OPEN Project

STREP Project FP7-ICT-2007-1 N.216552

Title of Document: Final communication and context management solution for migratory services

Editor(s): Rasmus L. Olsen

Affiliation(s): Aalborg University

Contributor(s): Miquel Martin (NEC), Carmen Santoro (ISTI-CNR), Björn Schindler (ClU), Simone Mazzei (Vod), Anders Nickelsen (AAU), Kim Højgaard-Hansen (AAU), Huan Cong Nguyen (AAU), Sergio Labeaga (AAU), Ahmed Shawky (AAU), Hans Rune Bergheim (AAU), Rasmus L. Olsen (AAU)

Affiliation(s): Institute of Information Science and Technologies, Italy (ISTI-CNR), NEC Europe Ltd./Heidelberg, Germany (NEC), Aalborg University, Denmark (AAU), Vodafone, Italy (Vod), Clausthal University, Germany (ClU)

Date of Document: February, 2010

OPEN Document: D3.4

Distribution: EU

Keyword List: architecture, design, migration platform, reference

Version: 1.0

OPEN Partners:

CNR-ISTI (Italy), Aalborg University (Denmark), Arcadia Design (Italy), NEC (Germany), SAP AG (Germany), Vodafone Omnitel NV (Italy), Clausthal University (Germany)

EXECUTIVE SUMMARY

This deliverable document the work carried out in Work Package 3 and is the final deliverable for the work package. Deliverable 3.1 which was written after one year, focused on state of the art, initial requirement analysis, and initial solution drafts. This deliverable contains the final outcome of the Migration support platform which covers Context Management, Migration Orchestration, Trigger Management, Mobility Support and clock/flow synchronization. Further, additional study on Quality of Service and Network performance monitoring has been done which are located in the appendices.

The deliverable starts by giving an introduction and overview of how it relates to other parts of the project, and gives an overview of the support platform and how it works as seen from a high level. This should give the reader a good understanding on how the various modules work together, and how the rest of the report is structured. This is followed by detailing the results of each module, where there is allocated one chapter per module.

The deliverable should be read in conjunction with Deliverable 3.3 which contains details on how the prototypes that has been developed are setup and working. Whenever relevant, proper referencing to other deliverables (and external resources) is given.

A lot of work has been carried out throughout Work Package 3 activities, and a lot of experiences have been gained with respect to supporting migratory services. However, there are still integration tasks to be done in Work Package 4 and evaluation of the platform which will be carried out in Work Package 6 in the remaining project period.

TABLE OF CONTENT

1. INTRODUCTION TO OPEN ARCHITECTURE

1.1. INTRODUCTION

An important aspect of ubiquitous environments is to provide users with the possibility to freely move about and continue the interaction with the available applications through a variety of interactive devices (including cell phones, PDAs, desktop computers, digital television sets, and intelligent watches). With such freedom envisioned one big potential source of frustration is that people have to start their session over again from the beginning at each interaction device change.

The purpose of the OPEN project is to enable migration of services and applications, so that users or systems may reconfigure applications to operate in a suitable way to the given circumstances, for example shifting a video stream from a mobile phone to a larger display in order to utilize the possibility of seeing the stream on a larger screen, changing input modality from a mobile phone to a keyboard to make typing easier, or shifting among different networks to utilize the best of multiple possibilities, all without having to stop and restart the application itself. The concept is summarized as follows

Migration = Changes + Adaptation + Continuity

Changes refer to the differences applied in the configuration and operational behavior of the application needed to work correctly before, during and after the reconfiguration of application and/or the execution environment. *Adaptation* relates to the reconfigurations required for the application to utilize resources effectively, whereas *Continuity* relates to the desired functionality of an application to continue without the user to restart his/her application usage, even after changing the condition on which the application works, e.g. running completely or partially on a different device than originally started or with a different network configuration.

The work presented in this deliverable is focused on the technical details on a support platform that enables applications and services to migrate by offering a set of functionalities needed to carry out the process. This relieves the application developer from reinventing code that is used to achieve same functionality, making it easier to develop new applications that may migrate as needed.

In Deliverable 3.1, [OPEN D3.1], the outcome of the first year work of Work Package three were presented. This included an overview of already existing technology and their strong and weak points, as well as an introduction to the initial platform support system, its functionality and interfaces. This was well supported by a set of prototypes demonstrating the work and described in Deliverable 3.2, [OPEN D3.2]. In Deliverable 3.3, [OPEN D3.3], we presented the final outcome of prototypes developed in Work Package 3, and should be read in conjunction with this deliverable as a reference to what has been implemented. A very brief overview of the content of Deliverable 3.3 is given in Appendix A for easier references, however, details should be found in Deliverable 3.3 instead of Appendix A. Parts of those prototypes in Deliverable 3.3 are also parts of the integrated demos, which therefore also acts as a documentation of the link between work package 3 and the rest of the project.

In this final deliverable, Deliverable 3.4, we present the final architecture and output from Work Package 3. The work has been carried out in close collaboration with other technical work packages, namely Work Package 2 and 4 which are focused on other aspects of the migration process, namely user interface and application reconfiguration. Feedback with respect to the overall system architecture, as well as technical requirements have been fed into Work Package 1, while also having received feedback from Work Package 4 in terms of the integration process and adjustments to interface specifications. The interaction with Work Package 5 has been focused on the actual application support interface, and the collaborative work done here has resulted in three proof-of-concept applications that utilize the platform developed in Work Package 3. Finally, output from Work Package 3 is planned as input to Work Package 6 where performance study will be carried out to get first indicators of the support systems efficiency in terms of defined performance metrics, e.g. timing, network overhead, etc. Some results will, however, be presented in this deliverable, but those are focused to individual components of the platform, whereas results showing in Work Package 6 relates to the platform as a whole.

1.2. OVERVIEW OF THE PLATFORM

An overview of the OPEN Migration Platform and system architecture was presented in Deliverable D1.4 [OPEN D1.4], and is for convenience presented also here in Figure 1.

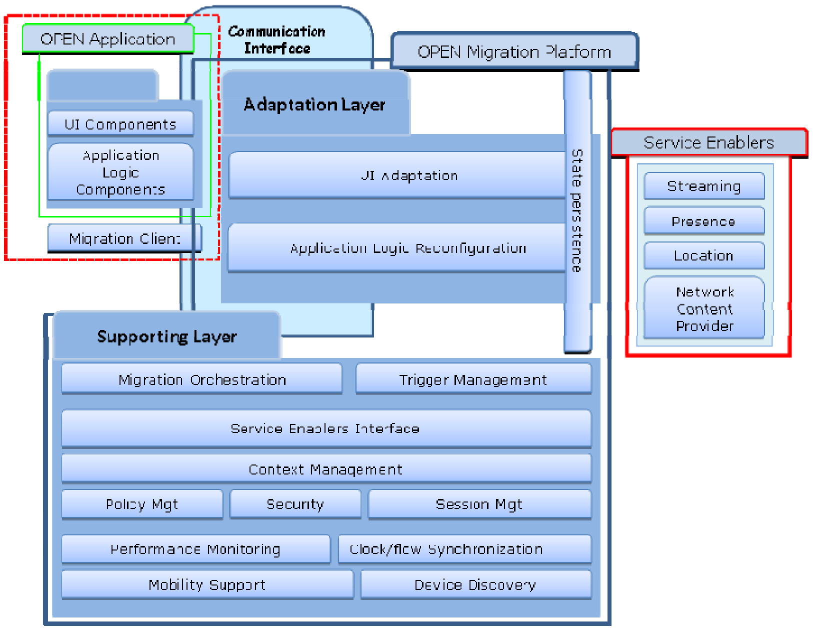


Figure 1: Functional diagram of the OPEN Migration Platform as defined in [OPEN D1.4].

As seen, the OPEN Migration Platform contains two major groups of functionalities, namely the Adaptation Layer and the Supporting Layer. The groups of functions in the Supporting Layer are implemented in a set of modules which are described later in Section 1.3, wherefrom the structure of the report is based. Within these two groups, the required functionalities needed by applications and services to easily migrate are located. Work Package 3 is focused on the Supporting Layer functionalities. In the following, a short overview of the functionality of the Supporting Layer is presented:

Context Management

The situation of the user, user's activity, the network state or other information that describes the situation of potential candidate devices for target migration, are good indicators on when and where to migrate and is the basis for the decisions taken in the Trigger Management. The Context Management system in the platform ensures easy access for other modules, applications and services to distributed, dynamic information of various types within the network. The Context Management is explained in Chapter 2.

Migration Orchestration

At the heart of the platform there is the Migration Orchestration, which is a function that ensures the migration process proposed by the Trigger Management is being executed. It orchestrates the involved entities in the process, for example the application by starting, stopping and initiating state transfer as needed in a timely correct manner. This is explained in Chapter 3.

Trigger Management

The Trigger Management is responsible for proposing a migration scenario to be executed. It bases this proposal on various information and knowledge about the current context which the user, network and potential involved devices are in. The Trigger management is explained in Chapter 4.

Policy management

Policies are used and enforced in various locations of the OPEN platform, e.g. in the Trigger Management module for making checks if a proposed migration is against or acceptable to a user, or in the Context Management system where the sharing of context information may need an additional privacy control check before being distributed. Policy Enforcement is therefore actions that take place in various components, in particular the Trigger Management, while Policy Management is a process of distributing the policy descriptions to relevant network nodes. In effect, the Context Management Node (see Chapter 2) with its distributed storage capability is used as a policy manager. In most cases when we discuss Policy Enforcement we mean the enforcement which is carried out in the Trigger Management, i.e. see Chapter 4, unless otherwise specified.

Security

The application or service may contain sensitive and personal data within its internal states potential for migration, which should not be migrated to the wrong devices; hence security mechanisms are needed. Security cannot be treated as a single module or functionality, but is treated within individual module as to address the various requirements and challenges to the respective module. The chapters found within this document, describing the various migration functionalities, contain a specific section that addresses the security concerns and solution for the specific module.

Mobility support

In scenarios where devices may change network as a part of the migration process, e.g. due to congestions in the currently used network or due to mobility, there is a need for the support of making this change transparent to the application server (and client) in order for them not to require a reconfiguration of e.g. IP addresses. This is addressed in Chapter 5. Further, a comparison study of how SIP can be applied is provided in Appendix B.

Session Management

Special care for the network connection between applications is needed to keep the session uninterrupted. The task of the Session Management is to ensure that active sessions are not lost due to the change of device or network, both before, during and after the migration process. This functionality is closely linked to the mobility support, whereas this topic is addressed along with the mobility support in Chapter 5.

Clock/Flow synchronization

Applications which have parallel data streams such as some types of video, sound, games or real time traffic may require synchronization between the data streams of the target and source device(s). Further, clock synchronized devices will be beneficial for not only this process, but also for the Performance Monitoring as it allows to measure bi-directionally delay. This topic is investigated in Chapter 6.

Device Discovery

A key factor for a migration to happen is that the devices are aware of each other, which capabilities they have and what applications and components and functions are available on the device. The Device Discovery ensures this information is available to the relevant devices. This functionality is explained along with the Migration Orchestrator (due to its close interaction with this module) in Chapter 3.

The network is a limited resource, and blindly triggering a migration may lead to undesired effects such as congestions or increased packet loss rate (migration from wired to wireless network). Performance Monitoring is used for monitoring the state of the network and provide valuable information to make decisions on the time of migration in the Trigger Management. An overview of the possible methodologies for performance monitoring is provided in Appendix C. Further, a close linked topic, namely Quality of Service is important to ensure that the end user gets a user pleasant experience of the migration process; a comparison study of different approaches to achieve QoS is provided in Appendix D.

Furthermore, in Appendix A, a brief overview of the various application prototypes are presented to ensure a complete overview of where and how the platform supports services. Further details are found in Deliverable 3.3 [OPEN D3.3] (for details on what has been implemented) and Deliverable 5.3, [OPEN D5.3].

Finally, the deliverable concludes the work done in Work Package 3 as presented here and provides an outlook on supporting platforms for migratory services may or should look like.

1.3. OVERALL PLATFORM FUNCTIONALITY AND INTERACTION

In this section we provide the basic network setup and assumptions that we have set for the OPEN Platform. We introduce the major network entities that constitute the platform and provide an overview of where various functionalities exist in the platform and on what devices. Finally, we also provide a rough overview of how a simple migration process will work with this structure. A more comprehensive interaction and component interface specification can be found in [OPEN D4.2]. It should be noticed that depending on the applications, these scenarios may vary, but with the provided example we argue for where and why functions are needed in the platform.

1.3.1. GENERAL NETWORK ARCHITECTURE AND ENTITY DEFINITIONS

The OPEN Platform assumes a set of entities connected through an interconnecting infrastructure that allows the necessary remote control and mechanisms to be executed between the devices involved in the migration process. The architecture is shown in Figure 2, which shows there are two major OPEN specific entities in the infrastructure domain, namely an OPEN Mobility Anchor Point and the OPEN Server. Furthermore, OPEN specific client software must also be running on the OPEN clients in order for the OPEN Server to be able to fetch needed information and control the application client.

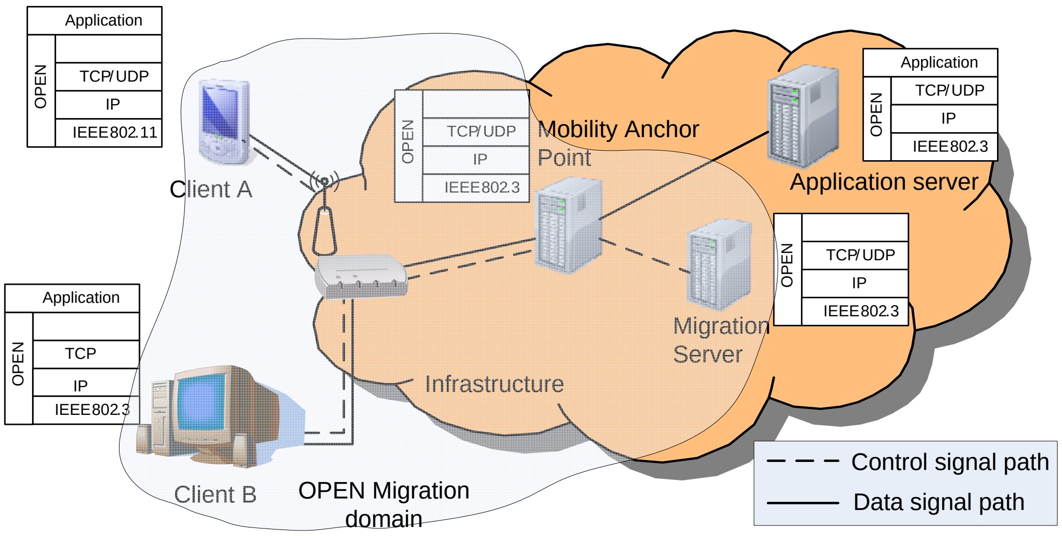


Figure 2: Network architecture for the OPEN Migration Platform.

The architecture puts no assumptions on the application server in order to keep the impact on the migration process to the application developer at a minimum. Instead, via the OPEN Mobility

Anchor Point (MAP) and OPEN Server, the migration process is constrained to a domain that includes the clients, MAP and OPEN Server entities, but excluding the application server. The definitions of the various entities found and discussed are listed in the following table.

OPEN Client	**Clients execute the applications. Applications executed on the clients can be OPEN Platform-aware or not aware.**
OPEN Server	**An OPEN specific network element that supports the migration between clients.**
OPEN Mobility Anchor Point	**An OPEN specific network element that ensures data streams at transport layer are directed to the correct device, transparent to the application server.**
Application Server	**Any type of application server that provides services to client applications.**

In Deliverable 3.1 we discussed various types of networks which could be envisioned as a basis for the migration process, namely:

- Network operator / Cellular networks
- Enterprise networks
- Home networks
- Ad-hoc / Peer-to-peer networks.

Focus in OPEN has been on scenarios where the MAP and OPEN Server have fixed locations in the network and available to the clients at all times. This is not the case in ad-hoc networks for example where connectivity between the key entities is not always guaranteed, in enterprise networks where firewalls may become a challenge or in home networks were NAT problems have to be addressed in order to perform the operations necessary to carry out the migration process. We did not focus on such issues, but rather on refining the existing architecture and go in depth with the support platform components to achieve a complete understanding of their functions, interfaces and interaction with the rest of the project components.

In Figure 3 an overview of the modules implementing the support layer functions of Figure 1 is presented. A module may implement one or more of the functions shown in Figure 3, which will be made clear in the respective section throughout the deliverable.

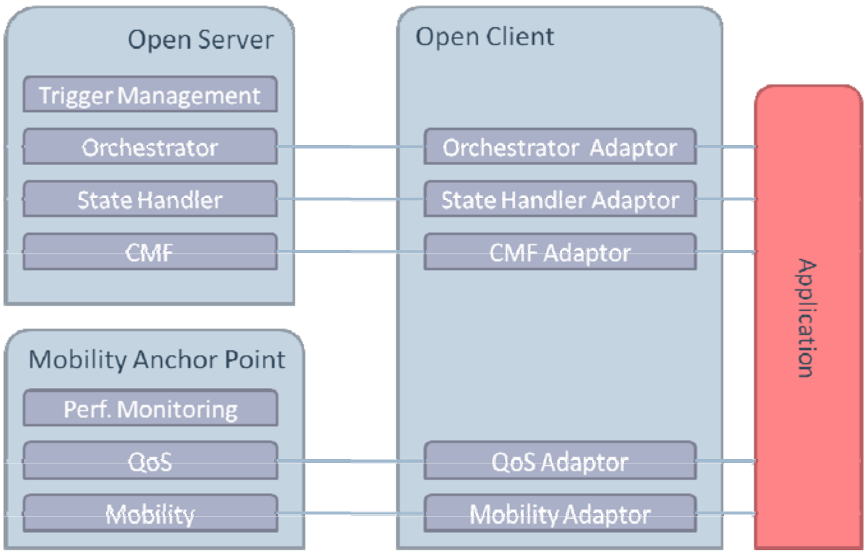


Figure 3: Modules distributed in the OPEN platform on the different entities which are the focus of this deliverable.

As it can be seen, there will be several modules running on the different entities within the OPEN domain. Since the main development platform is Java, using a complete JVM to run each component intuitively leads to a large processing overhead. Therefore modules can with benefit be run in an OSGi environment [OSGi], whereas several independent modules can coexists in one JVM. Interaction between modules becomes easier because they will share same Java environment (e.g. class path). Further, some modules as the Context Management Framework will benefit from the flexible environments OSGi offers in terms of life cycle control for internal components of the module (e.g. retrievers and processing units). All this will be described in details in Section 2.1.

The remaining part of the deliverable will be structured according to this picture, in the sense that each of the modules shown in Figure 3 will be elaborated in subsequent chapters. However, before that, an overview of how the Support Layer collaborates internally with the rest of the OPEN Platform is provided.

1.3.2. OPEN MIGRATION PLATFORM INTERACTIONS

This section provides an overview of how the entities in the OPEN Platform are supposed to interact by describing a full lifecycle of the whole OPEN platform. It should be noticed that the description here only provides a high level interaction, and some applications do not necessarily require all steps described here. The more detailed interaction between the entities and components is specified in the respective chapter, except for the ALR (Application Reconfiguration Logic) and UI Adaptation which is not the scope of this deliverable, but described in [OPEN D4.1] and [OPEN D2.2], respectively.

Figure 4 shows how the core OPEN entities interact with the client devices and the users of the OPEN platform. The interaction diagram assumes all entities are familiar with the IP address of the Migration Server, e.g. via DNS lookup, service discovery or a static configuration.

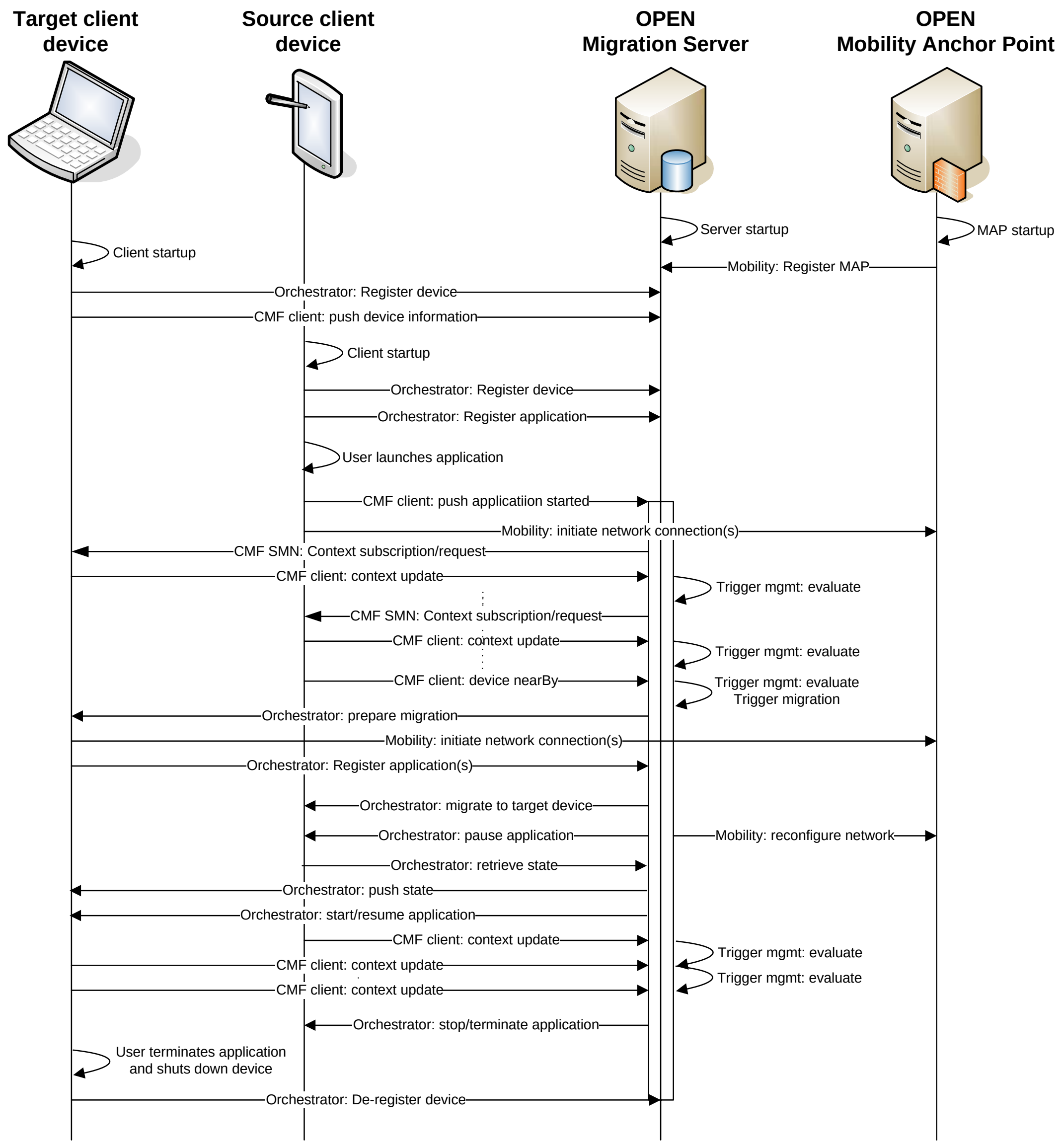

Target client device
Source client device
OPEN Migration Server
OPEN Mobility Anchor Point
Server startup
MAP startup
Client startup
Mobility: Register MAP
Orchestrator: Register device
CMF client: push device information
Client startup
Orchestrator: Register device
Orchestrator: Register application
User launches application
CMF client: push applicatiion started
Mobility: initiate network connection(s)
CMF SMN: Context subscription/request
CMF client: context update
Trigger mgmt: evaluate
CMF SMN: Context subscription/request
CMF client: context update
Trigger mgmt: evaluate
CMF client: device nearBy
Trigger mgmt: evaluate
Trigger migration
Orchestrator: prepare migration
Mobility: initiate network connection(s)
Orchestrator: Register application(s)
Orchestrator: migrate to target device
Orchestrator: pause application
Mobility: reconfigure network
Orchestrator: retrieve state
Orchestrator: push state
Orchestrator: start/resume application
CMF client: context update
CMF client: context update
CMF client: context update
Trigger mgmt: evaluate
Trigger mgmt: evaluate
Orchestrator: stop/terminate application
User terminates application and shuts down device
Orchestrator: De-register device

Figure 4: Generic high level interaction of the migration process.

The OPEN Server is first started and initializes all internal modules after which the OPEN Mobility Anchor Point can be started and registered at the OPEN Server. With these two major entities running, the OPEN Clients can utilize the network by having the client orchestrator registering with the OPEN Server orchestrator. The information required registered includes device capabilities, OPEN enabled applications and their respective modules (if any).

When the application starts, a startup event will be pushed from the source client to the OPEN Server. This will make the Trigger Management start evaluating different evaluation parameters. Trigger Management will periodically evaluate the available context information obtained via the Context Management Framework. If another OPEN Client devices becomes available for migration and the Trigger Management decides that migration is beneficial to the user, the orchestrator at the OPEN Server will make the target device prepare for migration. This includes fetching the required application if it is not already present on the device, establishing a connection to the OPEN Mobility Anchor Point, and registering the application and components with the OPEN Server. Then the orchestrator will extract the application state from the source client device and transfer it to the application on the target device after which the application can be resumed on the target client device. The migration process can happen multiple times with multiple client devices and multiple users involved. At some point the user and client devices can choose to leave the OPEN Platform again by de-registering.

Transparent to these activities, are clock/flow synchronization module which ensures that multiple data streams are synchronized via time stamping the data streams and clock synchronizing the involved entities. The details are described in Chapter 6. Quality of Service is also considered for the purpose of prioritizing of data streams for improved experience of the migration process, e.g. if the state to be transferred is huge, this can be given high priority. Appendix D gives details of what mechanism can be used to achieve QoS. Finally, Performance Monitoring runs in the background, ensuring measurements of the network and link quality is provided to the Context Management Framework which in turn may be one of the metrics used to trigger a migration. Appendix C gives details on the different approaches for performing this monitoring of the network state.

As mentioned, the migration process is not always following these exact steps. For example the state transfer may not be necessary or be so complex it involves several activities, or the target device may not be visible for the OPEN Server before much later than shown in Figure 4, whereas it makes no sense to do the registry at the time shown in the figure, but only when it becomes visible. Therefore, detailed interaction diagrams for application specific cases are found in Deliverables 4.4 and 5.4, which at the time of writing is not yet available.

2. CONTEXT MANAGEMENT ARCHITECTURE

Context Information describes the situation of any entity and constitutes the information that is relevant to the entity itself and its interaction with the environment, [Dey00]. For the migration process, context information is a key indicator on when and how to migrate applications and services. Which information depends on the entity and its activity, whereas getting the information is a process that reuses a set of common functionality which are discovery and access to dynamic information distributed in the environment and accessed via a network infrastructure, being either wired, wireless or both. The common functionalities are search, discovery, access, distribution, caching, and defines context management. Thus context management relieves the application developer from these activities and allows him to focus on the usage of context information, [Bauer06]. The Context Management Framework (CMF) originates from the IST project MAGNET Beyond, where the core components of the module were developed. In OPEN we develop this framework even further whereas the key advances are

- Migration to an OSGi environment
    - Core components capable of handling the dynamicity of components arriving and departing at any random point in time
    - Easy updates/change of core components, e.g. device dependent storage module
    - Component life cycle management (concept only)
- Added functionality to processing manager to cope with multiple processing units
- Development of new retrievers and processing units to produce context information (see Deliverable 3.3, [OPEN D3.3]) required by the prototypes

In this chapter we describe the Context Management Framework for OPEN, first by introducing the concept and core functionality, followed by a description of how the framework will work in an OSGi environment. Later in the chapter we provide details on how the OSGi functionality can be exploited for a highly dynamic reconfiguration of the framework, otherwise not so easy achieved, is described. Since context management relates to access of remotely located dynamic information we also provide some insight into the reliability of the access to context information and how this knowledge can be used to tune systems for certain trade off decisions. In this section we will focus on location information as an example, both because it is a very common context information type but also because we directly use location in terms of a distance in our use cases, [OPEN D3.3]. Finally, we give a brief overview on how security is addressed for context management. Thus the Context Management system has a central role in the OPEN Platform. Figure 5 shows how it relates to the respective parts described within this deliverable.

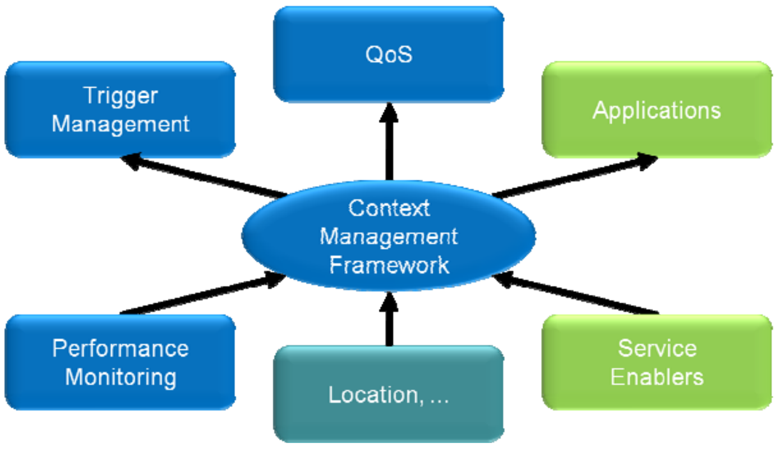


Figure 5: Relations between relevant parts of OPEN and the context management framework.

In relation to the other modules within this deliverable and the project, the Context Management Framework takes context as input from Performance Monitoring and other context sources like location, user orientation (see [OPEN D3.3]) or other service enablers and provides access to the information to the Trigger Management, QoS and applications, see [OPEN D5.3].

2.1. INTRODUCTION

The Context Management Framework being used in OPEN, was originally developed and a implemented in the IST project MAGNET Beyond in Java, [MAGWEB], [MAG D2.3.1], [MAG D2.3.2]. First we elaborate the basic and existing functionality, and how applications interact with the framework. Following this, we motivate for the changes required to fit the framework into the OPEN domain, most importantly to use OSGi, [OSGi] as a base environment for the framework. In Section 2.2 we describe the changes made.

2.1.1. DESCRIPTION OF FUNCTIONALITY

The major entity in the framework is the Context Agent, which implements the necessary functionality to ensure efficient gathering and distribution of context information. The framework assumes a context agent is running on each node within the operational domain. All agents are to be connected to an agent with the special role of knowing what context information is available within its network domain. We call this context agent for the Context Management Node (CMN). All context agents register its local information to this node type, and may use it to search for context information within the network domain. Further, the Context Management Node can connect to other Context Management Nodes via an overlay network of other Context Management Nodes, which allows a scalable setup of context agents. The Context Management Node can either be set via configuration files, or selected via a master selection algorithm. In OPEN the context agent running on the OPEN Server is a natural choice of being the Context Management Node. The details on how the framework works are described in [MAG D2.3.1] and [MAG D2.3.2]. Figure 6 gives an overview of the internal components of a Context Agent.

