermediate mobility management nodes can scale up to monitoring thousands of flows towards Mobile Hosts. Finally we discuss how combining passive and active measurements could further improve the solution.

Categories and Subject Descriptors
C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design

Keywords
Mobility management, measurement based, vertical handover, performance evaluation.

1. INTRODUCTION
Several solutions have been proposed in the last 15 years for mobility management in IP based heterogeneous networks, working at different protocol levels [1], from layer 2 up to application level. Nevertheless, mobility management is still an open issue for research and standardization and work is still actively ongoing in this area.

We consider a scenario in which the terminals have different wireless interfaces (e.g. WiFi, 3G/4G, WiMax). We assume that multiple interfaces can be active at the same time, therefore a decision must be taken on which interface will be selected at a given time (handover decision). A review of the handover decision process can be found in [2], the decision process can be terminal based or network based and several factors can be taken into account, from received signal strength on the radio interface, to cost of connectivity, desired QoS, battery usage and so on. Among these, it is relatively easy to evaluate the radio link performance on a given wireless access networks. Unfortunately, this does not provide a reliable indication of the level of service provided to the Mobile Host when its application flows are handed over that wireless access network. In order to achieve such indication, measurements should be taken at IP level on the path from the Mobile Host up to the intermediate node that handles the mobility or up to the Correspondent Host if the mobility is handled end-to-end.

Gathering such measurements in a timely, effective and efficient way is not an easy task. We will focus on a mobility management solution called UPMT (Universal Per-application Mobility Management using Tunnels) [3], but this issue is common to all mobility management solutions that combine heterogeneous networks using IP (e.g. Mobile IP [3], HIP – Host Identity Protocol [5]). Therefore our findings are of general value in this respect.

The UPMT solution (introduced in section 2) is based on tunneling over UDP and can be applied to different scenarios. In particular in this paper we will consider two scenarios: i) Internet access provided to a Mobile Host over heterogeneous access networks; ii) Networked Robot System, i.e. a machine-to-machine communication scenario using heterogeneous wireless networks. These scenarios will be described in section 3. Section 4 will describe the tunnel performance measurement mechanisms, designed in order to minimize the complexity and the state information to be maintained. Sections 5 and 6 will respectively deal with the implementation of the designed mechanisms in user space and in kernel space under Linux OS. The CPU processing performance of the two implementations are compared in section 7. Section 8 provides some hints for the design of a mixed active and passive measurement approach and finally section 9 reports some conclusions.

2. UPMT BASICS
UPMT is a solution for mobility management over heterogeneous networks based on IP in UDP tunneling. In this section we shortly recall its main features, further details in [3][7][8]. A Mobile Host establishes IP in UDP tunnels over its active network interfaces with its “correspondent” UPMT node. This correspondent UPMT node

can be another Mobile Host (see Figure 4), an "Anchor node" (see Figure 3) or a correspondent UPMT aware Fixed Host. The tunnels are used to exchange the IP packets according to the format shown in Figure 1. The "external" packet has IP source and destination addresses corresponding to the IP addresses of the interfaces of the Mobile Host and of the correspondent UPMT node. The internal encapsulated packet can keep the same IP source and destination addresses irrespective of the interfaces used for sending and receiving the packet. This allows seamless handovers of flows among multiple tunnels setup between the Mobile Host and the correspondent UPMT node.

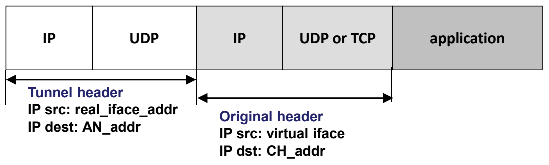

Figure 1. UPMT packet format

In our Linux implementation of UPMT, the UPMT kernel module provides a virtual interface called UPMT0 as a regular networking device, as shown in Figure 2. A "virtual" IP address can be assigned to it and the legacy applications will see a standard networking device. The UPMT encapsulation and mobility management is completely transparent for the applications that can use plain sockets to communicate.

