Application Mobility with HIP

Teemu Koponen, Andrei Gurtov, Pekka Nikander
Helsinki Institute for Information Technology
Email: firstname.lastname@hiit.fi

***Abstract*— Process migration, or application mobility as we call it, is the act of moving a process or an application between hosts during its execution. It enhances load management, fault resilience, and data access locality. Regardless of its impressive potential, it can't be considered a commercial success due to its currently limited deployment. A reason for the failure is the complexity process migration creates in order to operate transparently from application's perspective. Transparent network communication for migration capable operating systems has traditionally required either custom application or kernel implemented protocols, both breaking general interoperability. Perhaps surprisingly, mobility and multi-homing driven identifier/locator split architecture model provides an elegant solution for this classic problem.**

In this paper, we present an initial architecture for application mobility utilizing perhaps the most promising candidate for the identifier/locator split, Host Identity Protocol (HIP). Our novel use of the protocol introduces new requirements for HIP itself. The paper identifies these requirements and based on them introduces a protocol extension to enhance HIP's suitability for application mobility.

I. Introduction

Process migration is the act of moving a process between hosts during its execution. It enhances load management, fault resilience, and data access locality [1]. In addition to these classic benefits, process migration facilitates modern user demands, namely user session mobility[2], [3] and energy conservation of mobile devices[4].

Regardless of its impressive potential, process migration can't be considered a commercial success due to its currently limited distribution. A reason for the failure is the complexity process migration creates in order to operate transparently from applications' perspective[1].

The term migration itself is rather overloaded. Albeit rarely used nowadays, *host migration* actually refers to host mobility[5]. We shall not use the term further, but instead we refer to mobility if movement of hardware is meant. Service migration[6] is more abstract; it typically refers to relocation of a service from the clients' perspective. No detailed assumptions are made whether the service moves due to process migration or simple stop and restart cycle.

Several techniques akin to the classic process migration require further clarification. *Load distribution* is about balancing the load of a set of hosts. In a system capable of dynamic load distributing process migration supports distribution by moving executing processes from overloaded hosts to less loaded. In *load balancing* a group of servers gets a public virtual address which is actually the address of a load balancer distributing transport layer connections and application layer sessions for the servers. The distribution occurs at the creation time of the connections and sessions[7]. Therefore, load balancing is a static load distribution mechanism as server processes don't relocate themselves while running.

Although not called migration, *checkpointing* is a limited form of process migration; an application is periodically freezed for a short period of time and the operating system (or the application itself) stores application state to a persistent storage.[1] The key difference to the process migration is exactly this clear unidirectional interface of process state transfer; checkpointing copies the state to a file while process migration requires a more complex bidirectional interface between the source and the target host. Enabling application recovery to the last checkpoint is the primary motivation for checkpointing; however, it can be used to relocate a process too. After storing the process state, the process is stopped, its state is transferred to a new host, and a new process is constructed and started based on the state. As the previous implies, differentiating process migration and checkpointing can be rather fuzzy in practice as [8] illustrates.

The final technique worth of mentioning is *code mobility*. It concerns moving the code only, and no execution state is moved[9]. Therefore, an application must restart from scratch on a new host. Although, it must be noted that such applications are usually aware of the possibility of being relocated and similarities to checkpointing may be significant.

Encouraged from the fuzziness between the checkpointing and the process migration, we raise the level of abstraction and redefine process migration to refer to *application mobility* instead. This avoids us from fixating ourselves to a certain specific technique to relocate a running application. Thus, a reader familiar with the classic process migration should slightly re-orientate herself while reading the rest of the paper.

In this paper we present an initial application mobility architecture exploiting Host Identity Protocol (HIP)[10] to solve the classical communications transparency problem of process migration. Therefore, instead of building a protocol from scratch as early process migration research often did (e.g., [11]), we use a more generic protocol for solving this specific problem. HIP introduces a layer into the TCP/IP protocol stack between transport and network layers. This together with our strict focus in communications aspects only enables us to clarify our definition of process migration further; we define the transferred process state to also include the state of transport layers and layers above that. Therefore, session mobility (such as the architecture presented in [12]) is

[1]In a way, checkpointing introduces transactional properties to process state; a checkpoint can be seen as a commit and recovery as a rollback. A resemblance with database transactions is crisp.

not our principal interest. In fact, process migration should be rather transparent to the layers above the HIP layer.

