**H-0264 (H0811-004) November 19, 2008**
**Computer Science**

# IBM Research Report

## IP Mobility to Support Live Migration of Virtual Machines Across Subnets

**Ezra Silvera*, Gilad Sharaby, Dean Lorenz, Inbar Shapira**
IBM Research Division
Haifa Research Laboratory
Mt. Carmel 31905
Haifa, Israel
(ezra, giladsh, dean, inbar_shapira)@il.ibm.com

* Corresponding author

**Research Division**
**Almaden - Austin - Beijing - Cambridge - Haifa - India - T. J. Watson - Tokyo - Zurich**

# IP Mobility to Support Live Migration of Virtual Machines Across Subnets

***Abstract*** *– User-transparent live migration is one of the most interesting features of Virtual Machine (VM) environments. Current live-migration technologies require that the VM retains its IP network address; therefore, are typically restricted to movement within an IP subnet. The growing number of portable computing devices has led to the development of IP mobility solutions that enable uninterrupted network connectivity while moving between different IP subnets. In this paper we study the application of current network mobility approaches to VM cross-subnet live-migration. We show that although the core problems are similar, there are significant differences between these domains, in terms of both assumptions and requirements. We present a specific solution for live migration of a VM across IP subnets, and introduce a new framework for synchronizing migration and network configuration, which allows better optimization for different scenarios of live migration.*

## 1. INTRODUCTION

Virtual Machine (VM) environments provide a wide range of benefits including resource consolidation, performance isolation, and user-transparent migration. Among them, user-transparent live migration is one of the most interesting features, as it helps separate the hardware and software management and consolidate clustered hardware into a single coherent management domain. It allows for better resource load balancing, performance optimizations, and disruption-free hardware maintenance.

Current live-migration technologies (e.g., [1],[2]) allow for the live migration of virtual machines without any significant downtime on their provided service. However, these technologies require that the VM retains its network address, in order to keep the migration transparent from a network perspective and maintain uninterrupted network connectivity. Internet Protocol (IP) assumes that a host's IP address uniquely identifies the host's point of attachment to the Internet. Thus, if migration is not to be restricted to movement within an IP subnet, additional measures must be taken to allow a VM to seamlessly use its pre-migration IP on a different subnet.

The problem of maintaining uninterrupted network connectivity while migrating between different IP subnets has received significant attention from the network community, in the context of mobile communication. Mobile nodes change their network point of attachment as they move (e.g., roam between wireless access points, change laptop attachment between office and home); but, want seamless network connectivity (i.e., want to maintain transport and higher-layer connections). With a growing number of portable computing devices like laptops and PDAs, the need for seamless connectivity to the global Internet is driving the acceptance of different mobility solutions.

In this paper we study the application of current network mobility approaches to VM cross-subnet live-migration. We show that although the core problems are similar, there are significant differences between these domains, in terms of both assumptions and requirements. We present a specific solution for live migration of a VM between two different IP subnets, and a general framework which allows better optimization for different scenarios of live migration.

Mobility support approaches have considered many aspects of the problem, including steady-state performance (e.g., optimal routing), handoff latency, security, and network overhead (including signaling). This paper is focused only on handoff latency, leaving detailed discussion of other aspects for future work. Handoff is the process that allows a Mobile Node (MN) to seamlessly change its point of attachment from one Access Point (AP) to another, while maintaining running communications. Handoff latency is the primary cause of packet loss and the resulting performance degradation, especially in the case of reliable end-to-end communication. As a result, numerous methods of minimizing the handoff latency have been proposed in the literature. Seamless handoff has also been the focus of VM migration design.

The remainder of this paper is organized as follows. In Section 2 we provide some background on the different types of network mobility solutions and, in particular, the issue of handoff. In Section 3 we discuss the applicability of mobility solutions to migration and the specific properties of VM migration in this context. We introduce a new framework for synchronizing migration and network configuration. In Section 4 we discuss the implementation of IP mobility for VM migration, using Mobile IP as a case-study. In Section 5 we present evaluation results and some insights from our experiments. Finally, we conclude in Section 6.

## 2. BACKGROUND

In this section we overview some of the existing mobility protocols and handoff mechanisms.

## 2.1 Mobility schemes

### 2.1.1 Mobile IP

The IETF Mobile IP protocols [3][4] are the current basis for solutions for mobility management in IP networks. Mobile IP is a simple and scalable extension to the IP, allowing a mobile host to communicate with other hosts after changing its point of attachment to the Internet, yet without changing its IP address. Any network application executing on a mobile host with mobile IP support can continue to run regardless of any change in the mobile node's point of network attachment. This is achieved by allowing each mobile node to have two IP addresses: one of the IP addresses is the permanent home address that is assigned at the home network and is used to identify communication endpoints; the other is a temporary care-of address that represents the current location of the host. Mobile IP transparently maintains the binding between the two addresses with the help of mobility agents: a home agent (HA) on the mobile node's home network, and a foreign agent (FA), on the network to which the mobile node is currently attached. When the mobile node is away from its home network, the HA and FA reroute (tunnel) its traffic between the permanent and care-of addresses.

### 2.1.2 Mobility support above the Network Layer

Mobility can be implemented in higher layers of the network protocols stack. There are protocols supporting mobility at the transport layer, adding migration semantics for TCP connections in the protocol stack on the end hosts (e.g., SCTP [5], TCP-MH [6] , DCCP [7], and Snoeren and Balakrishnan [8]). Mobility support at the transport layer tries to avoid the complexity and cost of the transparent solution provided by Mobile IP. The argument given is that not all the applications require the level of transparency and generality provided by Mobile IP.

Mobility support has been added to even higher layers, such as the session and application layers (e.g., Snoeren [9], MSOCS [10], SIP [11]).

### 2.1.3 Mobility support bellow the Network Layer

Mobility can be also supported at lower layers of the network protocols stack. LAN switches and bridges can seamlessly handle host movement between access points (or ports) on the same sub-net, automatically refreshing MAC forwarding tables to reflect the new location. Cellular networks support intra-domain micro-mobility by using mobile specific routing, without tunneling, to route packets inside the access network (e.g., Cellular IP [12], HAWAII [13]). Cisco Local-Area Mobility (LAM) [14] enables mobile nodes to move from their local subnet to another location within an enterprise network while maintaining transparent connectivity. LAM does not require any software changes on the hosts. A router configured for LAM is able to detect foreign nodes by inspecting traffic on its LAN interfaces in order to determine whether there are directly connected hosts that do not belong to the local IP subnet. When this router sees locally originated traffic from a host that does not match the address and mask configured on that interface, the router installs an ARP entry for this mobile host. This router then also installs a host route that points toward the particular interface and propagates this route to the enterprise through its routing protocols. Even hosts on the home subnet can communicate with the mobile host, because if a router knows the route to a device and sees an ARP request on the LAN, it proxy ARPs [15] in reply.

