A Flexible Framework for Complete Session Mobility and its Implementation

Marc Barisch, Jochen Kögel, and Sebastian Meier

Institute of Communication Networks and Computer Engineering
Universität Stuttgart
Pfaffenwaldring 47, 70569 Stuttgart, Germany
{marc.barisch, jochen.koegel, smeier}@ikr.uni-stuttgart.de

Abstract. Users with several devices need a convenient mechanism to transfer running service sessions from one device to another device. This paper proposes a framework that allows session mobility without modifications on the communication partner's system from application layer down to network layer. That means we can transfer ongoing sessions with minor interruptions of the communication and thus call it complete session mobility. Due to the framework's flexibility we support a multitude of technologies across all layers. The architecture has been verified by a prototype that has been implemented on a Linux system.

1 Introduction

Nowadays a user owns several devices that are connected to the Internet and that can be used to consume services. Currently, these devices are independent of each other, which decreases the usability from the user's point of view. In particular, a user cannot easily transfer ongoing service sessions from one of his devices to another one. For example, a user wants to exploit the diverse capabilities of his devices and transfer a running video session from the TV at home to the mobile phone when he leaves the house. Moreover, the limited battery power of mobile devices motivates the need to transfer service sessions between devices. If one device runs out of power the service session can be transferred to another user device and the user can continue to use the service.

In order to accomplish the transfer of ongoing service sessions between devices of a user, we have to design a suitable framework. Such a solution should also take the following requirements into account:

High degree of flexibility: The solution has to cover a wide range of different applications. We target to support multimedia sessions, like voice, and video calls, web sessions and online games.

No involvement of communication partner: Existing solutions, e.g. based on SIP [1] require the exchange of signaling messages with the communication partner, i.e. the service provider, to transfer a service session between devices. This requires that the communication partner supports such session transfers. In addition, the communication partner is explicitly informed about the session transfer, which can be critical for privacy-aware users.

We propose a flexible and pluggable framework that allows session mobility. This framework takes into account that a service session consists of application state and communication state, which need to be transferred between devices. Since application state is very specific we demand that a service supports well-defined interfaces for state export and import. Regarding the communication state, which exists between the user's device and a communication partner, we need to redirect the traffic and to re-establish the corresponding states on the destination device. For redirecting the traffic we employ mobility techniques on the network layer. The re-establishment of the network state requires the extraction of information from the operating system's (OS) network stack of the network layer as well as of the transport layer.

The remainder of this paper is structured as follows. Section 2 provides an overview on the terminology used throughout this paper. Afterwards Section 3 presents the related work for the architecture proposed in Section 4. The partial implementation of the architecture is elaborated in Section 5. Finally, we discuss the open issues of our architecture in Section 6.

2 Terminology

Since we propose a layered session mobility framework based on mobility techniques on the network layer, we introduce the relevant terminology first. Based on a well-established mobility terminology introduced in Section 2.1, we elaborate on our cross-layer session concept in Section 2.2.

2.1 Mobility Terminology

The term mobility is widely used in communication networks with different meanings. In our understanding, mobility is "the ability to use services, irrespective of changes of the location and technical equipment" [2]. For its realization, suitable mechanisms have to be in place to support service usage despite of changes of network connections or change of terminals.

Specifically, we can distinguish three fundamental types of mobility according to the moving object: terminal mobility, personal mobility, and session mobility. Terminal mobility allows to maintain the network connection if the terminal is moving, i.e. keep the network address if the terminal changes its location. With user mobility, a user can consume services in the same way independently of the chosen terminal. E.g., a user can change the terminal and is still reachable with the same identifier. With session mobility, a user can continue to use an application session, even if he changes the terminal.

Orthogonal to the mobility types we classify mobility approaches according to the extent of continuity: services or sessions are either stopped before and continued after movements (discontinuity), or continue to be active during the movement (continuity). In the latter case, the movement can either lead to noticeable impact (non-seamless handover) or unnoticeable impact (seamless handover).

Our solution aims at session mobility in conjunction with user mobility. This means that we want to keep network connections with its identifiers persistent and move running application sessions from one device to another. Depending on the application and network or device performance, the continuity will go as far as enabling seamless handover. We call this complete session mobility.

2.2 Session Terminology

In order to transfer a session between two user devices we need to define a session concept. Fig. 1 shows that a service session consists of one application session at least. In case of non-communicating services, e.g. applications running on a user device without interaction across the network, or services that only communicate temporarily, there is no need for a communication session.