There seems to exist a relationship of hardware mobility and application mobility. One could consider application mobility as a form of multi-homing with rather limited communication facilities between multi-homed host's physical network interfaces; in the end the result is same, a remote peer has changed its preferred address. However, the relationship and its consequences are rather vague, at least for us. Thus, in this paper we also attempt to deepen our understanding of this relationship. We crystallize our two-staged research question as follows:

1) What kind of implications does HIP have to network communications of process being migrated and can HIP offer at least a partial remedy to a classical problem of process migration, namely, transparency of communications?
2) What implications the above has to HIP i.e., does the above create new requirements for HIP?

The paper is structured as follows. In Section II, we analyze architectural issues related to improving the process migration communication transparency. Section III presents a detailed design of the architecture. Section IV draws conclusions.

II. Architectural issues

This section attempts to identify the essential forces affecting to the architecture and its need to support two key functions: communication during migration and resolution of addresses after migration.

A. Delegation

End-points identities are assumed to remain the same for the whole lifetime of HIP associations. We liberate ourselves from this current restriction and shall next consider the implications if HIP had support for changing of end-point identities without breaking an association between end-points from applications' or transport layer's point of view.

In our scenario the source entity *delegates* responsibility of its HIP end-point identity to the destination entity. As a result, the client end-point assumes the destination entity to *represent* the same end-point that was earlier in the source host, even its end-point identity has changed. This sounds rather simple, at first at least.

The consequence is the same as if the identity were moved. All processes bound to the end-point being delegated, must be migrated to avoid an extra relay in the communications. In fact, it may be stating obvious but at this point, when both identity moving and delegation have been considered, it seems one cannot migrate a subset of an end-point without causing residual dependency. Clearly, an entity, which all applications bound to the end-point belong to, is the smallest unit of mobility even if application mobility is introduced. Clark et al. have claimed the same [13].

Maintaining full transparency in the communications API was identified to be a challenge. Changing the host identity challenges this; clearly, the semantics of the API change if we were to allow the end-point identifier to change together with end-point identity from processes' point of view. After all, an application must be able to depend on the static properties of a local binding of a socket, at least with the de facto standard BSD socket interface[14].

At this point, an option to guarantee transparency for the processes is address translation. In other words, let HIP association delegation and application mobility together trigger address translation in both hosts to hide the migration from applications referring to a changing end-point identity. The HIP base specification[15] already identifies a similar need, in the context of supporting legacy applications, though. Therefore, no new functional component is required as such and the implementation is likely to be trivial. In fact, the current HIP socket API proposal probably simplifies the task even further due to its extra indirection layer protecting applications from future protocol stack changes[16].

B. End-point resolution

Until now, we haven't considered the problem of address resolution after process migration. In the following, we shall analyze address resolution from both end-point identity movement and delegation point of view.

Moving an end-point and its identity seems to morph to the regular late binding address resolution process of HIP; location update procedure must be executed to responsible rendezvous servers to guarantee their ability to redirect the initiators' first packets to the current location of an end-point. Moving an end-point seems not change to the address resolution at all from regular multi-homing.

However, if the responsibility for an end-point identity is delegated, the scenario becomes slightly more complex as relocation of a network attachment point is not the only change in the association end-points have; the identifier of the (service) end-point also changes. HIP DNS extensions[17] become problematic, as DNS can't anymore be expected to have the identity of a service in touch, even though it were updated during the migration. Hence, avoiding opportunistic HIP handshake becomes difficult. The issue shall be discussed in more detail later.

We stress that the service can still be *contacted* after the migration by means of rendezvous mobility support[18], if its rendezvous information is properly managed. However, end-point's identity can't be verified to correlate with the service name. In fact, the security would drop below the current model of IP addresses with dual meaning[13]. Clearly, the delegation decision of the source host should be more persistent and it should even leave auditable traces for future association creations. A method for the destination host would be to store the delegation order and hand it for anyone trying to connect the destination host with an old host identifier.