### 2.1.4 Overlay Networks

A similar low-layer approach to mobility support is to provide an IP sub-net that is wide enough to cover all desired points of network attachment. This sub-net is a logical LAN that is overlaid across several physical LANs. The most common overlay technology is (Layer 2) Virtual Private Networking (VPN), which includes several sub-types, such as 802.q Virtual LAN (VLAN) [16], Virtual Private LAN Service (VPLS) [17][18], and IP-Only LAN-Like Service (IPLS) [19]. Also common are application-level overlays which are based on virtual network appliances (i.e., virtual bridges, virtual routers, etc.). There are many implementations, including some which were specifically designed to support the networking requirements of virtual machines (e.g., VNET[20], VIOLIN [21]).

## 2.2 Handoff

The protocol involved in changing a mobile node's network attachment point is called handoff. The key QoS factors are low packet lost and low latency during the execution of the handoff protocol. Packet lost significantly affects TCP throughput; moreover, due to TCP's flow control, it takes a relatively long time to recover even from a single packet lost. Unfortunately, most of the IP mobility schemes were not initially designed to support seamless low latency handoff.

### 2.2.1 Location propagation

During handoff, any mobility scheme must propagate and bind the new mobile node's location; a process that typically takes at least one round trip time (RTT) between the current location and the home network of the mobile node. Accordingly, hierarchical schemes (e.g., Hawaii [13], Hierarchical Mobile IP [22]) were proposed to shorten binding time for micro-mobility (i.e., mobility over short distances). Binding typically requires collaboration from mobility agents (e.g., FA), which must first be

discovered and contacted. Binding time may grow even longer when authentication and authorization are needed; either explicitly (e.g., registration at a new HA) or implicitly (e.g., for setting up a new secure tunnel).

### 2.2.2 Movement detection

Another contributor to handoff latency is movement detection. Most mobility schemes start the handoff protocol only after they identify a mobile node away from its home network; that is, either the mobile node discovers that it is on a foreign network, or the foreign network detects that a guest node has attached. The discovery requires L2 connectivity between the mobile node and the foreign network (and mobility handling entities, such as FA); therefore, discovery can typically start only after L2 handoff is complete.

Handoff latency is also governed by various timeouts. For examples, in mobile IP, a mobile node discovers that it is on a foreign network by listening to periodic FA advertize messages. L2 mobility handoff requires the expiration of ARP caches at routers and hosts and of MAC forwarding caches at switches and bridges; these timeouts are the most significant factor in total handoff latency.

### 2.2.3 Triggers

There have been many attempts to shorten handoff times and reduce packet loss [22][23][24][25][26]. There are two key observations. First, packets which are sent to the old point of attachment, after the mobile node has already moved, may be forwarded directly to the new location instead of being lost. This requires soft handoff; namely, collaboration between the old and the new points of attachment. Second, location propagation starts only after the mobile node attaches (and registers) at a new point and that L2 schemes wait for the first packet sent by the mobile node at the new location. Mobile IP must wait for the L2 handoff to complete, before starting L3 handoff and then it executes a registration process between the mobile node and the FA, before starting address propagation. Simple VPN schemes start the handoff process only upon host request to join the VPN (e.g., invoking a VPN client at the host).

For fast handoffs the network layer needs information from link layer so that it can re-establish connectivity as quickly as possible. The network layer needs to know from the link layer that handover is imminent or at least it has just happened. Link layer events can help anticipate mobile node movement and help prepare the mobile node and network in advance (e.g., pre-registration); furthermore, it is possible to exploit the interval between the time the mobile node detaches from the old location and the time it attaches to the new location.

An abstraction of such event notifications from link layer with other relevant parameter information can help reduce handover latencies [27]. Indeed, fast-handoff mobility schemes require special signals (triggers) to inform them of anticipated movement or "Advance Trigger" (AT), L2 detachment or "Link Down" (L2-LD), and L2 attachment or "Link Up" (L2-LU). These triggers may be invoked by either the network or the mobile node. Figure 1 shows the timeline of mobility with and without triggers. In Figure 1*a*, the movement is detected after L2 handoff is complete and only then the new address is propagated and bound. In Figure 1*b*, an AT trigger allows propagation of the new address and its binding to start before the detachment from the old address. The LU trigger allows shorter movement detection, which in turn triggers the final stages of binding. The overall outage time can be shortened considerably.

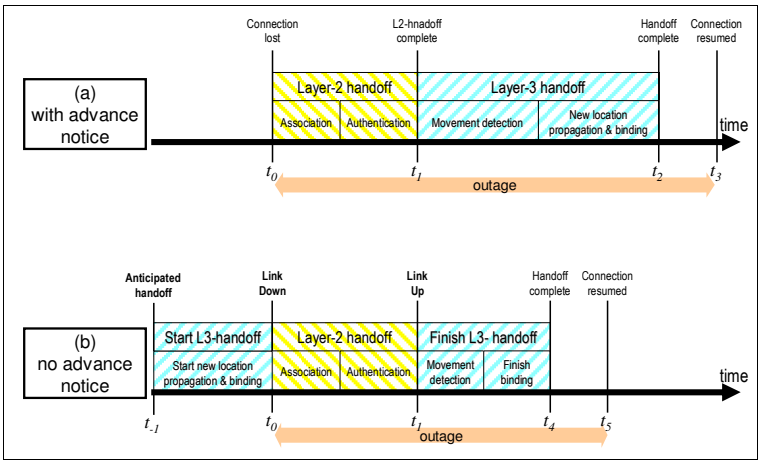


**Figure 1: Handoff & Triggers**

## 3. MOBILITY PROTOCOLS AND VM LIVE-MIGRATION

In this section we discuss the applicability of IP Mobility solutions to migration and the specific properties of VM migration in this context. We introduce a new framework for synchronizing migration and network configuration.