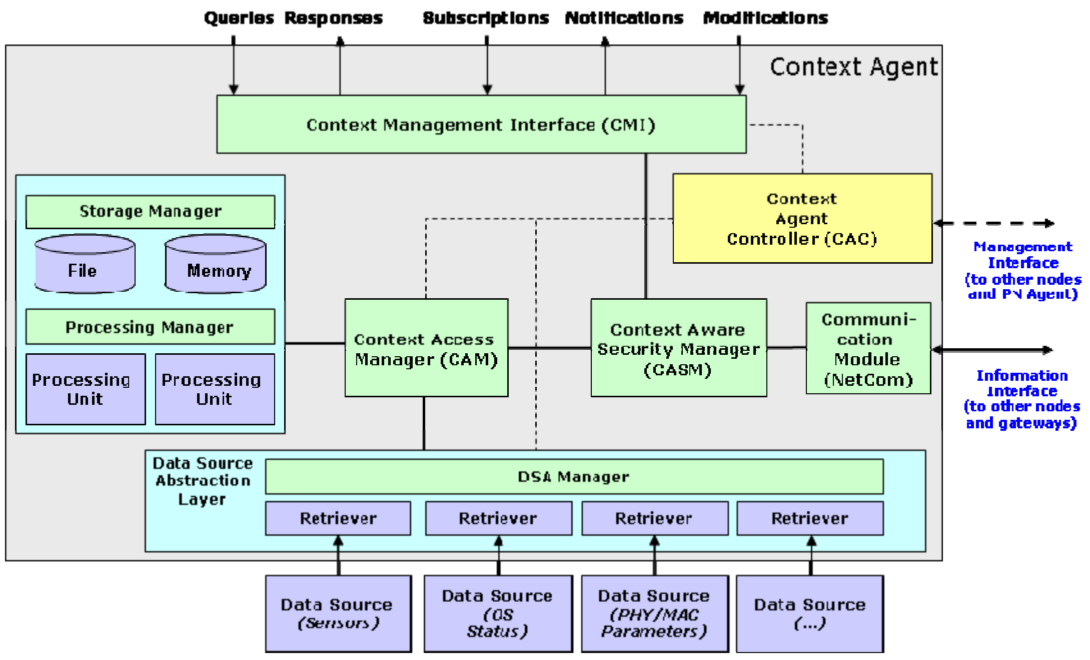

Queries
Responses
Subscriptions
Notifications
Modifications
Context Agent
Context Management Interface (CMI)
Context Agent Controller (CAC)
Management Interface (to other nodes and PN Agent)
Storage Manager
File
Memory
Processing Manager
Processing Unit
Processing Unit
Context Access Manager (CAM)
Context Aware Security Manager (CASM)
Communi-cation Module (NetCom)
Information Interface (to other nodes and gateways)
Data Source Abstraction Layer
DSA Manager
Retriever
Retriever
Retriever
Retriever
Data Source (Sensors)
Data Source (OS Status)
Data Source (PHY/MAC Parameters)
Data Source (...)

Figure 6: Overview of a Context Agent and its internal components. Processing Units and Retrievers are dynamically plug-able components, while the remaining parts are components needed at all time. The storage is, however, configurable to use either a file or memory.

In the middle of the Context Agent, the *Context Access Manager* (CAM) is responsible for constructing and maintaining a list of information locally available for the Context Agent. For Context Management Nodes, this list also includes available information in the given network. To the right, the *Context Aware Security Manager* (CASM) ensures that all input/output is being checked with respect to privacy and security requirements, e.g. requests for location information may have associated privacy policies that may need to be enforced at the CASM. The *Communication Module* (NetCOM) ensures that internal data objects are (de)serialized and transported effectively between Context Agents. The Context Management Interface (CMI) ensures the interface between the internal components and the external client applications and services that uses the Context Agent. This interface is defined by the so called Context Access LAnguage (CALA) which is an XML formatted query language, [MAG D2.3.1]. Then, there is the *Context Agent Controller* (CAC), which basically ensures correct configuration of the internal components via dedicated configuration files. Finally, there are the components *Data Source Abstraction Manager* (DSAM) and *Processing Manager* (PM) which maintains and controls the pluggable modules of retrieves and processing units. Retrievers are small software components which create the interfaces between some measurable information, e.g. ambient temperature, noise level, OS status, or network conditions and the internal of the Context Agent. Processing Units do not themselves interact with data sources in the same way as Retrievers do, but are still producing and offering information via inference, logic, or other heuristic algorithms, and bases its produced context information on already existing information in the framework. To get required information the Processing Units are requesting or subscribing to context information in the (nearly) same way as any other applications and services (the main difference is that processing units uses internal java objects and methods for communications, while applications and services uses the XML-RPC interface). Once they have the information needed, they execute which ever heuristic algorithm they implement in order to produce new types of context information. As a part of the processing component, there is also a Storage, which offers a simple database functionality to the applications, services and processing units that may need it. For example we deduce whether a device is either *nearby* or *farAway* to the user based on Bluetooth signal strength measurements and device threshold levels which are stored in the Storage, as a part of our demo implementation, see [OPEN D3.3].

2.1.2. APPLICATION INTERACTION WITH CONTEXT MANAGEMENT

From the application point of view, the CMF provides a single interface, that is, the XML-RPC at the CMI. At this level, the workings of a context agent can be summarized as shown in Figure 7.

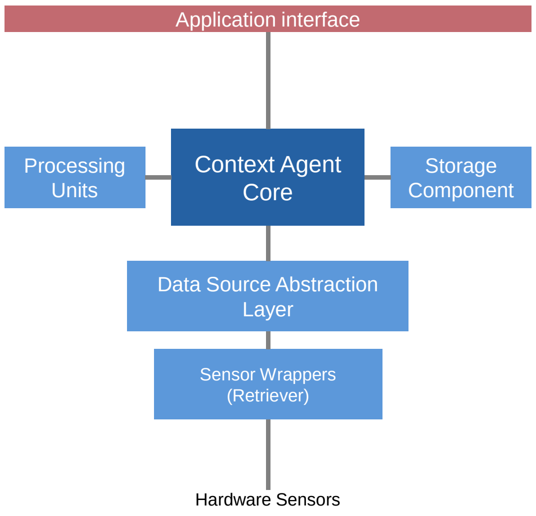


Figure 7: A Context Agent from the application's point of view.

Not only can this interface be used to retrieve the information stored at the chosen Context Agent, but also in any other Agent in the same CMF deployment. This is due to the internal distribution of information availability in the Context Management Framework, [MAG D2.3.2].

When application requests information, the Retrievers, Processing Units and Storage of every Context Agent in the network are called upon to retrieve the needed data. Scalability issues are solved by exchanging the available data indexes among Context Management Nodes and thus preventing the activation of unnecessary Agents (see Figure 8).

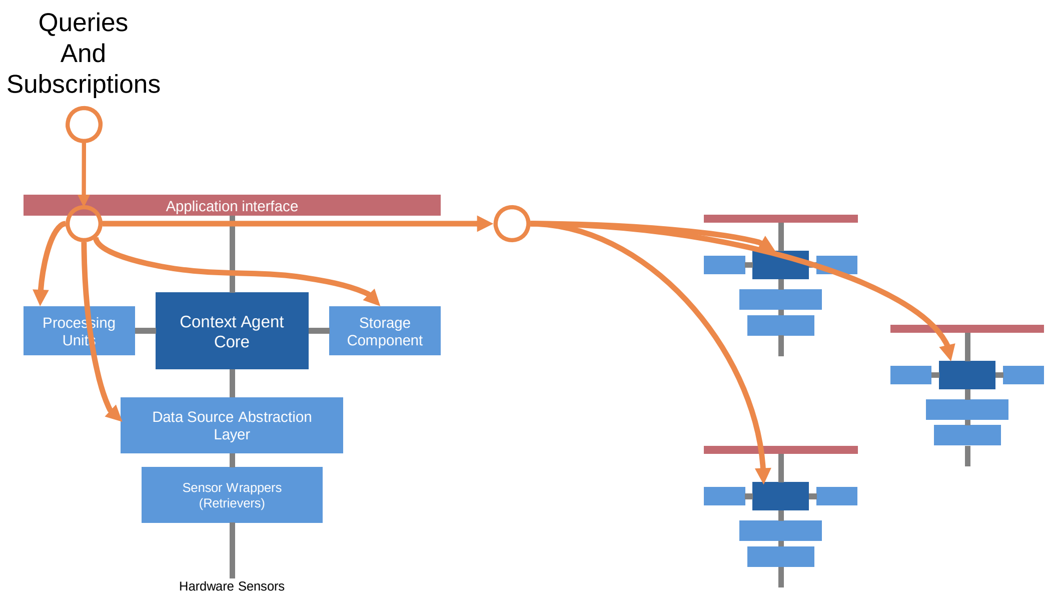


Figure 8: Distributed querying of the Context Agents.

A key feature with the interaction of the XML-RPC queries that the applications can make, is that it is possible to scope queries so that applications have some control of where queries are focused (e.g. node local, same network domain, overlay network, in time and/or space). The above is merely a brief overview of the roles and functionalities of the internal components in the Context Management Framework, and to get the full extent of the interaction and workings of the framework, see [MAG D2.3.1] and [MAG D2.3.2].

2.1.3. AN OSGI CMF IN THE OPEN WORLD

The mobility in the OPEN scenarios calls for the dynamic addition and removal of sensors. When an application is migrated, the devices in the new environment have to be taken into consideration: new sensors might be available which enrich the user interaction with the app or define its behavior. Likewise the user might unplug the keyboard when he takes a device away, or might walk off the line-of-sight of the display he was using.

All these events can be monitored by the CMF, which would then work together with the device discovery module, or with other applications that require context information, like the trigger management.

It is therefore clear that supporting the addition and removal of devices and sensors provides for greater adaptability and a tailored, as small as possible deployment at all times.

Additionally, being able to reconfigure context agents, and the knowledge of the environment where they run, provide for advanced features, such as running processing tasks in the most powerful, or less battery-dependent devices, or perform operations that require high amounts of data as close to the data source as possible, thus minimizing bandwidth usage and therefore both cost and battery. Further, the argument of having multiple java component and applications running on only one Java Virtual Machine instead of one each is also a part of the considerations. the For these reasons the existing framework is not sufficient to address these dynamic situations as envisioned, hence it was decided to shift the framework to an OSGi environment to utilize the aspects and functionalities in this framework to overcome the reconfiguration problem. In the following we describe how this is done and the perspectives of using OSGi.

2.2. CONTEXT MANAGEMENT FRAMEWORK IN OSGI ENVIRONMENT

The original framework from MAGNET Beyond was developed in java, and was running on a single Java Virtual Machine. Each time a Context Agent was started, a configuration file describing what retrievers and processing units should be started, was required. Once these were started, it was no longer possible to start new ones, except if the Context Agent was restarted, hence a limitation to the OPEN scenarios. In the following we elaborate how OSGi is used to address this problem.

2.2.1. SHORT INTRODUCTION TO OSGI

OSGi provides a flexible environment where configuration problems (such as java class path issues) no longer are a problem, which in particular for those dynamically needed components is very useful. What it basically offers, is an environment where *services* can be installed, started, stopped and uninstalled as needed, i.e. basic service life cycle. These different states are also shown in Figure 9. Services are wrapped in a kind of package called bundles, which may include more than just the service itself such as other used Java packages.

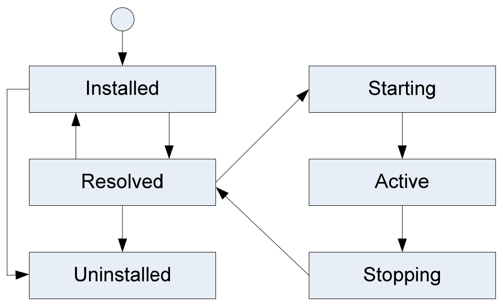


Figure 9: OSGi different states that a service bundle can be in.

All bundles start their lives by being installed into the OSGi container, meaning that OSGi becomes aware of them and handles classloaders to make them available throughout the container instance. Before the bundle can be executed, the bundle is checked against its dependencies, meaning that if bundle A depends on package B, then the framework will ensure that bundle A can only be executed before it has found a bundle which offers package B, at which point the bundle is marked as Resolved. It is here worth noticing that in OSGi, bundle interfaces are defined by whole packages and not classes, e.g. bundle C may offer package B as a part of its interface. Once all dependencies are ensured, the bundle can be started and activated. In this way, OSGi ensures only bundles which have access to needed packages are allowed to start. Starting a service within a bundle is done by a signal, e.g. as a command line input from the user, and where after the bundle is told to enter its Starting state. In this state the service may instantiate and initialize what it needs before it enters the Active state. The service (and bundle) then stays in the Active state and does whatever it is programmed to do, until it reaches the end, is stopped by a stop signal from the OSGi environment, e.g. a command line input, or an error forces it stop. In the Stopping state, the service may need to do things like shutting down a database, closing sockets or similar terminating operations. After that, the bundle is put back into the Resolved state, where it stays until it is once again activated or eventually uninstalled from the environment.

It is worth noticing that the basic OSGi is node local only, and offers no remote communication method in itself. However, bundles may be installed that offers services needed for any type of remote communication, e.g. for discovery and download purposes as will be discussed later on.

2.2.2. THE CONTEXT AGENT INTERNAL STARTUP AND MAINTENANCE

For the CMF each internal component is represented as a service, e.g. the CAM or the CASM, to be used by the Context Agent. Since some of the components are needed at all times, this means that some services need to be run and not stopped unless the Context Agent needs to be stopped as well. The components which are allowed to be started/stopped as needed are the Storage, Processing Units and Retrievers. This is not a coincidence and it is in fact the whole idea of the extension that we can control and manipulate the life cycle as needed for exactly these components. Further, the modularization of the internal components into services allows us to run different versions of a certain component, should a particular target platform impose special restrictions (e.g. a more powerful database backend for the Storage system). Making components into services that are run individually, but with interdependencies, also puts strict requirements to the internal synchronization procedure among the core components in the startup phase. For example, starting the Processing Manager (PM) before the Context Access Manager (CAM) creates potentially a problem, since the PM will send registry of what processing units it has available to the CAM, as it assumes the CAM is up running and listening for incoming registry information. Similar, e.g. service references to the CASM is needed at the CMI and CAM which are only available once the CASM is in a running state. If, or while, the references are not available, the methods of the respective component are not available and lead to (null pointer) errors!

To facilitate the integration of new modules, we have developed a simple base activator which enforces the application-layer dependencies of the Context Agent, ensuring both the correct order of initialization, and the uniqueness of the singleton components, which could result from a faulty configuration.

2.3. RETRIEVER AND PROCESSING UNIT COMPONENT LIFECYCLE

OSGi allows as mentioned, to install, run, stop and uninstall bundles at runtime which in turns gives additional possibilities for controlling active components as retrievers and/or processing units. In the following part of this section we describe conceptually the major aspect of a support system that allows a dynamic component lifecycle management which was not possible in earlier version of the context management system.

2.3.1. SUPPORT SYSTEM

The approach at which components are treated as individual services enables the system to automatically ensure that the most updated components are in place. Each bundle is marked with a version number, which enables the system to check if the required version of a given component is being used or not. It also allows, if checked against a repository to see if the newest version is being used, or updates are required. This concerns the core components, as well as the more dynamic components as retrievers and processing units.

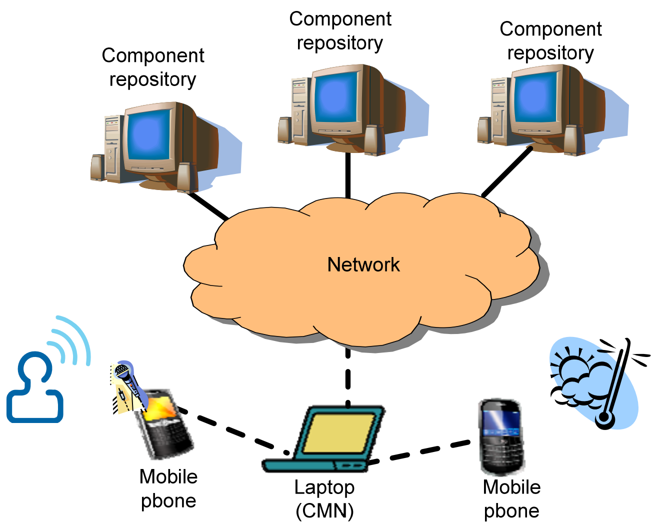


Figure 10: Example of support architecture for retriever and processing component discovery and installation.

Figure 10 shows an example of how a context management support architecture would look like. In the infrastructure, e.g. the Internet, repositories of retriever and processing components are located from which Context Management Nodes may become aware of their availability. The repositories can be maintained by business entities offering context information, for example operators or small businesses.
Once the key components of the Context Agent are running, needed retrievers and processing units can be downloaded and installed from the repositories known a priori to the Context Management Node. Thereafter, they can be activated and the Context Agent can now offer the given context information. This process is illustrated in Figure 11, and is only required when the retriever/processing units are not initially available node local.

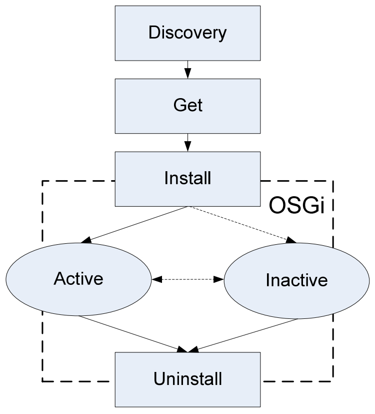


Figure 11: Component life cycle for retrievers and/or processing units. In principle, updates to existing internal components may follow similar diagram.

2.3.2. COMPONENT DISCOVERY REQUIREMENT

The question how to select the appropriate retriever or processing component (or both) is not a trivial question to answer. In the following we do not answer how to do this, but we will instead sketch the problems and non-triviality of selecting the best retriever/processing unit, and what parameters to consider in this selection process.

During runtime, the Context Access Manager, which receives the requests or subscriptions for context information needs to make a decision between the two options it has

A) Try to discover the information and retrieve it from other Context Agents (hence, no need to download a retriever and/or processing component). The context management framework already supports this option.

B) Try to get the relevant retriever and/or processing component from one of the repositories

The latter option may be even more interesting when considering the case, where needed information may be inferred via processing units based on existing available information on the node. For example a processing unit capable of transforming received Bluetooth signal strength into an estimated distance metric, which may already be present at the local node.

Since access delay time is a critical factor in accessing and using context information, it is also critical that the Context Agent makes the right decision. Obviously, if the information is already locally available, i.e. a retriever or processing unit is present and can deliver the information requested for this is not a problem at all.

Initially, it may seem like the discovery of information is the best option, but the Context Agent must also consider that once the initial cost of installing a discovered component, the cost of getting this information in subsequent requests are hugely reduced. A major issue needed to be addressed is, however, that a Context Agent cannot use any arbitrary retriever since this is usual specific to the hardware or software interface from which it gets its information from. Hence, for retrievers there are specific requirements to the underlying interface to the retriever. For example, a device with a Context Agent may need the position of itself in relation to another device, assuming that the Context Agent decides to locate a retriever for this there may be several options

- A Bluetooth retriever capable of discovering other Bluetooth devices (leading to a simple within Bluetooth range distance metric).
- A Bluetooth retriever capable of getting the RSSI value and establish a distance metric based on this.
- A WLAN retriever capable of reading signal strength values and establish a distance metric based on this information
- A Location tracking system retriever, capable of interacting with a specific existing and external location tracking system.
- A GPS retriever if information is required outdoor. However, for indoor positioning this retriever is not useful.

Some retrievers may even require an additional processing unit, or being aware of external systems being available, e.g. indoor tracking systems, in order to function properly. As it is shown later in Section 2.5 this selection is not a trivial matter of choosing the retriever with the highest accuracy, since if it relies on time consuming algorithm, an additional question of reliability of the information comes into considerations. So although the simple Bluetooth discovery approach is simple and robust, its inaccuracy may render it useless. On the other hand, a retriever for a high accurate indoor system may not be useful if the reliability of the retrieved information becomes too high.

Hence, there is a need for a discovery solution which can take additional parameters into account:

- Type of retriever/processing unit
- Attributes that it is offering
- Interface requirements (in case of processing units, this is not needed)
- Performance requirements; processing capabilities, memory requirements, delay and reliability and other specific performance metrics (e.g. accuracy of position estimate)

Once retrievers and processing units has been discovered and downloaded, they may be installed and activated which enables the respective Context Agent to provide the information produced. Later on, this may be stopped and started again as needed, and finally uninstalled when there is no longer a need for the information, or the retriever/processing unit should be updated or becomes obsolete via other components.

2.4. RELIABILITY ASPECTS OF CONTEXT MANAGEMENT

Access to context information does not always happen locally, i.e. on the same node/device which the requesting application is submitting its request from. In that case, the context agent will redirect the request to a device which provides the information (if such is present). In this remote access some delay will be experienced from the request has been accepted at the local context node, until a response from the corresponding context agent has been received. During this time interval, changes may have happened to the information itself due to an underlying (stochastic) process, leading the accessed information to be mismatching. The probability for that happening we call mismatch probability. In [Hansen10] we investigated different access strategies, and their impact on this reliability metric. Our findings at that time showed that this metric is impacted by several factors, namely

- The base access strategy taken;
    - Reactive; with and without caching of accessed data
    - Proactive; periodic updates or event driven updates with either full or incremental updates. In the case of periodic updates, the update time interval is the influencing parameter, while the inter arrival times of events are the factor wrt. the event driven mechanism.
- The underlying stochastic process that defines when the values of the information elements are changing, i.e. the inter event times and values. What defines an event, we will later in Section 2.4.3 specify.

- The delay and the distribution of delay times between the requesting context agent and the context providing node. The delay could also be a part of pre-processing of data.
- The request rate does affect the mismatch probability in the case when a cache is being applied to the reactive strategy.

In the following we focus on the reactive access strategy.

2.4.1. SCENARIOS FOR MISMATCH PROBABILITY ON LOCATION INFORMATION

In the Twitter Wall application, [OPEN D5.3], for example, the distance between the user and the screen is used to trigger a migration. This is achieved by either mapping a measured RSSI value into a *nearby* or *faraway* attribute, or via the Bluetooth discovery whether the device is within range or not, [OPEN D3.3]. When triggering a migration automatically, it is important that it does not happen at inappropriate times, e.g. when the user has moved away from the screen again, or that it is due to mismatching information is triggered to another screen further away. In the following we look into the reliability of the location information provided for automatic triggering of the migration process and how we can use this metric to tune the positioning system for reliable context information. The scenario, which is a subset of the Twitter Wall application, is sketched in Figure 12.

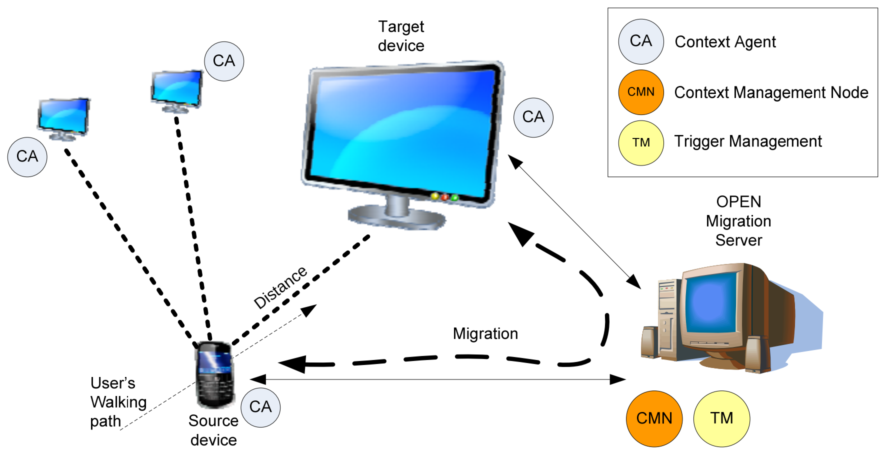


Figure 12: Part of the scenario in the Twitter wall application which including the usage of distance between the screen and the mobile device to trigger a migration process.

The Context Agents on the source and target devices are in this scenario responsible for obtaining the location of the device, via one of the mentioned approaches (which assumes that either the RSSI or discoverability can be directly linked to the distance between the source and target devices without any other influencing factors). For the screen this can be assumed static and therefore not prone to information mismatch between the OPEN Migration Server and the screen location, hence it is only a mismatching location of the source device that leads to errors in the distance, which is why in the following we focus ourselves on the location of the source device.

Another scenario in OPEN which benefits from this analysis is the Device Discovery Map, see [OPEN D3.3], which provides information regarding the availability of the devices within vicinity of the user, taking into consideration also the orientation of the user. Figure 13 shows the conceptual idea from a user's point of view.

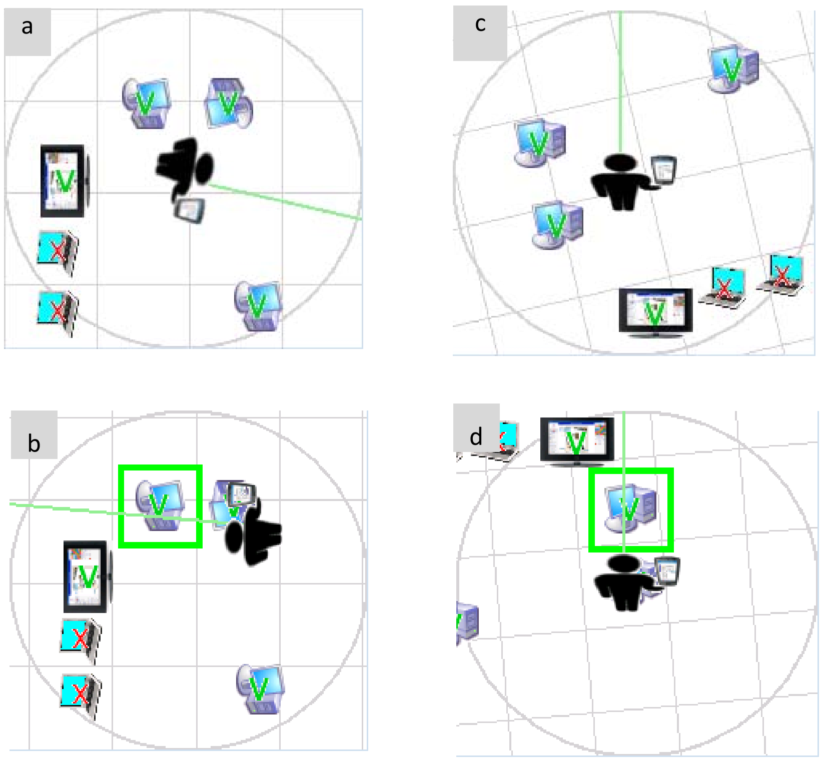


Figure 13: Device Discovery Map used to discover relevant devices within the vicinity of the user.

For this scenario, a mismatch in the position acquired will lead to the user getting wrong information of the devices shown available, i.e. if the user is shown case a) on the screen, then in fact it may be case d) if the user has moved around. Although not fatal, this is still a deviation of the expected functionality, and therefore undesired. It is important to notice in this scenario, that it is only the position and not the user's orientation part of the Device Discovery Map where this can be applied to. If the orientation was to be taken into account then further analysis of how often the user changes orientation and an analysis of the delay it takes to get the orientation from the sensor would be needed.

For the following investigation on the mismatch probability we need to introduce two terms, following the measurement terminology of [Bondavelli07]

- **Accuracy**: the representative for the closeness of the measured value to the best estimated value of the measurand (the measurand being the position in a coordinate system). We use the standard deviation of the estimated value and the notion σ.
- **Resolution**: The smallest variation of a measurand that can be appreciated. We use the grid size in a given coordinate system and the notion R.

Typically the achieved accuracy of a given location estimation system for static nodes are associated with the time it takes to do the estimation, meaning that an improved accuracy typically leads to longer delays and vice versa. The position estimate is mapped into a coordinate system, e.g. a triangulated signal strength measurements from a set of access points, [Figuerias] are mapped into a building local coordinate system useful for the Device Discovery Map or Twitter Wall application. In the same time, when nodes are mobile a high resolution of the coordinate system may not be useful, since the movement of the node may lead the system to provide erroneous coordinates. Consider the following, if the positioning system estimates the user to be in (0,0) then due to mobility the user has moved to (2,2), the context management framework may falsely provide the coordinate (0,0) to the application. By adjusting the resolution (and thereby the required accuracy of the system) we can control the probability of getting erroneous position estimates, which is what we will put focus on in the following part of the section, and how it can be applied.

2.4.2. CONTEXT ACCESS STRATEGY MODEL

The two scenarios may be mapped into a timing diagram illustrating the access of the position information from the client terminal itself, or from any system that provides the location of the client terminal. The timing diagram is shown in Figure 14.

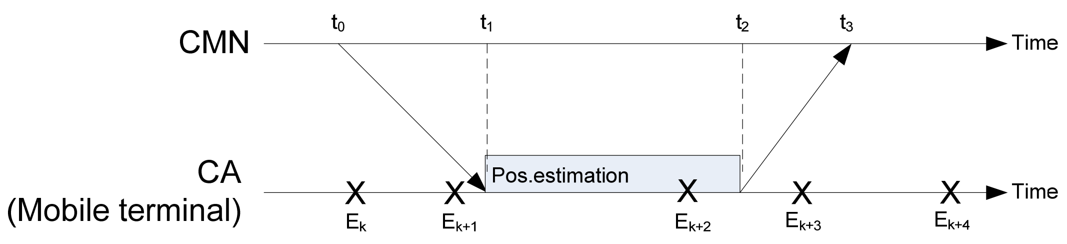


Figure 14: Timing and interaction diagram of the CMN and Context Agent on the node from which the position is required to trigger the migration process.

At the time t_0 the CMN is being requested for the position of the mobile terminal. This triggers a request to the Context Agent on the mobile terminal for the position received at t_1, which in turn triggers some positioning algorithm that provides the position at t_2. The value is then returned to the CMN at t_3 which is used, in conjunction with other context information to determine whether a migration should be triggered or not.

In the same time, the user along with the mobile terminal is moving around, and therefore changes position. An event process, E, indicates the points in time whenever a user and therefore the mobile terminal has changed position in terms of a changed coordinate set. If this happens during the estimation process (between t_1 and t_2) and/or during the return time (between t_2 and t_3) we state the information may not match the desired position which is the one at time t_1. In Figure 14 this happens at the time of event E_{k+2}. However the user may move so that a mismatch is returned back to a match, e.g. E_{k+3} render the mismatch due to E_{k+2} into a matching value. The probability of getting mismatching information is our target performance metric for the following analysis. Thereafter we need to model the delay and event process to finally present numerical results which we can use for effective configuration of our location tracking system. An event that happens in the time interval t_0 and t_1 does not influence the mismatch probability since the information of interest is the one at t_1, [Olsen06]. However, it does influence the waiting time to access the location information.

2.4.3. THE EVENT PROCESS

The device mobility is the basis for the event process. Each time the device changes its coordinate, we say an event has happened. We distinguish between two mobility models,

- **a device mobility** model described by an NxN Markov state space (modeling a discrete version of a Brownian motion) each state covering a d_0xd_0 area as shown in Figure 15,
- **an observation model** with MxM states representing a (md_0)x(md_0) area with n being an integer number describing the amount of states of the device mobility model covered by a single state in the observation model. Therefore, m≥1 and M = N/m.

With this model, the events marked in Figure 14 are indicating the times at which the Markov chain changes state, and a difference in the true and known state value determines whether the information is mismatching or not.