An application session is made up of at least one application state part. Several application state parts can exist to distinguish for example between control and media flows in case of VoIP communication. An application state part in turn can have associations to communication flows that make up the communication session. A communication flow needs to be identified, e.g. by the so called 5-tuple that consists of source and destination IP address, source and destination port and transport protocol. For example in case of TCP also a flow state exists. Moreover it might have an associated security context, e.g. to reflect a TLS [3] association. Communication flows of different communication sessions might depend on each other. For example in case of a security association on the network layer between two end systems all flows depend on each other.

The introduced model shows that we need solutions to transfer the application state and the state contained in the transport and network layer. Therefore, Section 3 introduces relevant proposals that have influenced the design of our framework.

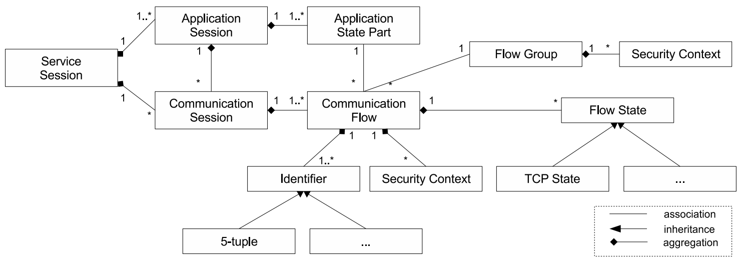

Fig. 1. Session Model

3 Related Work

For the transfer of service sessions we have investigated relevant technologies on the network layer that allow the redirection of traffic (c.f. Section 3.1), transport layer concepts (c.f. Section 3.2) and concepts that support session mobility on the application layer (c.f. Section 3.3). Moreover, we discuss several integrated approaches in Section 3.4.

3.1 Network Layer Mobility

At the design time of IP, hosts were static and IP address changes were rather seldom. Therefore, IP addresses are used to identify a host as well as its location in the network. With the advent of mobile devices, there is a need for solutions that allow the seamless change of IP addresses, and thus the redirection of the traffic to a different topological location, during a running communication session. A common approach on the network layer is to decouple the duality of the IP address and to introduce separate identifiers for hosts and their location in the network. Akyildiz et al. [4] provide an overview on mobility solutions on the network layer.

Basically we can distinguish two different approaches to support mobility: Tunnels and Signaling. Tunnel approaches [5] use a fixed anchor point in the network which redirects the traffic to the current IP address of the mobile device. With Signaling approaches [6, 7] an explicit end-to-end signaling protocol is used to inform the the communication partner of the IP address change.

In Mobile IP (MIP) [5] without route optimization all traffic sent from the communication partner is forwarded via a tunnel by the home agent to the mobile node. Thus, the home agent serves as a fixed anchor point for the mobile node and maintains the mapping to the current IP address. Moreover, MIP supports an explicit signaling between the mobile node and the correspondent to overcome the drawbacks of triangular routing.

Approaches based on end-to-end signaling are Shim6 [6] and the Host Identity Protocol (HIP) [7]. Both approaches introduce some kind of additional layer between the transport and the network layer to eliminate the dual-use of IP addresses. This allows changing the IP address based on the exchange of signaling messages without interrupting an ongoing communication session. Both solutions have in common that the communication partner has to support the protocol. In contrast to Shim6, which solely operates on existing IP addresses, HIP introduces additional host identifiers to identify hosts.

All introduced approaches fit in our framework in order to redirect network traffic from one user device to another. In case of Mobile IP we need to update the binding of the home address, whereas in case of HIP and Shim6 we need to signal the update of the IP address and to transfer the corresponding state.

3.2 Mobility on the Transport Layer

Two mobility approaches can be identified on the transport layer: multihoming and device transition. Multihoming provides persistent transport connections despite changing IP addresses. Various techniques for supporting explicit handoff signaling in transport protocols are presented in [8]. In most cases such solutions are based on protocol extensions that need to be supported by the communication partner. Instead of extending the transport layer protocol [9, 10] introduce an additional session layer between the application and the transport layer. Such solutions require the introduction of additional addressing concepts.

Beyond IP address changes, device transition enables the migration of transport connections between devices. We can differentiate solutions that extend existing protocols, e.g. TCP [11], and solutions without any involvement of the communication partner. This is done by extracting, manipulating and transferring OS internal transport protocol state information. Several sample implementations for Linux exist, such as SockMi [12] and tcpcp [13] that allow the transfer of TCP sockets. Since our framework is intended to support device transition, we focused on the latter approach for transport layer mobility.

3.3 Application Layer Mobility

Session mobility on the application layer has been thoroughly considered by different research communities. The SIP community has addressed the need to transfer a multimedia session from on device to another device based on the existing SIP protocol [1].