## 3.1 Properties of VM migration

VM migration is a special case of host mobility and has properties which can be exploited to support simpler and more efficient handoffs.

(a) ***Network QoS:*** Mobility is most often associated with wireless connectivity, which implies high error rates, low bandwidth, and long delays and requires link-level security (e.g., wireless association). VM migration provides much better network QoS and physical-like link-level security.

(b) ***Soft handover:*** VM migration inherently supports soft handover. Migration is a collaborative effort between source and target hypervisors that requires a direct connection between the source and target hypervisors.

(c) ***Trust:*** Trust is also inherent. The target hypervisor must authenticate and authorize the source of migration requests and the target must be trusted to allow a VM to run on it. Since trust must be managed by the migration code, it can be exploited to establish trust at the network mobility layer. Generic mobility protocols do not assume much about trust on the foreign network, thus suffer from significant overhead during registration and binding.

(d) ***Mobility agents:*** The source and target hypervisors can assist with the handoff, by either acting as mobility agents or pre-establishing connectivity with these agents. Furthermore, the hypervisors completely control the VM's network, allowing for more powerful and more transparent mobility agents.

(e) ***Triggers:*** VM migration is an informed process. Unlike mobility which is triggered by movement detection, the source and target hypervisors are always aware of migration well before the VM is actually suspended and disconnected from the network. They have full knowledge and can easily provide all the necessary mobility triggers. Indeed, even in the simpler case of intra-LAN migration, the target hypervisor initiates an unsolicited ARP message on behalf of the migrating VM. This is the equivalent of an L2-LU trigger and is used to shorten L2 handoff latency (without this trigger, one would have to wait for the ARP entries to expire). Note that, with VM migration, there is relatively more time to exploit between L2-LD and L2-LU triggers (at least as long as the time it takes to transfer VM state).

(f) ***Locks:*** The hypervisors have full control over the VM and its network. This allows extending the trigger concept to support *locks*; namely, block mobility events until they are released. For instance, one may want to wait with the detachment event (or even the VM suspension) until pre-registration is complete. Another example is waiting for tunnel setup before allowing the VM to resume on target (e.g., to avoid premature back-off by the VM's TCP flow control).

## 3.2 Synchronization of Migration event and network configuration

In this section we describe a new framework that is specifically designed to help with the issues involved in migration of virtual machines.

A VM migration process is composed of several phases. Each phase has particular characteristics in terms of VM and service availability, duration, etc. We suggest leveraging the knowledge about these unique phases in order to optimize the VM migration process.

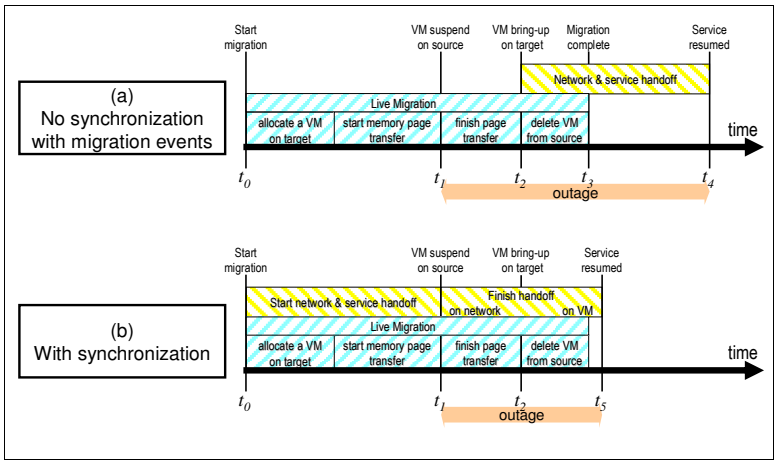


**Figure 2: Migration events**

Figure 2 illustrates the main migration events. Usually the migration starts with a memory transfer phase in which memory pages that belong to the VM are transferred from the source hypervisor to the target hypervisor. During this phase the VM is fully operational at the original location and hence the service is fully available to users. It should be noted that the service quality and availability may be impacted from the migration process also at this phase due to the resource used by the migration process (e.g., bandwidth); however, these performance aspects are already addressed by improvements of the migration process itself (e.g., limiting the BW assigned for migration).

After the initial transfer phase the VM is suspended at the source location and all the remaining memory pages and CPU state is transferred to the target hypervisor. Obviously, because the VM is down the service is also not available during this phase.

Finally when all the necessary information is on the target machine the VM is brought up on the target machine while the VM at the source is destroyed.

In theory the service downtime should be identical to the VM downtime ($t_2$-$t_1$). However, in practice the service availability does not depend only on the state of the VM (i.e., up or down) but also on additional aspects, most notably network connectivity. Therefore, as it can be seen in the diagram, it take additional time ($t_4$-$t_2$) from the moment the VM resumed operation on the target location to the time the service is restored and available to the user. Our goal is to minimize this additional time and therefore decrease the overall service downtime. Note that the additional time depends on the specific service and the details of the connectivity (e.g., network topology), and may vary from a few milliseconds to seconds and even to minutes in extreme situations.

When inspecting the tasks that need to be performed to resume the service (once the VM is up), we can usually identify both disruptive and none distractive operations (i.e., disruptive from the service perspective). We propose a migration synchronization framework that will allow a user to register to any of the migration events to enable synchronization of network and service management operations with the various migration events. For example, suppose we have a disruptive configuration operation with duration $T$ (e.g., change a routing path). If the user would activate it in the regular way (i.e., just after the VM is brought up on target) the service downtime will span at least between time $t_1$ and time ($t_4$=$T$+$t_2$). If, on the other hand, the operation is triggered by the "suspend VM on source" event (at time $t_1$), then the service downtime can end by time ($T$+$t_1$); in other words, we can reduce the service downtime by ($t_2$-$t_1$). Moreover, if $T \leq (t_2-t_1)$ than the service downtime would not be affected at all and the extra operation would be completely "hidden". We should note that ($t_2$-$t_1$) is dependent on many factors (e.g., rate of dirtying pages, bandwidth used for migration, etc.) and may become significant. Figure 2 shows that we can decrease the service downtime even more if we utilize a "migration start" event (at time $t_0$). Since ($t_1$-$t_0$) is relative long, there is more than enough time to perform pre-migration service and network handoff operations. Post-migration operations can start as soon as the VM is suspended on source (at time $t_1$). Only operations that involve the VM itself (e.g., refreshing its ARP cache entries) must start after the VM is brought back up (at time $t_2$). Therefore, we expect the service resumption time with utilization of VM migration events, ($t_5$ in Figure 2), to be significantly earlier than the service resumption time without them ($t_4$ in Figure 2).