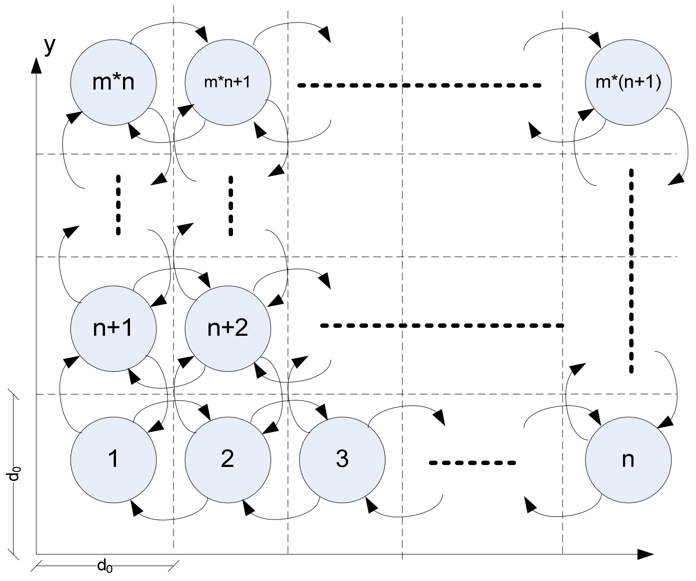


Figure 15: Mapping of rectangular room to a Markov chain representation with each state representing an area.

The state holding times for the different states relate to the velocity of the object moving in the area. Obviously, there is a model limitation here as the state holding times assumes matrix exponentially distributed random times, whereas the moving may not necessarily be following this assumption. Considering the device mobility model, then by defining the mean velocity as the time it takes to move a certain path (N(t)) inside the Markov chain and derive an expression of the rates, and assuming that d_0 are all the same in the Markov structure shown in Figure 15, we can make a relation between the rates defining the generator matrix **Q**, the velocity and the distance between the states as (assuming Diag(Q) contains the constant value $-\mu$)

$$\bar{v} := \frac{E(N(t))d_0}{t} \Rightarrow \bar{v} = \mu d_0$$

For other structures and non homogeneous area models, this relation may not be so simple. The resolution of the positioning information is given such that the location system is capable of tracking an object to one single state or a group of states, i.e. the resolution is determined by an integer multiple of d_0, i.e. we define

$$R := md_0\,,\ n \in \mathbf{Z} > 0\,.$$

With this definition of the resolution, we basically hide some states to what is experienced by the user. That is, instead of determining whether the system is in state 1, 2 or 3, we may say the system is in state (1, 2, n+1, n+2) or (3, 4, n+3, n+4) and so forth. When we mean groups of states we will indicate this by a ~, and otherwise we refer to the individual states shown in Figure 15.

2.4.4. THE DELAY PROCESS

The delay which is relevant for the considered type of reactive access is defined as the time the request has been received at the Context Agent on the source device, until the response has been received at the requesting entity, i.e. the time interval from t_1 to t_3. For the correlation between the achievable accuracy (the standard deviation, σ) of the positioning system and the estimation delay, t_2-t_1, we assume in the remaining part the following relation which is based on a study of a Bluetooth positioning system studied in [Figuerias] from where we use the values c_1 = 0.568 and c_2 = 3.146.

$$\bar{D}(\sigma) = \frac{c_1 + c_2\sigma^2}{\sigma^2} \qquad (1)$$

Other positioning systems may have other relations than the one described here. Although the time interval constitute two delays caused by two independent processes, and should be treated like this, then for simplicity we will combine the two delays into a single delay which has an exponential distribution with a mean value given by the above equation.

In reality, a system providing a location estimate will most likely not take an exponential random delay, but some more complex distribution. Here we investigate the case where we use a Bluetooth tracking system and perform an evaluation of the mismatch probability based on this system, which shows there is surely an impact of the distribution of the delays to the resulting mismatch probability.

2.4.5. MISMATCH PROBABILITY

In [Hansen10] we showed that the mismatch probability for a reactive access strategy (see Figure 14), i.e. the Context Management Node sends a request to the Context Agent at the source device to get the position of the mobile terminal, without caching the received position is defined as

$$mm\Pr = 1 - \int_{t=0}^{\infty} \Pr(E(t_3) = E(t_1) \mid t_3 - t_1 = D) f_D(t)dt \tag{2}$$

Assuming stationarity we may set $t_1 = 0$ and simplify the probability distribution $f_D(t)$ of the delay $D = t_3\text{-}t_1$. Now we extended an expression of the mismatch probability for a case where the information is taking value according to one or more states in a Markov chain; hence a model described by a generator matrix ***Q*** and steady state probability vector π. Taking this into account, Equation (2) is expressed by

$$mm\Pr = 1 - \int_{t=0}^{\infty} \sum_{i=1}^{M^2} \tilde{\pi}_i \exp(Qt)\tilde{e}_i' f_D(t)dt \tag{3}$$

with M as the number of states defined in the observed model (see Section 2.4.3), which is constrained by the relation between the geographical area observed by the positioning system (A) and the resolution (R), i.e. $M = A / R^2$. Then the following vectors are defined (with *j* indicating the *j*th column of the device mobility model, and *i* the *i*th of the observation model, s indicates the state in the device mobility model and ~s the state in the observation model)

$$\tilde{\pi}_{j,i} = \begin{cases} \pi_j & s_j \in \tilde{s}_i \\ 0 & s_j \notin \tilde{s}_i \end{cases}, \qquad \tilde{e}_{j,i} = \begin{cases} 1 & s_j \in \tilde{s}_i \\ 0 & s_j \notin \tilde{s}_i \end{cases}$$

and finally $f_d(t)$ is the delay distribution. In cases where the delay distribution can be expressed as Matrix Exponential distributions [Lipsky92], where distributions are described by a system matrix **B** and an entry vector **p**, Equation (3) can be expressed as

$$mm\Pr = 1 - \sum_{i=1}^{M^2} [\tilde{\pi}_i \otimes (p_D B_D)][(-Q) \oplus B_D]^{-1} [\tilde{e}_i' \otimes \varepsilon_D'] \tag{4}$$

with ε_D as a vector of ones and dimensions of the delay distribution vector/matrix pair.

Equation (4) simplifies the calculations of the mismatch probability, since using Equation (3) otherwise in most cases would involve numerical integration. Further, our simplification of the delay by modeling it as one exponential random distributed variable, Equation (4) reduces to

$$mm\Pr = 1 - \lambda \sum_{i=1}^{M^2} \tilde{\pi}_i (Q - I\lambda)^{-1} \tilde{e}_i' \quad (5)$$

where λ is the rate of the delay distribution.

2.4.6. NUMERICAL RESULTS

In this section we evaluate the impact on the mismatch probability with varying parameters, delay and velocity of the tracked object. Figure 16 shows the results of the mismatch probability for three different resolution values, each for two velocities with a varying delay as input. Results are obtained using Equation (4) with an area of 36x36 m, and a d_0 of 1 meter meaning that the two curves at the bottom with an R of 18 meter, are referring to a system that is only able to determine if the user is in one of four corner areas of the room, whereas the upper two curves with an R of 1 meter, determines the user's position to a 1 meter area.

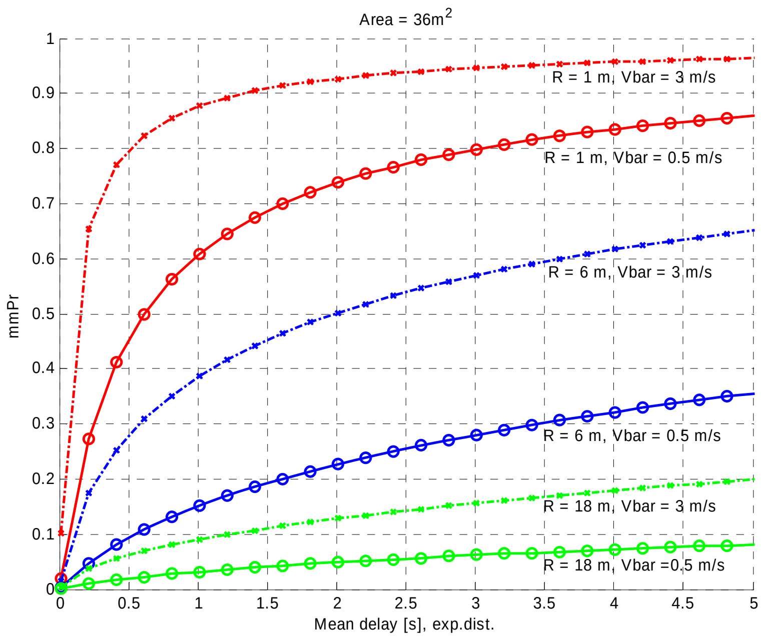


Figure 16: Mismatch probability as a function of the mean delay, shown for various resolutions and object speed.

From Figure 16 the trade-off between a desired resolution and the mismatch probability is clearly shown, i.e. if a high resolution is desired for the position, then this must lead to an increased probability of getting mismatching information. It should be noted here, that the mmPr parameter only states the probability of having erroneous data, but not how much error there is. This scenario relates to cases where the resolution of a system may be given beforehand and fixed during runtime, which may often be the case. It may also be the case that the resolution may be changed only one time in an initial startup phase, and then be fixed remaining time. Then this type of evaluation may be useful to determine if the resolution in relation to the mobility and reliability is acceptable.

There may be cases where the resolution can be changed online. In Figure 17 we assume we can change the resolution in order to ensure we do not exceed a given boundary of the mismatch probability $mmPr_c$ of 0.1, 0.25 and 0.5, respectively. In this case we need to solve Equation (5) with respect to the delay rate λ and further solve Equation (1) with respect to the achievable accuracy, which we do numerically. For comparison, one could simply state a Rule of Thumb (RoT) that within the estimation time, the object shall not move outside the given accuracy of the estimation, hence we say our resolution is given by $R = vt$ where $t = E(D(t))$.

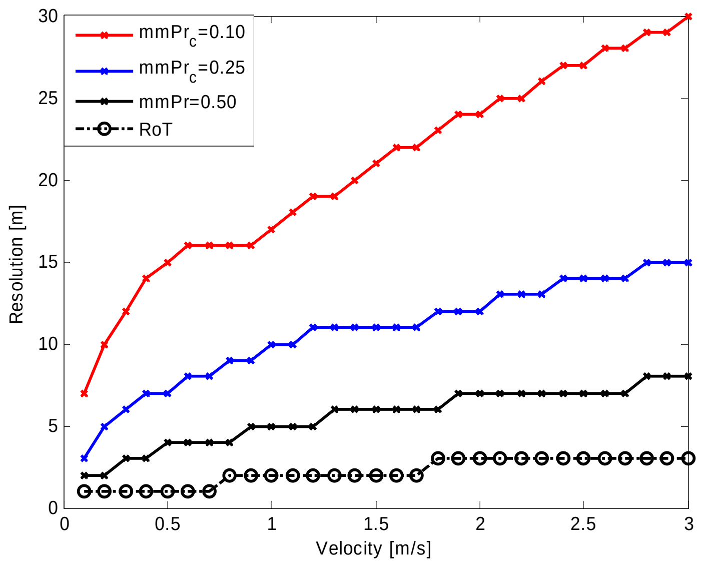


Figure 17: Required resolution of system in order to maintain reliability requirements on the achieved information.

From Figure 17 we see by altering the resolution of the system we are able to maintain a certain mismatch probability, which is shown in Figure 18. The tougher restrictions on the mismatch probability we have, the lower resolution we have to accept. In comparison to solve Equation (1) and (5), using the simplistic rule of thumb there is no or little control of the resulting mismatch probability which in this case it becomes very high and in some cases up to 0.90, meaning that 90% of the response messages is outdated when received!

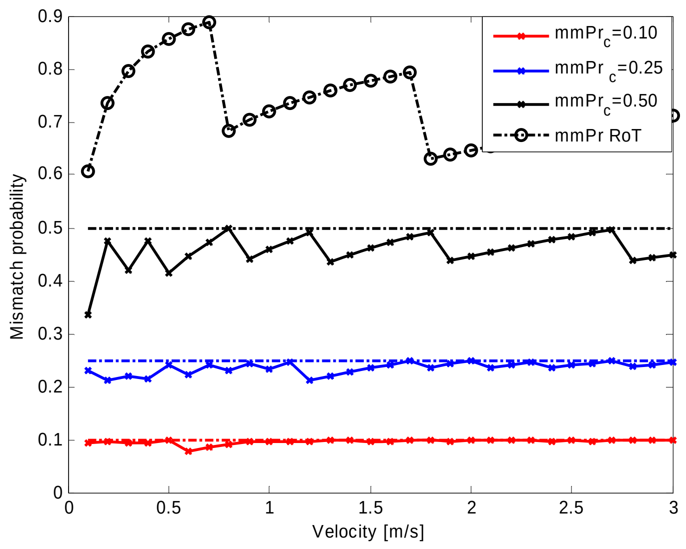


Figure 18: Resulting mismatch probability by using the resolution shown in Figure 17.

2.4.7. POTENTIAL USAGE AND OUTLOOK

The results shown in this section can be applied in general to positioning estimation systems to argue numerically for the choice between reliability and the resolution of the positioning estimate provided. For the Twitter wall application the results shown in Figure 16 can be used to tune the filtering mechanism that happens when mapping raw signal strength measurements of the Bluetooth connection to the device into a distance. The response time of the filter directly applies to the delay of the estimation process as described in the process. For the Device Discovery Map, the results shown in Figure 17 and Figure 18 to find a reasonable resolution for the grid of the shown display which takes into account expected user mobility and maximum probability of showing wrong services at wrong locations.

It is clear that there are still limitations of this mismatch probability work, e.g. the mobility model here is restricted to a very specific one, restrictions of access method to a pure reactive strategy, the delay distribution of the network and estimation process and the relation between the estimation accuracy and delay can be expected to be highly technology specific, and must be taken into consideration when applying.

2.5. SECURITY AND PRIVACY CONTROL

The main security concerns of the Context Management Framework are

- the protection of gathered context information against malicious parties that may exploit the knowledge of context for e.g. blackmailing people, identity theft or stalking.
- avoiding false information injection that may be done in order to render context aware systems useless, annoy the user or to trick the user to do specific things in the interest of

the attacker, e.g. to guide the user to a bad location in the city or an advanced type of phishing.

- denial of service (DoS) attacks by requesting context information at all time as fast as possible in order to make the intention of the context management framework useless. This can happen at both the application side (pulling of information) but also at the information provider side (pushing of information).

The security means directly related to the CMF are twofold; 1) what is provided by Java and OSGi in itself which aims to limit code exploitation, and 2) the Context Aware Security Manager (CASM) in terms of privacy protection against policies when distributing context information.

The Virtual Machine provides some security in itself. The JVM is designed to restrict the capabilities of a Java application during run time. Dangerous constructs, like pointer manipulations and unchecked access to arrays, are removed from the programmer's vocabulary. This choice makes it impossible to use, for example, buffer overruns to attack the environment. Buffer overruns are known to be a way of executing malicious code or inserting false data into the system, which aids to ensure injecting of false information or misuse the framework. Buffer over-run attacks are just not possible in Java, [OSGi].

For the case of OSGi environment there may be many other applications running on the same Virtual Machine posing possible threats. Java allows access modifiers to declare the allowed callers of the code. This access control mechanism allows multiple Java applications to reside in the same VM and cooperate while at the same time providing a protection shield for code that should be kept private. Further, each bundle can be limited in its capabilities by giving it permissions for only those resources that it should have access to. Finally, the OSGi Framework strictly separates bundles from each other, so that each bundle need appropriate permissions to even detect other bundles on the OSGi Service Platform. This protects to a certain extend the integrity of the internal components of the Context Management Framework against trojan type of components, and leaves the focus on authentication of third party plugin components to processing units and retrievers. In turn, this is solved by trust establishment solutions in the support network as described in Section 2.3, e.g. by certification, [RFC2459].

Second mechanism is the CASM module, which ensures privacy rules are being enforced when remote devices are requesting context information, e.g. if a remote Context Agent asks for the position of a given device, this device may have a policy to provide a less accurate position estimate than actually available to obfuscate the user's position giving some privacy to the user. A solution was developed in MAGNET Beyond, [MAG D4.3.3] which was basically a no or full access solution), but considering the diversity of context information this seems very restricted as for example information can be obfuscated by change of accuracy or anonymized.

External to the CMF there are firewalls, data link and network encryption methods (secure tunnels etc.) for establishment of connections to trusted entities. For certain DoS type of attack Firewalls or IP filtering may be able to detect and block remote of nodes requesting access to node local services suspiciously frequent, see [RFC4732]. For Distributed DoS attacks (DDos) the protection may not be as simple, since service requests are not coming from a single source but multiples e.g. bot nets. To some extend white listing IP addresses may be useful.

2.6. SUMMARY

In this chapter we presented the Context Management Framework used in the OPEN platform, which is based on an existing framework from the project MAGNET Beyond, [MAGWEB]. To this framework we made necessary changes in order to fulfill the requirements of reconfigurability in OPEN. The main changes related to the relocation of the framework to run in an OSGi environment to accommodate the reconfigurability needed. This allows control of the individual component such as installation, start, stop and uninstallation of the components that ensures the access of the context information. This could be useful for controlling processing or energy consumption on the local node. The change to OSGi also leads to potential further enhancement of the framework to fully cover the life cycle of a context management framework, namely in terms of automatic discovery, installation, running, stopping and uninstallation of required active sub components (retrievers and processing units) as needed.

We described in this respect required support architecture conceptually and requirements to discovery criteria when context agents requests for new information types. With this concept in place, the context management will go from being a static setup to become a dynamic auto reconfigurable system to support the platform with the best context information possible.
We also provided insight into reliability of context information and in this case with focus on location information, which is used in several of the demonstration for triggering a migration and discovery of services within the vicinity of the user. That the information is provided from the context management framework is reliable is in the scenarios important for correctly triggering of the migration process and correctly showing the discovered devices in a map. In the section we provide insight into how to tune a location tracking system in order to provide reliable information, and show how to apply this into a given scenario.

Finally, in the chapter we provide some insight into the security means with respect of the context management framework. Since it has not been the focus of the work, it is only stating security means which is inherited from the OSGi and Java environment, and what has already been considered for the existing Context Management Framework in MAGNET Beyond, [MAGWEB].

3. MIGRATION ORCHESTRATION

The Migration Orchestration module coordinates all of the phases of a migration in OPEN Migration Service Platform. It supports manual migrations (i.e. triggered by end users) and automatic migrations (i.e. triggered by the Trigger Management module). Moreover, some support functionalities are provided for the management of OPEN applications and OPEN devices.

The Migration Orchestration module receives migration triggers from the Trigger Management, interacts with the Application Logic Reconfiguration and with the Mobility Support to make possible needed reconfigurations, and uses the State Handler to preserve the internal state of migrated applications.

Moreover, the Migration Orchestration module interacts with OPEN Applications in order to manage them and perform required migrations.

Figure 19 shows the interaction between the Migration Orchestration, the other modules of the OPEN Migration Service Platform, and OPEN applications.

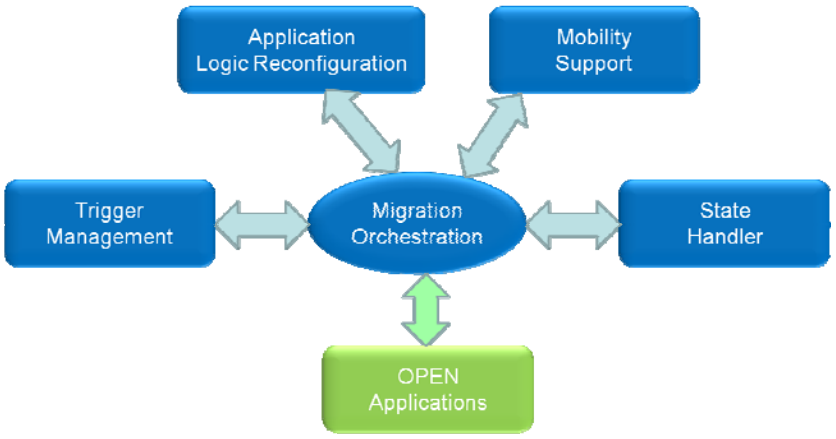


Figure 19: Migration Orchestration module and interaction with other components.

3.1. INTRODUCTION

The Migration Orchestration module coordinates all of the tasks needed to perform a migration in the OPEN Migration Service Platform.

The following functionalities are provided:

- **Device Management**: The OPEN orchestration manages all the devices used in the platform. For each device a set of properties is stored.
- **Application Management**: The OPEN orchestration module manages all of the applications that use the OPEN platform. Every application, in order to grant the support of partial migrations, is made of one or more components.

- **Support for application reconfiguration**: The orchestration module interacts with the Application Logic Reconfiguration module in order to support the application reconfiguration when it is needed.
- **Support for network reconfiguration**: The orchestration module interacts with the Mobility Support module in order to support the network reconfiguration when it is needed.
- **Migration management**: The orchestration module manages every kind of migration performed using the OPEN platform. The migration of desktop and web applications is supported.

3.1.1. MIGRATION ORCHESTRATION SERVER

The Migration Orchestration Server is the server side component of the Migration Orchestration. It is a Java application that implements the OPEN Orchestration Server interface, defined in D4.2.

Apache XML-RPC java implementation has been used for the management of communications with OPEN clients.

The module is able to manage devices and applications used in the OPEN migration service platform. Moreover, it is possible to perform several migrations involving different application components at the same time.

The following data are stored by this module:

- registered devices
- registered applications
- registered application components
- ongoing migrations

Java objects with the properties defined in [D4.2] have been employed for the representation of devices, applications, and components. In order to improve the module efficiency during the search of devices, applications, and components, HashMaps have been used for the storage of tables of devices, applications and components. Due that each object is univocally identified by its ID property (please read the sections 3.2 and 3.3 for further details), IDs have been uses as keys of the HashMaps.

Pending requests to source and target devices are stored using a “PendingRequest” object. It contains the reference to a method (called waitingMethod) and a vector. It is possible to add/remove elements to the vector and, when it becomes empty, the waitingMethod is automatically executed.

Moreover, some classes for the serialization of OPEN objects have been implemented. They perform a conversion between the internal representations of OPEN objects (i.e. as java objects) and their external representations (i.e. with a STRUCT element, according to XML-RPC standard).

3.1.2. MIGRATION ORCHESTRATION CLIENT

The migration orchestration client is an OPEN adaptor running on OPEN devices. Every OPEN Orchestration Client must implement the OPEN orchestration server interface in order to allow OPEN applications to interact with the OPEN platform. Moreover, it must also implement the OPEN orchestration client interface, that is used by the Orchestration Server. The protocol used to communicate with the Migration Orchestration server is XML-RPC. For further details about client and server interfaces, please read [D4.2].

The implemented version of the Orchestration Client is a java application that runs on a PC. It, as the server side component, uses “pending request” objects and serialization methods described in the previous section. In [D5.3] a description of how applications should be developed to use the this Orchestration Client is provided.

3.2. DEVICE MANAGEMENT

Every device used in the OPEN migration service platform has to be registered on the OPEN Orchestration.

The method “registerDevice” is used. It uses the properties of the device to register it into the platform and associates it a univocal device ID.

The main properties of a device Object are:

- Device name: An intelligible name of the device.
- URL: The URL that will be used to contact the client via XML-RPC. It, obviously, is required when an XML-RPC server is available in the client device.
- Capabilities: A set of software or hardware capabilities of the device.

The current implementation of the OPEN orchestration client, automatically registers the current device on the service platform. Some device properties (for example the device URL or the device name) are retrieved from a configuration file.

When a device is no longer available it has to cancel its registration using the unregisterDevice method.

3.3. APPLICATION MANAGEMENT

In the OPEN architecture, an application is a set of application components running on one or more devices. This means that an OPEN application is not associated to a single device, but it has to have at least one component running on a device. Instead, a component is an instance of an application component running on a device. Every component has to belong to an application and to run on a device. For example, in the case of the social game prototype, the “Social Game” application, see [OPEN D5.3], could contain the racing game component running on a PC and the chat component running on a mobile phone. In an OPEN platform, more than one instance of an application can be supported. For example, in the case there are two Social game instances (Social Game 1 and Social Game 2), it is possible to have a chat component for the social game 1 and a chat component for the social game 2.

3.3.1. APPLICATION REGISTRATION

When an OPEN application is manually started on a device, it must register itself in the OPEN platform.

The method “registerApplication” is used. It uses the properties of the application to register it into the platform and associates it a univocal application ID.

The main properties of an application Object are:

- Application name: An intelligible name of the application (e.g. Social Game, Emergency, etc.).
- Application URL: The URL that will be used to contact the application via XML-RPC. It, obviously, is required when an XML-RPC server is included in the application.
- Launch command: The shell command that should be executed to launch the application on a device.

An OPEN application has to implement the Orchestration Client interface [OPEN D4.2] and can have an XML-RPC server or it can periodically retrieve pending requests from the Orchestration using the proper dispatcher. It should allow to launch, terminate, play and pause its components. Moreover, the methods “setState” and “getState” have to be offered by the application in order to make possible the state transfer from a component to another one during a migration. For a detailed guideline on the development of OPEN compliant applications please read [OPEN D5.3].

When an application component is migrated from a device to another one, there could be two options:

1. The application is already running on the target device (i.e. there is at least one component of the application running on the target device). In this case, the client interface offered by the application is used to start the required component.
2. The application is not running on the target device (i.e. there is not any component of the application running on the target device). In this case, the shell command contained in the “launch command” property is executed on the target device in order to launch the application. When the application is ready, its client interface is used to launch the migrated component.

When the application does not have any running component, it must be unregistered from the OPEN platform.

3.3.2. COMPONENT REGISTRATION

When an OPEN application is registered in the OPEN Orchestration server, also its components have to be registered. The method “registerApplication” is used. It uses the properties of the component to register it into the platform and associates it a univocal component ID.

The main properties of a component Object are:

- hasName: An intelligible name associated to the application component (e.g. Betting, Racing Game, etc.)
- hasRunningStatus: The running status of the component (i.e. Launch, Run, Paused, Terminated).
- belongsTo: The application ID that contains the current component. This property is mandatory.
- isRunningOnDevice: The device ID where the component is currently running. This property is mandatory.
- hasRequirement: A set of requirements that a device must fulfill to support the component. A requirement in made mainly by a capability name, a capability value and a comparator (e.g. less-than, greater-than, equal-to, etc.). For example, if a requirement with capability name “Screen Width”, value set to “480 pixels”, and a comparator “greater-than” is defined, then the component can only run on devices with a screed larger than 480 pixels. The “getDevicesSupporting ” method of the Migration Orchestration module uses the “hasRequirement” property and devices capabilities to return the devices that support a set of components.
- hasComponentRelationship: A set of relationships with other components of the application. This property can be used for the launch or the termination of components. For example, if the component A is dependent from the component B, when the component B is terminated, also the component A must be terminated.

When a component is terminated, it is unregistered from the platform.

3.4. APPLICATION RECONFIGURATION

As depicted in Figure 20 the Orchestrator registers the application and the application components at the application logic reconfiguration (ALR), when they become available.

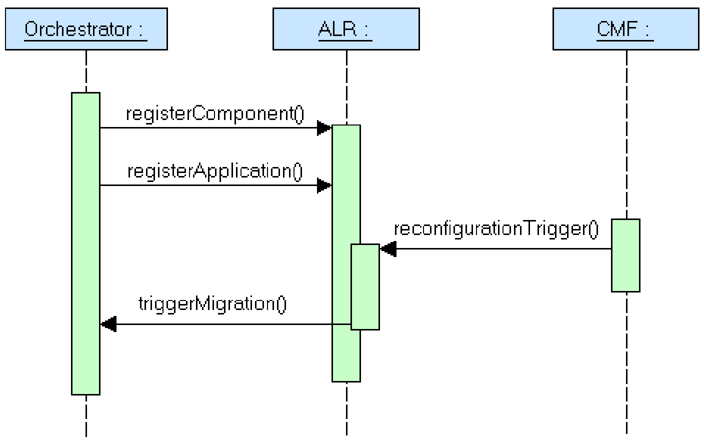


Figure 20: Application logic reconfiguration and its interaction with the orchestrator.

The ALR computes a sorted list of configurations based on the registered components and context information. The context information is provided by the context management framework (CMF). The best configuration of the ALR view is the first configuration of the resulting list.

A reconfiguration trigger can be initiated by the CMF caused by an update of a context value. The ALR updates the sorted list of configuration and selects the best configuration out of the ALR view. Afterwards it triggers the migration at the orchestrator.

More detailed information on the Application Logic Reconfiguration can be found in [OPEN D4.1].

3.5. NETWORK RECONFIGURATION

In OPEN, network interfaces are viewed as objects within a device, which must be registered when the device registers itself with OPEN Migration Server (MS), or when a new interface is added to the device (for example, a Bluetooth adapter is plugged into USB-port). The device also updates regularly the availability, the performance and other properties of its network interfaces to OPEN MS, so that appropriate decision can be made (e.g. triggering of migration process).

Figure 21 illustrates the registration of network interface(s) from the source and the target Open Client to the Open MS. The method “registerNetworkConfiguration” is used, which takes the device’s univocal ID and a networkConfiguration object as parameters. The networkConfiguration object has been defined previously in D4.2 to represent a connection between two devices. In this deliverable, it is redefined to represent a network interface, and a new Connection object shall be created later to denote connection(s). The networkConfiguration object shall have the following main properties:

- ID: Identification of this network interface
- isAvailable: True if the interface is having connectivity, false if the interface is down.
- isOfClass: The class of this network interface (such as wireless LAN, wired LAN, 3G…)

- performanceStatus: The status according to performance monitoring module.
- hasCapability: The set of this interface capabilities.

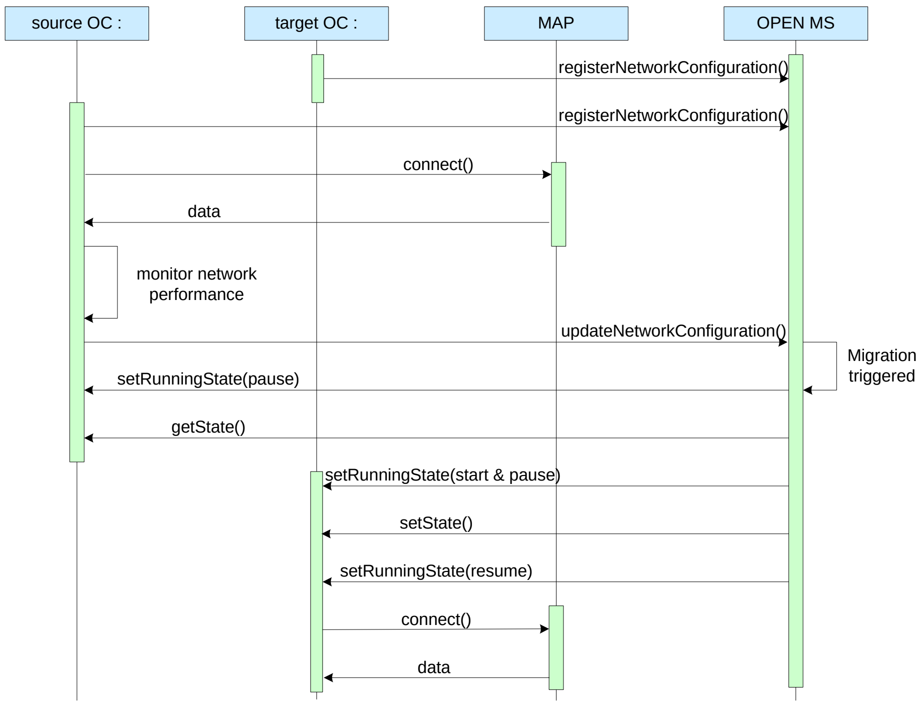

source OC :
target OC :
MAP
OPEN MS
registerNetworkConfiguration()
registerNetworkConfiguration()
connect()
data
monitor network performance
updateNetworkConfiguration()
Migration triggered
setRunningState(pause)
getState()
setRunningState(start & pause)
setState()
setRunningState(resume)
connect()
data

Figure 21: Reconfiguration process of the network during migration process

Figure 21 indicates that a migration is triggered after the performance monitoring module in the source OC reports a network problem to the OPEN MS (network congestion, for example). The updateNetworkConfiguration() method is used to inform the OPEN MS about network configuration changes. The OPEN MS shall use the getState() to obtain state information from the source device, including the connection(s) currently used by the source device. It also employs setState() method to transfer the state information to the target device, so that application in the target device can make the required connection(s). Connection object, which is included in getState() and setState(), has the main properties as follows:

- ID: Identification of this network connection
- endpointAddress: The socket and protocol to connect to in order to use this connection
- sourceDevice: A Device Object contains information of the source device
- targetDevice: A Device Object contains information of the target device

The application in the target device shall issue a connect() request to the Mobility Anchor Point (MAP), which acts as a “fixed” network’s attachment point for mobile devices. For more details on session mobility handling during migration, please see Chapter 5 in this deliverable.