Even the delegation order should be semi-persistent[2], its validity period must be finite. A finite validity period protects

[2]With semi-persistent we mean the order should remain available as long as migrated application is running; therefore, storing to a truly persistent storage, such as a hard drive, is unnecessary.

source hosts from security breaches of its destination hosts; without the finite validity period, with a stolen end-point identity an attacker could reuse the delegation order and impersonate a source host even the migration and the current DNS would not contain the stolen destination end-point identity. The choice of ending time shall be discussed shortly. Clearly, a finite validity period requires also a start time; the current time is acceptable for this purpose. The enforcement of the validity interval renders an unfortunate side-effect; peers must have a real time clock that is in sufficiently right time. Moreover, the global nature of Internet also requires the timestamps in coordinated universal time (UTC) format, and thus, the peers must have time zone information too.

As identified earlier, the migration may occur multiple times sequentially. In other words, the current home of a process may have to store and present several delegation orders. Fortunately, the required storage time is finite as it's ultimately defined by the time DNS stores outdated information of a service. Sequential migrations have an implication regarding rendezvous servers too; a mechanism is needed to enable foreign end-points to update the current IP address of an end-point.

III. Design

This section introduces the architecture instantiated based on the key issues identified earlier; it shall focus the earlier discussion into a specific architecture. As pointed out in the introduction, we shall emphasize the communications aspects while presenting the architecture.

A. Overview

Next we shall be introduce main functionalities and a component of our process migration architecture capable of communication transparency:

- *Address resolution enhancements* to HIP are required to guarantee secure end-point verification;
- Maintaining *application transparency* induces internal protocol stack modifications; and
- *A migration component* is responsible for suspending, transferring, and resuming a process state.

B. Address resolution

The end-point identity responsibility delegation approach creates two new challenges regarding the HIP association establishment after a process migration. First, how do initiating end-points resolve a DNS name of a migrated end-point to its current IP address? Second, even if the resolving could be done, how can the migrated end-point show its migration history in a trusted manner to the HIP association initiators?

A semi-persistent method to indicate the delegation is required to solve the latter challenge. The source of the application being migrated should *certify* the new end-point identity of the application to act on behalf of itself using its end-point identifier. Certification is required as the HIP end-point identifier, called as a Host Identifier Tag (HIT), is a hash of end-point identity's public key. In other words, once the end-point identity changes, one can not use a hash-function to validate the relationship of an end-point identifier and a end-point identity. Therefore, we require the source of the process to sign an "end-point impersonating certificate"[3] for the destination, which can then include it to HIP R1 packets while responding to handshake requests directed to the source. Once included, the destination may use the source end-point identifier in HIP headers, as after processing the certificate there is a valid relationship between the end-point identifier and the host identity of the destination. Figure 1 depicts the delegation of an end-point identifier.

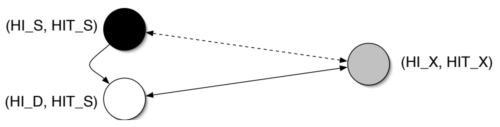

Fig. 1. A certificate (a curve) authorizes a destination (a hollow circle) to use the end-point identifier of a source (a black circle) for the communication with another host (a grey circle).

As the process migration may occur multiple times sequentially, we do not restrict the number of certificates as such, but instead, require initiators to support construction of a trust chain using the certificates from the end-point identifier they intended to connect to the end-point identifier they got a response from. From performance point of view, it's obvious the introduction of the certificates raises the processing resource consumption in initiating peers; as the length of the trust chain increases, the consumption of resources increases in a linear fashion. The exact side-effects are vague at this point of time.

Rendezvous servers principally exist to evade the limitations of DNS; namely, the inability of DNS to remain up to date with rapid information changes. In its basic form, a rendezvous server merely redirects initiators I1 packets to a current IP address of an end-point.[4] Thus, the rendezvous model of HIP fits nicely to the application mobility scenario as the certificates limit the problem of proving the delegation not to be a problem of a rendezvous server.