We propose a new framework, in which the VM migration code is augmented to allow registration to different migration events (triggers) on either the source or the target hypervisors:

(a) ***MIGRATION_START:*** This event indicates that a migration process has been started, a connection has been established between source and target, and service migration is imminent. It also ensures that the operation has been authorized. At this point, preparation for network and service migration can start; however, traffic must still be directed to the source which still provides the service.

(b) ***SOURCE_SUSPEND.*** This event indicates that the source VM is down and VM "movement" phase is taking place. At this point, service is no longer provided at the source, so traffic can be directed to the target. All network and service migration operations that do not require interaction with the migrating VM, may be performed. It is even possible to bring up the target NIC (before the VM); however, this depends on the particular hypervisor and requires some code changes.

(c) ***TARGET_RESUME.*** This event indicates that the VM has finished "moving" to the target hypervisor. At this point, network and service migration should be finalized.

(d) ***MIGRATION_DONE.*** This event indicates that the migration process has finished and all resources on source have been freed. At this point, any forwarding code left on the source should be terminated.

In addition, in our framework, the user may temporarily suspend the migration process at certain events; that is, all these events should support both *informative* and *blocking* modes of operation. In informative mode, the events are fired, but do not affect the VM migration code itself (i.e., it continues to run before and after the event). In blocking mode, the event is fired synchronously and any further operation is blocked until the event callback returns.[1] Blocking events allow the user to synchronize network and service migration operations with a particular phase of the VM migration process. As we will show later, this ability allows the user to achieve much better performance (i.e., shorter downtime) than just using the informative events. In addition, on some occasions, if we cannot block the migration then we cannot take advantage of the migration events. For example, if we want to perform an operation just before the machine is suspended then, unless we block the migration, we cannot guarantee that the operation will complete before the VM is suspended. Accordingly, blocking events are fired right before the migration action itself (e.g., before the VM is suspended); hence, we denote these events by a "PRE_" prefix (e.g., PRE_MIGRATION_START, PRE_SOURCE_SUSPEND, etc.). Informative events, on the other hand, are fired right after the corresponding migration action is performed.

In the next section we will show how we used the ideas presented in this section in order to allow live migration of a VM across different subnets while keeping a relatively short service downtime.

[1] Obviously, if the migration is blocked for too long then the migration process itself might be affected (e.g., if a PRE_SOURCE_SUSPEND event is blocked for too long then more VM state would have to be transferred after the VM is suspended).

# 4. DESIGN AND IMPLEMENTATION

In this section we present an IP tunnel based solution that enables a live migration of a VM across different IP subnets. We will show how, using the framework presented in Section 3.2 we can optimize the solution and minimize the service downtime.

We demonstrate the advantages of a framework that synchronize migration events with configuration and management tasks through a simple tunneling solution that is loosely based on the principals of Mobile IP. Our solution is tailored specifically for live VM migration. While Mobile IP is a relatively simple technology, many features of Mobile IP are not needed for VM migration. Movement detection is explicit and can be provided by the hypervisors; trust relations with FAs can be assumed *a-priori*; FA discovery can be provided by the target hypervisor (if not the FA itself). Thus, in the context of VM migration, Mobile IP boils down to creating a tunnel between HA and FA (but initiated at both the source and the target, saving RTT), and to providing the relevant L2 triggers (AT, LD, LU).

As mentioned above, the main objective is to allow migration across different subnets with minimal performance degradation during the migrations (e.g., minimize the service downtime). In addition to this basic requirement, we also defined several special guidelines and requirements for our solution:

(a) ***Hypervisor independent.*** The base solution should be independent of any particular virtualization technology (e.g., works on both Xen and VMWare).

(b) ***Transparent to host.*** We do not *require* any changes in the Hypervisor. This property helps achieve a hypervisor independent solution, as specified above. On the other hand, we allow hypervisor-specific extensions, when possible, in order to gain improvements over the base solution. One such extension is the event framework we described – hopefully in the future these events will be exposed by all hypervisors enabling users to take advantage of this mechanism without the need of changing the hypervisor code.

(c) ***Transparent to guest.*** We do not perform any changes to the guests – the guest should not be aware of the solution. This means that not only no special software should be installed on the guest (e.g., Mobile IP support), but also no interaction with the guest is allowed (e.g., changing routing tables on the guest).

(d) ***Automated.*** We use only procedure and steps that can be fully automated and do not require any user interaction.

## 4.1 Architecture and design

A high level architecture diagram is presented in Figure 3. The system is composed of 3 main entities: source subnet agent, target subnet agent, and a Migration Manager.

The Migration Manager main role is to perform all the necessary network configuration setting and changes. In addition, the Migration Manager monitors all the relevant migration events and is in charge to coordinate all the operations needed for the network configuration changes with these migration events. The manager interacts with both source and target tunnel agents and with the source hypervisor (to initiate the migration command), using standard Secure Shell (SSH) sessions.

The main roles of the Subnet Agents are to mange the IP tunnel (i.e., create/destroy), and perform the necessary forwarding and routing to push the relevant packets to the desired destination. This is a very simple task and in practice a regular Linux machine can perform all the necessary tasks. In our implementation for this solution we used a regular SLES 10 machine without any special code.