For the OS community it has been important to migrate complete processes [14] between different devices. This includes the transfer of the complete code and stack of the running process. We restrict ourself to the extraction of the application state via well-defined interfaces. This allows to support heterogeneous applications complying with a defined interface standard.

3.4 Integrated Approaches

Abeille proposed the Virtual Terminal concept [15] that supports session mobility with mobility solutions on the network layer in order to redirect traffic between different devices. In contrast to our architecture, the concept does neither consider the transfer of the application state nor the consequences on the transport layer. Similar to [15] is the approach proposed by [16] that is based on HIP. They have focused on extensions for HIP to transfer an HIP association between different hosts and claim that the framework can be extended to complete session mobility.

4 Architecture

4.1 Overview

Fig. 2 shows the architecture of our pluggable framework for session mobility. It consists of one central component per device, the Relocation Manager (RM), which coordinates the session transfer between two devices. In order to transfer a session it extracts all the required session state from so called adapter components via a generic interface A. Available adapter components register with the RM. Basically, three different kinds of adapter components can be distinguished

– **Network Adapters:** The network adapters turn commands via the A interface into technology specific actions in order to redirect the network traffic from one device to the other device. For example in case of HIP, a HIP network adapter it is required to transfer the corresponding security association from one device to the other device and to trigger an update of the IP address to redirect the traffic.

– **Transport Adapters:** The transport adapters extract transport layer specific information from the OS. In case of the TCP transport adapter all necessary state information is extracted from the OS in order to re-establish the socket on the other device.
– **Application Adapters:** On the application layer, the application adapters take care of the state extraction from the applications. An application needs to provide an application adapter specific interface for state export and import.

A session transfer is initiated and controlled via the Controller component that connects to the RM via the *U* interface.

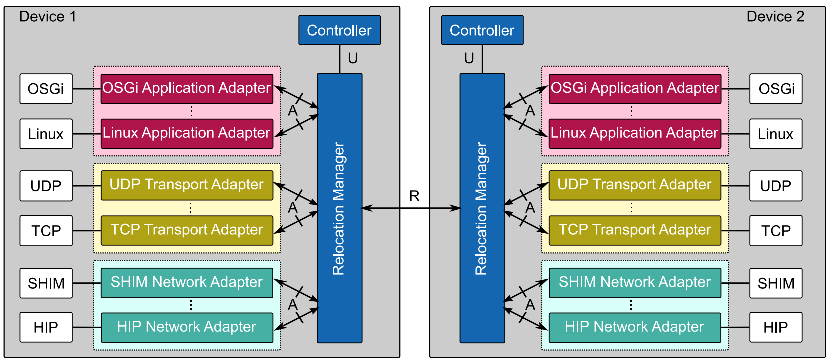

Fig. 2. Overview on the architecture

4.2 Definition of Interfaces

Adapter Interface - *A* The generic interface *A* between the adapters and the RM is used to import and export state from the network, transport and application layer. In our architecture, *A* provides the following primitives.

– *Suspend*: Freezes layer dependent state information which is going to be transferred from the exporting device (ED) to the importing device (ID).
– *Resume*: Unfreezes state information. An ID may resume work when called. Furthermore, an ED may resume work in case of session migration failures.
– *Shutdown*: Instructs an adapter to free all resources upon successful session migration.
– *Get Resources*: Queries for resources used by a network stack component.
– *Reserve Resources*: Checks and reserves resources for further use.
– *Get State*: Exports internal state information.
– *Set State*: Imports internal state information.
– *Do Presignaling*: Triggers signaling between ID or ED and peer device before transferring the session.
– *Do Postsignaling*: Triggers signaling between ID or ED and peer device after successful session migration.

Relocation Manager Interface - *R* The R interface is defined between two RMs. It provides primitives to control the session transfer from ED to ID.

- *Check Preconditions*: This primitive is used to verify whether it is possible to transfer a session in advance of the actual session transfer. It is needed to check whether sufficient resources are available on the destination device.
- *Initiate Session Transfer*: Initiates the actual session transfer and transports the hierarchical state description from the ED to the ID.
- *Activate Session*: After the session state has been transferred, the ED triggers the activation of the session on the destination device.

With these three primitives it is possible to transfer a session from one device to another device. This is elaborated in Section 4.3.

For the description of the state we defined an hierarchical exchange format as illustrated in Fig. 3. It reflects the session model shown in Fig. 1. We distinguish the transfer of the application state, which is assumed to be a binary large object (BLOB), the transport state that reflects in most cases serialized structures out of the OS kernel, and the network state. Thus, the complete state description looks like this:

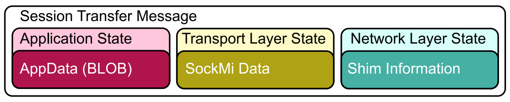

Fig. 3. Exchange format for session transfer

All message exchanges via the R interface are security critical, because they might contain sensitive information. Before the transfer of a session between devices can take place, devices need to mutually authenticate each other. Moreover, the messages exchanged need to be encrypted to prevent eavesdropping.