Two options exist to inform rendezvous servers about the relocation of an end-point, and thus, to solve the first challenge. Either a source of the application executes the location update procedure or a destination executes it. We are for the latter, as it resembles more closely the way transparency is maintained for applications (discussed later); implementations are likely to benefit from the similarity. An extension needed is one to enable the new end-point of a process to readdress the old end-point of the process to the new address.

We require the readdressing end-point to include necessary delegation certificates to a I1 packet it sends to a rendezvous server while establishing a rendezvous association with the server. Including only the delegation certificates obtained

[3]These certificates have nothing to do with X.509v3 certificates; instead, SPKI certificates[19] are a closer match.

[4]Certain limitations to the scope of the paper are imminent; therefore, we shall consider only the basic rendezvous scenario, that is, the redirection of HIP I1 packets.

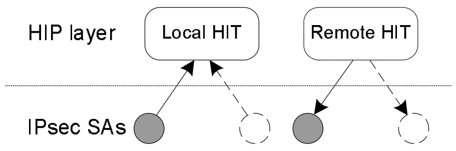

Fig. 2. The mapping replacements a destination host causes within a remote peer after a successful HIP association establishment. A solid arrow represents a SA to HIT (or vice versa) mapping, while a dashed arrow represents the new mapping after the new HIP base exchange. The remote HIT refers to the HIT a remote source and a remote destination share.

since the last rendezvous readdressing is necessary. Thus, in practice, I1 packets to rendezvous servers contain only a single certificate and the construction of the trust chain remains lightweight. Once the certificates have been included, the HIP header may contain the end-point identifier of the source end-point as above.

We shall next consider the modifications the above delegation requires to a protocol stack to maintain transparency from upper layer's point of view.

C. Protocol stack modifications

IPsec Security Parameter Index (SPI) [20] provides a compression of HITs [10]. In other words, within an end-point a SPI translates to a HIT, and vice versa. This enables HIP to multiplex and demultiplex between IPsec Encapsulating Security Payload (ESP) traffic and their respective HITs. Therefore, a HIP capable end-point has sort-of address translation in place by default. Thus, as we are speaking of the required functionality to maintain transparency for layers above HIP, nothing would prevent us to modify the SPI and HIT relationships. In such a model, a destination host would contact without attaching signed certificates to a handshake. Thus, the destination would not appear as a representative of the source for a remote application until the parameter sent by the source host had been processed and mappings were altered use IPsec Security Associations (SAs) established with the destination host.

However, the above sounds a bit hackish; an end-point identity is effectively changed after the association establishment. Architecturally it is more sound to let a destination represent a source already during the creation of an association; in this way delegation extensions highlight the symmetrical nature of HIP associations[10]. As an additional benefit, it is identical with the way destinations inform rendezvous servers.

Once a destination end-point end-point includes the certificates to the handshake, from a remote peer point of view there could be several HIP associations with the entity represented simultaneously by both a source and the destination host. However, as HIP end-points are expected to share a single HIP association, the new IPsec SAs established with the destination simply replace the old SAs with the source within a remote peer[10]. This renders the application mobility completely transparent for the layers above HIP within the remote peer. Figure 2 depicts the association replacement. As shown in the figure, IPsec SAs are unidirectional. Thus, actually two replacements occur within a remote peer: one to replace the mapping of a HIT to an outgoing SPI and one to replace the mapping of an incoming SPI to a HIT.

The above discussion applied to the remote peers not involved in the migration directly. For the destination host of the migration, it is easier: establishment of HIP associations together with the certificates to remote peers guarantees transparency for the layers above HIP. However, as the destination and source host share an end-point identifier, an unfortunate side-effect develops. If the destination were, accidentally, to choose the same SPI the source has assigned for a remote peer, from the remote peer's point of view, separation of the SA to the source and the SA to the destination would become impossible. This could potentially introduce problems within remote peers. Therefore, the destination and the source end-point must coordinate to prevent usage of the same incoming SPIs. It's convenient to let this communication about SPIs to happen out-of-band from HIP point of view. We feel that this is insignificant requirement, as the hosts involved in the migration must have a more complex migration protocol in place anyway.