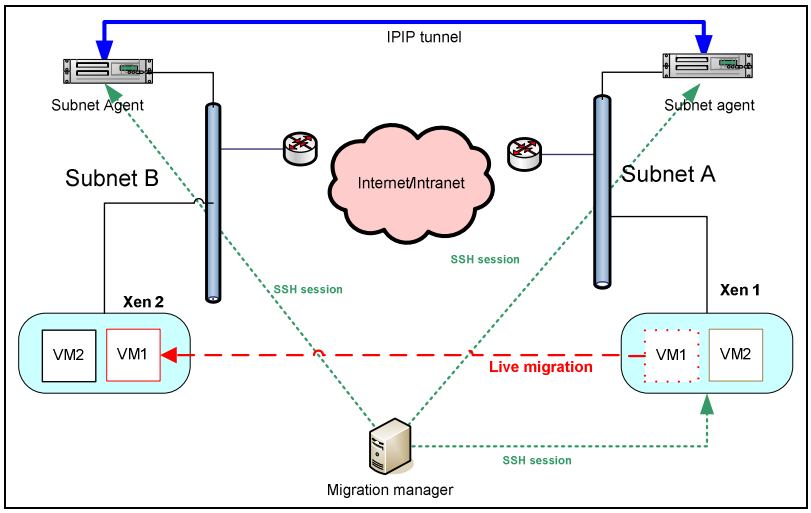


**Figure 3: Solution architecture**

When inspecting the tunneling solution architecture on Figure 3 and comparing it to the current intra-subnet migration topology, we can detect two significant differences that may impact the service downtime. For the intra-subnet case the VM

maintains its MAC address during the migration and therefore only the switch needs to be updated with the new physical location of the VM. In order to reduce the overhead of this update phase, which may be quite long, the VM initiates an unsolicited ARP message announcing its new location. In our cross-subnet tunnel solution, the source agent needs to receive all the traffic on behalf of the VM and forward it through the tunnel to the VM's new location on the target. Since, like Mobile IP, we use L3 tunneling; all the machines which reside on the VM's original subnet need to update the HW address associated with the migrating VM, by replacing the VM's MAC address with the MAC address of the source agent. Normally, this would be accomplished automatically, by setting an ARP proxy on the agent, which responds to ARP requests for the VM address; however, the overhead of such a "rediscovery" may become too big. In the cross-subnet setup, an unsolicited ARP message from the VM will not solve the problem, because the VM moved to a different subnet. The solution we implemented for this problem was to initiate (using our migration manager) an unsolicited ARP from the source subnet agent (which also acts as an ARP proxy for the VM). The Migration Manager monitors all the migration events and therefore can synchronize the ARP message with the appropriate migration event in order to optimize the performance. For example, the Migration Manager can initiate the ARP message as soon as the VM is suspended on the source hypervisor, in practice, "hiding" the update time behind the suspend time of the VM.

The second "update" issue occurs only on the cross-subnet setup – after the VM is brought up in the new subnet, it must send all its traffic through the target agent, in order to communicate with other machines. In order to achieve that, the target agent acts as an ARP proxy for all machines residing in the VM's original subnet (including the gateway). As in the previous case, in order to optimize the performance and reduce the service down time, the agent should initiate an ARP message publishing its MAC address as the address of all machines in the VM's original subnet. This problem does not occur in the intra-subnet case, because the VM stays in the same subnet and communicates with all machines directly; therefore, it does not need to update its ARP table with new MAC addresses. We used the Migration Manager to initiate such ARP messages as soon as the VM is resumed on the target side. Although this minimizes the downtime, we can further decrease the downtime by using the new events of our synchronization framework. A more detailed description of this problem and how the synchronization framework may help is given in Section 5.1.

## 4.2 Implementation details

We divided the network configuration steps into different phases, in order to match them with the relevant migration events, helping to minimize the service downtime. The Migration Manager performs the migration process in three phases:

(a) Pre-migration configuration phase

(b) Migration phase (this includes the actual migration phase on the hypervisor)

(c) Post-migration configuration phase

During the pre-migration phase, the VM is still at the original location (source hypervisor) and therefore does not contribute to the service downtime. Naturally, at this phase we should perform only operations which do not have any effect on the system connectivity. This phase does not contribute to the VM downtime or to the service downtime, as long as it completes before the VM is suspended. It is possible to perform this phase before initiating the migration; alternatively, during this phase, one can register a *blocking* registration for the "VM suspend on source" event.

During the VM migration phase, the VM is suspended on the source hypervisor; any disruptive operations that must be performed while the VM is not running can be scheduled to this phase. Again, a blocking event may be used to make sure the phase is complete before the VM is resumed on target.

The post-migration phase is used for disruptive operations that require the VM to be up and running. Obviously, we should try to minimize this time as much as possible. In the intra-subnet case, one can consider the VM's unsolicited ARP message as part of such a post-migration phase.

**Pre Migration**

The main tasks during this phase are the following:

(a) Build a bidirectional IP over IP tunnel between the source subnet agent and the target subnet agent. This typically requires initiating a tunnel from source to target and another tunnel from target to source.

(b) Create ARP proxies on both source and target agents.

(c) Configure routing rules with appropriate IP forwarding on both source and target agents.

It is easy to verify that all the tasks listed above do not affect the on-going networking operations of the source VM.

**Post Migration**

In this phase we complete the network reconfiguration by actively letting the ARP proxy at the source subnet to take over the VM's address. This is done by broadcasting an unsolicited ARP message from the source agent on behalf of the migrated VM.

## 5. EVALUATION

In this section we present and analyze the performance characteristics of the solution for cross-subnet migration. We also show how the performance can be improved utilizing the migration events, as described in Section 3.2.

In general, when analyzing the performance related to live migration, we can divide the possible impact into two classes:

(a) ***Performance impact during migration***: how the proposed solution affects the normal behavior during the migration process.

(b) ***Performance impact at steady state***: how the proposed solution affects the overall system performance after the migration process finishes and all the configuration changes are over.

The test configuration based on the architecture presented in Figure 3. We used HS20 blades (single Xeon 3*GHz*, 1*GB* RAM), for all our servers (source and target hypervisors, source and target agents). For all the Xen instances we used SLES 10 sp1 (kernel 2.6.16.46) and Xen version 3.0.4_13138, and allocated 256*MB* for DOM0. We used several different network topologies during our tests. For most of the tests, the servers were connected through a switched 100*Mbit/sec* Ethernet network (in the cases where we used a different network configuration, we will indicate this fact in the test description). For the guest virtual machines, we used a SLES 10 sp1 image and allocated 256*MB* of memory. The guest image file was 4*GB* and was located on an NFS server.

### 5.1 Service and VM downtime

In this test we measured the machine and service downtime. The impact of migration (within a single IP subnet) on the service downtime for various applications was already demonstrated and presented in several places ([1],[2]). Therefore, the following tests were mainly aimed on measuring the specific overhead which is directly related to the cross-subnet migration in general and to our specific solution in particular.