Controller Interface - *U* The controller interface provides the corresponding controller component with information on the availabe sessions and allows for the initiation of session transfers. The primitives of this interface are subject to further study. Currently a user has to trigger the corresponding transfer manually.

4.3 Message Flow

For the transfer of sessions between user devices, two message flows are essential. First, a mechanism is required to discover user devices that are available for session transfer, which is discussed subsequently. Second, the message flow for the actual session transfer is elaborated.

Device Discovery In order to know which devices are available for session transfer we assume for simplicity a centralized approach: All user devices register themselves with a central repository and update the registration periodically.

The central repository serves as anchor point and must always be available. It allows a device to retrieve a list of all currently available devices and present it towards the user, who then triggers the session transfer. This mechanism could be enhanced based on service discovery protocols like Service Location Protocol [17]. Additional improvements could use beacon mechanisms or exploit geographic information in order to propose only close-by devices for session transfer.

Session Migration Fig. 4 illustrates the message flow to transfer a session between the ED and the ID. For simplicity only one adapter is depicted. When the RM of the ED receives the *Export* command for a session, e.g. $S1$, it first sends a *Suspend* message to the adapter to freeze its internal state before obtaining information on the required resources. Afterwards the RM of the IM is contacted to check whether it has sufficient resources to take over the session. If the check is positive, the resources are reserved and confirmed. The *Do Presignaling* primitive provides the flexibility for additional steps that have to take place before the actual state can be extracted with *Get State*.

Afterwards, the complete session state is transferred from ED to ID, which subsequently imports the state into the corresponding adapters. If *Set State* is successful, both RMs have the possibility to trigger additional steps with *Do Postsignaling*. Finally, the EM's RM activates the session and releases the remaining resources with *Shutdown*. The in Fig. 4 depicted session migration delay t_{Mig} has to be short in order to minimize the packet loss.

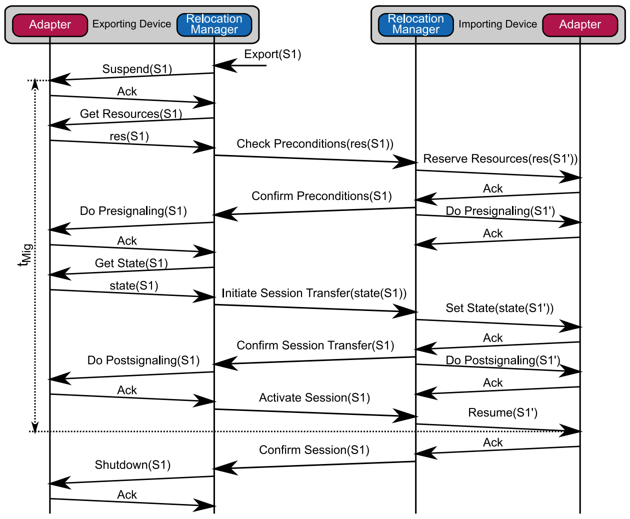

Fig. 4. Session migration message flow

5 Implementation

We implemented the above introduced architecture on a Linux system running kernel 2.6.24. The implementation consists of three different adapters and the

RM. Since we have not considered complex logic that triggers session transfers, user action to manually trigger session transfers is needed. On the application layer, a Linux application adapter allows the transfer of special adapted Linux applications. Such applications have to provide an interface that allows the import and export of the application state. For demonstration purposes we have implemented a simple echo server that provides the needed interfaces.

If the application has open TCP sockets, we are in the position to transfer the corresponding TCP transport layer state by using the SockMi [12] implementation in the TCP adapter. The transfer of open TCP sockets also requires the redirection of the corresponding traffic stream. We achieve this by exploiting the SHIM protocol and extended its implementation LinShim [18] to manipulate locator pairs.

Fig. 5 provides additional details on the implementation of the RM and the generic adapter design. The RM consists of an *Adapter Coordinator* that manages and communicates with the registered adapters based on the coordination by the *Session Administrator*. Data obtained from the adapters is (de-)serialized by the *Export* and *Import Dispatcher* in order to transport it by the R interface across the network.

The modular design of the adapters consists of functional blocks that are responsible for managing the resources required, the import and export of state, and the explicit signaling that needs to be triggered with the adapter backend. All activities are coordinated by the *Relocation Assistant*.