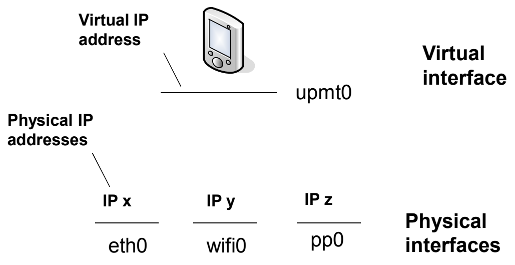

Figure 2. UPMT virtual interface vs. physical interfaces

3. SCENARIOS OF INTEREST

The UPMT solution can be applied to different scenarios, we will consider two of them in this paper. The first scenario is called "Internet access" and it is shown in Figure 3. A Mobile Host is connected to an "Anchor Node" via different access networks and it has to choose the "best" access network over time. If a given access network is used and the connectivity towards the Anchor Node through such access network fails, the active flows should be immediately handed over another access network. If the failure happens on any node or link behind the radio access point in the path toward the Anchor Node, it is undetectable using the radio link monitoring. The only option is to perform a continuous monitoring at IP level.

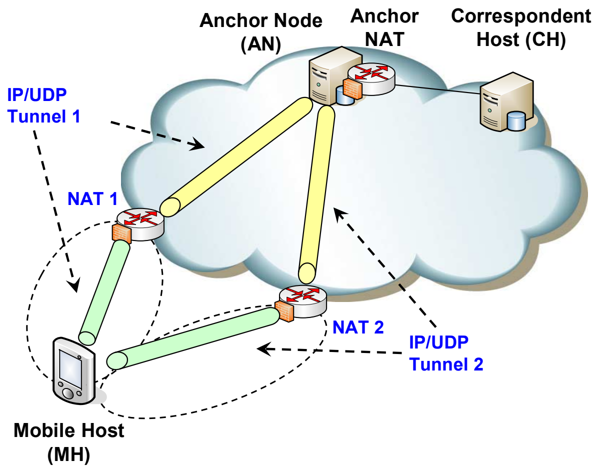

Figure 3. Internet Access scenario

In the second scenario (Figure 4), a set of devices, like for example Networked Robot Systems (NRS) [6] which are typically composed by a set of UGV/UAV (Unmanned Ground/Aerial Vehicle) or the devices of a WSN (Wireless Sensor Network) use multiple heterogeneous interfaces to communicate. We will refer to these devices as "Mobile Hosts" to keep the same notation of the previous scenario. Each Mobile Host could use up to three independent WiFi interfaces, a WiMax interface, two or three 3G interfaces of different operators, a VHF transceiver. The WiFi networks can operate in mobile ad-hoc mode with routing protocols like OLSR (Optimized Link State Routing Protocol) [11]. In this scenario the UPMT is used (i) to hide the plethora of underlying communication interfaces to the applications running on the devices, (ii) to improve the efficiency of the communication resource usage by forwarding application packets taking into account theirs urgency and criticality. The Mobile Hosts will only see a virtual interface called UPMT0 (we are using the Linux operating system) with a statically configured IP address. This IP address will be used by the devices to communicate each other, irrespective of the IP addresses that will be used on the physical interfaces. The multiple tunnels will be used to exploit diversity: if a Mobile Host has two active interfaces and its correspondent Mobile Host has also two active interfaces, four tunnels could potentially be established (see Figure 4).

We assume that ad-hoc routing protocols are used in the wireless mobile networks. In particular if the Mobile Hosts have multiple WiFi interfaces, different WiFi channels can be used for creating multiple (up to three) parallel mobile ad-hoc networks. This will create redundancy and different options to send packets from one Mobile Host to another. The UPMT module will hide this redundancy to the applications running in the Mobile Hosts. The applications will only see one interface and one static IP address despite a continuous change in the underlying wireless networks connections. The NRS Mobile Hosts operate on a "Virtual Mobile Network" that is dynamically mapped over multiple Physical Mobile Networks.

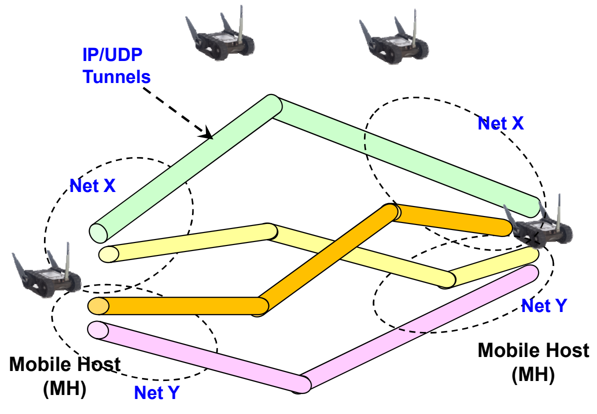

Figure 4. Networked Robot System (NRS) scenario

In both scenarios, there is the need to monitor which tunnels provide connectivity between each couple of Mobile Hosts and what is the performance (delay and loss rate) of the connected tunnels. The UMPT solution allows for the detection of a sudden loss of connectivity or a sharp decrease in performance on a connected tunnel improving the overall system reactivity to external event, a key properties for distributed real-time systems such as the NRS .