We used a simple socket based client server program for our timing analysis. The client opened an SSH session to the server and then, upon connection establishment, the server would sent a packet, every *T* milliseconds, to the client. By monitoring the delays between the messages and the corresponding ACKs, we could estimate the impact of the migration on the service downtime. As we explained in Section 4.1, the impact of cross-subnet migration is not identical for incoming and outgoing messages (i.e., to and from the migrated VM). Therefore, we performed two sets of tests - on the first test, we placed the server on the migrating virtual machine; on the second test, we placed the client on the migrated machine.

In order to get the reference performance numbers, we performed the same client-server test, while the virtual machine was migrated within a single subnet (i.e., both source and target hypervisors were located on the same IP subnet). As it can be seen in Table 1, for the intra-subnet case, we measured an average service downtime of 1290 milliseconds and VM downtime of 621 milliseconds. The differences between our test results and the results from [1], are due to the lower network bandwidth we used (i.e., 100*Mbit/sec* vs. 1000*Mbit/sec*).

We then performed a live migration on top of the configuration presented in Figure 3. During the migration we tried to measure all the relevant migration events (presented in Figure 2). We used TCPDUMP to trace all the network timing on the source hypervisor, target hypervisors, client machine, and server machine. In addition, we hooked into the Xen migration scripts, in order to monitor both the *SOURCE_SUSPEND* (on the source hypervisor) and TARGET_RESUME (on the target hypervisor) events.

**Table 1: Service and VM downtime (seconds)**

| | Cross IP Subnets | Intra-Subnet |
|---|---|---|
| Service Down Time | 1.656 | 1.291 |
| VM Down Time | 0.596 | 0.621 |
| Total Migration Time | 32.243 | 32.837 |

The events occur at different locations and therefore had to be taken from different physical machines. In order to synchronize these measurements into a unified timeline, we used the packets traversed between the hypervisors during the migration as a synchronization point. We identified a specific packet on two different physical machines and used its time-stamp to calculate the clock skew between the machines. We repeat this calculation for several such packets to make sure that this skew does not change much over time.

To measure the VM downtime, we pushed out the timing of the suspend event together with the TCP dump on the source hypervisor, and on the target hypervisor we pushed out the timing of the resume event together with the TCP dump. To get a unified timeline between these two events, we used the last Xen migration message, sent from the source hypervisor to the target hypervisor, as our reference. We measured the duration between this packet to both the suspend and resume points and, since the round trip time of the packet was less than one millisecond, we considered the last migration packet as a good enough synchronization point. As indicated in Table 1, the VM downtime was 596 milliseconds. One can see that the VM downtime during the cross-subnet migration was similar to the downtime in the intra-subnet case (i.e., our reference number). This is an

expected result, because all the tasks performed by our tunneling based solution are performed either before the VM is suspended or after it is resumed, and therefore, should not have any impact on the VM total downtime.

To measure the service down time, we used the client-server application described above with an interval *T=25* milliseconds (i.e., the delay between two successive messages), this interval enabled us to get a good enough accuracy for the downtime measurement. We then used the TCP dump, in order to identify and measure the actual delay corresponding to the migration.

In order to estimate the performance improvements due to the synchronization framework, we first performed an initial test that included just the mandatory configuration changes: creating the tunnels, setting up an ARP proxy on the both source and target agents, and configuring the necessary routing tables on the agents. All these steps could be performed before the VM was brought up on the target hypervisor. Although the solution worked and allowed the VM to get and send messages once the migration was over, the performance, however, was not good and the service downtime was above 10 seconds. Inspecting the TCP traffic indicated that, indeed, this long down time can be mainly contributed to the two "update issues", described in Section 4.1. After we added the unsolicited ARP messages and synchronized them with the migration events, the service downtime reduced to an average of 1695 milliseconds. This result can still be improved by optimizing the update procedure on the target subnet. The problem is that in order to update the ARP table of the migrated VM, the VM needs to be up and running and therefore, in our tests, we tied this operation to the *TARGET_RESUME* event. However, due to some race conditions, this is not the best solution – if the VM tries to send a message as soon as it is brought up and does it before we have a chance to update the VM's ARP table, then the send will fail and we would get a long delay due to the TCP retries. This can be solved by using the special *PRE_SOURCE_SUSPEND* blocking event to first force the VM to stop sending and receiving traffic (for example, by reducing the TCP window size to 0), and only then suspend the VM and move it. This way we can ensure that we can update the VM ARP table before the VM starts sending messages.

## 5.2 Steady-state latency

In this test we wanted to measure the impact of our tunneling based solution on the latency after the VM was migrated to the target hypervisor. There are three main components that may impact the latency:

(a) The packet forwarding at the source and target agents

(b) The IP over IP encoding and decoding at the source and target agents

(c) The extra path that the packet need to traverse.

The overhead due to the extra path is heavily dependant on the topology and specific network configuration (e.g., the number of routers and switches to go through, physical extra length, etc.). This overhead is well known and can be easily estimated and measured (e.g., using the "ping" utility); furthermore, this overhead relates to the route optimization aspects of network migration, which, as mention earlier, we have left for future work. Since we have not implemented any route optimization, the extra path overhead is inherent; therefore, in the following tests, we focused on the other contributors to the latency overhead.

We used a special testing setup in order to isolate only the forwarding and encoding/decoding overheads. The test setup is presented in Figure 4. The setup included 4 blades residing within the same chassis (i.e., all communication went through the internal 1*Gbit* switch). We used PETPERF version 2.4.4 client server application to measure the request/response time between machines A and B, and used the other two blades as the source and target agents (agent 1 and agent 2 in Figure 4).

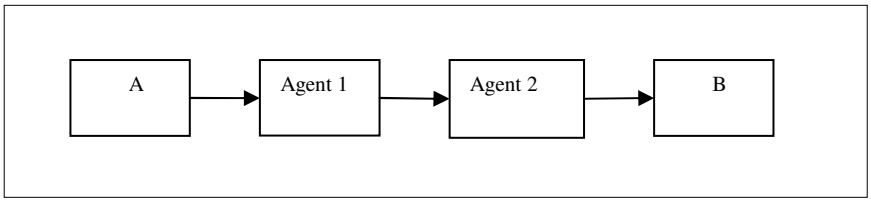


**Figure 4: Latency test setup**

We performed three different tests in order to estimate the overhead:

(a) Traffic goes directly from A to B (agents 1 and 2 did not participate)

(b) Agents 1 and 2 perform only forwarding (no tunnel is used). The traffic goes from A to Agent 1, Agent 1 forwards it to Agent 2, and Agent 2 forwards the packet to machine B.