3.6. MIGRATION MANAGEMENT

3.6.1. MIGRATION TRIGGERING

The migration of a group of components of an OPEN application can be:

1) Automatically triggered by the platform. In this case, the migration request is sent by the Trigger Management module (for a detailed description of this module please read the chapter 4). The Trigger Management module, when a migration is required (according to the current context) calls the "triggerMigration" method offered by the Migration Orchestration.
2) Manually started by the end user..In this case, the migration request is sent by the Orchestration Client or by an OPEN application to the Migration Orchestration Server. Also in this case, the triggerMigration method, offered by the Orchestration Server interface is used. This method allows to migrate one or more application components to a target device.

In order to allow a migration the following checks are performed:

- Every migrated component must support the migration. If it is not possible to migrate a component, than the migration request is rejected.

- No migrated component must be involved in an ongoing migration. If a component is already migrating to another device, the migration request is rejected.

- The target device must support all of the requirements of all of the migrated components. If there is a component requirement that is not fulfilled, the migration request is rejected.

There are two kinds of migration supported in the OPEN platform:

- The migration of a local application. In this case the application must be registered on the OPEN platform and to be aware of the migration.

- The migration of a web application. A generic web application can be migrated by accessing it using a proper proxy. In this case the application registration is not required.

3.6.2. MIGRATION OF A LOCAL APPLICATION

The Migration Orchestration module can manage the migration of a set of components from the devices where they are currently running to a target device.

For example, the component "C1" of the application "Demo" could be migrated from the device "A", to the device "B". As it is possible to see in Figure 22, the first operation that is performed is to pause the component "C1" in the source device and to launch a new component ("C3") of the same type of "C1" in the target device. If in the target device there are no components of the application "Demo" currently running, the application launch script is executed (please read [OPEN D5.3] for further details).

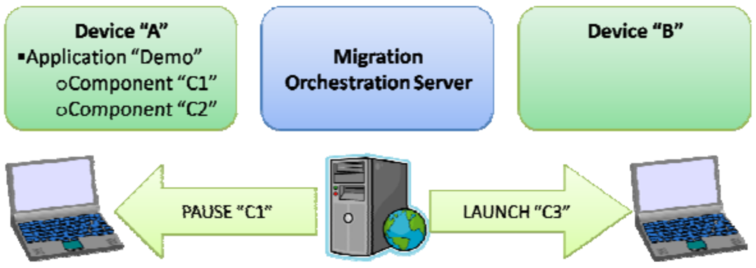


Figure 22: Migration of a component - phase 1.

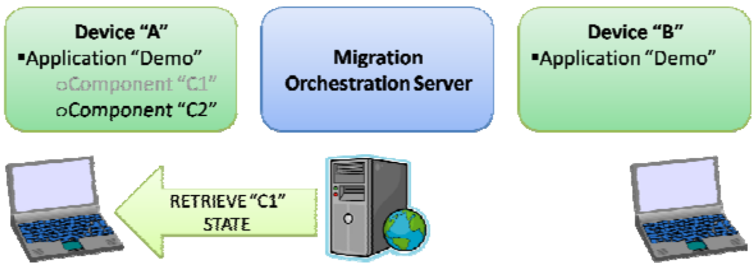


Figure 23: Migration of a component - phase 2.

When the component C1 is paused, its state can be retrieved and when the execution of the application "Demo" launch script has been completed, then the component "C3" is launched in the device "B" (Figure 23).

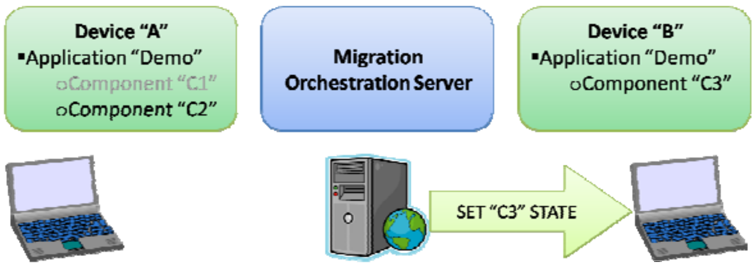


Figure 24: Migration of a component - phase 3.

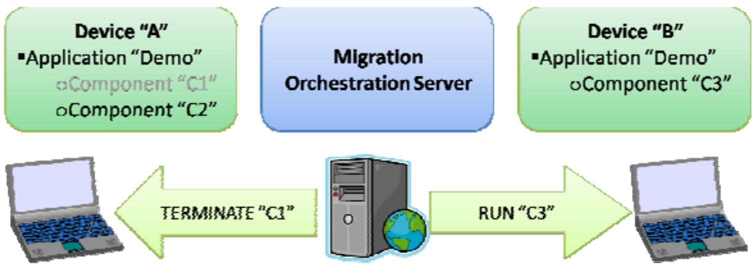


Figure 25: migration of a component - phase 4.

As it is possible to see in Figure 24 and in Figure 25, when the launch of the component "C3" is completed and the state is correctly retrieved from the component "C1", it is transferred to the newly launched component. Finally, the new component ("C3") is run in the target device and the source component ("C1") is terminated. In Figure 26 it is possible to see the configuration at the end of the migration, with the component "C1" that is no longer running and that is unregistered from the platform.

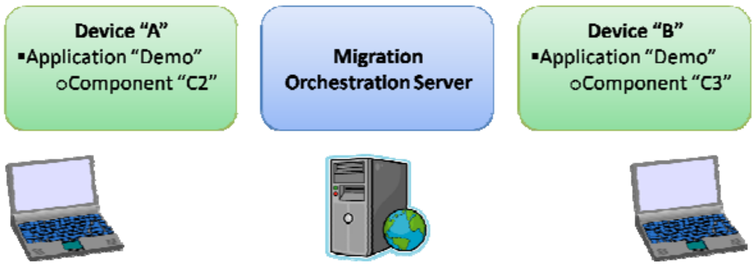


Figure 26: End phase, the migration process has completed.

3.6.3. PARTIAL MIGRATION

Further, migration of an application may happen only partially. In Figure 27 it is possible to see an example of how the Migration Orchestration module is used in the scenario of a partial migration started by the OPEN Migration Service Platform.

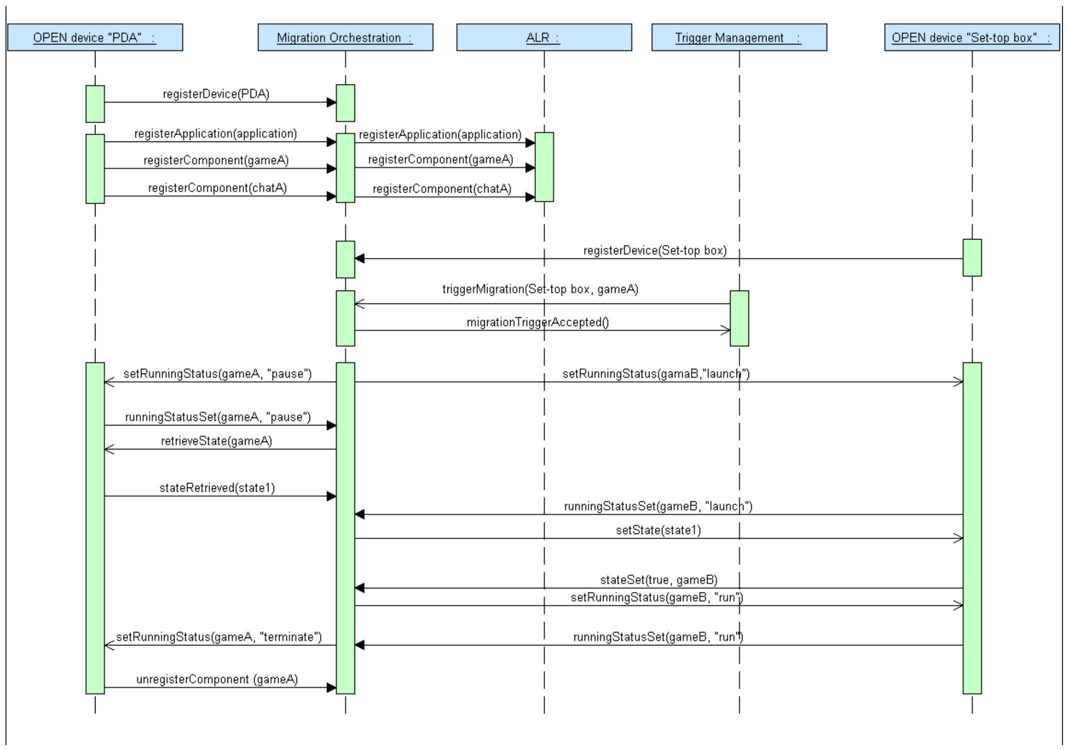

OPEN device "PDA" :
Migration Orchestration :
ALR :
Trigger Management :
OPEN device "Set-top box" :
registerDevice(PDA)
registerApplication(application)
registerComponent(gameA)
registerComponent(chatA)
registerApplication(application)
registerComponent(gameA)
registerComponent(chatA)
registerDevice(Set-top box)
triggerMigration(Set-top box, gameA)
migrationTriggerAccepted()
setRunningStatus(gameA, "pause")
setRunningStatus(gamaB,"launch")
runningStatusSet(gameA, "pause")
retrieveState(gameA)
stateRetrieved(state1)
runningStatusSet(gameB, "launch")
setState(state1)
stateSet(true, gameB)
setRunningStatus(gameB, "run")
setRunningStatus(gameA, "terminate")
runningStatusSet(gameB, "run")
unregisterComponent (gameA)

Figure 27: automatic migration of an application with two components

In this process, the following steps are performed:

1. A device (PDA) performs a registration to the Migration Orchestration module. This task can be performed by the client side component of the Migration Orchestration (a Migration Orchestration Adaptor) or directly by an OPEN application.

2. When an OPEN application with two components (chatA and gameA) is started by the end user on the device PDA, the application and every application component are registered on the platform. The application can directly send the registration request to the Migration Orchestration Server or use a Migration Orchestration Adaptor.

3. The Migration Orchestration registers the application and its components to the Application Logic Reconfiguration (ALR) module.

4. A new device (Set-top box) is available. It performs a registration to the Migration Orchestration.

5. The Trigger Management module analyzes some context information and it establishes that a partial migration of the application is required. In particular, the component gameA needs to be migrated from the device PDA to the device Set-top box. The migration request is sent to the Migration Orchestration module. The Migration Orchestration checks if it is possible to fulfill the request and then it signals to the Trigger Management the check result. In this example the migration request is accepted.

6. The Migration Orchestration pauses the component gameA in the device PDA and launches an instance of the application with only the game component (gameB).

7. A confirmation is received that the component gameA has the running status “pause”, and then an internal state request of gameA is sent.

8. When the state of gameA is retrieved and the launch of the new component (gameB) on the device B is completed, the state of gameA is transferred to gameB.

9. The migrated component “gameB” is started on the device Set-top box and, when this task is correctly completed, the source component “gameA” is terminated.

10. The application on the device PDA, unregisters the component gameA because it is no longer running.

3.6.4. MIGRATION OF A WEB APPLICATION

Figure 28 shows the sequence of communications occurring between the various modules involved in the OPEN Platform for a migration of a Web Application. As you can see from the figure, the application currently running on the source device asks the Migration OPEN Server Orchestrator to retrieve the user interface of a specified component (the ID of the component is passed as an input parameter to the *retrieveUI()* method of the OPEN Server Orchestrator). As a consequence of this calling, the OPEN Server Orchestrator asks the OPEN Server Web State Handler to actually get the UI of the specified component (see *getUI()* method), also specifying as an input parameter of this method an optional callback which will allow to discriminate the type of request: if the callback parameter is not specified the request will be synchronous, otherwise it will be asynchronous. After the Web State Handler retrieves the UI for the specific component and delivers it to the OPEN Server Orchestrator, the latter module sends it to the application on the source device. It is worth pointing out that this is the behavior occurring when a user accesses a web page, even without triggering the request of any migration.

However, at a certain point a request for migration could be triggered, either in an automatic way (see *startMigration()* method in Figure 28) or in a manual way (namely: user-activated, see *ManualMigrationRequest()* method in Figure 28). Please note that the fact that both of such possibilities are available at a certain point has been highlighted in the Figure 28 by putting the two associated methods at the same temporal level. In particular, the possibility to trigger a manual migration request comes from the Migration OPEN Client Orchestrator component (which is running on the source device) and it is directed to the OPEN Server Orchestrator, while the automatic migration trigger comes from the OPEN Trigger Manager module, and it is sent as well to the OPEN Server Orchestrator.

After receiving one of such requests, the OPEN Server Orchestrator asks the OPEN Server Policy module the needed authorization for issuing a migration (*allowMigration()* method in Figure 28) and, in case the authorization for carrying on the migration is received, the OPEN Server Orchestrator asks the OPEN Client Orchestrator of the source device to retrieve the state of a specified component (see *retrieveState()* method in Figure 28). After the state of such a component has been retrieved, the OPEN Server Orchestrator pauses the application on the source device and then it asks the OPEN Server Web Adaptation to adapt the user interface to the characteristics of the target device -both the retrieved user interface and the ID of the target device are passed as input parameters to the *adaptUI()* method. The latter method, carried out by the OPEN Server Web Adaptation module, will perform a reverse engineering process of the retrieved web page, so generating a logical UI description, and then it will semantically redesign this logical UI description for the new target platform. After this, the OPEN Server Orchestration module will ask the OPEN Server Web State Handler to set the state of the user interface: this will produce a new logical description of the UI for the target device in which the state has been updated with the state information that was saved at the time the migration was requested. After doing this, the OPEN Server Orchestration module asks the OPEN Server Web Adaptation module to actually transform such a logical description for the target device in a specification exploiting a particular implementation language (this is the final user interface that should be loaded on the target device). In order to perform this, the OPEN Server Orchestrator asks the OPEN Client Orchestrator (target device) the authorization to activate a migration on the target device (see *incomingMigrationRequest()* method in Figure 28). If this request is accepted by the target device, then the OPEN Server Orchestration module asks to close the application on the source device, and then uploads the final UI on the target device by asking the OPEN Client Orchestrator on the target device to set the received UI as the new UI (*setAdaptedUI()* method). After receiving the callback (*adaptedUISet()* method) from the target, the OPEN Server Orchestrator can start the application on the target device. From this point, the user can continue the interaction on the new device by interacting with an UI that has been adapted to the characteristics of the target device and which also contains the state updated up to the point when the migration was triggered on the source device.

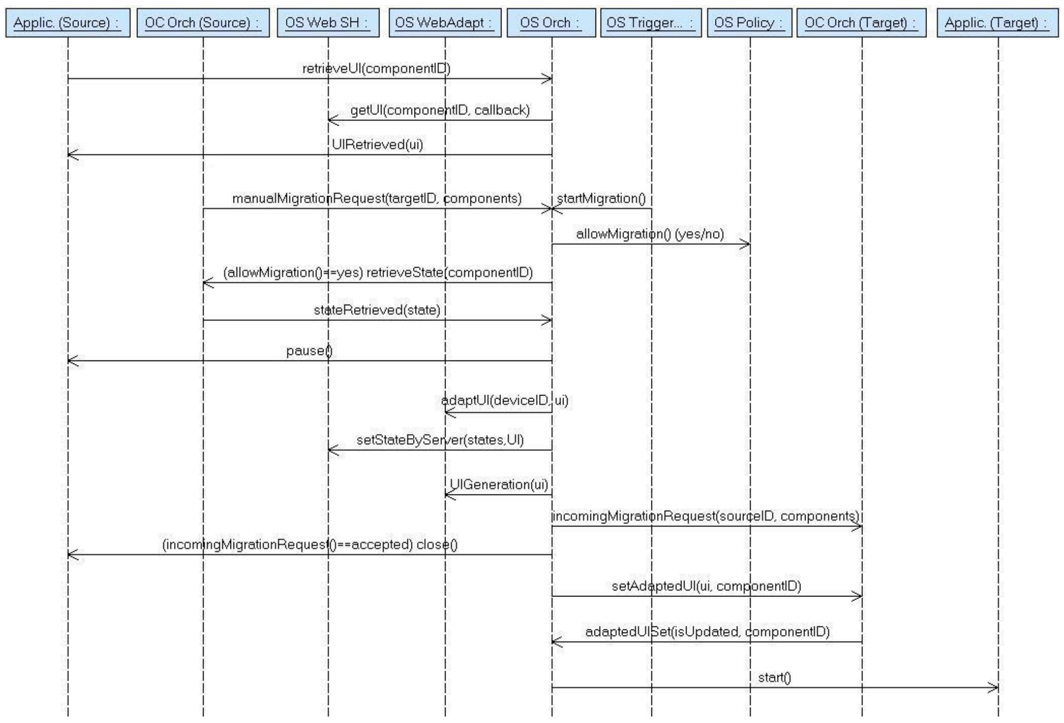

Applic. (Source) :
OC Orch (Source) :
OS Web SH :
OS WebAdapt :
OS Orch :
OS Trigger... :
OS Policy :
OC Orch (Target) :
Applic. (Target) :
retrieveUI(componentID)
getUI(componentID, callback)
UIRetrieved(ui)
manualMigrationRequest(targetID, components)
startMigration()
allowMigration() (yes/no)
(allowMigration()==yes) retrieveState(componentID)
stateRetrieved(state)
pause()
adaptUI(deviceID, ui)
setStateByServer(states,UI)
UIGeneration(ui)
incomingMigrationRequest(sourceID, components)
(incomingMigrationRequest()==accepted) close()
setAdaptedUI(ui, componentID)
adaptedUISet(isUpdated, componentID)
start()

Figure 28: The sequence of communications between the various modules in case of a web migration.

3.7. STATE HANDLING

During a migration, the application internal state is preserved. In this way, the continuity of the application usage on several devices is granted to OPEN users.

The OPEN Migration Service Platform offers the option to maintain the state of an OPEN aware application (i.e. an application that offers an OPEN client interface) and of a web application accessed using a proper proxy.

3.7.1. LOCAL APPLICATIONS STATE HANDLING

The state of an OPEN application component is represented using "State" objects. It is retrieved from a component using the method retrieveState, and it is set using the method setState. Both of the methods must be implemented by OPEN applications.

The main property of a State object is the string "hasState" that represents the current application state. There are no requirements on the state representation and its format is defined during the development of OPEN applications.

In the current version of the OPEN Migration Service Platform the state of a component is not permanently stored by the OPEN Server, but it is simply transferred from the source to the target device

3.7.2. WEB APPLICATIONS STATE HANDLING

The goal of this module is handling the state of the interactive application so that it will be captured on the source device and, after some adaptations, opportunely transferred to the UI that will be loaded on the target device, so allowing the user to seamlessly continue the interaction from the point it was left on the source device.

In the case of a web application, the state handler (client side) is implemented as a JavaScript module running on the source device and capturing the application state. Whenever a migration request is received, it sends the captured state to the OPEN Orchestrator. Then, the Web State Handler (server side) will manipulate such state information of the current web page and associate it to the logical description automatically generated for the target device, in order to deliver an adapted version of it. This adapted version of the logical UI which will be used for generating the implementation of the application that will be finally uploaded on the target device.

3.8. SECURITY ISSUES

The Orchestration Management module communicates (using XML-RPC over an IP network) with all of the OPEN devices and moreover it manages the state transfer during migrations. For that reason, some security issues have to be addressed,

The following security issues related to the Migration Orchestration module have been identified:

- Devices registration: It is important that only a restricted group of devices are allowed to register to the platform. For example, an authentication mechanism could be useful to avoid that malicious devices could be allowed to retrieve users' data. There could be two kinds of devices:
    - Personal devices. They are devices (for example a PDA, or a laptop) that are used by a single user. Each user should have his access credentials to be used during the registration of each of her/his personal devices.
    - Shared devices. A shared device (for example a smart wall or a set top box) is used to share information and functionalities among a group of users. In this case, during the registration of the device, the access credentials associated to the group of users have to be used. However, sensitive data about a single users should never be sent to a shared device.
- Communication security: The SSL (Secure Sockets Layer) cryptographic protocol could be used to encrypt client-server communications in the OPEN Migration Service Platform.

3.9. SUMMARY

The Migration Orchestration module has been developed for the management of all of the tasks required to perform a migration in the OPEN Migration Service Platform.

The main characteristics of the Migration Orchestration module are:

- The Migration Orchestration Server is a java application running on the OPEN Server. It is the main component of the Migration Orchestration module and it can receive requests from OPEN clients via XML-RPC.
- A java adaptor has been developed to be used as the client side component of the Migration Orchestration module. Its usage is not mandatory and it is possible to use another implementation or to include it in an OPEN application.
- The Migration Orchestration can perform a check of the device capabilities and the application components requirements in order to avoid the migration of a component to a device that does not support it.
- The Migration Orchestration can manage the migration of OPEN local applications and generic web applications. Moreover, it is possible to perform a partial migration of an application.
- Migration triggers are asynchronous and the Migration Orchestration can manage several migrations at the same time. The only limitation is that it is not possible to accept a migration request for an application component that is involved in an ongoing migration.

4. TRIGGER MANAGEMENT

Trigger Management is a module that is used to determine when and where to migrate applications. It is located in the platform in between the Context Management Framework, Migration Orchestrator and Application Logic Reconfiguration as shown in Figure 29.

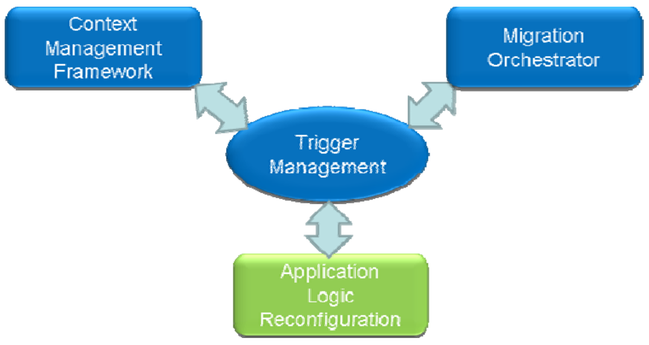

Context Management Framework
Migration Orchestrator
Trigger Management
Application Logic Reconfiguration

Figure 29: Interaction of Trigger Management Module

The details of the interaction are described in the following.

4.1. INTRODUCTION

The purpose of the trigger management (TM) module is to decide the configuration of the migratory system and its applications in order to fulfill the requirements of the user and the applications. If changes are needed in the way applications are running or deployed, a migration is performed. To decide upon a configuration, TM collects information about possible configurations of applications, devices and networks. These configurations constitute the options to choose between. The decision is made by comparing the requirements of the option to the capabilities of the platform and the environment – i.e. the context of the configuration/applications. Figure 30 depicts the external modules necessary for TM to collect the described information.

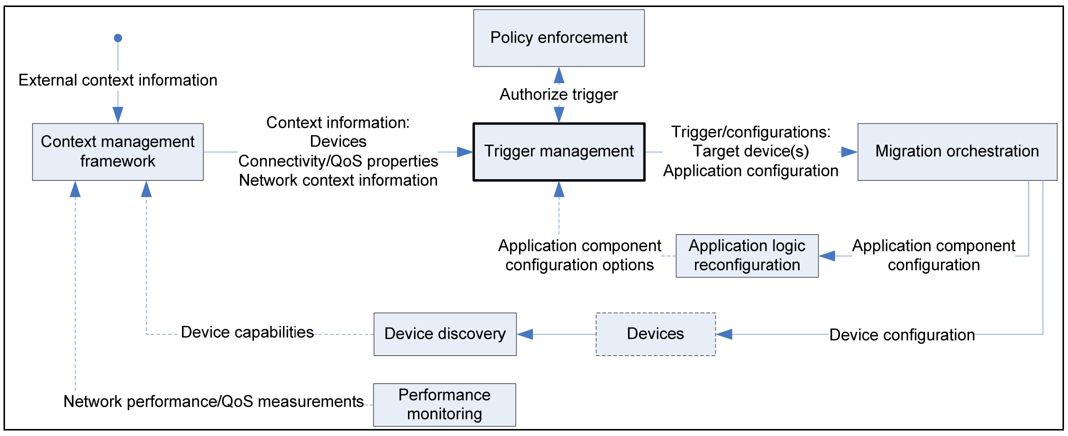

Policy enforcement
External context information
Authorize trigger
Context information:
Devices
Connectivity/QoS properties
Network context information
Context management framework
Trigger management
Trigger/configurations:
Target device(s)
Application configuration
Migration orchestration
Application component configuration options
Application logic reconfiguration
Application component configuration
Device capabilities
Device discovery
Devices
Device configuration
Network performance/QoS measurements
Performance monitoring

Figure 30: Overview of external modules necessary for trigger manangement to carry out its task

Knowledge about the requirements and context has different origins:

- Available configurations and requirements come from application logic reconfiguration.
- Device capabilities come from device discovery
- Network capabilities come from device discovery and performance monitoring
- User preferences may be defined externally e.g. Default Delivery Context (DDC), [DDC] from W3C which specifies minimum environment for experiencing the web, and be enforced through policy enforcement

Every piece of context information going into TM is formatted as XML in a CMF-object.

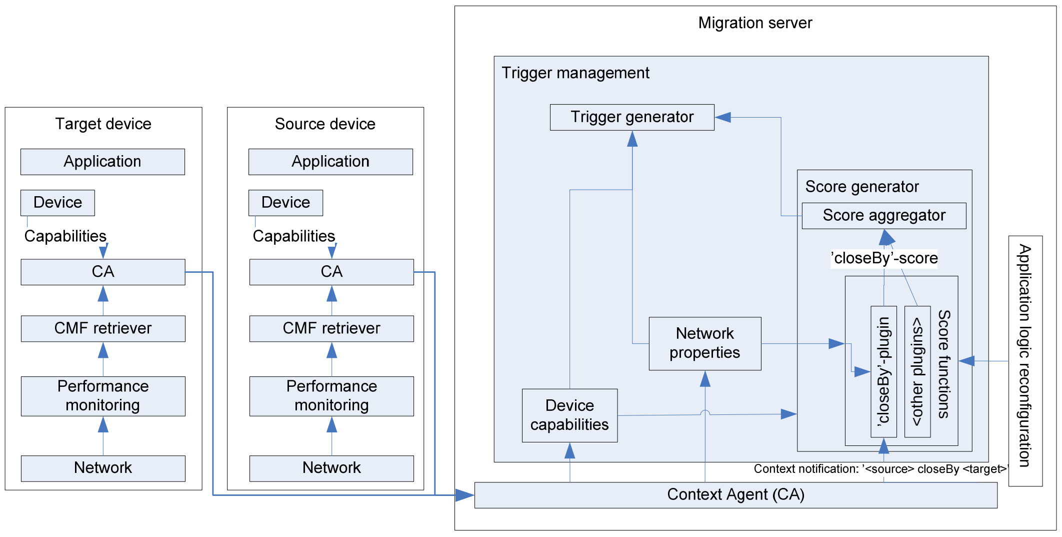


Figure 31: Overview of internal modules in trigger management and how they interact with the context management framework to collect network context information and ALR to have configuration options and component requirements

TM decides which configuration to choose based on a set of configurations made available by the ALR. The ALR collects information from the registered device about which application components are registered, and decides which combinations of components – constituting applications – are viable in certain settings. A *configuration* is a set of application components on a set of devices, which is determined able to run by the ALR component.

Deciding which configuration to used is done either manually based on either the user input (cf. section 4.2) or automatically based on observations of contextual information (cf. section 4.3). Figure 31 illustrates the internals of the TM module. The context information and the application configurations are input to the system mainly on the end terminals (some context information may come directly from system components) and the application configurations are provided by the ALR client-side components of the ALR framework [OPEN D4.1]. The later sections describe use of scores in order to rank configurations to make a decision, and the placement of these score functions are also illustrated as an internal module on the server-side of the TM. TM does not have client-side components in this solution, as it is expected that all input from the client-side is communicated from the input suppliers (manual trigger from applications or migration client, or context providers) through the CMF.

Output from TM is sent to the migration orchestrator for execution in the form of a configuration in the same format as received from the ALR.

4.2. MANUAL MIGRATION TRIGGERING

Migration triggers can be generated manually by the user in the application or on the migration client. This form of triggering is for the most part by-passing trigger management, as a manual trigger is interpreted as an intentional action from the user side.

Another situation where the user is involved is when the TM module automatically generates triggers, as described in the following section. In this case, the user may be asked to accept or reject a certain migration. If the user does not want to be asked each and every time a trigger is generated, authorization policies can be specified.

Such policies can be managed in the context management system and the decision about a certain trigger could be followed by a request to remember the decision on the trigger, which means that in the future, the user would not be consulted when the particular trigger is generated. This could also be more expressive in the sense that the policy could be on trigger types, devices, device types or alike instead of individual triggers. This would be specified in the same semantic language as the configurations from the ALR, since a migration means to change from one configuration to another, and the question asked to the user regards whether these individual transitions (/migrations) are acceptable.

4.3. AUTOMATIC MIGRATION TRIGGERING

TM can make its decision on which configuration to use in several ways, characterized by level of complexity and how much information is used to make the decision. In the following, two methods are described; a score-based approach that has a low complexity but is not able to incorporate much information, and a Markov Decision Process (MDP)-based approach that has a higher complexity because it uses more information.

In general, the decision making mechanism in TM relates observed context information to requirements of the configurations of the applications to migrate. TM chooses the configuration that has the best fit to the observed context, and if this configuration is another than the one currently running, a migration trigger is issued, with the selected configuration as target.

The context information is collected via the CMF and provided by various monitoring components in the OPEN framework. In this chapter, the performance monitoring component is heavily used as context information provider, but context information could be provided by other components such as device discovery or the interface to service enablers in the network. The configurations, and thus requirements, are provided by the ALR.

4.3.1. SCORE-BASED DECISION APPROACH

In the score-based approach, a score is calculated using a utility function for each configuration based on all its requirements and the state of the required parameters at a given time.

The utility function assigns a value to the user perceived application quality of the requirement being met by the system, and the extent to which it is met. The system is represented by the observed context information. The quality is represented by any quality-of-experience parameter, for instance Perceptual Evaluation of Speech Quality (PESQ) for speech applications or Perceptual Evaluation of Video Quality (PEVQ) or Peak Signal-to-Noise ratio (PSNR) for video applications. The utilities are all represented in the same scale using mean opinion score (MOS) values so every quality-of-experience parameter must be mapped into MOS values, if they are not already so. The MOS scale represents the user perceived quality of a given application in a scale from 1 to 5, 1 being very bad and 5 being excellent. As the quality is application specific, the mapping is performed by the application developer, and thus general to the migration framework. An example of mapping between PSNR and MOS is described in [Klaue03].

The final utility functions taking system parameters as input and giving quality as output are also specified by the application developers. These functions can be simple binary expressions (>,<,=) mapping input values or intervals to MOS, tabular expressions mapping input and output, or continuous or discrete functions. A general utility function is depicted in Figure 32. The system parameters could be observed in multiple ways, ranging from simple instantaneous samples, over moving averages and auto-regressive functions, to more complex (hidden) Markov models.