D. Migration implementation

In the classic process migration research, devising an protocol to transfer process state with minimal freezing time and maximal execution throughput, once resumed, has been an art of its own[1]. However, we confine ourselves to a simple migration protocol as our focus is in the communications aspects. The fundamental nature of the migration protocol requires a rather secure communication channel between a source and a destination host; after all, a transferred process state is subject to contain critical information. Moreover, as we wish not to limit mobility and multi-homing capabilities of the hosts involved, we are prone to select HIP as a basis. In this manner, the actual migration protocol may focus in plain process migration aspects only.

However, integration of HIP to a full Single System Image (SSI) migration environment such as [21] seems an overkill and unnecessarily complex at this point, while a plain checkpointing approach seems rather limiting from supported applications point of view. Checkpointing can't provide full transparency for applications; and as a result, applications depending on operating system allocated identifiers were outside of the scope. In [8] Osman et al. introduce a concept of *process domain*, which is a collection of processes that potentially depend on each other. Instead of migrating a process, they migrate a checkpointed process domain, which contains a virtual and static view for processes in the domain. Our future implementation shall build on top of this sound hybrid model combining full transparency and simplicity.

In our architecture an end-point actually refers to a process domain and not to an individual process. From the migration protocol point of view the requirements migration of communications renders are still the same. The protocol requires only a decent amount of message exchanges:

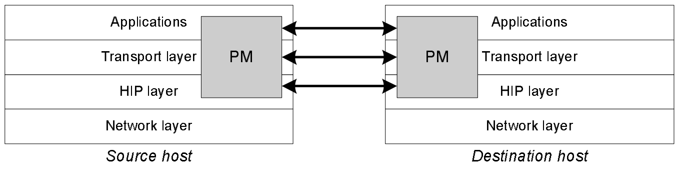

Fig. 3. The relationship of the application mobility architecture and TCP/IP protocol stack. The arrows represent the different protocols the process migration implementation (PM) needs between a source and a destination host. They are from bottom to up: HIP, TCP, and a custom process state transfer protocol.

1) A source transfers its delegation order (and the possible previous orders it has received) to a destination;
2) Together with the orders, the source transfers location and identity information of current remote peers of the process domain to the destination;
3) The destination acknowledges the transfer with a set of SPIs pointing to SAs it is about to establish with the peers;
4) The source acknowledges the SPIs, if they do not create conflicts, and the destination establishes HIP associations with the remote peers[5]; and
5) The source transfers a checkpointed process domain state including the state of transport layer sockets relevant for the migrated process domain.

Despite the above simplicity, the architecture has to integrate into rather many internal kernel components of an operating system. Figure 3 depicts the relationships from TCP/IP stack point of view. In addition, the implementation has to provide the process domain for applications, and thus, to virtualize basically the whole system call interface of the kernel. We aim to use existing code of [8] to manage issues not directly related to the TCP/IP protocol stack.

At this point of research we cannot introduce the process state transfer protocol in more details. Moreover, cross-layer aspects are too blurry to comment about in detail. However, it seems attractive to provide hints e.g., to TCP sockets about an occurred migration rather than only copy a TCP state as such from a protocol stack to another.

IV. Conclusion

We believe that the HIP architecture integrates with application mobility almost seamlessly. Application mobility requires only modest enhancements to the HIP protocol suite to obtain full communications transparency without any residual dependencies due to communications itself. To provide for application mobility, we proposed adding a new *delegation primitive* to the HIP protocol. Clearly, further research is required to understand the wider implications of the delegation primitive. Finally, we feel that this work also shows the importance of supporting multiple end-points within a host.

[5]In the worst case, a few additional message exchanges are introduced while agreeing about SPIs. However, as the SPI space is rather large, the probability of several negotiation messages is non-existent.