(c) An IP over IP tunnel was created between Agents 1 and 2. The traffic goes from machine A to Agent 1 and then forwarded through the tunnel to Agent 2, which forwards it to machine B.

**Table 2: Service and VM downtime (milliseconds)**

| | Latency | | | |
|---|---|---|---|---|
| | **Min.** | **Average** | **Max.** | **Stdev.** |
| Direct traffic | 0.104 | 0.146 | 0.214 | 0.018 |
| Forwarding only | 0.294 | 0.329 | 0.44 | 0.034 |
| Forwarding + tunneling | 0.299 | 0.332 | 0.402 | 0.036 |

It can be seen that the total overhead was 0.186 milliseconds (0.332 – 0.146), and most of it was due to forwarding. Similar results were obtained when using simple PING and analyzing the TCPDUMP traffic. We should also note that during all these tests, we could not see any change in the CPU load on the agents. This means that both the forwarding and the IP over IP encoding/decoding were performed quite efficiently.

Although, as indicated above, the total latency overhead is heavily dependent on the specific topology, we wanted to get some rough numbers of the latency overhead on our cross-subnet testing setup. We used the setup presented in Figure 3 and send PING messages to the migrated VM from a machine on the original subnet. For reference we created an additional virtual machine on the target hypervisor (with IP address which belong to the target subnet) and PING it from the same machine, this time the PING went directly to the VM without going through the agents and tunnel.

The average PING time via tunnel was 700 milliseconds, while the average PING time to a VM on the same location, but not through the tunnel, was 390 milliseconds. When combining with the results presented above, we can see that the network overhead due to the extra path on our specific testing system was around 0.2 milliseconds ( 0.2=(0.7-0.39) – 0.186 ).

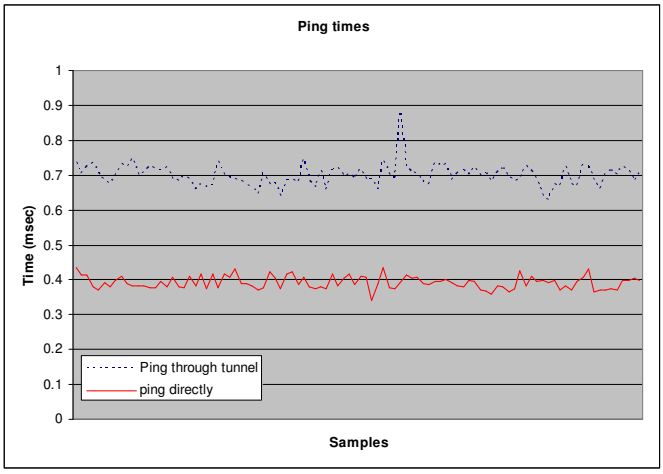


**Figure 5: Latency**

## 5.3 Steady state bandwidth

In this test we measured the bandwidth reduction due to the tunneling and forwarding.

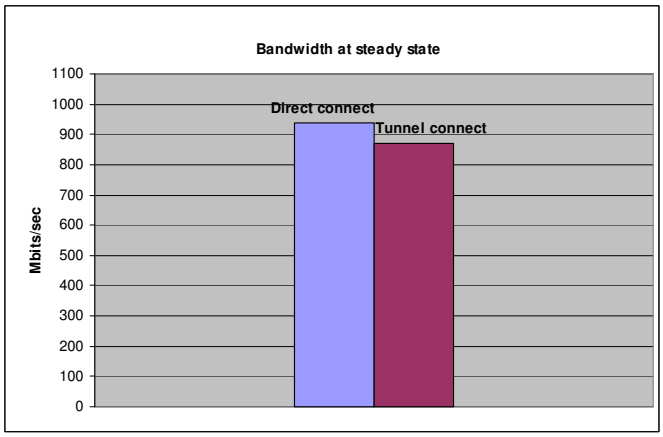


**Figure 6: Bandwidth**

We used the IPERF (version 2.0.2) on top of the testing setup described in Figure 4. When activating the IPERF between A and B directly, we observed a bandwidth of 940*Mbits/sec*, the CPU load on both A and B was 40%. We repeat the same test but this time the traffic went through agent 1 and agent 2; this time the bandwidth dropped to 880*Mbits/sec*. This is a 7.5% reduction in BW due to the overhead of our tunneling solution. Similarly to the latency case, during the test, we observed no change in CPU load on the agent machines. We repeated the tests with different configuration parameters for the IPERF (e.g., different number of IPREF threads) and got similar results.

In this test we measured the BW overhead due to one client using the tunnel. Obviously, if several clients use the same agent and all the traffic is going through a single NIC, it may lead to a reduction in the BW. We should note, however, that once again, this depends on the specific network topology. For example, if all communication between subnet A and subnet B goes through a single gateway capable of X *Mbit/sec*, then an agent capable of X *Mbit/sec* will not have an impact on the BW because it introduces a limitation similar to an already existing BW limitation. In addition, in case there are multiple paths between the two networks, we can use several agents and open several tunnels between the two subnets, once again eliminating the overhead of the agent itself.

## 6. CONCLUSIONS

Live migration of virtual machine is a very powerful feature. It allows sophisticated management operations like load balancing, hardware maintenance (by evacuating all VM's from the physical machine) and more. However, today live migration suffers from several limitations that prohibit users to take full advantage of the technology; most notably, migration is limited to within an IP subnet and requires shared storage. In addition, the performance of the migration in term of the service downtime is still not optimal.

In this paper we introduce a new framework to efficiently support live migration of virtual machine across IP subnets. We investigate existing approaches for IP mobility; namely, the support for seamless movement of mobile nodes between different IP subnets. We show how these approaches apply to live migration of virtual machines, and the special mobility related characteristics of live virtual machine migration. In particular, we incorporated the notion of *mobility triggers*, a key concept in efficient IP mobility, into the migration process. We introduce a new framework that enables the synchronization between network mobility and VM migration. Our proposed framework defines a set of migration events that support both *informative* and *blocking* usage modes. Blocking events demonstrate the unique characteristics of VM migration, as opposed to movement of mobile nodes. We discuss how the events, defined by our framework, can be tied to specific user operations to improve the overall migration process.