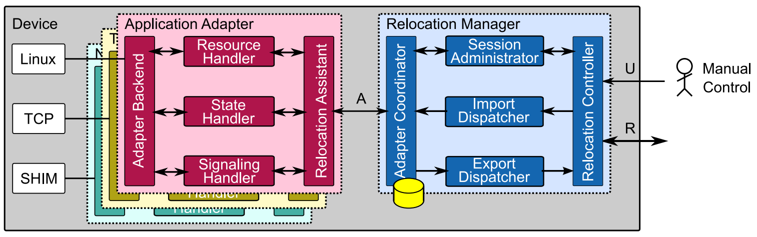

Fig. 5. Implemented architecture

6 Summary and Conclusion

Our approach for complete session mobility moves running applications between devices while maintaining network connections. This includes transferring all state of open connections (e.g. TCP) between devices, so that the application can continue to use the connections as on the old device, as well as switching network identifiers between devices (e.g. with MIP, HIP).

We proposed a framework that collects state from applications and the network stack, moves it to a different device, and reestablishes the state required to continue the session there. The framework is kept flexible with generic interfaces for pluggable adapters to the specific mobility protocol or applications. We proofed the applicability of this approach by implementing a prototype with adapters for SHIM6 and TCP on Linux.

While our first tests are promising, some issues still have to be solved. First, switching IP addresses in active TCP connections is uncommon in today's IP networks. Thus, devices in the network that track the connection state (firewalls, NAT) cannot associate address switched TCP packets with connections and will block traffic. This could be resolved with appropriate signaling. Second, we considered transfer of TCP state with Linux only, while transferring TCP state between operating systems with different stack implementation demands for suitable transformation of state information in order to work correctly. Moreover, we need to conduct thorough performance measurements on the prototype and elaborate on security issues regarding the transfer of sessions between devices.

Acknowledgments

This work was supported in part by the European Union under the FP7 programme (SWIFT project).

References

1. Schulzrinne, H., Wedlund, E.: Application-Layer Mobility Using SIP. Mobile Comp. Comm. Rev. **4** (2000) 47–57
2. ITU-T: ITU-T Recommendation Q.1706/Y.2801 (2006)
3. Dierks, T., et al.: The Transport Layer Security (TLS) Protocol Version 1.1. RFC 4346 (April 2006)
4. Akyildiz, I., et al.: A Survey of Mobility Management in Next-Generation All-IP-based Wireless Systems. IEEE Wireless Communications **11**(4) (August 2004)
5. Johnson, D., et al.: Mobility Support in IPv6. RFC 3775 (June 2004)
6. Nordmark, E., Bagnulo, M.: Shim6: Level 3 Multihoming Shim Protocol for IPv6, draft-ietf-shim6-proto-12.txt. IETF Internet draft (February 2009)
7. Moskowitz, R., Nikander, P., Jokela, P., Henderson, T.: Host Identity Protocol. RFC 5201 (April 2008)
8. Atiquzzaman, M., et al.: Survey and Classification of Transport Layer Mobility Management Schemes. In: IEEE PIMRC 2005. Volume 4., IEEE (September 2005)
9. Suri, N., , et al.: Mockets: A Comprehensive Application-level Communications Library. In: Proc. IEEE Military Communications Conference MILCOM 2005. (17–20 Oct. 2005) 970–976
10. Landfeldt, B., et al.: SLM, A Framework for Session Layer Mobility Management. In: Proc. Eight International Conference on Computer Communications and Networks. (11–13 Oct. 1999) 452–456
11. Sultan, F., et al.: Migratory TCP: Connection Migration for Service Continuity in the Internet. In: Proc. 22nd International Conference on Distributed Computing Systems. (2002) 469–470
12. Bernaschi, M., et al.: SockMi: A Solution for Migrating TCP/IP Connections. In: Proc. 15th EUROMICRO PDP 2007
13. Almesberger, W.: Tcp connection passing. In: Proceedings of the Ottawa Linux Symposium 2004. (2004)
14. Milojičić, D.S., et al.: Process Migration. ACM Comput. Surv. **32**(3) (2000)
15. Abeille, J., et al.: MobiSplit in a Virtualized, Multi-Device Environment. In: IEEE ICC 2007, Glasgow, Scotland
16. Koponen, T., et al.: Application Layer Mobility with HIP. In: ICT 2005. (2005)
17. Guttman, E., et al.: Service Location Protocol, Version 2. RFC 2608 (June 1999)
18. Barré, S.: LinShim6 - Implementation of the Shim6 Protocol. Technical report, Université Catholique de Louvain (2008)