In general, such monitoring and measurement can be done using an *active* approach (i.e. sending probe packets) or with a *passive* approach (i.e. trying to infer connectivity status and tunnel performance from the observation of existing traffic). In principle, the passive approach is preferable because it does not introduce additional traffic into the network. Typically, it is not feasible to only rely on passive measurements, because measurements and monitoring are needed also in absence of traffic. Therefore the choice is between using active measurements only (simpler but less efficient in terms of network and CPU load) or a combination of active and passive measurements (more complex but more efficient).

In this paper we discuss the mechanisms to perform connectivity check and performance measurements considering also the practical implementation of these mechanisms in the UPMT solution. In particular we focus on the CPU processing load that is imposed by these procedures on Mobile Hosts and on mobility management nodes where present.

4. DESIGN OF THE PERFORMANCE MEASUREMENTS PROCEDURES

We refer to the procedure that monitors the connectivity over a tunnel and evaluates the network performance as the "Keep alive" procedure. Let us first describe this procedure in the Internet access scenario. The procedure is periodically executed by the Mobile Host which sends a "probe" packet towards the Anchor Node for each active tunnel. The Anchor Node sends back a "probe reply" packet.

In the Internet access scenario this procedure can also serve the purpose of maintaining the tunnel connections active through NAT (Network Address Translation) boxes in the path between the Mobile Host and the Anchor Node.

The "Keep alive" procedure is designed to perform at the same time a connectivity check and an evaluation of network performance parameters like Round Trip Time (RTT) and loss rate. Our design goal for this procedure is to keep it simple and to minimize the amount of state information to be maintained by the Mobile Host and by the Anchor Node.

The Mobile Host numbers the probe packets with a sequence number and adds a timestamp when sending the packet. The sequence number is per tunnel, therefore a state variable is needed for each tunnel. The Anchor Node will copy this information (sequence number and client timestamp) in the probe reply packet and will add its own timestamp. In this way, the Mobile Host can evaluate the RTT delay from the received probe reply packet without keeping a state information. This "per packet" RTT information can be accumulated using an EWMA (Exponentially Weighted Moving Average) so that a single state variable per tunnel can represent the RTT performance of the tunnel. It is also possible to use two different EWMA state variables using different smoothing factors in order to have the information at two different time scales (for example a shorter time scale in the order of few seconds and a relatively longer time scale in the order of few tens of seconds).

The details for the evaluation of the RTT EWMA are as follows. The definition for the EWMA S_k of a variable x available at regular time intervals $\{t_k\}$ with period T ($t_k = k \cdot T$) is:

$$S_k = \alpha \cdot x_{tk} + (1-\alpha) \cdot S_{k-1}$$
$$S_0 = x_0$$

where S_k is the EWMA of x at time $t_k = k \cdot T$, and α ($0< \alpha <1$) is the "smoothing factor". A higher α implies a higher weight of more recent observations of x. We extend this definition for the general case in which the values of the variable x are available at non regular intervals. Let $\{t_k\}$ be the sequence of time instants at which an observation x_{tk} is available. Let $\Delta_k = t_k - t_{k-1}$. Given a reference time interval T, we can define the EWMA as follows:

$$S_k = \left[1-(1-\alpha)^{\Delta tk/T}\right] \cdot x_{tk} + (1-\alpha)^{\Delta tk/T} \cdot S_{k-1}$$
$$S_0 = x_0$$

The smoothing factor now depends on the time interval between two different observations. If $\Delta_k = T$ the smoothing factor is exactly α, like in the simpler case.