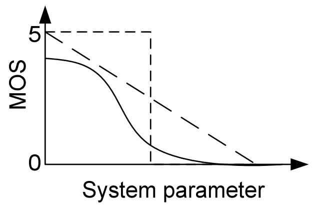


Figure 32: Examples of utility functions mapping system parameters to user perceived quality (MOS). Packet loss rate in the network or delay are examples of parameters.

The general algorithm for making decisions based on scores it as follows:

1. Observe the system parameters *P*
2. Calculate the corresponding utility for each application configuration for each parameter
3. Average all utility values for a configuration C into one score using the following equation:

$$S(C,P_C)=\frac{1}{|P_C|}\sum_{i\in P_C} w_i U_{C,p_i}(p_i(t))$$

Here w_i are weights that can be used to rank utilities against each other, decided by the application developer and $|P_C|$ the number of parameters in configuration C.

An assumption in the expression is that there is no correlation between system parameters, which may not always hold (for instance, there is correlation between packet loss and delay in congested networks). These correlations can be included in the expression as additional utility functions, but have been left out here for simplicity.

4. Select configuration with maximum score: $C^* = \max_C \left(S(C, P_C) \right)$

In the following we present an example of how to use the score based approach to make decisions in TM.

The example system consists of one video streaming application that can operate in two configurations; high quality (HQ) and low quality (LQ). The HQ video stream is more sensitive to dropped packets (due long delays or drops in the network) than the LQ video stream. If the network becomes very congested packets are dropped or delayed in the bottleneck queue. Late packets at the video client are dropped by the application. The trigger management system continuously chooses which configuration to use based on observed packet drops. A dropped packet in a streaming application is identified by being missing when due for playback.

We need 2 utility functions when we have 2 configurations evaluated over 1 system parameter; U_{HQ}*(packet drop rate)* and U_{LQ}*(packet drop rate)*. These functions are derived from other works (voice example is shown in [Schwefel07]), since such quantification is not within the scope of this project, and plenty of research has investigated these relations before. The two functions used in the example here are illustrated in Figure 33. Other examples of utility function could be MOS-values in table form for screen resolution:

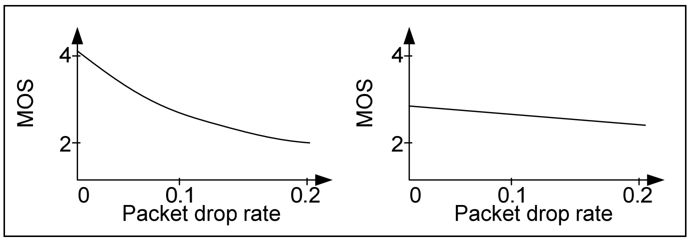


Figure 33: Example utility functions for voice.
Left is HQ stream (from [Schwefel07], right is LQ stream (created from left).

Screen resolution [px]	*Utility [MOS]*
320x240	1
640x480	2
800x600	3
1024x768	4
1600x1200	5

Table 1: Example utility function for screen resolution in a given video application.

Example mappings of video PSNR are depicted in Figure 34.

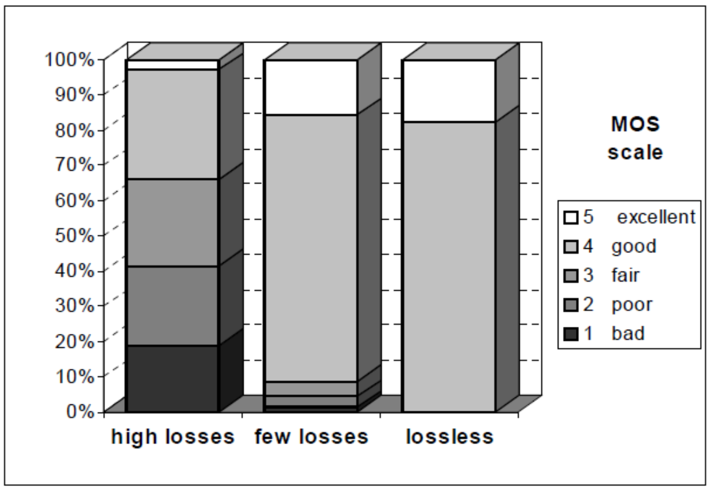


Figure 34: Example MOS utility from video quality under different network conditions (high loss = 25% packet loss, few loss = 5% packet loss) (from [Klaue03])

When the system is running, TM constantly evaluates the system parameter – in this example the packet drop rate – and calculates the instantaneous utility of all configurations. To illustrate this, two scenarios are considered; one *normal* (few losses, packet drop rate 0.01) and one *congested* (many losses, packet drop rate=0.2). We can then use the utility functions to calculate the utilities of all configurations for both scenarios:

- Scenario *normal*:
  - HQ: U(mode=HQ, packet drop rate=0.01) = 4
  - LQ: U(mode=LQ, packet drop rate=0.01) = 2.8
  - → selected configuration: HQ; score = 4
- Scenario *congested*:
  - HQ: U(mode=HQ, packet drop rate=0.2) = 2
  - LQ: U(mode=LQ, packet drop rate=0.2) = 2.8
  - → selected configuration: LQ; score = 2.8

We see that using the score based approach we have a TM module that can make decisions on what configuration to use based on observed system parameters.

Advantages:

- Utility functions are fast, since they do not require complex computation, but are merely lookups or function output calculation from input
- The TM architecture supports any kind of score function, as long as the score function and the corresponding context information are available.

Disadvantages:

- Since a score is based on the instantaneous calculation of utilities at time *t*, the selected configuration can only be considered best in the that moment of time.
- The score approach does not incorporate expected future events/predicted observations as a simple extension – this would require some form of

extrapolation/prediction of observations, based on heuristics or models of the environment

- The score-based approach is not able to consider future decisions in the quality evaluation (e.g. given I choose A next time and B after that, A is the best choice now)

Including predicted utility in the score function in the score-based approach, i.e. taking into account estimated future parameter values, can be performed in simple cases. Simple cases mean situations where only few future values are needed, for instance one point in time t+Δt into the future.

The score function would then look as follows:

$$S(C,P_C)=\frac{1}{|P_C|}\sum_{i\in P_C} w_i\left(U_C(extrapolate(t,p_i(t),t+\Delta t))\right)$$

Here, an additional parameter extrapolation is introduced and used in the calculation of the utility. The size of Δt is a design decision as it depends on the prediction model quality.

4.3.2. MODEL-BASED DECISION APPROACH

In the previous section, we illustrated how decisions can be made fast in the TM by calculating scores based on utilities for each configuration based on the current state of the system parameters. This method is fast and straight-forward, once the utility functions are in place, but it is not possible to include the effect on the system (or the application quality) of making a decision.

In this section we show how a more complex framework of Markov Decision Processes (MDP) can be used to integrate knowledge about the environment in the decision making. The objective of the framework is to find the optimal decision policy, which inherently considers (all) future decisions. Additionally, the MDPs include a notion of reward, which on one hand represents the utility of being in a state, but also can represent the cost of making a transition between states.

In order to make decisions based on a MDP, all system states must be modeled in a Markov chain, including system states *S* and possible actions *A* to take in each state. Moreover, transition probabilities *P(s,a,s')* between all states are specified, and cost/reward *R(s,a)* for taking an action in a state is specified.

The example application and scenarios specified in the previous section is depicted as a MDP in Figure 35. Application modes are HQ and LQ video, and network state is either *normal (N)* or *congested (C)*, defined by the packet drop rate specified in the previous section.

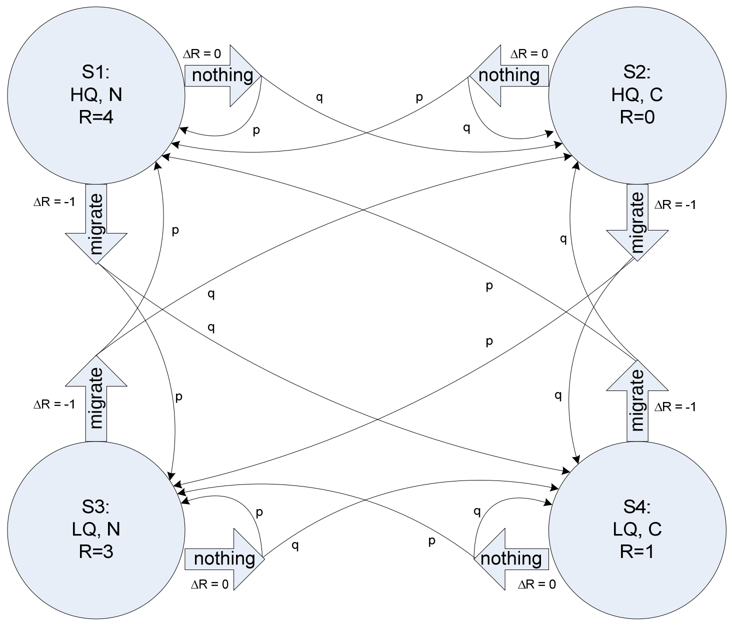


Figure 35: Example application and scenarios modeled using a MDP.

The TM can choose to migrate between the application modes (LQ/HQ), but not control the state of the network, so the network state may change independently.

P(s, a, s') for the example is listed in Table 2 for *a='nothing'*, which means that the TM does not trigger a migration.

s/s'	*1*	*2*	*3*	*4*
1	p	q	0	0
2	p	q	0	0
3	0	0	p	q
4	0	0	p	q

Table 2: P(s, *a='nothing'*, s').

P(s, a, s') for the example is listed in Table 3 for *a='migrate'*, which means that the TM chooses to migrate, i.e. change the quality of the video stream, either from HQ to LQ or vice versa.

s/s'	*1*	*2*	*3*	*4*
1	0	0	p	q
2	0	0	p	q
3	p	q	0	0
4	p	q	0	0

Table 3: P(s, *a='migrate'*, s').

In the model, the knowledge of the dynamics of the underlying network is represented by *p* = P(s(t)=N | s(t-1)=N) and q = 1-*p*.

Reward of being in a state is determined by the score functions, expressed in MOS values, because a state represents an application mode and a system parameter at a certain value, and the reward expresses the utility of being in that state, in the example defined as
R(s) = [4 0 3 1]. The rewards are calculated by mapping the utility functions output specified in the previous section to the states and actions in the MDP.

Migrating between modes has a cost as it interrupts the user, which is represented by a lower reward when taking an action that results in a migration compared to taking an action where no migration happens. Migration cost = 1 → rewards R(s, a) = Table 4. As the rewards are represented as MOS values, we employ a lower bound on the reward at 0, as the MOS value -1 is undefined.

s/a	*nothing*	*migrate*
1	4	3
2	0	0
3	3	2
4	1	0

Table 4: Rewards R(s,a) for the example application calculated from the utility functions.

The solution to the MDP is a decision policy π(s), which specifies which action *a* to take in state *s.* By definition, the solution is a optimal policy in the sense that following the policy maximizes the expected reward of being in a state on a long-term basis. To calculate the reward, the following function is maximized

$$V(s,a) = R(s,a) + \gamma \sum_{s'} P(s,a,s')V(s',a)$$

and the policy to use for making decisions is found using

$$\pi(s) = \arg\max_{a} \left\{ R(s,a) + \gamma \sum_{s'} P(s,a,s')V(s',a) \right\}.$$

Here, γ is a discounting factor, $0 < \gamma \leq 1$, in this example set to 0.9.

For the simple two-mode, two-action problem described here, with the rewards as specified and the value spectrum for *p*, the optimal decision policies are depicted in Figure 36.

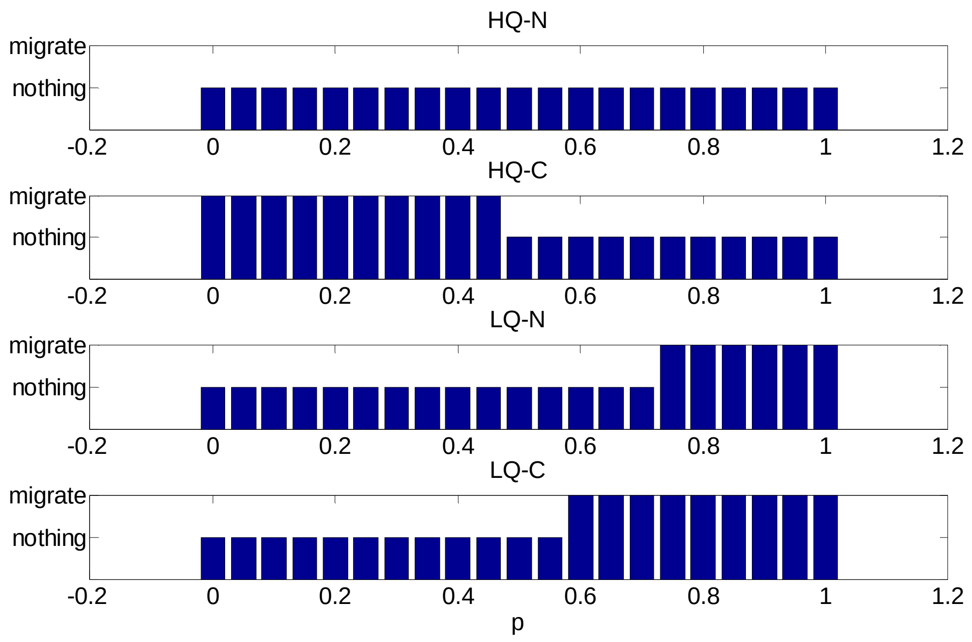


Figure 36: Decision policies as given by solving the MDP, for different values of *p*.

The result shows that the optimal decision, i.e. the one that maximizes the expected utility, depends on the value of *p* and not just the state of the network, in this example *normal* or *congested*. If we define the network states in the MDP by the packet drop rate from the score based example, we see the differences between the two approaches. In the score-based approach, we can define a threshold around which we get certain decisions. If U(HQ)>U(LQ), then we use HQ, otherwise we use LQ. Here hysteresis can be applied to avoid transitions too often, which could also disturb the user unnecessary. If we define the threshold from the utility functions governing HQ and LQ, we obtain a threshold value where the two graphs intersect, around 0.1. This threshold defines the two states N (<0.1) and C (<0.1) This means that below a packet rate at 0.1, we choose HQ, and above 0.1 we choose LQ. In the MDP we add the dimension of the probability of being either state. From Figure 36 we see that the MDP is able to capture this feature and use it in the decision making, whereas the score-based approach is not.

Two approaches can be taken to use the found decision policy; an offline or an online version. In the offline version, the MDP is solved beforehand and the policy/policies are deterministic. The network is assumed to behave as modeled, meaning that the transitions between *normal* and *congested* state is governed by *p* for the chosen policy. As an extension, *p* could be estimated along with the current state *s*, and the (p,s) combination could by lookup be used to first determine the policy, and next to determine the action to take.

In the online version, the MDP is re-calculated, or solved, at given time intervals, incorporating the knowledge from the observed context.

One thing that needs to be considered in order to implement the model-based approach is that MDPs assume known state of the system, so any uncertainties of the observation process complicates the process either by extending the model to include true and observed state (which would require constraints on the solution, as the true state still cannot be known [Qvist07]) or by using a partial observable MDP (POMDP) [Cassandra1995], which integrate uncertainty directly in the model as probability distributions over states.

4.4. SUMMARY

The role of TM is to decide which configuration delivers the best user experience. This chapter described the architecture and modules necessary to provide such a decision framework. TM uses input from several sources. Depending on the autonomy of the system, TM uses either several system components or direct input from the user. Relevant system components are the context monitoring system, in OPEN managed by the CMF (delivered by different context providers such as performance monitoring and device discovery) and the application logic reconfiguration component, which has the knowledge of the eligible application configurations.

The chapter also described to methods for making decisions based on the input; a simple scoring-based and a more complex model-based using MDPs. Through example both methods were explained and proven useful for decision making in the TM. One important fact here is that the specific requirements and parameters used in the process are irrelevant to TM as it generalizes decisions to be based on MOS, so basically any application configuration and any type of system parameters can be used, as long as they are supplied by the application developer.

The model-based approach for generating automatic triggers has the advantage of outputting optimal decision policies (if they exist). The fact that the decision policy of the score-based method did not appear among the optimal policies of the model-based approach, show that – although applicable – the score-based method does not optimize the user experience in the given example. The size of the gain of using the model-based approach over the score-based is evaluated in the context of WP6 and the results are reported in D6.7.

5. MOBILITY SUPPORT

5.1. INTRODUCTION

This chapter discusses the analysis and architecture of Mobility Support module for the OPEN Migration Service Platform (MSP). As migration process involving device and network changes, the primary objective of Mobility Support is to ensure that such changes do not affect the communication between the client and the application servers. Figure 37 indicates the interaction of Mobility Support within OPEN to achieve the above-mentioned goal.

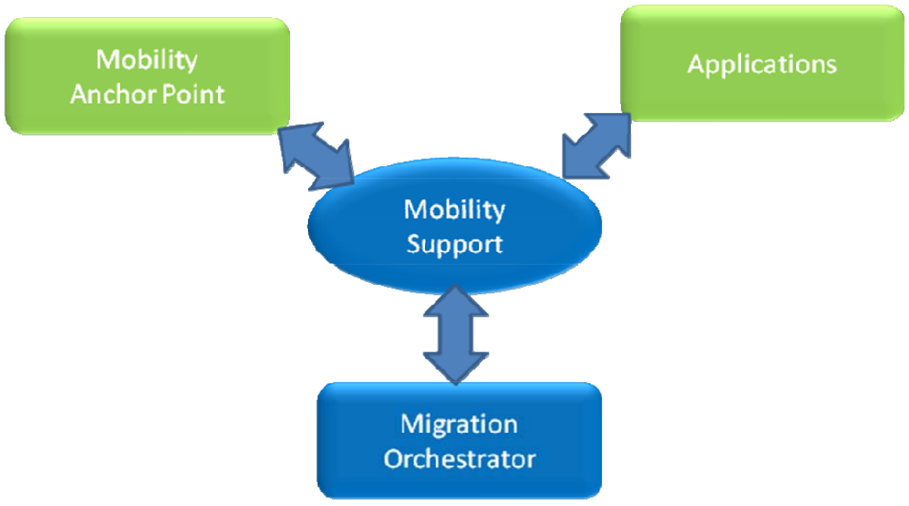


Figure 37: Interactions of Mobility Support with other components

First, the requirements for the Mobility Support module in OPEN are listed in Section 5.1.1. The solutions for Mobility Support are discussed in details in sections 5.2 and 5.3, where terminal and session mobility are solved, respectively. In Section 5.4, architecture for the Mobility Support module in OPEN is proposed based on the conclusions from Section 5.2 and 5.3. Section 5.5 discusses security related aspects of the Mobility Support while Section 5.6 concludes on the analysis and proposed architecture.

5.1.1. REQUIREMENTS FOR MOBILITY SUPPORT MODULE

The Mobility Support module is defined as “the support provided for the user to continue working while mobile, moving from one place to another”, and also “the ability to continue the user’s current service seamlessly across multiple devices”. According to [OPEN D1.3], the OPEN Mobility Support is mainly concerned with:

- **Terminal mobility**: The user’s ability to take his terminal, move across heterogeneous networks, and continue to have access to the same set of subscribed services.
- **Session mobility**: The user’s ability to maintain an active session while switching between terminals. For example, a user may join a multi-party teleconference session using his PC at work. When he leaves to go home, he can switch the session to his mobile phone

without interruption, and can continue participating in the conference during his walk home.

The requirements for Mobility Support module depend on the type of applications. For example, an OPEN prototype is analyzed in Appendix B, which supports resuming at application level when mobility or migration has occurred. The resume here provides the needed functionality that Mobility Support would have provided. However, not all application servers are OPEN-aware, and not all of them support resume at application level. Therefore, the Mobility Support module must make network changes transparent to non OPEN aware applications, meaning terminal and session mobility should be handled whenever the application itself is not able to handle this. Furthermore, the Mobility Support must not break applications which are able to handle mobility themselves.

5.2. TERMINAL MOBILITY

Terminal mobility has been receiving much research attention in the past years, and therefore a number of solutions already exist, for example the Mobile Internet Protocol (MIP), the mobile Stream Control Transmission Protocol (mSCTP) and the Session Initiation Protocol (SIP). In this section, they are briefly discussed, and a solution will be chosen to support terminal mobility in OPEN.

5.2.1. THE MOBILE INTERNET PROTOCOL

Mobile Internet Protocol is designed to allow a mobile device to roam between different administrative domains (sub network), and yet maintaining the same IP address. This is done by introducing the Home Agent (HA), a static network router at the home network of the Mobile Node (MN), to defend the MN’s home address while the MN is roaming. Packets sent to the MN’s home address shall be tunneled to the MN, using the MN’s foreign addresses, which are given to the MN by the foreign networks. The MN updates its foreign address record at the HA by sending Binding Update (BU) message every time it obtain a new address, or the current binding has expired.

5.2.2. THE MOBILE STREAM CONTROL TRANSMISSION PROTOCOL

Stream Control Transmission Protocol (SCTP) is a new reliable transport protocol, featuring ‘multi-streaming’ and ‘multi-homing’. ‘Multi-streaming” refers to the capability of SCTP to transmit several independent streams of chunks in parallel, for example transmitting Web page images together with the Web page text. ‘Multi-homing’ means one (or both) endpoints of a connection can consist of more than one IP interface, enabling possible failover between redundant network paths. Connection between two SCTP endpoints is referred to as an association.

RFC 5061 defines Stream Control Transmission Protocol (SCTP) Dynamic Address Reconfiguration, which allows an SCTP stack to:

- Dynamically add an IP address to an SCTP association,

- Dynamically delete an IP address from an SCTP association, and
- To set the primary address the peer will use when sending to an endpoint.

Such extension makes SCTP a candidate for mobility support in OPEN MSP. In [Seok04], it is referred to as Mobile SCTP (mSCTP), and the mSCTP's soft-handover is defined:

- Terminal obtains new IP address at new location
- Adding this new IP address to SCTP association
- Changing the primary IP address to the new IP address
- Delete the old IP from SCTP association

Such procedure allows the MN to move from one access router to another, and still maintain the active connections with CNs.

5.2.3. SIP

The SIP protocol was designed to support establishing, controlling and tearing down multimedia sessions at the application layer and is therefore also a candidate for solving terminal mobility in OPEN.

As stated in [Wedlund99], if terminal mobility occurs while a SIP session is established, the mobile host will have to send a new INVITE to the correspondent host with the same session identifiers as the original SIP session. The new IP address should be put in the Contact field of the SIP INVITE message which tells the correspondent host where subsequent SIP messages should be sent to. To redirect the data flow, the new IP address is indicated in the SDP field where the transport address is specified.

The disadvantages of using SIP for terminal mobility is that it requires end-to-end SIP support at the application layer and that the delay of re-establishing a SIP sessions is higher than e.g. MobileIP.

5.2.4. COMPARISIONS OF TERMINAL MOBILITY APPROACHES

As we have pointed out, the above-mentioned approaches allow for the MN to move from one sub network to another, while maintain the active connection. However, they differ in many aspects, which are summarized in Table 5.

	Mobile IP	**Mobile SCTP**	**SIP**
Support terminal mobility	Yes, via binding update	Yes, via dynamic address reconfiguration	Yes using re-INVITE
Number of updates	One	Multiple (one per SCTP association)	Multiple (one per SIP association)

Support UDP and TCP	Yes	Possible, via tunneling	Yes
Support IPv4 or IPv6 or both	Yes (with dual stack)	Yes	Yes
Mode of operations	Via 3rd party (Home Agent)	End-to-end	End-to-end
Require protocol support at Correspondent Node	No	Yes	Yes
Require protocol support at Mobile Node	Yes	Yes	Yes

Table 5: Comparison between MIP, mSCTP and SIP.

When a change of IP address occurs, both MIP, mSCTP and SIP require some signaling overhead to update the new IP address: Only one BU is needed to update the HA in case of MIP approach (the route optimization mechanism of MIP is not discussed here for the sake of simplicity), while multiple updates are requires in case of mSCTP and SIP, each update per one active mSCTP association or SIP session.

MIP supports all transport protocols, namely UDP, TCP and also SCTP while SIP is only guaranteed to support TCP and UDP but can use other transport protocols. However, for mSCTP, support of UDP and TCP requires extra tunneling and encapsulation, which increases the overhead and computational complexity. All protocols support both IPv4 and IPv6.

The approaches also differ in their mode of operation: MIP redirects the traffic through the third party (the HA), while mSCTP and SIP sends data directly between two hosts. Both operations have pros and cons. End-to-end (or direct) communication is faster, less overhead and independent from failure of the third party. However, end-to-end communication requires that the CN implement the protocol stack (such as mSCTP, SIP or MIP - in case of route optimization), which might not be applicable in many cases. Since the OPEN MSP aims at also supporting legacy AS, it is not safe to assume that the AS understands mSCTP, SIP or MIP.

Based on the above comparison and analysis, we choose MIP as terminal mobility solution for OPEN MSP. It can be implemented transparently to the OPEN MSP, from both application and controlling perspectives. The only requirements are the MIP stack to be installed at the OPEN-aware client, and the HA is setup at the home network, which will defend the client's home address. No interface is required between OPEN MS and the MIP stack or the MIP's HA.

5.3. SESSION MOBILITY

Unlike the terminal mobility, which can be solved transparently to OPEN MSP, the session mobility must be integrated into OPEN to allow for seamless migration process. Therefore, this type of mobility will be the main focus in this section.

In this section it is assumed that there is an OPEN MAP present in the network path from the AS to the target devices. The AS cannot be assumed to be OPEN aware, while the client applications can be either OPEN aware or not.

In [Moh07] applications are classified into 5 different categories according to their mobility management requirements. Because this classification is too broad (e.g., for different types of mobility), we define a new classification which focuses on session mobility as following:

- Class A - Applications with no session notion: This class contains all IP-based applications which do not require session notion. These include, for example, DNS resolution or Ping.
- Class B – Applications with "soft" session: This class contains TCP or UDP-based applications that do have session, and their sessions can be either short-lived or long-lived, but in general the cost of re-establishing their sessions is low. Here the cost should be measured from various aspects, such as network overhead, delay and also customer's experience. Examples of this class are Web browsing, email or telnet session. Some multimedia application, such as video-on-demand, can also be considered as "soft" session, since the client can request the server to resume from a previous position with minimal overhead.
- Class C – Application with "hard" session: This class consists of TCP or UDP-based applications with long-lived session, and the cost of re-establishing the session is relatively high. For example, the cost of re-establishing session of a VoIP call is dropped and interrupted communication, or in an online game is a game over if the client is disconnected.

Class A and B often do not need session mobility support, since the cost of resume the session is minimal. Session mobility support, however, makes sense for applications in Class C. The aim of introducing session mobility in OPEN is to reduce the cost of re-establishing session for applications in class C.

5.3.1. MOBILE IP FOR SESSION MOBILITY

The MIP protocol is designed for terminal mobility, and therefore it does not support session mobility by default. Assume that there are multiple applications running on the Source device, and all of them are connected to Internet from one network interface with a common IP address. Suppose that the user wants to transfer only one application and its related network connection(s) to the Target device. If the standard MIP is used to redirect traffic from Source device to Target device, all connections from / to the Source device's IP – no matter which application they belong to – will be migrated. This is due to the fact that the standard MIP is layer 3 protocol, and it does not understand the "data flow" notation. The complete migration of the Source device's IP to new device affect the other applications running on the device, as well as bringing a half to communication between the OPEN MS and the device.

Nevertheless, such shortage of MIP can be overcome by using the special MIP extensions, namely the “Multiple care-of-address registration” [Wakikawa09] and the “Mobile IPv6 flow routing over multiple care-of-addresses” [Soliman09] and [Gundavelli09]. The former supports mapping of multiple care-of-address into one home address, and the latter allows distributing data flow(s) selectively among those care-of-address. From now on, the extensions plus the standard MIP will be referred to as the Extended MIP in this document.

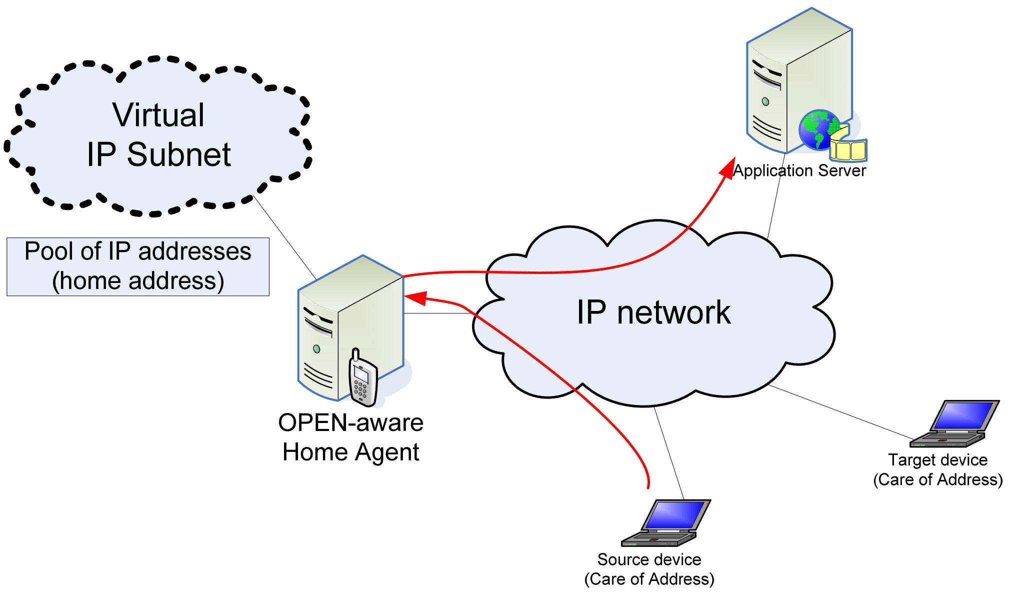


Figure 38: OPEN - Aware home agent and its virtual subnet.

Another issue needed to be solved before the Extended MIP could be used for session mobility support in OPEN is the Home Agent’s coverage. In order to redirect traffic from Source device to Target device, the two devices must share the same Home Agent, which controls the redirection. This might not be the case in the OPEN scenario, where the two devices are assumedly located in different IP subnets. To force these devices sharing the same Home Agent, a virtual IP subnet is proposed in Figure 38. An OPEN-aware Home Agent possesses a pool of IP addresses, which shall be assigned to both Source and Target devices. The Source device shall consider this IP address as its home-address, while the IP provided by its current subnet is considered as care-of-address. As a result, the network connection between the Source device and the Application Server shall always tunnel through OPEN-aware Home Agent, and so do the Target device’s connections.