References

[1] D. S. Milojičić, F. Douglis, Y. Paindaveine, R. Wheeler, and S. Zhou, “Process Migration,” *ACM Computing Surveys*, vol. 32, no. 3, pp. 241–299, 2000.
[2] R. A. Baratto, S. Potter, G. Su, and J. Nieh, “MobiDesk: Mobile Virtual Desktop Computing,” in *Proceedings of the 10th Annual International Conference on Mobile Computing and Networking*. Philadelphia, PA, USA: ACM Press, Oct. 2004, pp. 1–15.
[3] K. A. Bharat and L. Cardelli, “Migratory applications,” in *Proceedings of the 8th Annual ACM Symposium on User Interface and Software Technology*. Pittsburgh, PA, USA: ACM Press, Nov. 1995, pp. 132–142.
[4] A. Rudenko, P. Reiher, G. Popek, and G. Kuenning, “Saving Portable Computer Battery Power through Remote Process Execution,” *Mobile Computing and Communication Review (MC2R)*, vol. 2, no. 1, pp. 19–26, Jan. 1998.
[5] F. Teraoka, Y. Yokore, and M. Tokoro, “A Network Architecture Providing Host Migration Transparency,” in *Proceedings of the 1991 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. Zurich, Switzerland: ACM Press, Sept. 1991, pp. 209–220.
[6] H. Balakrishnan, K. Lakshminarayanan, S. Ratnasamy, S. Shenker, I. Stoica, and M. Walfish, “A Layered Naming Architecture for the Internet,” in *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. Portland, OR, USA: ACM Press, Aug. 2004, pp. 343–352.
[7] M. Rabinovich and O. Spatscheck, *Web Caching and Replication*. Addison-Wesley, Dec. 2001.
[8] S. Osman, D. Subhraveti, G. Su, and J. Nieh, “The Design and Implementation of Zap: A System for Migrating Computing Environments,” in *Proceedings of the 5th USENIX Symposium on Operating System Design and Implementation (OSDI 2002)*. Boston, MA, USA: ACM Press, Dec. 2002, pp. 361–376.
[9] A. Fuggetta, G. P. Picco, and G. Vigna, “Understanding Code Mobility,” *IEEE Transactions on Software Engineering*, vol. 24, no. 5, pp. 342–361, 1998.
[10] R. Moskowitz and P. Nikander, “Host Identity Protocol Architecture,” IETF draft-ietf-hip-arch-00, June 2004.
[11] M. F. Kaashoek, R. van Renesse, H. van Staveren, and A. S. Tanenbaum, “FLIP: An Internetwork Protocol for Supporting Distributed Systems,” *ACM Transactions on Computer Systems (TOCS)*, vol. 11, no. 1, pp. 73–106, 1993.
[12] M. A. C. Snoeren, “A Session-based Architecture for Internet Mobility,” Ph.D. dissertation, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, Feb. 2003.
[13] D. Clark, R. Braden, A. Falk, and V. Pingali, “FARA: Reorganizing the Addressing Architecture,” *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 4, pp. 313–321, 2003.
[14] D. E. Comer and D. L. Stevens, *Internetworking with TCP/IP, Vol. III: Client-Server Programming and Applications – BSD Socket Version*, 2nd ed. Prentice Hall, Mar. 1996.
[15] R. Moskowitz, P. Nikander, P. Jokela, and T. Henderson, “Host Identity Protocol,” IETF draft-ietf-hip-base-01, Oct. 2004.
[16] M. Komu, “Application Programming Interfaces for the Host Identity Protocol,” Master’s thesis, Helsinki University of Technology, Telecommunications Software and Multimedia Laboratory, Sept. 2004.
[17] P. Nikander and J. Laganier, “Host Identity Protocol (HIP) Domain Name System (DNS) Extensions,” IETF draft-ietf-hip-dns-00, Oct. 2004.
[18] J. Laganier and L. Eggert, “Host Identity Protocol (HIP) Rendezvous Extensions,” IETF draft-ietf-hip-rvs-00, Oct. 2004.
[19] C. M. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylönen, “SPKI Certificate Theory,” IETF, RFC 2693, Sept. 1999.
[20] S. Kent and R. Atkinson, “Security Architecture for the Internet Protocol,” IETF, RFC 2401, Nov. 1998.
[21] A. Barak, O. La’adan, and A. Shiloh, “Scalable Cluster Computing with MOSIX for Linux,” in *Proceedings of Linux Expo ’99*, Raleigh, NC, USA, May 1999, pp. 95–100.