It is interesting to observe that the RTT is a "bidirectional" delay measurement, as it takes into account the transit delay in the tunnel in both directions. In the described solution, no state information is needed on the server, only a state variable is used per tunnel in the client in order to accumulate the EWMA of the RTT. Measuring the One Way Delay (OWD) would require clock synchronization between the Mobile Host and the Anchor Node.

Let us consider now the estimation of loss rate. We describe a simple procedure that can measure the "Round Trip" loss rate l_{RT}, i.e. the probability that a packet is dropped when travelling from the Mobile Host to the Anchor Node and then back from the Anchor Node to the Mobile host. l_{RT} can be expressed in function of the uplink loss rate l_{up} (from Mobile Host to Anchor Node) and of the downlink loss rate l_{down} as follows:

$$l_{RT} = l_{up} + l_{down} - (l_{up} * l_{down})$$

When l_{up} and l_{down} are small, $l_{RT} \approx l_{up} + l_{down}$

Obviously, knowing l_{RT} is suboptimal in case one needs a separate estimation of l_{up} and l_{down} but there is a great saving in the complexity of the procedure and in the state information to be maintained.

The Mobile Host evaluates the Round Trip loss rate every K probes, i.e. over a time interval equal to $K \cdot T_{KA}$ where T_{KA} is the configured interval for the Keep Alive procedure. For each tunnel, the Mobile Host will simply count the number of probe replies received in the in the time interval using a state variable called ReceivedProbes, while OutSeqNum is the state variable counting the number of sent probes. The following pseudo code describes the algorithm performed when sending out a probe and receiving a probe reply.

When sending out a probe

```
OutSeqNum = OutSeqNum + 1
If OutSeqNum mod K = 0 {
   LossRate = max [(K-ReceivedProbes)/K , 0]
   ReceivedProbes = min [ ReceivedProbes – K , 0]
}
```

When receiving a probe reply

```
ReceivedProbes = ReceivedProbes +1
```

Note how the ReceivedProbe state variable is reset when evaluating the loss rate every K probe intervals. Due to the RTT delay, a probe reply could not be received in time, in this case the algorithm will measure a loss event over the observation period. If the RTT remains constant, in the next period the number of probe and probe reply will match and no loss will be detected. If the RTT decreases, one can receive a number of replies larger than K in an observation interval. In this case the excess probe replies are accounted for in the next observation period.

The Round Trip Loss rate that is evaluated on each period can be accumulated using an EWMA like the RTT. In this case another state variable will be added per each tunnel.

Overall, the state variables that need to be maintained in the Mobile Host per single tunnel to be monitored are:
(OutSeqNum, ReceivedProbe, RTT-EWMA, l_{RT}-EWMA).

No state information per tunnel (related to the performance monitoring) needs to be maintained in the Anchor Node. Therefore the solution is scalable with the number of Mobile Hosts that need to be handled by an Anchor Node, as far as the state information is concerned.

Let us now move from the “Internet access” scenario to the “Networked Robot System” scenario. In this case there is no notion of Mobile Host/Anchor Node, but the two Mobile Hosts are “peers” each other. Both Mobile Hosts can be interested to measure the performance of the tunnels in order to decide how to send the flow directed toward the other Mobile Host. One possibility is that both Mobile Hosts independently perform the Keep Alive procedure, taking their own measurements. This will duplicate the probe traffic load on the network. A smarter solution is that only one of the two Mobile Hosts will perform the Keep alive procedure and will continuously report the results to the other side. Given that for each tunnel one of the two ends plays the role of the client (the one that has sent out the Tunnel Setup Request) and the other end plays the server role, the former Mobile Host can take the responsibility to send the probe packets. Two main modifications are needed in this case: i) the probe packets will also carry the evaluated RTT-EWMA and l_{RT}-EWMA so that these values will be communicated to the Mobile Host playing the server role; ii) the Mobile Host playing the server role needs to implement a timer for each tunnel that will be triggered when no probe requests arrive for a certain interval, meaning that the tunnel is under a critical failure.