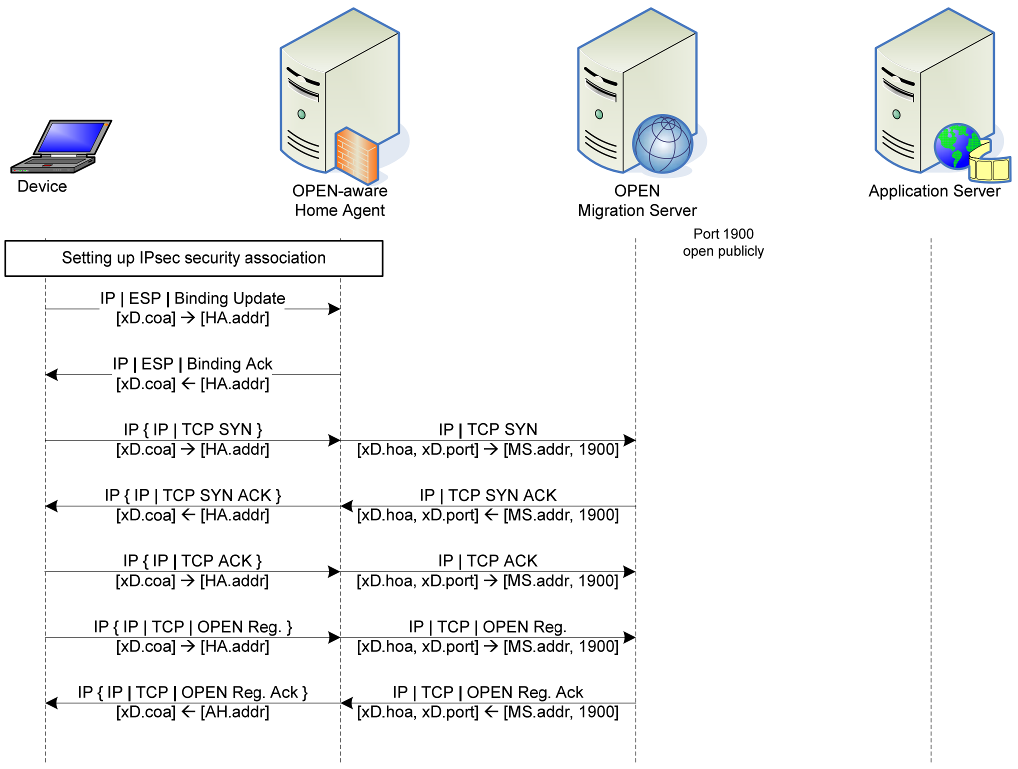

Device
OPEN-aware Home Agent
OPEN Migration Server
Application Server
Port 1900 open publicly
Setting up IPsec security association
IP | ESP | Binding Update [xD.coa] → [HA.addr]
IP | ESP | Binding Ack [xD.coa] ← [HA.addr]
IP { IP | TCP SYN } [xD.coa] → [HA.addr]
IP | TCP SYN [xD.hoa, xD.port] → [MS.addr, 1900]
IP { IP | TCP SYN ACK } [xD.coa] ← [HA.addr]
IP | TCP SYN ACK [xD.hoa, xD.port] ← [MS.addr, 1900]
IP { IP | TCP ACK } [xD.coa] → [HA.addr]
IP | TCP ACK [xD.hoa, xD.port] → [MS.addr, 1900]
IP { IP | TCP | OPEN Reg. } [xD.coa] → [HA.addr]
IP | TCP | OPEN Reg. [xD.hoa, xD.port] → [MS.addr, 1900]
IP { IP | TCP | OPEN Reg. Ack } [xD.coa] ← [AH.addr]
IP | TCP | OPEN Reg. Ack [xD.hoa, xD.port] ← [MS.addr, 1900]

Figure 39: Registration with Home Agent and OPEN Migration Server.

Figure 39 shows how OPEN device registers itself with the Home Agent and the Migration Server. In the diagram, the abbreviations in Table 6 are used.

xD	Device
SD	Source Device
TD	Target Device
HA	Home Agent
MS	Migration Server
AS	Application Server
SP	SOCKS Proxy
addr	IP Address
coa	IP Care-of Address
hoa	IP Home Address
IT | TCP | HTTP	IP datagram contains TCP and HTTP protocols
IP { IP | TCP | HTTP }	IP tunnel contains IP datagram with TCP and HTTP protocols
port	Transport Layer's port number

Table 6: Abbreviations used in the diagrams.

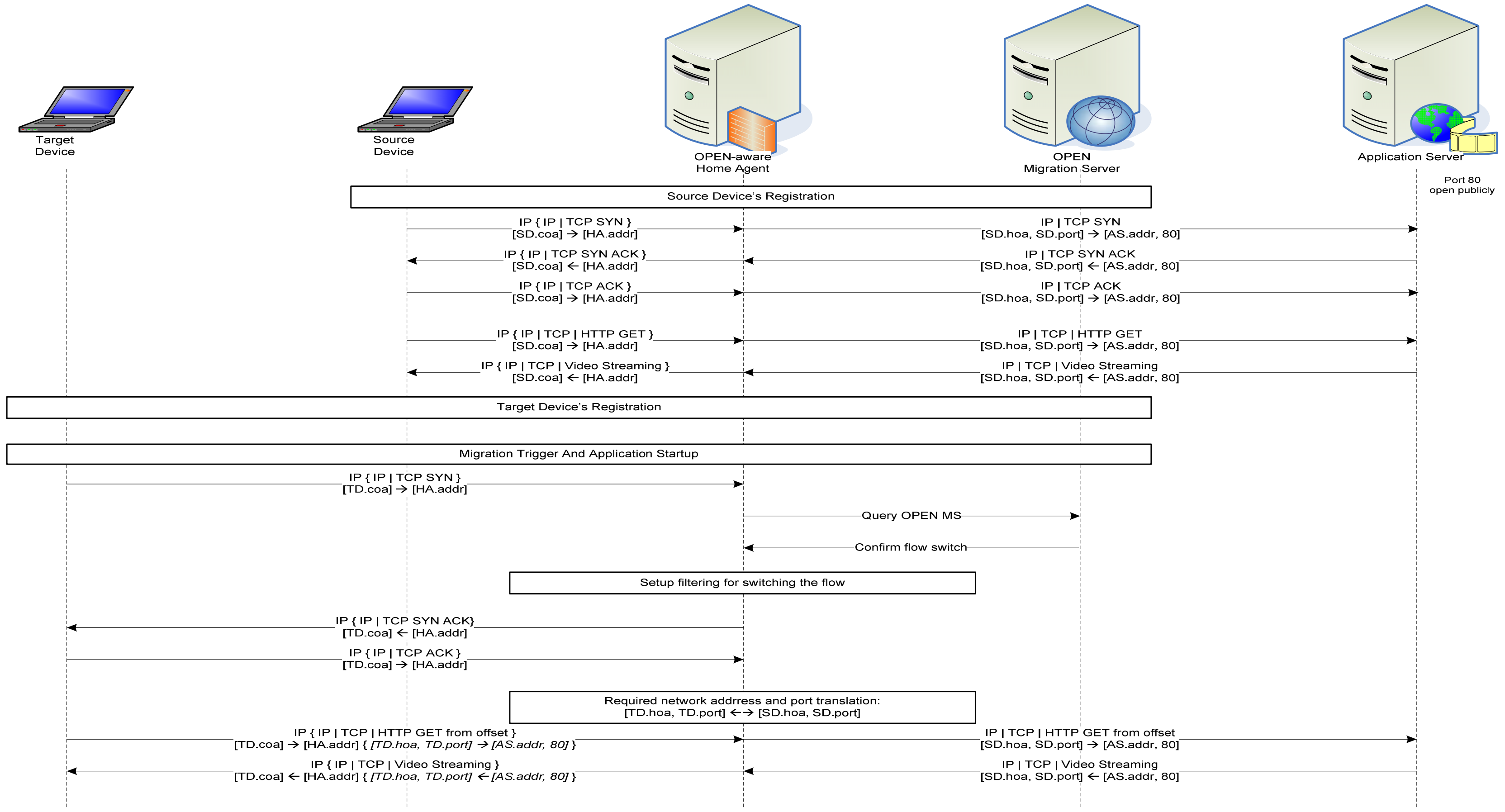

Target Device
Source Device
OPEN-aware Home Agent
OPEN Migration Server
Application Server
Port 80 open publicly
Source Device's Registration
IP { IP | TCP SYN } [SD.coa] → [HA.addr]
IP | TCP SYN [SD.hoa, SD.port] → [AS.addr, 80]
IP { IP | TCP SYN ACK } [SD.coa] ← [HA.addr]
IP | TCP SYN ACK [SD.hoa, SD.port] ← [AS.addr, 80]
IP { IP | TCP ACK } [SD.coa] → [HA.addr]
IP | TCP ACK [SD.hoa, SD.port] → [AS.addr, 80]
IP { IP | TCP | HTTP GET } [SD.coa] → [HA.addr]
IP | TCP | HTTP GET [SD.hoa, SD.port] → [AS.addr, 80]
IP { IP | TCP | Video Streaming } [SD.coa] ← [HA.addr]
IP | TCP | Video Streaming [SD.hoa, SD.port] ← [AS.addr, 80]
Target Device's Registration
Migration Trigger And Application Startup
IP { IP | TCP SYN } [TD.coa] → [HA.addr]
Query OPEN MS
Confirm flow switch
Setup filtering for switching the flow
IP { IP | TCP SYN ACK} [TD.coa] ← [HA.addr]
IP { IP | TCP ACK } [TD.coa] → [HA.addr]
Required network addrress and port translation: [TD.hoa, TD.port] ←→ [SD.hoa, SD.port]
IP { IP | TCP | HTTP GET from offset } [TD.coa] → [HA.addr] { [TD.hoa, TD.port] → [AS.addr, 80] }
IP | TCP | HTTP GET from offset [SD.hoa, SD.port] → [AS.addr, 80]
IP { IP | TCP | Video Streaming } [TD.coa] ← [HA.addr] { [TD.hoa, TD.port] ← [AS.addr, 80] }
IP | TCP | Video Streaming [SD.hoa, SD.port] ← [AS.addr, 80]

Figure 40: MIP-based session mobility.

Figure 40 illustrates the implementation of session mobility support for OPEN MSP using the Extended MIP protocol. After registration, the Source device is tunneling its request to the Application Server (AS) via the OPEN-aware HA. The HA acts like a router, which sends packets from an IP under its management to the AS, and thus the whole process is transparent to the AS. When the migration process is started, the Target Device will send a TCP SYN, through the HA's tunnel, to the AS to create the network connection. The HA will intercept this request, and before forward the packet to the AS, the HA will query the OPEN MS whether this is a fresh connection request, or a migrated one. The OPEN MS can check its list of on-going migrations to make the correct decision. If the OPEN MS instructs the HA to perform a migration, the HA will alter its filter rules, so that the flow(s) of interest will be switched from the Source to the Target device.

The Extended MIP implementation in OPEN scenario is at network layer, and therefore is transparent to protocols used by higher layers. It can support both TCP and UDP, and also application layer protocol such as HTTP or RTP. Terminal mobility is also inherently supported, i.e. this implementation will support both terminal and session mobility.

However, there are several disadvantages of this solution. First, there is a scalability issue: The Home Agent must possess a large pool of public IP addresses (or ports, in case NAT is used), so that it can assign to each of the OPEN-aware devices. Secondly, since MIP is a layer-3 solution, it needs modification to support the switching at layer 4. For example, it must be able to detect the TCP SYN packet, so that it knows when to query the OPEN MS. And it must also be able to generate a "fake" TCP SYN ACK to reply to client, if a switch of flow was done, and no actual new connection was made. Thirdly, the switch of flow for session mobility support is fundamentally different from the standard terminal mobility: the switch of flow process involves two MNs with different home addresses and care-of-addresses, which means its care-of-address and home address are both changed. As a result, the HA cannot simply tunnel the packet from the AS to the target device, like it does in case of terminal mobility. The HA must be able to modify the destination address of the packet to set the new home address. This process might compromise the integrity of IP packet sent between the AS and the device.

5.3.2. PROXY-BASED SOLUTION FOR SESSION MOBILITY

In this section, we consider the proxy-based solution to provide Session Mobility Support for the OPEN MSP. An OPEN-aware proxy is placed in the network path between the Application Server (AS) and user's devices, allowing data flow to be switched from one device to the other during migration. There are several types of proxy server:

1) *Network Address Translator:* This device operates at the network layer of the Open Systems Interconnection (OSI) model. It basically maps IP addresses from one address realm to another, providing transparent routing to end hosts.

An extension of NAT is Network Address and Port Translator (NAPT), which operates at the transport layer of the OSI model. NAPT allows a number of private hosts to be multiplexed into a single public IP address using transporting identifiers (e.g. TCP and UDP port numbers, ICMP query identifiers). Typically, NAPT functionality is implemented at the end-point gateway, which interfaces the local network and the public network.

2) *Application layer proxy server:* An application-layer proxy sends requests to application server, and then receives responses on behalf of the client. HTTP and SOCKS proxy are the most common application-layer proxy:

- HTTP proxy (or web proxy) focuses on HTTP traffic. It only works with HTTP requests and responses.

- SOCKS is an Internet protocol that facilitates the routing of network packets between client-server applications via a proxy. It provides a handshaking procedure to inform the proxy about the connection the client is trying to make, and therefore the client can use any form of TCP or UDP socket connections. There are several versions of SOCKS, but our interest is SOCKS v5 because it supports authentication, IPv6 and UDP.

As described earlier, the NAPT solution works best if the Source and Target devices are located within a local network, under the coverage of an OPEN-aware NATP router. However, in OPEN scenarios, the two devices are more likely to be located in different IP subnets, which can be different access techniques (e.g. GPRS, WLAN or Ethernet etc). Therefore, we only discuss the usage of application-layer proxy to provide Session Mobility Support for the OPEN MSP, especially the SOCKS proxy.

Figure 41 describes the registration process, in connection with the SOCKS proxy. The device will first authenticate with the SOCKS Proxy, and then registered with the OPEN MS (The abbreviations used in the diagram are explained in Table 6):

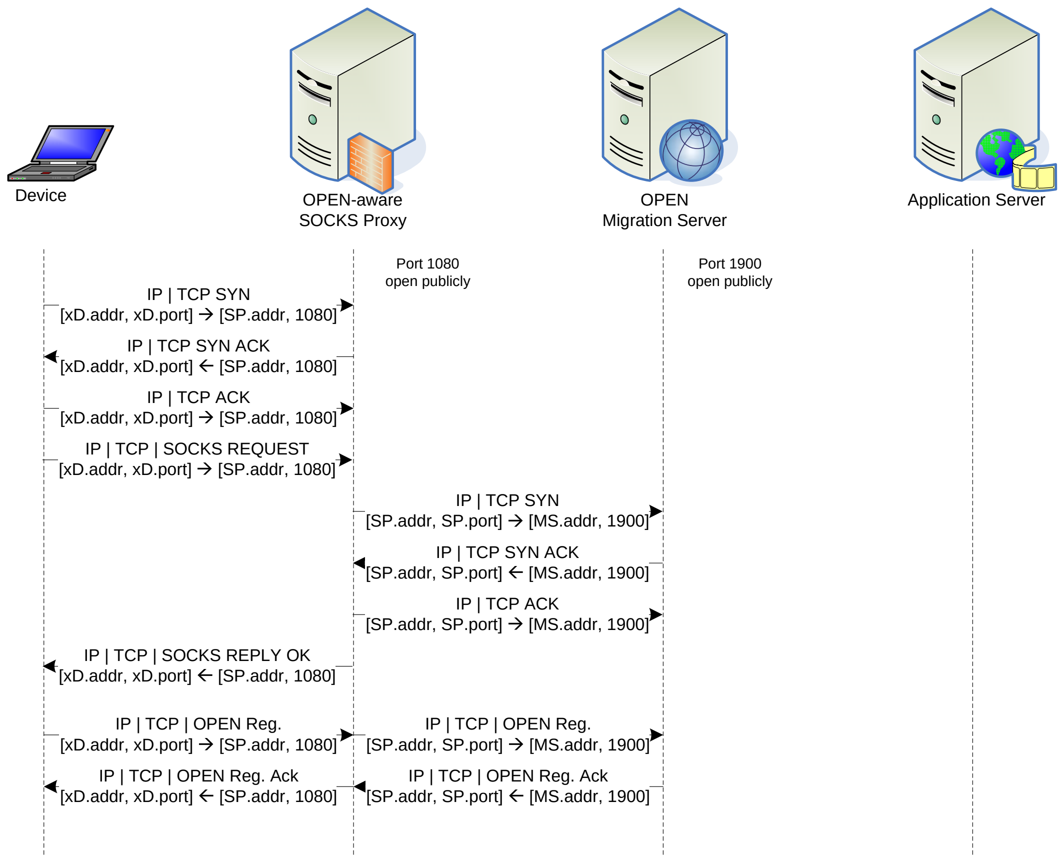
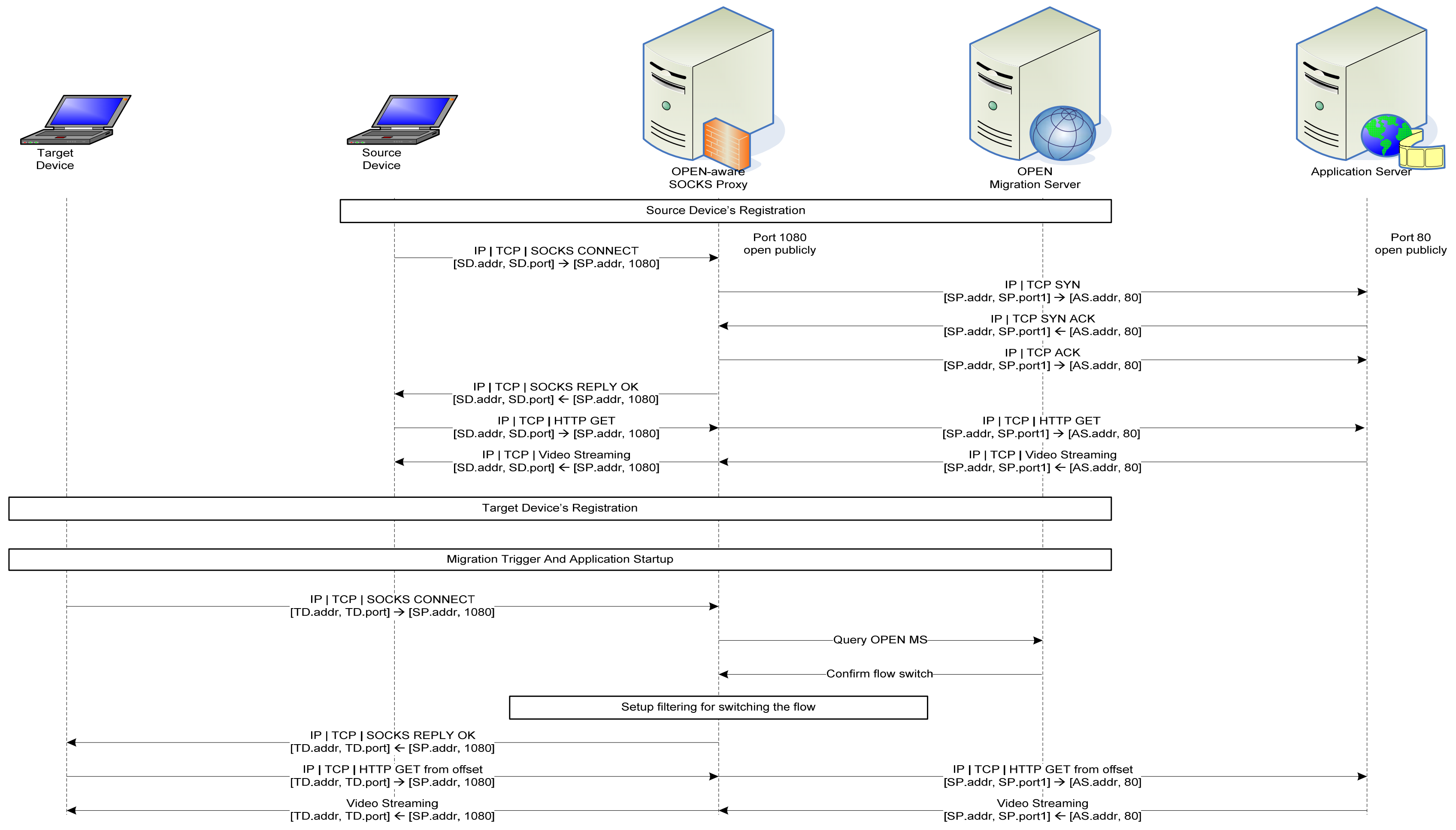

Device
OPEN-aware SOCKS Proxy
OPEN Migration Server
Application Server
Port 1080 open publicly
Port 1900 open publicly
IP | TCP SYN [xD.addr, xD.port] → [SP.addr, 1080]
IP | TCP SYN ACK [xD.addr, xD.port] ← [SP.addr, 1080]
IP | TCP ACK [xD.addr, xD.port] → [SP.addr, 1080]
IP | TCP | SOCKS REQUEST [xD.addr, xD.port] → [SP.addr, 1080]
IP | TCP SYN [SP.addr, SP.port] → [MS.addr, 1900]
IP | TCP SYN ACK [SP.addr, SP.port] ← [MS.addr, 1900]
IP | TCP ACK [SP.addr, SP.port] → [MS.addr, 1900]
IP | TCP | SOCKS REPLY OK [xD.addr, xD.port] ← [SP.addr, 1080]
IP | TCP | OPEN Reg. [xD.addr, xD.port] → [SP.addr, 1080]
IP | TCP | OPEN Reg. [SP.addr, SP.port] → [MS.addr, 1900]
IP | TCP | OPEN Reg. Ack [xD.addr, xD.port] ← [SP.addr, 1080]
IP | TCP | OPEN Reg. Ack [SP.addr, SP.port] ← [MS.addr, 1900]

Figure 41: Registration with SOCKS Proxy and OPEN Migration Server.



Target Device
Source Device
OPEN-aware SOCKS Proxy
OPEN Migration Server
Application Server
Source Device's Registration
Port 1080 open publicly
Port 80 open publicly
IP | TCP | SOCKS CONNECT [SD.addr, SD.port] → [SP.addr, 1080]
IP | TCP SYN [SP.addr, SP.port1] → [AS.addr, 80]
IP | TCP SYN ACK [SP.addr, SP.port1] ← [AS.addr, 80]
IP | TCP ACK [SP.addr, SP.port1] → [AS.addr, 80]
IP | TCP | SOCKS REPLY OK [SD.addr, SD.port] ← [SP.addr, 1080]
IP | TCP | HTTP GET [SD.addr, SD.port] → [SP.addr, 1080]
IP | TCP | HTTP GET [SP.addr, SP.port1] → [AS.addr, 80]
IP | TCP | Video Streaming [SD.addr, SD.port] ← [SP.addr, 1080]
IP | TCP | Video Streaming [SP.addr, SP.port1] ← [AS.addr, 80]
Target Device's Registration
Migration Trigger And Application Startup
IP | TCP | SOCKS CONNECT [TD.addr, TD.port] → [SP.addr, 1080]
Query OPEN MS
Confirm flow switch
Setup filtering for switching the flow
IP | TCP | SOCKS REPLY OK [TD.addr, TD.port] ← [SP.addr, 1080]
IP | TCP | HTTP GET from offset [TD.addr, TD.port] → [SP.addr, 1080]
IP | TCP | HTTP GET from offset [SP.addr, SP.port1] → [AS.addr, 80]
Video Streaming [TD.addr, TD.port] ← [SP.addr, 1080]
Video Streaming [SP.addr, SP.port1] ← [AS.addr, 80]

Figure 42: SOCKS-based session mobility.

Figure 42 illustrates the migration process with SOCKS-based proxy. Every time a device sends a SOCKS CONNECT request to the proxy, it will query the OPEN MS to check whether this is a migrated connection. The OPEN MS can look into its list of current migration processes, and then make the correct decision. If the OPEN MS confirms, the SOCKS proxy will not make new connection to the AS on behalf of the Target Device, but it switches the Source Device's connection to the AS to the Target Device. A SOCKS REPLY OK will be sent back to the Target Device to acknowledge the successful establishment of the connection. The other end of the connection, the AS, does not know that the other end has been switched.

The SOCKS-based session mobility solution has several advantages. First, it does not require any extension to the current SOCKS standard, which means that we can use any standard SOCKS library at Source and Target Devices. Secondly, the required modification of SOCKS proxy is internal, straightforward and relatively less complex than that of the MIP-based Home Agent. The SOCKS proxy does not have to inspect every packet for TCP SYN, or modify the destination IP address as required by the MIP, which results in a more efficient and simpler solution. Therefore, the SOCKS proxy shall be selected as session mobility solution in OPEN.

5.3.3. SIP

SIP has built in mechanisms that can be used to provide session mobility support. There are two ways this can be achieved, either via third party call control or by using the SIP REFER mechanism.

THIRD PARTY CALL CONTROL EXAMPLE

In the example (see Figure 43) Alice is in a media session with Bob at his mobile device. Now Bob wants to move the session to a fixed device instead. Here the third party will be the bob@mobile since it will stop being a part of the media session. Bob@mobile sends an INVITE to bob@fixed with the session parameters such as IP address for the remote party alice. Bob@fixed replies with a 200 OK message containing SDP information, which bob@mobile can then forward to alice in another INVITE message. The 200 OK message returned from Alice contains the SDP from Alice which can then be forwarded to bob@fixed with the ACK message.

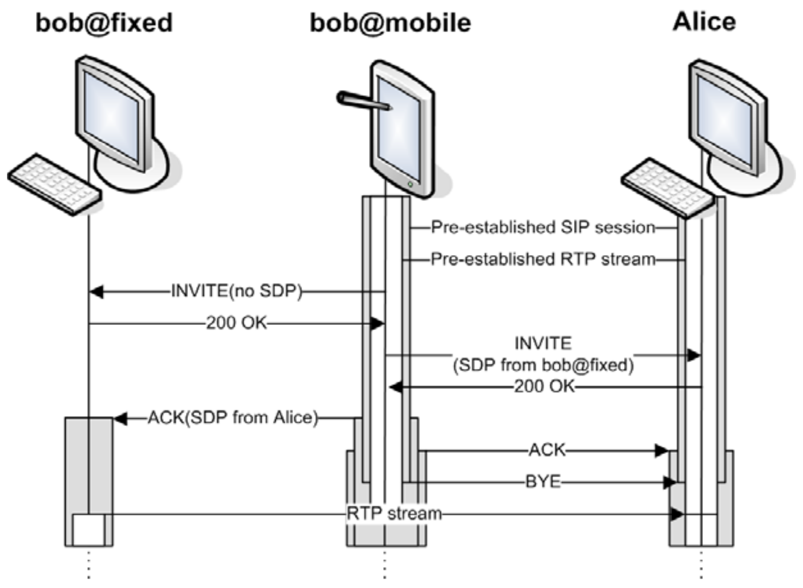


Figure 43: Third party control example using SIP.

The disadvantage of this approach is that bob@mobile has to remain involved in the session since this device established the two new SIP sessions, resulting in that bob@mobile will have to be contacted to change or terminate the SIP session.

It is also possible for bob to split the session into components with different receivers by establishing more SIP sessions.

SIP REFER METHOD

Using the REFER method of SIP it is possible to transfer a SIP session between devices. In the example shown in Figure 44 Bob@mobile sends a REFER message to Alice, which states that she should contact bob@fixed. Alice then negotiates a SIP session with bob@fixed using the normal INVITE procedure.

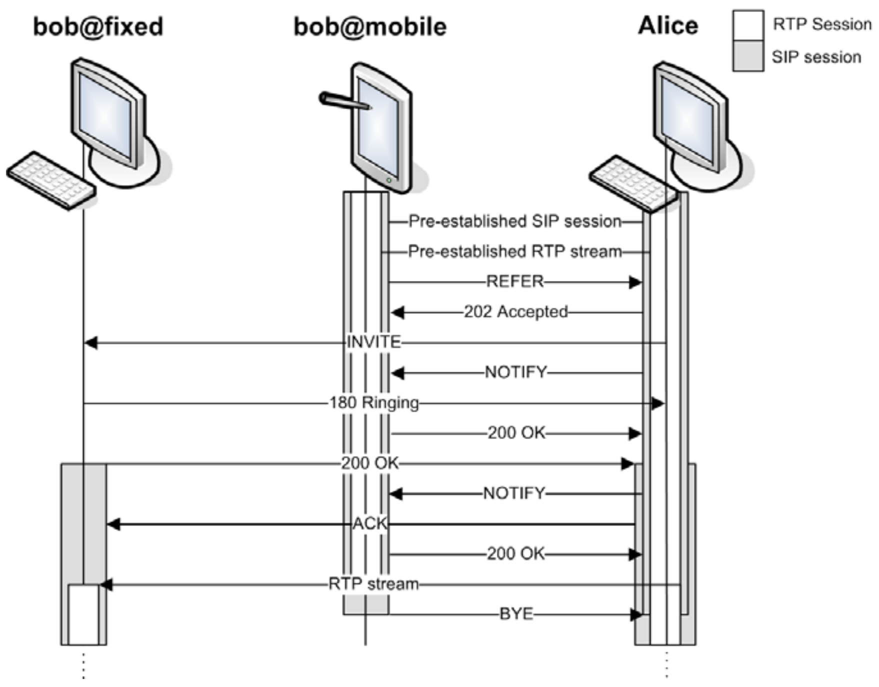


Figure 44: Transferring SIP sessions using the REFER method.

It is also possible that bob@mobile sends the REFER to bob@fixed instead.

If the session should be split into multiple parts, bob has to invite both e.g. bob@audio and bob@video.

The REFER mechanism can also be used to push a HTTP GET as shown in Figure 45.

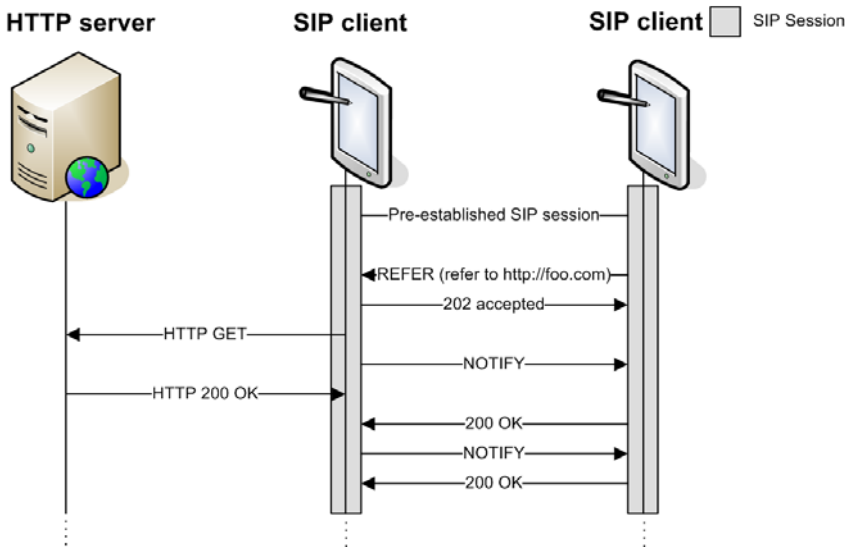


Figure 45: Using SIP REFER method to push a HTTP GET request.

SIP IN OPEN

A common SIP setup is illustrated in Figure 46. Two SIP User Agents (UA) need to establish a media session. First UA 1 will register in the local network by sending a SIP REGISTER message to the registration server. The information about the location of the SIP user is then sent to and stored at a location server for later retrieval by SIP proxies. Both UA1 and UA2 are preconfigured to utilize the proxies present in their domains. The proxy will typically authenticate the UA and may pull up a profile of the user and apply outbound routing services.

When UA1 initiates a SIP dialog with UA2 it will use the SIP proxy present in the local network, which will lookup a proxy server in the other domain using DNS SRV queries. This (inbound) proxy will then be able to route the SIP messages to the receiving UA using the location information registered for the user. Unless the “Record-Route” field is used, further SIP messages are sent directly between the two UAs.