Utilizing our framework, we implemented a simple IP tunnel based extension to live virtual machine migration that allows migration across IP subnets (overcoming the current limitation of live migration for both Xen and VMware). We then showed, through analysis and evaluation, how to leverage our framework to improve the migration performance; in particular, we significantly shortened the machine and service downtime during migration.

In this study we used our framework to *enable* connection migration across subnet during live virtual machine migration. We are currently working on leveraging our framework to synchronize between VM migration and advanced storage management (e.g., mirroring and data relocation) in order to eliminate the need for shared storage. We are also working on further improving VM migration by incorporating other mechanisms from IP mobility; specifically, routing optimization.

## REFERENCES


[1] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. "Live Migration of Virtual Machines". In *2nd Symposium on Networked Systems Design and Implementation (NSDI), USENIX,* May 2005.

[2] M. Nelson, B. Lim, and G. Hutchins. "Fast transparent migration for virtual machines". In *Proceedings of the USENIX Annual Technical Conference*, April 2005.

[3] C. Perkins. "IP Mobility Support for IPv4". RFC 3344, *IETF Network Working Group*, August 2002.

[4] D. Johnson, C. Perkins, and J. Arkko. "IP Mobility Support in IPv6". RFC 3775, *IETF Network Working Group*, June 2004.

[5] Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson. "Stream Control Transmission Protocol". RFC 2960*, IETF Network Working Group*, October 2000.

[6] A. Matsumoto, M. Kozuka, K. Fujikawa, and Y. Okabe. "TCP Multi-Home Options". *IETF Internet-Draft, draft-arifumi-tcp-mh-00.txt*, October 2003.

[7] E. Kohler, M. Handley, and S. Floyd. "Datagram Congestion Control Protocol (DCCP)". *IETF Internet-Draft, draft-ietf-dccpspec-06.txt,* February 2004.

[8] A. C. Snoeren and H. Balakrishnan. "An end-to-end approach to host mobility". In *Proceedings of the 6th ACM/IEEE Int. Conf. on Mobile Computing and Networking (MOBICOM-00)*, Boston, MA, 2000.

[9] A. Snoeren. "A Session-Based Approach to Internet Mobility". PhD Thesis, Massachusetts Institute of Technology, December 2002.

[10] D. A. Maltz and P. Bhagwat. "MSOCKS: Architecture for Transport Layer Mobility". In *Proceedings of IEEE INFOCOM*, pages 1037–1045, March 1998.

[11] J. Rosenberg, H. Schulzrinne, G. Caramillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. "SIP: Session initiation protocol". RFC 3261, *IETF Network Working Group*, June 2002.

[12] A. Campbell, J. Gomez, C-Y. Wan, S. Kim, Z. Turanyi, and A. Valko. "Cellular IP". *IETF Internet-Draft, draft-ietf-mobileipcellular-00-txt*, January 2000.

[13] R. Ramjee, T. La Porta, S. Thuel, K. Varadhan, and L. Salgarelli. "IP micro-mobility support using HAWAII". *IETF Internet-Draft, draft-ietf-mobileip-hawaii-00.txt*, June 1999.

[14] Cisco Systems. "Cisco IOS Local-Area Mobility—A Cisco IOS Software Solution to Business Needs to Enable Mobility within the Enterprise Network". White Paper http://www.cisco.com/warp/public/cc/pd/iosw/ioft/lam/tech/lamso_wp.pdf, Cisco Systems, April 2000.

[15] S. Carl-Mitchell and J. S. Quarterman. "Using ARP to Implement Transparent Subnet Gateways". RFC 1027, *Network Working Group*, Oct. 1987.

[16] 802.1Q. IEEE Standard for Local and Metropolitan Area Networks: Virtual Bridged Local Area Networks, 2005.
[17] K. Kompella and Y. Rekhter. “Virtual Private LAN Service (VPLS) Using BGP for Auto-Discovery and Signaling”. RFC 4761, *Network Working Group*, Jan. 2007.
[18] M. Lasserre and V. Kompella. “Virtual Private LAN Service (VPLS) Using Label Distribution Protocol (LDP) Signaling”. RFC 4762, *Network Working Group*, Jan. 2007.
[19] H. Shah, K. Arvind, E. Rosen, G. Heron, and V. Radoaca. "IP over LAN Service (IPLS)". *IETF Internet-Draft, draft-shah-ppvpn-ipls-02.txt*, June 2003.
[20] A. Sundararaj and P. A. Dinda. “Towards virtual networks for virtual machine grid computing”. In *Proceedings of the 3rd USENIX Virtual Machine Research and Technology Symposium* (VM' 04), USENIX, May 2004.
[21] X. Jiang and D. Xu. “VIOLIN: Virtual internetworking on overlay infrastructure”. In J. Cao, L. T. Yang, M. Guo, and F. C.-M. Lau, editors, *ISPA, volume 3358 of Lecture Notes in Computer Science*. Springer, 2004.
[22] E. Gustafsson, A. Jonsson, and C. Perkins. “Mobile IP Regional Registration”. *IETF Internet-Draft, draft-ietf-mobileip-reg-tunnel-02.txt*, March 2000.
[23] K. Malki, P. Calhoun, T. Hiller, J. Kempf, P. McCann, A. Singh, H. Soliman, and S. Thalanany. “Low Latency Handoffs in Mobile IPv4”. *IETF Internet-Draft, draft-ietf-monileip-lowlatencyhandoffs-v4-04.txt, Mobile IP Working Group*, 2002.
[24] R. Koodli. “Fast handovers for mobile IPv6”. *IETF Internet-Draft, Mobile IP Working Group*, 2002.
[25] S. Sharma, Z. Ningning, and C. Tzi-cker. “Low-Latency Mobile IP Handoff for Infrastructure-Mode Wireless LANs“. IEEE Journal on *Selected Areas In Communications*, Vol. 22, No. 4, May 2004.
[26] S. Goswami. “Simultaneous Handoff of Mobile-IPv4 and 802.11”. *IETF Internet-Draft, draft-goswami-mobileipsimultaneous-handoff-v4-01.txt*, 2002.
[27] A. Yegin. "Link-layer Event Notifications for Detecting Network Attachments*". IETF Internet-Draft, draft-ietfdna-link-information-03.txt*. Oct 2005.