5. USER SPACE IMPLEMENTATION

The Keep-alive procedure described in the previous section have been first designed and implemented at the user level. The UPMT software is composed of a kernel module dealing with encapsulation of packets into tunnels and of a Java application that offers a GUI to the user and manages the signaling messages between the UPMT remote entities. The signaling is based on the SIP protocol and implemented using the Open Source MjSip stack [10]. Therefore we relied on this software infrastructure and implemented the Keep-alive probe packets using SIP MESSAGE methods [11]. The SIP MESSAGE is a SIP request message that does not create a session, but can be used to transfer any information. The receiving entity will reply with a SIP 200 OK message according to the SIP protocol rules. The SIP stack implementation will manage multiple retransmission of the request if no reply comes in within a timeout. We enhanced the SIP stack adding a new SIP header to the messages, called Timestamp. When performing the Keep-alive procedure, the Mobile Host will send a SIP MESSAGE toward the correspondent UPMT node, adding the Timestamp header (time is expressed in millisecond since Jan 1 1970). The initial part of the SIP MESSAGE is reported in Figure 5, showing the new Timestamp header.

```
MESSAGE sip:160.80.103.66:5060 SIP/2.0
Via: SIP/2.0/UDP 5.6.7.8:40000;rport;branch=z9hG4b
K809f1ea0
Max-Forwards: 70
To: <sip:160.80.103.66:5060>
From: <sip:1.2.3.30>;tag=251807832719
Call-ID: 314335872631@5.6.7.8
CSeq: 1 MESSAGE
Expires: 3600
User-Agent: mjsip 1.7
Timestamp: 1339598185957
```

Figure 5. SIP MESSAGE for the Keep alive probe

Implementing the above solution was relatively easy as we reused the available code structure of UPMT. Unfortunately, when we performed scalability tests (as reported in section 7) we found out that the solution was using a relatively high amount of CPU power. We could have switched to a simpler solution in application space, taking out the overhead added by using the SIP protocol. Taking also into account the latency requirements typical of NRS we adopted a kernel space implementation. In fact, part of the load is due to the continuous switching from kernel to the application when sending and receiving packets.

6. KERNEL SPACE IMPLEMENTATION

We designed and implemented the Keep alive procedure with all the performance measurements done within the UPMT Linux kernel module. Linux kernel timers are used to schedule the sending of probe packets for each active tunnel. It is possible to activate/deactivate the Keep-alive procedure for each tunnel by sending configuration commands from a user space application.

The probe packet is a UDP packet incapsulated within the tunnel. The external IP destination address and UDP destination port are the ones of the tunnel. The internal IP destination address is the same of the tunnel, the UDP source and destination ports are used to distinguish a Keep alive probe packet.

When originating a probe packet, the kernel module incapsulates the inner probe packet into a UDP packet and sends it. When receiving a probe packet, the kernel module decapsulates the packet like any other packet received on the tunnel. Then a matching with UDP destination and source ports is performed to recognize the probe packets. If the packet is recognized as a probe, it will not be forwarded to a UDP socket to be delivered to user space but it will be dropped in the kernel. In this case the kernel module will generate the probe reply packet (copying the timestamp from the received packet) and will encapsulate it into a UDP packet to be sent back to the sender of the probe. The probe (and probe reply) packet format is shown in Figure 6.

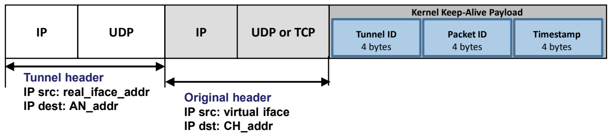

Figure 6. Probe packet format in the kernel implementation

As a possible evolution, we plan to define a UPMT management packet that can be sent instead of IPv4 and IPv6 packets within a tunnel. In this case we will not need the internal IP and UDP headers, but we will send such generic UPMT management packet as payload of the IP/UDP external packet. Both generation and "matching" of the probe/probe request packets will be much faster in this case.

7. PROCESSING PERFORMANCE

There is clearly an advantage in setting the Keep alive rate at the highest possible value: it allows to have a more precise estimation of RTT and of Round Trip loss rate and to react in a faster way to changing network conditions. Unfortunately, there are two factors that limit the potential increase or the Keep alive rate: the CPU load (on Mobile Hosts and intermediate mobility management nodes, if present) and the network load. Of the two factors (CPU load and network load) we believe that the most critical one is the CPU load. As a thumb rule, a Keep alive rate in the order of 2-3 Keep alive per second could be enough to fulfill the requirements of a precise and timely estimation of RTT and loss. From the network load perspective this would correspond to few hundred bits/second, i.e. one order of magnitude less than a VoIP call. On the other hand we will show hereafter that the CPU load may become critical even at these relatively low rates if an inefficient implementation is used.