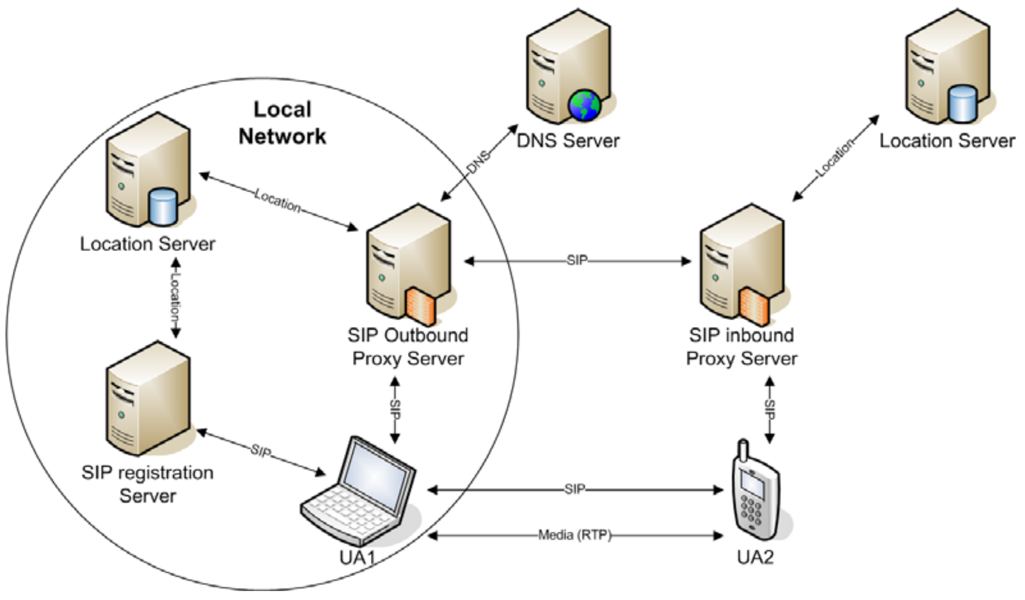


Figure 46: A common SIP setup

Since it is chosen that OPEN should not change anything for the Application Servers, the OPEN platform can communicate with either a SIP enabled AS or a non-SIP AS. These two possibilities are further analyzed in Appendix B.1 while an analysis of how SIP signaling could be used for the OPEN platform is detailed in Appendix B.2.

5.3.4. SESSION MOBILITY ANALYSIS CONCLUSION

Using SIP for session mobility when the AS does not support SIP is similar to the other proxy based solutions in that a SIP UA will terminate the SIP signaling path before the traffic is exchanged with the AS. However if the AS does support SIP, SIP becomes an advantage since the data traffic can be exchanged directly between the SIP end-points (OPEN client and AS). SIP also has the advantage that it operates at the application layer and is therefore independent of the transport protocol.

It is chosen to use a SOCKSv5 proxy based solution for session mobility support in OPEN since the SOCKSv5 protocol implementation is much simpler than SIP (one proxy vs. lots of SIP entities) and since SOCKSv5 is already directly supported in many applications. This means that unless the SOCKSv5 protocol is extended, session mobility can only be provided completely for TCP and UDP based applications.

5.4. ARCHITECHTURE OF MOBILITY SUPPORT MODULE

Based on the analysis a concrete architecture for the Mobility Support module in OPEN is proposed here. The architecture is illustrated in Figure 47 and consists of: (a) OPEN Migration Server, where device discovery and migration logic are performed, and (b) OPEN-aware Mobility Anchor Point, where terminal and session mobility are handled. The Mobility Anchor Point is the OPEN-aware SOCKS proxy, which acts as a "fixed" network point for the mobile device. Prior to data transmission, the device shall register with the OPEN Migration Server and the Mobility Anchor Point (MAP).

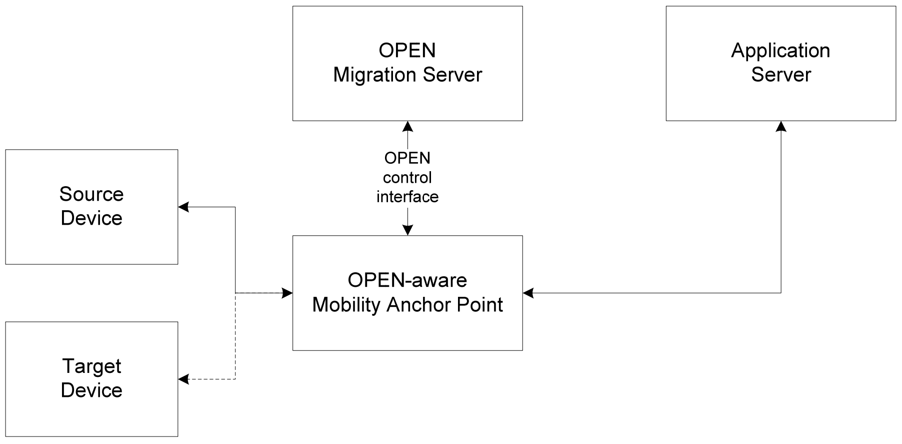


Figure 47: The elements for Mobility Support in OPEN.

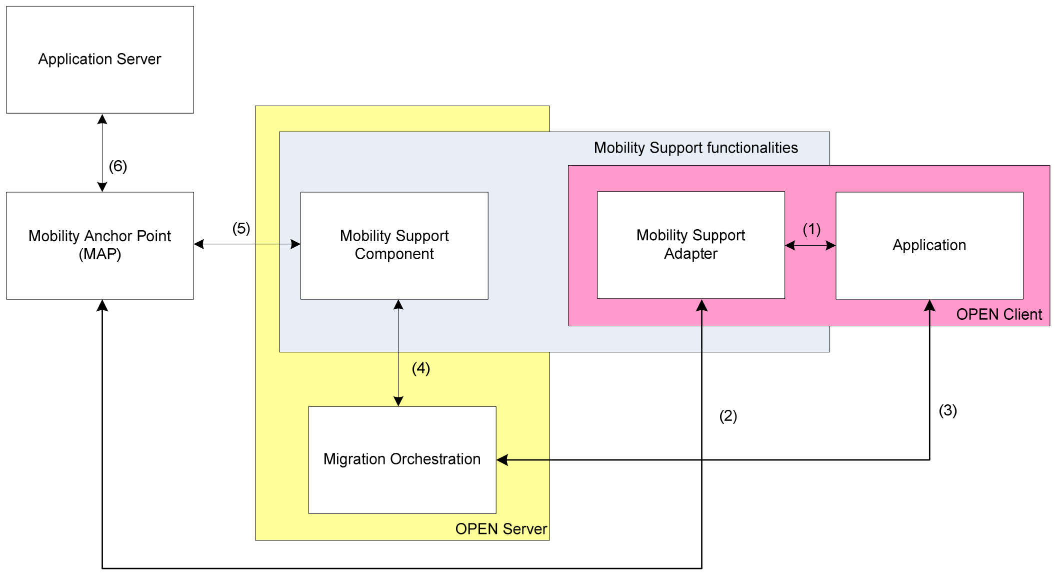


Figure 48: Mobility Support interacts with other OPEN's components.

Figure 48 illustrates the interfaces between the Mobility Support and the other components and modules in the OPEN MSP. Note that the Mobility Support module consists of Mobility Support Adapter, which resides at the OPEN client, and Mobility Support Component, which is located in the OPEN Server. Each interface is marked with a number, and they are described in detailed as follows:

(1) Interface to Application on client-side: Mobility Support Adapter provides a SOCKSified socket API to application, and the application should use this API instead of operation system's standard socket API, so that it can enjoy the mobility support functionality.

(2) Interface towards MAP: This interface is used by Mobility Support Adapter to send SOCKS request to the MAP on behalf of the application.

(3) Interface between the Migration Orchestration and the application: This interface allows the Migration Orchestration to get and set the application state, which includes the information on the required network connections.

(4) Interface between the Migration Orchestration and Mobility Support Component: This interface is for the Migration Orchestration to inform the Mobility Support Component which connection needs to be migrated. The Mobility Support Component maintains a list of connections awaiting migration, and it will refer to this list to answer the MAP's query.

(5) Interface between the MAP and the Mobility Support Component: This interface allows the MAP to confirm with the Mobility Support module whether it should perform migration on a specific connection request. If the Mobility Support Component says yes, the MAP will start switching the source device's data flow with the target device's one, so that traffic from the AS will be redirected to the target device.

(6) Interface from MAP to AS: This is normal TCP or UDP connection from MAP to the Application Server, or vice versa, which the MAP creates on behalf of the application.

5.5. SECURITY

In order to make both session and terminal mobility transparent to the applications, additional software and network entities have been introduced into the OPEN Framework. It is important that security is not compromised because of this, and in this section we address the security issues related to Mobility Support.

The proposed solution for handling terminal mobility is Mobile IP which have built in security mechanisms as specified in the RFCs about mobile IP: [RFC3344], [RFC4721] and [RFC3775].

The proposed solution for session mobility uses a modified SOCKSv5 server and a SOCKSv5 aware middleware layer at the clients to make it possible to move a session from one network path to another. One possible threat to this solution could be if it is possible to maliciously redirect traffic from one OPEN user to another. The SOCKSv5 protocol specifies a general framework for the use of arbitrary authentication mechanisms in the initial SOCKS connection setup to enable a trusted relationship between the SOCKSv5 server and client. [RFC1961] and [RFC1929] specify specific authentication protocols for use with SOCKSv5. Using authentication for the SOCKSv5 combined with proper authentication of users in OPEN will make sure only the user owning a specific connection is able to migrate this.

The MAP needs to query the OPEN server about what to do for new connections established for client devices, which means a trusted relationship between those two entities is needed. This can be solved using e.g. SSL certificates to enable authentication and integrity of the connection.

If privacy and integrity of the network traffic going from the clients through the MAP needs to be ensured, although this is beyond the scope of the mobility support function, but it can be achieved by having higher layer encryption enabled.

5.6. SUMMARY

This chapter devoted to analyzing and providing solutions for Mobility Support Function in OPEN MSP. The main task of Mobility Support Function is to allow the user's current service to be continued across multiple networks and multiple devices.

The mobility in OPEN MSP can be categorized into two types: terminal mobility and session mobility. Several alternatives have been considered, and two solutions have been selected for implementation in OPEN MSP, namely the Mobile IP and SOCKS-based proxy. The Mobile IP - which can be implemented transparent to OPEN MS, client and AS - shall provide the required support for terminal mobility. For session mobility, a new entity, the OPEN-aware Mobile Anchor Point, is introduced to act as a "fixed" network attachment point for the mobile device. The MAP is a modified SOCKS-base proxy, which is interfaced with OPEN MS, to facilitate the migration of data flow(s) from the source device to the target device, when migration occurred.

6. CLOCK/FLOW SYNCHRONIZATION

Certain parts of the OPEN platform either needs, or would perform better with, a mechanism to ensure synchronization.

The orchestrator module could make use of a global time to control the migration process, while the performance monitoring module can use a global clock to make delay measurements in the network. The CMF module could use a global time to make quality assessments about context information. Another synchronization aspect arises when multiple streams of data have a time dependence on each other. Some mechanism is needed to make sure such streams are kept in synchronization when these are e.g. split an transmitted to multiple devices.

The clock/flow synchronization module is defined to take care of these aspects in the OPEN platform. As shown in Figure 49 the Clock/Flow Synchronization module interfaces with the Orchestrator, the CMF and the Performance monitoring modules as well as applications which may need access to a synchronized clock.

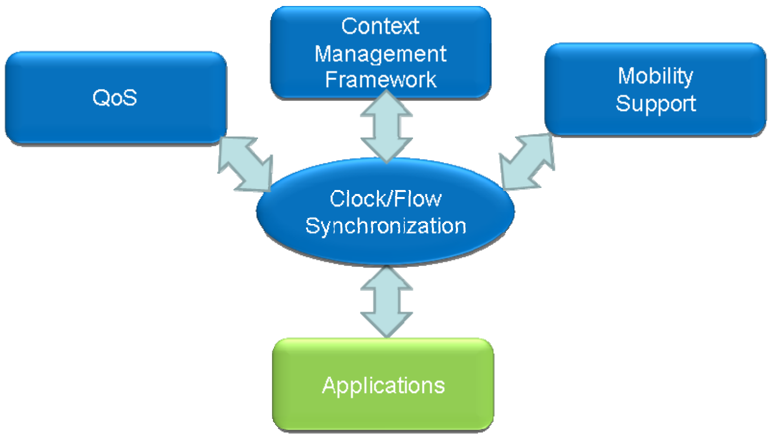


Figure 49: The clock/flow synchronization block and its interaction with other modules

In Section 6.1 a scenario where flow synchronization could be useful is detailed followed by a problem definition. In Section 6.2 a method to achieve clock synchronization is discussed and in Section 6.3 methods for achieving flow synchronization is discussed. Section 6.4 sums up the work done on the Clock/Flow Synchronization module.

6.1. INTRODUCTION

A user (Eric) is at home, watching a movie on his cell phone. The movie is streamed from a remote internet-server which is not open-aware. Eric is connected to his home network via a Wi-Fi connection to the local router, which has WiFi capabilities (WR), and the entire movie stream is sent through the OPEN-aware Mobility Anchor Point (MAP), which has been introduced for session mobility. The MAP and client devices are controlled by an OPEN migration-server (MS). The MS and MAP may be located in the home network, or at a remote site. There is a large LCD-TV in the living room, which is connected to a media-center-computer. The media-center is connected to the home network via Ethernet to the local WR. This setup can be seen in Figure 50.

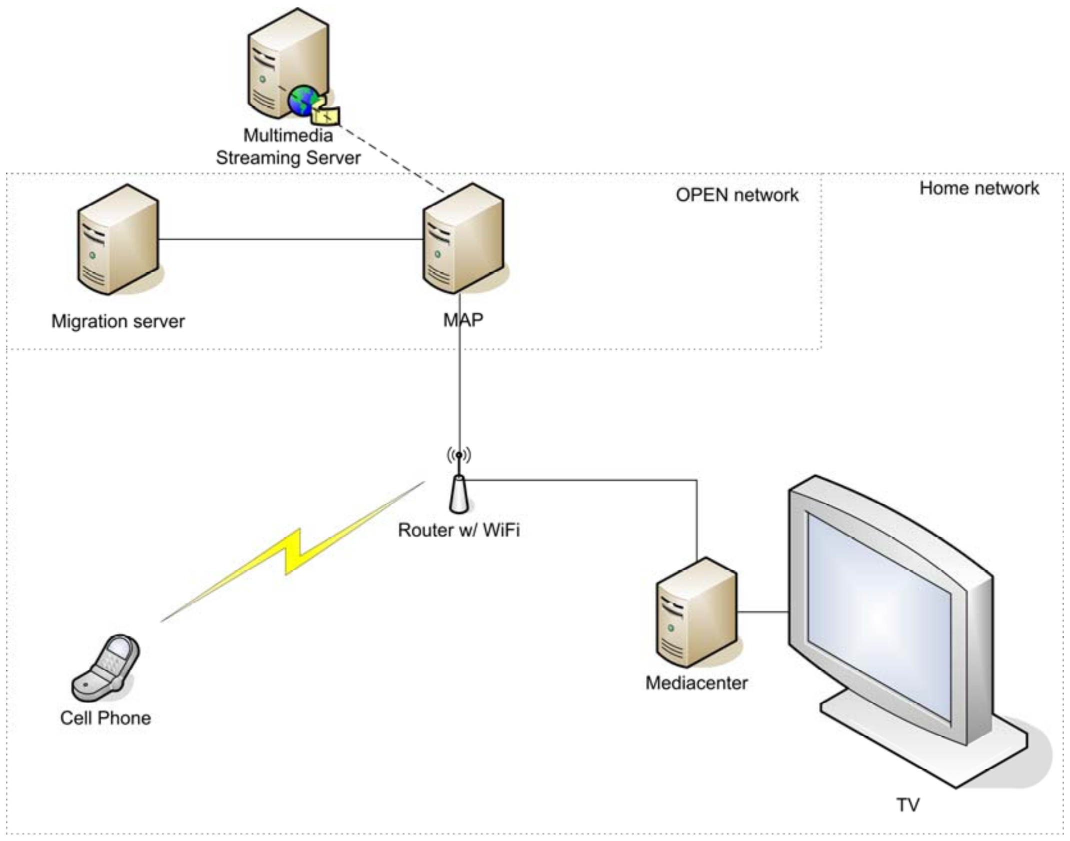


Figure 50: The OPEN-aware home network and the remote multimedia streaming server.

The user wants to display video of the movie on the LCD-TV, as he thinks the cell phone screen is too small for watching a full movie. He still wants the movie audio to be played on the phone, so he can use his headset. For convenience he wants to continue controlling the movie playback using the phone. He brings up the OPEN-client on his phone and chooses the TV from the list of migration devices, and sets it to only migrate the video-stream. He is now able to continue watching the movie seamlessly.

In this chapter it is assumed that no firewalls or NATs prevent communication in the home-network.

6.1.1. SYNCHRONIZATION PROBLEM DESCRIPTION

When video and audio is played back from two different devices, we need the two devices to synchronize their playback in order to provide a pleasant experience to the user.

Experiments from IBM European Networking Centre [Steinmetz96] show that humans perceive the notion of synchronicity differently, and that the playback does not need to be in perfect sync (0 ms delay) to appear synchronized to human eyes and ears. From the experiments it can be concluded that playback is perceived as synchronized with delays from -80 ms (audio behind video) to 80 ms (audio ahead of video). The OPEN software must therefore ensure that the playback from the devices involved in presenting the media to the user is within these delay limits. It also needs to start, stop and pause the playback simultaneously.

A media stream is a continuous stream with media information, such as audio and video. This stream is normally split into small information units, the size of which is controlled by the application and the type of media. As an example, an information unit of an audio stream could contain 512 audio-samples. This way, the unit is better suited for transportation over a computer network. In this discussion the term Media Units (MUs) will be used for these units.

Assume that the MAP starts the transmission of the media-streams MS1 and MS2 simultaneously to the cell phone and the media-center. Then the two devices will receive the MUs and start buffering and playback independent of each other. During this process (from start of transmission to playback), the streams incur several delays, affecting the playout times. The playout time is the moment the data is presented to a user. An illustration of this situation can be seen in Figure 51.

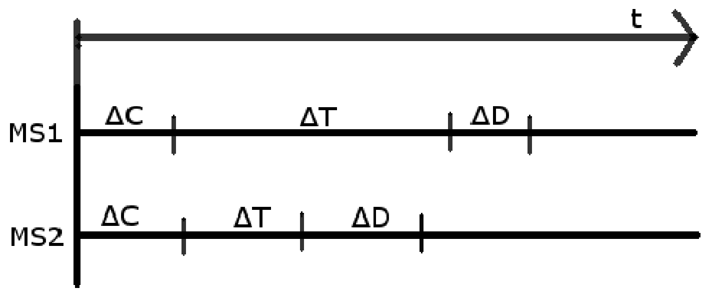


Figure 51: Delays considered in a two stream setup.

In Figure 51, it is possible to see the delays incurred on stream MS1 and stream MS2. ΔC is the collection delay (sampling, encoding and splitting into MUs at the stream source), typically a processing delay. ΔT is the transmission delay. This is a major and unpredictable factor, due to the heterogeneity of the transmission media. The Ethernet-connection of the media-center is low latency and stable, compared to the unpredictable Wi-Fi channel of the cell phone. This results in different transmission delays. ΔD is the Delivery delay, which is the time it takes before the information is represented to the user. This is typically induced by the presenting application, and will vary according to hardware, load and the type of media to be presented. As an example, this delay will normally be smaller for an audio MU than a video MU, due to less processing power needed. If the playout times of the different streams differ more than 80 ms, they will appear unsynchronized to the audience.

It is therefore necessary to look into ways of eliminating or limiting the effects of network delay on playback. Depending on how strict the synchronization requirements are how the synchronization mechanism is implemented, it is possible to limit the effects of the delays depicted in Figure 51.

Synchronization of the playback can be more easily achieved if devices are following a global clock. The next section therefore looks at a way to achieve clock synchronization of the devices in the OPEN-network.

6.2. CLOCK SYNCHRONIZATION

The goal of clock synchronization is to achieve a global system time, which the clock of all devices in the system are set to. There are many ways to achieve this goal, depending on the accuracy needed by the system, and the resources available in the system. We will only discuss one of the most available solutions: NTP.

6.2.1. NETWORK TIME PROTOCOL

The Network Time Protocol (NTP) is the most commonly deployed time service and protocol, providing accuracy in the ms range [Verissimo]. Its structure is inspired by a master-slave round trip synchronization protocol [Christian89], and can be seen in Figure 52. The NTP-servers are organized in a descending hierarchy of stratums, where stratum 1 is the most accurate to the global UTC time. The servers below synchronize to the servers with one higher stratum. The servers of a stratum can also compare data to improve their accuracy. The servers also run correction algorithms to account for clock drifts. NTP implementations are available on multiple software platforms and free of charge [Ntp].

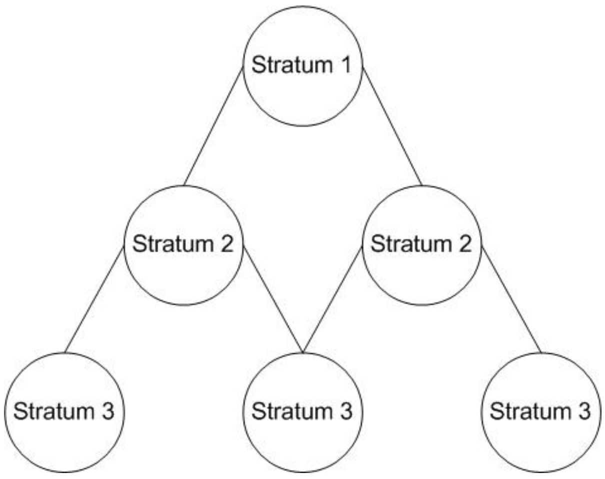


Figure 52: Hierarchical structure of clock synchronization points organized in stratums.

NTP appears to be a good solution to make the OPEN-aware devices follow a global clock, which is important for synchronized playback. A full implementation of NTP is however resource intensive, due to the correction algorithms, and might therefore not be suited for battery-powered devices.

6.3. FLOW SYNCHRONIZATION

In the literature ([Escobar94] and [Boronat08]), the media synchronization problem is commonly divided into three parts:

1. Intra stream synchronization refers to the temporal relationship between MUs of a time-dependant media stream. It is assumed here that this synchronization is handled by the applications buffering algorithms.
2. Inter-stream synchronization refers to the synchronization between two or more media streams that belong together. Lip synchronization is a common example of this kind of synchronization, where the media application synchronizes the audio stream to the video stream. It will be assumed here that the media application handles this type of synchronization when both streams are on the same device. If the streams are split out to multiple devices part 3 is used as the classification.
3. Group synchronization refers to synchronization of the playout of one or more media streams by *several receivers* at the same time. In the OPEN project, this kind of synchronization is called flow synchronization, which is referring to the media stream as a flow of MUs.

The methods described in the following are inspired by [Escobar94] and [Boronat08]. None of these can be used directly in an OPEN-environment, so the proposal described here is a mixture of these, applying them to the described scenario.

The flow synchronization assumes synchronized devices, which can be achieved by the means described in the last section. The idea is to achieve flow synchronization by adding control of the playout time (t_{r1} and t_{r2}) at the clients based on delay measurements, by implementing an equalization delay, ΔE, and estimating each receiver's delivery delay, ΔD. Using the information about the delays, it is possible to calculate when the protocol should release the MUs in order for the presentation time of the two streams to be simultaneous ($t_{p1} \approx t_{p2}$). This can be seen in Figure 53.

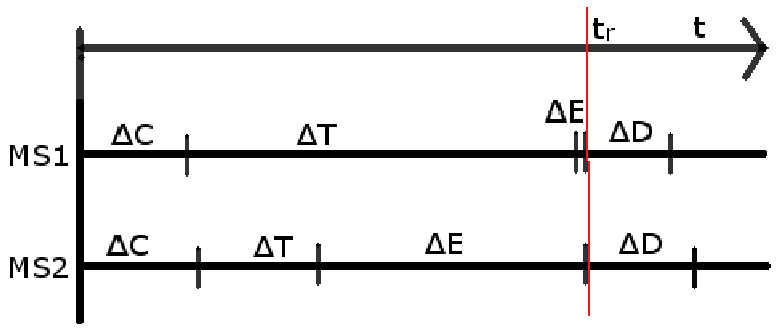


Figure 53: Release time in a two stream setup.

An overview of the different entities in the solution can be seen in Figure 54. The initiator is the entity through which the user controls the distributed application, and may reside on any of the clients. There are one or more Source processors (Si) that sends and timestamps data, and one or more Destination processors (Di, Dj) that receive the data and buffer it. The Destination processors that should be synchronized belong to the same synchronization group. The processors in the synchronization group periodically share information with each other and agree on a common maximum delay, ΔK.

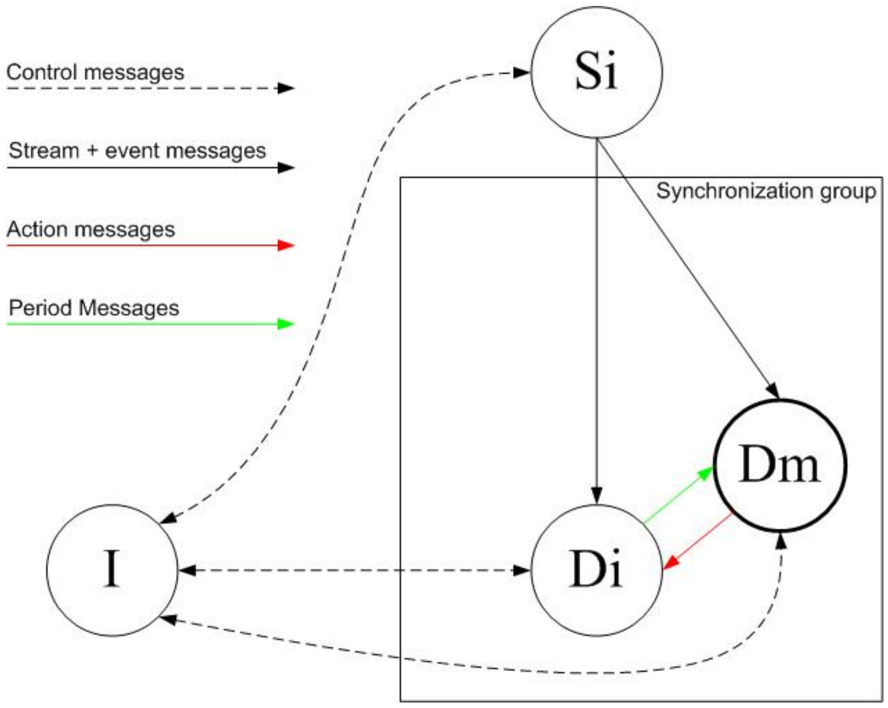


Figure 54: Overview of entity groups in a potential synchronization solution for OPEN.

6.3.1. USE CASE

The user is watching the movie on the cell phone. The MAP-server is forwarding a media-stream from the remote multimedia server and functioning as the Source processor “Si”. It also timestamps some of the MUs upon transmission. The initiator and the destination processor Dm is located on the phone.

The user will initiate migration of the video-stream through the application. The migration orchestrator controls the initiator and contacts the media-center and asks it to prepare a destination processor for the media-stream. The media-center replies when it has set up a destination processor Di. The initiator adds Di to the synchronization group, and sends the information to all parties. Si receives the message and starts duplicating the stream.

At the moment Si starts duplicating the stream, it sends a message notifying the phone from which time the video will be presented at another location. An application policy chooses when to cut the video on the phone relative to when it is displayed at the TV. This can be achieved as a combination of Trigger Management enforcing the policy and the Orchestrator executing the migration. It may for instance be a more comfortable transition for the user, if the video is displayed for 5 seconds on both devices.

The destination processors chooses the parts of the stream it wants to process ("video", "audio" or "both"), and then negotiate the release time, t_r , by communicating their delay-information using periodic messages. The periodic messages will be exchanged throughout the playout of the movie, and are used for estimating the playout points of the destination processors. If the differences between the playout points exceed a threshold-value, action must be taken to keep the presentation in sync. These actions are communicated through action messages and are described more thoroughly in the section "Handling asynchrony".

6.3.2. CALCULATION OF RELEASE TIME

For the purposes of this section, it is assumed that the devices are synchronized to a global clock. It is assumed that information about the transmission delay can be accessed through the performance monitoring functions of the OPEN-platform. An overview of the various times introduced is shown in Figure 55.

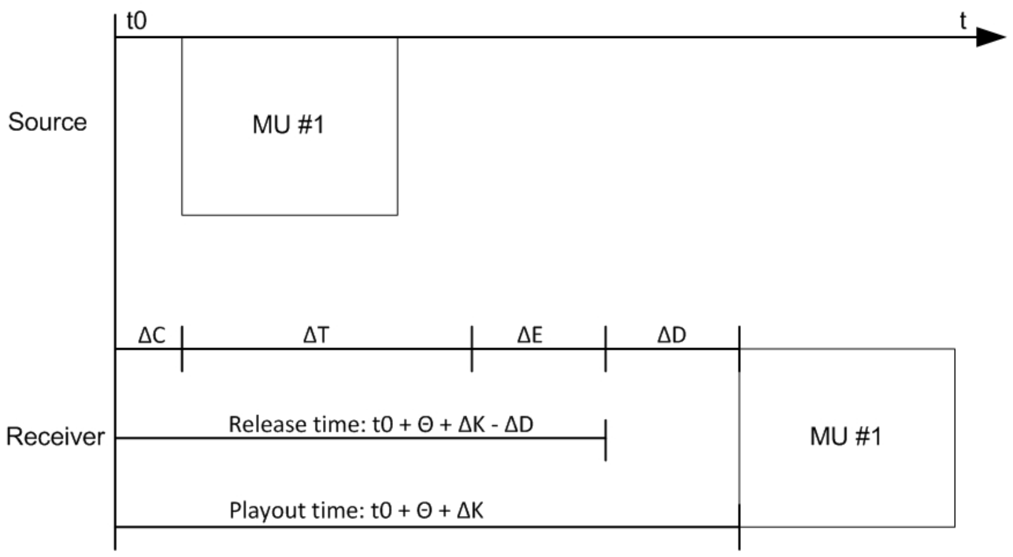


Figure 55: Overview of times in release time calculations.

ΔK is negotiated amongst the destination processors in the group, and defined as slightly larger than the maximum delay d^imax. Each destination processor keeps track of their own d^i which is defined as: $d^i = \Delta C^i + \Delta T^i + \Delta D^i$, the sum of the collection, transmission and delivery delay experienced by destination processor i.

Slightly before the transmission of the source stream begins, Si creates a reference time t_0. The timestamps for the stream are then calculated as: $\Theta = t - \Delta C^i - t_0$, where t is the current local time and ΔC^i is the estimated collection delay of the processor source processor Si. The destination processors receive the timestamps and calculate a release time as follows: $t_r = t_0 + \Theta + \Delta K - \Delta D^i$.

6.3.3. HANDLING ASYNCHRONY

The delay information in the periodic messages is used to estimate the playout points of the destination processors. In case the playout points of the destination processors exceed a threshold value, some mechanism must be implemented to re-synchronize the processors.

One possibility is to announce one of the streams the master-stream, and have the slave-streams synchronize to it. The synchronization can be achieved by skipping (if it is lagging behind), or pausing (if it is before) MUs. A more subtle approach is to adjust the playout rate of the streams, so that they will catch up with the master stream. This is less noticeable to the user than skipping or pausing MUs.

The master-stream can be selected dynamically based on network performance measuring, or it could be specified by the application. An example could be a video-telephony application, where changes in audio playback are more noticeable than changes in video. It may therefore be wanted to have the video stream synchronize to the audio master-stream.

When it has been decided what streams that needs to do adjustments, the destination processor with the master-stream will send action messages to other processors, instructing them how to adjust their playback.