In both the Internet access and NRS scenarios, the CPU load has an impact on the battery usage for the Mobile Host. Even if the CPU load due to the monitoring of few tunnels would be low in absolute terms, a reduction of this load has a positive impact on battery duration as the performance monitoring procedure needs to be continuously executed when the Mobile Host is connected. In the Internet access scenario, the CPU processing due to the monitoring procedures can even be the bottleneck for the mobility management node (i.e. the Anchor Node).

We set up our testbed with virtual machines running on VirtualBox [12] in a PC with a Intel® Core™2 Quad CPU Q8400 processor running at 2.66Ghz (4GB RAM). We focused on the Internet access scenario and considered the CPU utilization in the Anchor Node. One virtual machine was running an Anchor node, while the Mobile Hosts were running in different virtual machines. We executed the Keep alive procedure at different rates both for the user space and for the kernel space implementation. We measured the CPU utilization using the *sar* command, a part of the sysstat package.

Figure 7 shows the results for the CPU usage vs. the overall Keep alive rate received by the server with the user space implementation. The experiments were repeated with a different number of Mobile Hosts (1, 2 and 3) for the same aggregated Keep alive rate, showing a small dependency on the number of clients (i.e. receiving a total of 3 Keep alive requests per second from 3 Mobile Hosts is slightly heavier that receiving 3 requests per second from a single Mobile Host). The results show that the CPU utilization grows linearly with the sending rate of the probes. From the results we can estimate the maximum Keep alive rate that an access node can sustain within a given CPU utilization threshold (e.g. 50%). This maximum Keep alive rate is in the order of 100/s. If we assume 2 Keep alive per second per tunnel and 2 tunnels per client the maximum number of clients for the Anchor Node is in the order of 25.

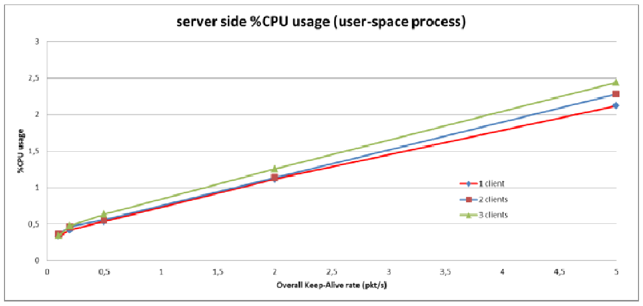

Figure 7. CPU usage for the user space implementation

Clearly this result is dependent on the specific hardware that we have used for the experiment, but what is of general interest is the ratio between the supported number of flows in the user space solution and the one in the kernel space solution. Figure 8 reports the CPU usage versus the Keep alive rate in the server for the kernel space implementation. We did not consider more than one client making the request, because the procedure on the server side is now completely independent of the number of clients as each probe packet is handled in a stateless way (the dependence from the number of clients in the user space implementation was due to the management of SIP protocol for different Mobile Hosts). Also in this case the load on the server grows linearly with the Keep alive rate, but the sustainable rate is much higher. If we consider the same maximum CPU utilization threshold (50%) we can evaluate that 50000 Keep alive per second can be handled by the Anchor Node. By making the same assumptions considered above, this would turn in 12500 Mobile Hosts that can be supported. The gain that we have obtained with respect to the user level implementation is in the order of 500 times.

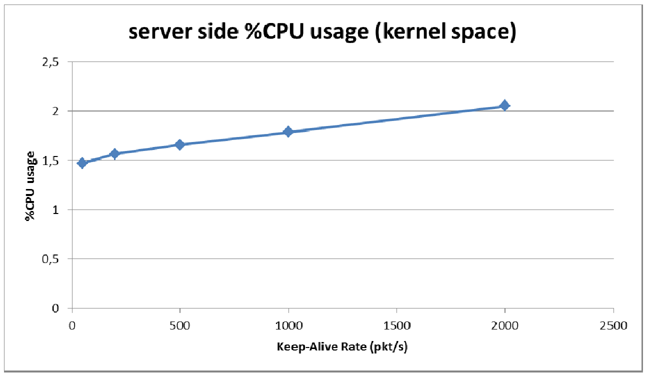

Figure 8. CPU usage for the kernel space implementation

As we mentioned above, this is an important indication also for the CPU processing load in the Mobile Host side, which we have not explicitly measured. We can expect that such a large reduction of the processing load will have a benefic impact on the duration of the battery.

8. PASSIVE MEASUREMENTS

The tunnel performance measurement procedures proposed in section 4 and implemented as described in sections 5 and 6 are only based on the active measurement approach. Passive measurements, i.e. the capacity to exploit the existing traffic to gather performance information could improve the efficiency of the solution. It could reduce CPU load and network load and/or increase the accuracy and timeliness of performance monitoring for the same CPU and network load. Our idea for taking passive measurements is to add timestamp and packet counter information to packets in transit on a tunnel. As our UPMT tunneling modules operate in kernel space we can perform this operation with a minimal CPU overhead while encapsulating and decapsulating the packets in the tunnels. As anticipated in section 6 we plan to define a UPMT packet format that will be transferred within the tunnel (i.e. inside the UDP payload of the outer packet). We can easily differentiate between i) plain IPv4 and IPv6 packets, ii) UMPT active probe packets that only carry the Keep-alive Payload as described in section 6; iii) UPMT passive probe packets that carry both information related to performance measurement procedures and an IPv4/IPv6 packet. Hereafter we present the high level design of the passive measurements procedures. Detailed design and implementation is ongoing. The passive probe packets use relatively "short" IPv4 and IPv6 packets so that adding the performance measurement related data will not exceed the Maximum Transmission Unit of crossed links. The combined active/passive probe mechanism sends active probe packets only when normal traffic is not flowing on a tunnel (i.e. it reduces the sending rate of the active probes considering the rate of passive probes). On each transmitting end of a tunnel, the total number of transmitted packets is counted. This information is added in the probe packets, allowing to measure tunnel loss rate with much higher accuracy and timeliness, including the evaluation of the "unidirectional" loss rate in the two directions.

9. CONCLUSIONS

In this paper we have presented the definition and implementation of performance monitoring procedures to drive interface selection in heterogeneous network. We have proposed a solution to estimate RTT and Round Trip loss rate with minimal amount of state information and very simple processing. We have implemented the solution in user space and in kernel space using the Linux OS. We measured the CPU processing load of both solutions and showed that the kernel space solution is dramatically more efficient. If we consider a mobility management node handling thousands of Mobile Hosts, a kernel space implementation seems to be the only scalable solution. As for the implementation in a Mobile host that should handle a limited number of tunnels, the dramatic reduction in processing load helps increasing the battery duration.

10. AKNOWLEDGEMENTS

This work was supported in part by italian regional funding FESR 2007-2013 by Regione Puglia for the project "Sistema per la bonifica di aree critiche basato su sciame di robot (BEE SAFE)".
The authors wish to thank Marco Bonola for his work on the design of UPMT and his precious suggestions and Marco Galvagno for the implementation of the user space monitoring solution.

11. REFERENCES

[1] D. Le, X. Fu, D. Hogrere, "A Review of Mobility Support Paradigms for the Internet", IEEE Communications surveys, 1s t quarter 2006, Volume 8, No. 1

[2] Meriem Kassar, Brigitte Kervella, Guy Pujolle, "An overview of vertical handover decision strategies in heterogeneous wireless networks", Computer Communications, Volume 31, Issue 10, 25 June 2008, Pages 2607–2620

[3] M. Bonola, S. Salsano. "UPMT: Universal Per-Application Mobility Management using Tunnels", IEEE GLOBECOM 2009

[4] C. Perkins, Ed., "IP Mobility Support for IPv4, Revised", IETF RFC 5944, November 2010

[5] R. Moskowitz, P. Nikander, T. Henderson, "Host Identity Protocol", IETF RFC 5201, April 2008

[6] D. Calisi, F.Fedi, A.Leo, D. Nardi, "Software Development for Networked Robot Systems", 7th IFAC Symposium on Intelligent Autonomous Vehicles, September 2010.

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

[9] T. Clausen, P. Jacquet, Eds. "Optimized Link State Routing Protocol (OLSR)", IETF RFC 3626

[10] MjSip home page: http://www.mjsip.org

[11] B. Campbell (Editor),"Session Initiation Protocol (SIP) Extension for Instant Messaging", IETF RFC 3428, December 2002

[12] Oracle VM VirtualBox, http://www.virtualbox.org