Applications normally have different requirements when it comes to flow specific parameter. There has to exists some policies (see Chapter 4) for how to act when these requirements are not fulfilled. An example is upper bounds on total delay. If the total delay gets large, it will result in an unresponsive program. This may not be a problem when the user is watching a movie, which has little interaction, but in case the user was playing a game, it may very well render the game unplayable. Situations like these needs to be handled by Trigger Management, e.g. the solution could be that the game application is migrated fully to the device with the best network conditions.

6.3.4. DISCUSSION

A synchronization technique where the synchronization is done on the client-side, to limit the processing power needed at the MAP-server, has been proposed. Some overhead is imposed on the network however, by the fact that the streams are duplicated to all recipients. In some cases multicast may be used to reduce this, for instance if all client devices are in the same wireless network. In this case, the bandwidth demand does not increase as the multicast packets are broadcasted to all clients by the WiFi network.

The accuracy of the synchronization is naturally depending on the underlying clock synchronization technique and on the rate of period messages and corrections. Experiments can be done to map the effects of different configurations. The settings can then be controlled by the application or be dynamically changed in response to the environment.

Two main assumptions was made in order to make this solution possible, first that it is possible to achieve a global time on OPEN enabled devices and secondly that performance monitoring can measure one way delays on network traffic. Since NTP is a way of achieving synchronized clocks which then enables the performance monitoring to measure one way delay times instead of RTT, the proposed solution should be feasible.

The synchronization mechanism can also be useful when a full migration of a media streaming application is performed. Instead of stopping the playback, transferring the state, retrieving state and starting playback, the synchronization can be utilized to create a seamless transfer between the two devices. If a full migration is initialized, the procedure described in the use case in Section 6.3.1 is followed. From the moment the two streams are in synchronization, the original device can be stopped. It is assumed that all other state-information of the program is possible to transfer while both programs are running.

6.4. SUMMARY

In this chapter a solution for synchronizing time dependant multimedia streams on different devices was described. The method proposed relies on the availability of synchronized time which can be achieved in OPEN by using NTP, and which would be useful for a number of other modules as well to enhance their functionality e.g. making it possible for performance monitoring to measure one way delays.

Since the synchronization is done on the client-side, the solution only requires small resources at the MAP-server (stream duplication and event messages). The accuracy of the synchronization is naturally depending on the underlying clock synchronization technique, but since the requirements for synchronized multimedia streams are within 80 milliseconds and the accuracy of NTP is in the millisecond range, NTP should be sufficient.

7. CONCLUSIONS AND OUTLOOK

This deliverable in conjunction with Deliverable 3.3 contains the major results obtained within work package 3 within the planned time period in close collaboration with the rest of the project. The work has since the writing of Deliverable 3.1 been focused on refining and detailing the respective functionality and interfaces for modules and components located within the support platform belonging to Work Package 3.

In Chapter 1 we provide an overall view of the platform and how it connects to the rest of the project, and gave an introduction to the core functionalities that has been of focus in Work Package 3, such as Migration Orchestration, Trigger Management, Context Management, and Mobility support. We also here introduce the key network entities we have identified; OPEN Mobility Anchor Point for terminal and session mobility, OPEN Server for migration support and the client devices. In this aspect we focus on scenarios where the Migration Anchor Point and OPEN Server are static and known a priori, whereas dynamic scenarios e.g. ad hoc networks have not been investigated as focus has been refining and integration of existing functions rather than introducing new ones required by ad hoc scenarios. Finally, we describe the overall interaction between these entities which also shows the high level interaction between the modules described in the subsequent chapters.

In Chapter 2 we detail the Context Management with in particular focus on the changes that has been made to the applied framework that comes from the IST project MAGNET Beyond. The changes made focuses on shifting the existing framework into an OSGi environment, which allows a much higher degree of flexibility in terms of configurations. The existing framework from MAGNET Beyond did not meet the requirements of handling the dynamicity of runtime reconfiguration as needed, which allows remote repositories of context producing components to be easily installed, started, stopped and uninstalled from the migration platform. With location as being key context information for the prototypes we also provide a methodology to meet reliability requirements on location information via adjusting accuracy on the position estimate. This is useful for system triggered migrations where certain reliability on correct triggering of the migration process should be ensured.

Chapter 3 contains the details of the Migration Orchestration. The main characterisation of this module is that it is responsible for correct timely execution of the migration process by sending various control signals such as start, stop, wait, to the relevant module and components on relevant devices involved in the migration process. The chapter details interactions of migration processes in various situations; full or partial migration and web migration, and how application states are handled in the process.

The triggering of a migration process is described in Chapter 4, were focus is on the automatic triggering. The idea is to suggest triggering of applications to the user when there are good opportunities, in the chapter an example with a video stream application is used with the quality of the links involved are used as base information to decide whether to run the video stream in a high or low quality. A methodology to achieve such decisions is presented along the details of the Trigger Management module responsible for detecting and deciding upon migration triggers.

As the migration involves changes in the execution and communication environment of the application, support for terminal and session mobility is required. Chapter 5 investigates different terminal mobility support solutions; Mobile IP, SCTP and SIP, whereas it is concluded that Mobile IP can be used without modifications and integrated transparently to the rest of the OPEN Migration Platform as opposed to the other solutions which requires some modifications. For terminal mobility, Mobile IP, SIP and SOCKS were investigated, where a modified SOCKSv5 proxy solution was decided upon. A prototype of the proposed solution has been implemented and documented in Deliverable 3.3. SIP was found interesting as solutions only when the application server and client already are using SIP which would be the case if the application server is deployed in an IMS backend.

Chapter 6 focuses on more specific scenarios where two or more data streams requires to be synchronized during and after the migration process, e.g. real time audio and video streams. The solution proposed is relying on synchronized entities, which can be achieved with NTP using the Migration Server as a synchronization point.

To finally reflect on the initial definition of Migration in OPEN, that

$$\text{Migration} = \text{Changes} + \text{Adaptation} + \text{Continuity}$$

we have in Work Package 3 developed a platform which support changes in terms of network mobility, adaptation in terms of context management and continuation in terms of migration orchestration. Several more functions that support the migration have also been developed, which together with the project provides a migration platform addressing set requirements for the process, and shown working via various prototypes. This said there are still issues to be addressed in order for a functional platform should be considered ready. Some issues are

- Differentiation between multiple application sessions and how these are registered in order to support multiple applications from multiple users. Herein are also problems related to ensuring privacy of potential sensitive data among different users.
- Ad hoc scenarios where the assumptions on fixed Mobile Anchor Point and OPEN Server are removed and evaluation of added cost in terms of time and network traffic to select and change these entities.
- Further integration of modules into the platform; clock/flow synchronization, Quality of Service and Performance Monitoring have not yet been included in the platform.

Finally, since this deliverable concludes the work in Work Package 3, there will be no further development in the OPEN platform, whereas focus is changed toward final integration of modules in Work Package 4 to be used for final testing, validation and evaluation in Work Package 6. Results from these will appear in Deliverable 4.4 and 6.7.

REFERENCES

[OPEN WEB] http://www.ict-open.eu/

[OPEN D1.3] Susan Marie Thomas, Kay-Uwe Schmidt, et.al., Final Requirements for OPEN Service Platform, Deliverable 1.3, June 2009

[OPEN D1.4] Susan Thomas, et.al., Final OPEN Service Platform architectural framework, Deliverable 1.4, August 2009

[OPEN D2.2] Fabio Paternò, Carmen Santoro, Antonio Scorcia, Giuseppe Ghiani, Armin Jahanpanah, Stefano Marzorati, Document about Architecture for migratory user interfaces, February 2009

[OPEN D3.1] A. Nickelsen, R. L. Olsen, M. Martin, E. Kovacs, G. Ghiani, H. Klus, S. Marzorati, A. Grasselli, M. Piunti, Detailed network architecture, Deliverable 3.1, February 2009

[OPEN D3.2] Anders Nickelsen, Rasmus Olsen, Miquel Martin, Holger Klus, Giuseppe Ghiani, System support for application migration, Prototype deliverable 3.2, December 2008

[OPEN D3.3] Rasmus Olsen, Giuseppe Ghiani, Miquel Martin, Anders Nickelsen , Kim Højgaard-Hansen, Huan Cong Nguyen, Simone Mazzei, Björn Schindler, System support for application migration (revised), Prototype deliverable 3.3, December 2009

[OPEN D4.1] H. Klus, D. Niebuhr, B. Schindler, M. Deynet, C.Deiters, Solutions for Application Logic Reconfiguration, Deliverable 4.1, January 2009

[OPEN D4.2] Miquel Martin, et.al., Migration Service Platform Design, Deliverable 4.2, 2009

[OPEN D5.3] Giancarlo Cherchi, Francesca Mureddu, et.al., Final application requirements and design, Deliverable 5.3, January 2010

[MAGWEB] MAGNET Beyond, online http://www.magnet.aau.dk

[MAG D2.3.1] Martin Jacobsson, Venkatesha Prasad, Martin Bauer, Jan Stoter, Rasmus L. Olsen, Jorge Lanza, Luis Sanchez, Juan Rico, Yanying Gu, Majid Ghader, Marc Girod Gene, Djamal Zeghlache, Interim PN Framework Solution and Performance, D2.3.1, MAGNET Beyond, IST-027396

[MAG D2.3.2] Martin Jacobsson, Venkatesha Prasad, Martin Bauer, Jan Stoter, Rasmus L. Olsen, Jorge Lanza, Luis Sanchez, Juan Rico, Yanying Gu, Majid Ghader, Marc Girod Gene, Djamal Zeghlache, PN Framework Solution and Performance, D2.3.2, MAGNET Beyond, IST-027396

[MAG D4.3.3] Dimitris Kyriazanos, Andreas Pashalidis, Jan Stoter, Jasper Goseling, Shahab Mirzadeh, Puri Novelti, Anggraeni, Neeli Prasad, Antonietta Stango, Solutions for Identity Management, Trust Model and Privacy for PNs, D4.3.3, MAGNET Beyond, IST-027396

[Dey00] Dey, A.K., Providing *Architectural Support for Building Context-Aware Applications*, Ph.D. thesis, Georgia Institute of Technology, 2000

[Pascoe99] Pascoe J., Ryan N.S., Morse, D.R., *Issues in developing context-aware computing*. Proceedings of the International Symposium on Handheld and Ubiquitous Computing (Karlsruhe, Germany, Sept. 1999), Springer- Verlag, 208-221.

[Schilit94] Schilit, B.N., Adams, N.I. & Want, R. (1994). *Context-aware computing applications*. Proceedings of the Workshop on Mobile Computing Systems and Applications, pp 85-90. IEEE Computer Society, Santa Cruz, CA.

[Bauer06] M.Bauer, R.L.Olsen, L.Sanchez, J.Lanza, M.Imine, N.Prasad, "*Context management framework for MAGNET Beyond*", invited paper, Joint MAGNET Beyond, e-SENSE, DAIDALOS and CRUISE IST workshop, Myconos, Greece, June 2006.

[Salber99] Salber, D., DEY, A.K., and Abowd, G.D. The CONTEXT TOOLKIT:. Aiding the development of CONTEXT-enabled applications. Proceedings of CHI'99, 1999

[OSGi] OSGi Alliance, http://www.osgi.org/

[Hansen10] Martin Bøgsted, Rasmus L. Olsen, Hans-Peter Schwefel, *Probabilistic models for access strategies to dynamic information elements*, Performance Evaluation, Volume 67, Issue 1, January 2010, Pages 43-60, ISSN 0166-5316, DOI: 10.1016/j.peva.2009.08.015.

[Bondavelli07] Andrea Bondavalli, Andrea Ceccarelli, Lorenzo Falai, Michele Vadursi, "Foundations of Measurement Theory Applied to the Evaluation of Dependability Attributes," Dependable Systems and Networks, International Conference on, pp. 522-533, 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07), 2007.

[Olsen06] Rasmus L. Olsen, H.-P. Schwefel and M. B. Hansen, *Quantitative analysis of access strategies to remote information in network services*, 2006, In proceedings of Globecom 2006

[Figuerias] J.C.P.Figueiras, Accuracy Enhancements for Positioning of Mobile Devices in Wireless Communication Networks, Ph.D. Dissertation, Aalborg University, January 2008, ISBN: 87-92078-38-9

[Lipsky92] L. Lipsky. *Queueing Theory: A Linear Algebraic Approach*. McMillan and Company, New York, NY, 1992.

[Schwefel07] Schwefel, H.P. and Praestholm, S. and Andersen, SV, “Packet Voice Rate Adaptation Through Perceptual Frame Discarding”, IEEE Global Telecommunications Conference, 2007. GLOBECOM'07, pp. 2497--2502

[Klaue03] Klaue, J. and Rathke, B. and Wolisz, A., “Evalvid-a framework for video transmission and quality evaluation”, Lecture notes in computer science, pp. 255--272, 2003

[Qvist07] Qvist, M; Schwefel, HP; Hansen, MB, “A quantitative decision model for bottleneck link upgrades in packet switched networks“, 14th International Conference on Analytical and Stochastic Modelling Techniques and Applications, ASMTA 2007, pp. 171—177

[DDC] Default Delivery Context, recommendations for mobile web, W3C Mobile Web Best Practices Working Group, Mobile Web Best Practices 1.0, July 2008, http://www.w3.org/TR/mobile-bp/

[Cassandra1995] Cassandra, A.R. and Kaelbling, L.P. and Littman, M.L., “Acting optimally in partially observable stochastic domains”, Proceedings of the National Conference on Artificial Intelligence, 1995

[Seok04] Seok Joo Koh, Moon Jeong Chang, and Meejeong Lee, “mSCTP for Soft handover in Transport Layer”, IEEE Communication Letters, Vol. 8, No. 3, March 2004

[Wedlund99] E. Wedlund and H. Schulzrinne, “Mobility support using SIP”, The second ACM International workshop on Wireless Mobile Multimedia, pp. 76-82, Aug. 1999.

[Moh07] Shantidev Mohanty, Ian F. Akyildiz, “Performance Analysis of Handoff Techniques Based on Mobile IP, TCP-Migrate, and SIP”, IEEE Trans. On Mobile Computing, Vol. 6, No. 7, July 2007

[Wakikawa09] R. Wakikawa, V. Devarapalli, G. Tsirtsis, T. Ernst, K. Nagami, Internet Draft: “Multiple care-of-address registration” - http://tools.ietf.org/html/draft-ietf-monami6-multiplecoa-14, October 2009

[Soliman09] H. Soliman, G. Tsirtsis, N. Montavont, G. Giaretta, K. Kuladinithi, Internet Draft: “Mobile IPv6 flow routing over multiple care-of-addresses” - http://tools.ietf.org/html/draft-ietf-mext-flow-binding-03, July 2009

[Gundavelli09] S. Gundavelli, K. Leung, K. Srinivas, Internet Draft: Multiple Tunnel Support for Mobile IPv4 - http://tools.ietf.org/html/draft-gundavelli-mip4-multiple-tunnel-support-01, July 2009

[RFC3344] C. Perkins, RFC3344 - IP Mobility Support for IPv4, August 2002

[RFC4721] C. Perkins, P. Calhoun, J. Bharatia, RFC4721 - Mobile IPv4 Challenge/Response Extensions (Revised), January 2007

[RFC3775] D. Johnson, C. Perkins, J. Arkko, RFC3775 - Mobility Support in IPv6, June 2004

[RFC1961] P. McMahon, RFC1961 - GSS-API Authentication Method for SOCKS Version 5, June 1996

[RFC1929] M. Leech, RFC1929 - Username/Password Authentication for SOCKS V5, March 1996

[Steinmetz96] Ralf Steinmetz, Human perception of jitter and media synchronization. Selected Areas in Communications, IEEE Journal on,14:61–72,1996.

[Verissimo] Paulo Verissimo and Luis Rodrigues, Distributed Systems for System Architects, Kluwer Academic Publishers, 2004, ISBN 0-7923-7266-2

[Christian89] Flaviu Cristian, Probabilistic clock synchronization, Distributed Computing, Volume 3, Number 3, September 1989, p.146-158, Springer Berlin/Heidelberg, ISSN 0178-2770 (Print) 1432-0452 (Online)

[Ntp] Network Time Protocol, online at http://en.wikipedia.org/wiki/Network_Time_Protocol

[Escobar94] Escobar, J., Partridge, C., and Deutsch, D. 1994. Flow synchronization protocol. IEEE/ACM Trans. Netw. 2, 2 (Apr. 1994)

[Boronat08] Boronat Seguí, F., Guerri Cebollada, J. C., and Lloret Mauri, J. 2008. An RTP/RTCP based approach for multimedia group and inter-stream synchronization. Multimedia Tools Appl. 40, 2 (Nov. 2008)

[Andreozzi06] S. Andreozzi, D. Antoniades, and P. Trimintzios. On the Integration of Passive and Active Network Monitoring in Grid Systems. CoreGRID 2006

[Jain02] M. Jain, and C. Dovrolis. Pathload: a measurement tool for end-to-end available bandwidth. Proc. of Passive and Active Measurement Workshop 2002

[Ostermann09] Shawn Ostermann (Ohio University), TCPTRACE, http://www.tcptrace.org/, Accesed October 2009.

[Pfeiffer] René Pfeiffer. Measuring TCP Contestion Windows. Online at Linux Gazzette (http://linuxgazette.net/), March 2007.

[Jiang02] H. Jiang, and C. Dovrolis. Passive Estimation of TCP Round-Trip Times. SIGCOMM Comput. Commun. Rev., 2002

[Michael02] L. Michael, and G. Lior. The effect of Packet Reordering in a Backbone Link on Application Throguhput. Network IEEE 2002.

[Prasad03] R. Prasad, C. Dovrolis, M. Murray, and K. Claffy. Bandwidth Estimation: Metrics, Measurement Techniques and Tools. IEEE Network 2003

[But05] J. But, U. Keller, D. Kennedy, and G. Armitage. Passive TCP Stream Estimation of RTT and Jitter Parameter", The IEEE Conference on Local Computer Networks, 2005.

[Jaiswal04] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D Towsley, Inferring TCP Connection Characteristics Through Passive Measurements, Proceedings of IEEE InfoCom2004, March 2004.

[Padhye01] J. Padhye and S. Floyd, On Inferring TCP Behaviour, Proceedings of ACM SIGCOMM2001, August 2001

[Jiang02] H. Jiang and C. Dovrolis, Passive Estimation of TCP Round-Trip Times, ACM SIGCOMM Computer Communications Review, vol. 32 no. 3, 2002, pp. 75-88

[Aikat03] J. Aikat, J. Jaur, F. Smith and K. Jeffay, Variability in TCP Roundtrip Times, Proceedings of ACM SIGCOMM Internet Measurement Workshop, October 2003, pp. 279- 284

[Jaiswal03] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose and D. Towsley, Measurement and Classification of Out-of- Sequence Packets in a Tier-1 IP Backbone, Proceedings of IEEE Infocom2003, April 2003

[Torino09] Politecnico di Torino – TCP Statistic and Analysis Software Tool, http://tstat.tlc.polito.it/index.shtml, Accessed October 2009.

[Marchese07] Mario Marchese, QoS Over Heterogeneous Networks, April 2007.

[Gary07] Gary A. Donahue, O'Reilly Media, Network Warrior, June 2007

[RFC2212] S. Shenker, C. Partridge, R. Guerin, RFC2212 - Specification of Guaranteed Quality of Service, September 1997

[RFC2475] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss, RFC2475 - An Architecture for Differentiated Services, December 1998

[RFC2597] J. Heinanen, F. Baker, W. Weiss, J. Wroclawski, RFC2597 - Assured Forwarding PHB Group, June 1999

[RFC2459] R. Housley, W. Ford, W. Polk, D. Solo, Internet X.509 Public Key Infrastructure - Certificate and CRL Profile, RFC2459, January 1999

[RFC4732] M. Handley, E. Rescorla, Internet Denial-of-Service Considerations, RFC4732, November 2006

A. OVERVIEW OF PROTOTYPES

The purpose of this appendix is only to provide a short overview of what has been implemented from a Work Package 3 view. The blocks encircled with red lines in Figure 56 are indicating the functionalities which have been actually implemented.

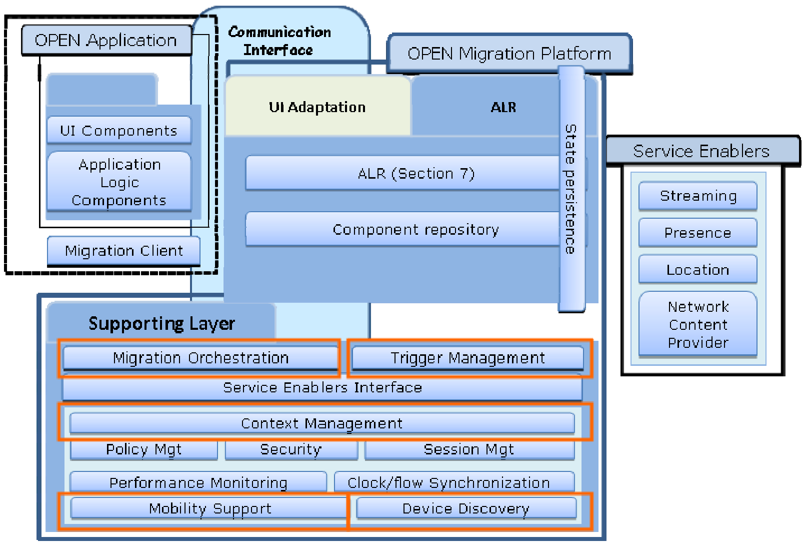


Figure 56: Overview of complete OPEN platform, with Work Package 3 focus area at the supporting layer, and red encircled functionalities shown via prototypes.

The prototypes that have been done focus on are the following

- Context Management Framework
  - Context Information access and distribution
    - Location/distance estimation (RFID and Bluetooth based)
    - User orientation
  - Profile and information management
- Device discovery using user orientation
- Migration Orchestration
  - Device Discovery
  - Device Registering
  - Migration orchestration
- Trigger Management
- Mobility Support

With these prototypes Work Package 3 has demonstrated the platform's capability to support the migratory applications and services that have been developed in Work Package 5, and integrated as a part of Task 4.2 with other components coming from Work Package 2 and Task 4.1. The prototypes developed showing parts of the platform, is described in more details in Deliverable 3.3, [OPEN D3.3].

B. MOBILITY SUPPORT - APPLICATION SERVER WITH AND WITHOU SIP

B.1 APPLICATION SERVER WITH SIP CAPABILITIES

If the AS is SIP enabled and the OPEN MS acts as a SIP proxy both session and terminal mobility can be handled by SIP. The SIP messages which need to be sent could perhaps be utilized in the OPEN platform to carry the internal OPEN signaling e.g. by letting the OPEN migration server act as the SIP proxy in the OPEN network. It is also possible that the OPEN platform can utilize some of the information present in the SIP signaling.

If the OPEN MS acts as a SIP proxy for the OPEN clients, it is possible to have all SIP signaling going through the OPEN MS by using the “Record-Route” field in the SIP messages.

A standard SIP dialog could look like depicted in Figure 57. Both SIP UAs needs to be registered in order for the proxies to be able to look up the inbound proxy. When UA1 wants to contact UA2 (the Application Server with SIP support) it will send the SIP INVITE through the outbound proxy for the UA1 domain, which will look up the inbound proxy for UA2 using DNS SRV. The SIP INVITE is then forwarded to the inbound proxy and further on to UA2. All SIP messages are routed through the proxy in the domain of UA1 by using the “Record-Route” header field. Otherwise the SIP ACK, BYE and last 200 OK would have been sent directly between the two UAs

If a migration is triggered and SIP is used to make that happen from a network perspective, this would look like illustrated in Figure 58. Here there is no proxy depicted since it works the same way only the messages are routed through the SIP proxy. When a migration is triggered a SIP REFER message is sent to the AS which can then use a SIP INVITE to establish a dialog with the target device.

Having a SIP enabled AS would be the same situation as when OPEN is interacting with IMS.

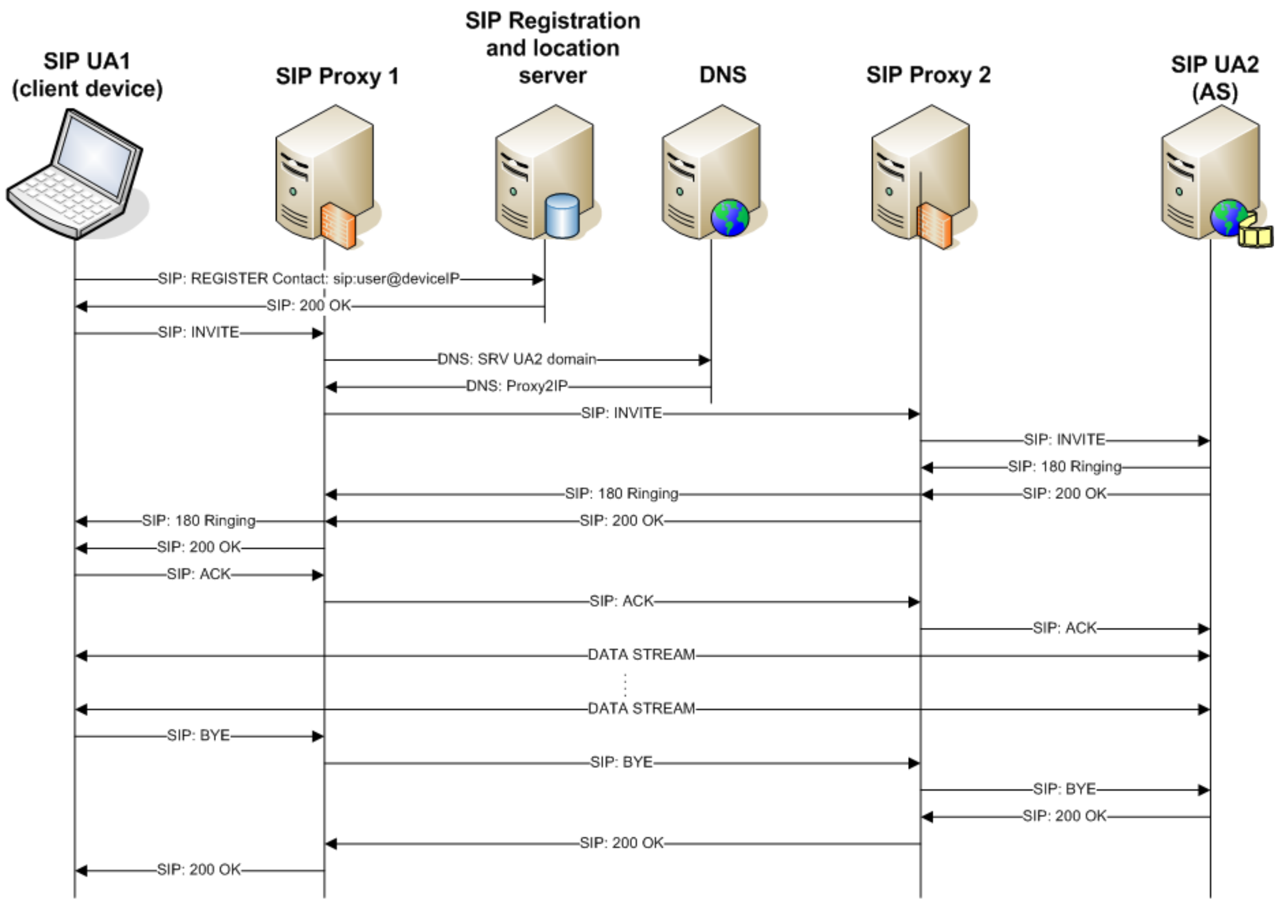


Figure 57: SIP dialog using proxies

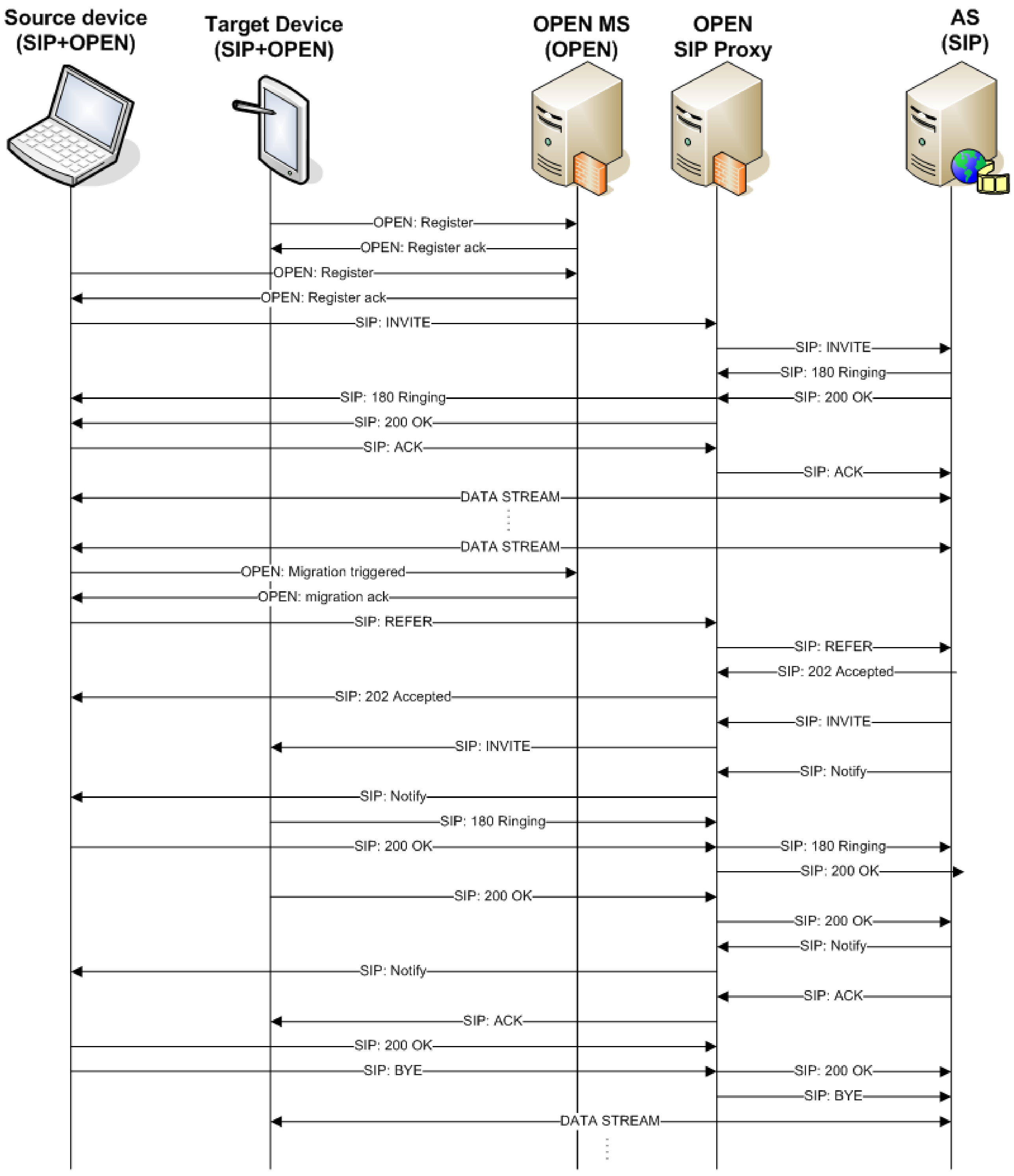


Figure 58: SIP dialog used in a migration process.

B.2 APPLICATION SERVER WITHOUT SIP CAPABILITIES

When the AS does not have SIP capabilities the SIP messages can only be used inside the OPEN network. This means the SIP signaling will have to terminate at the border of the OPEN network as illustrated in Figure 59 where a SIP UA is interacting with the application server outside the OPEN network.

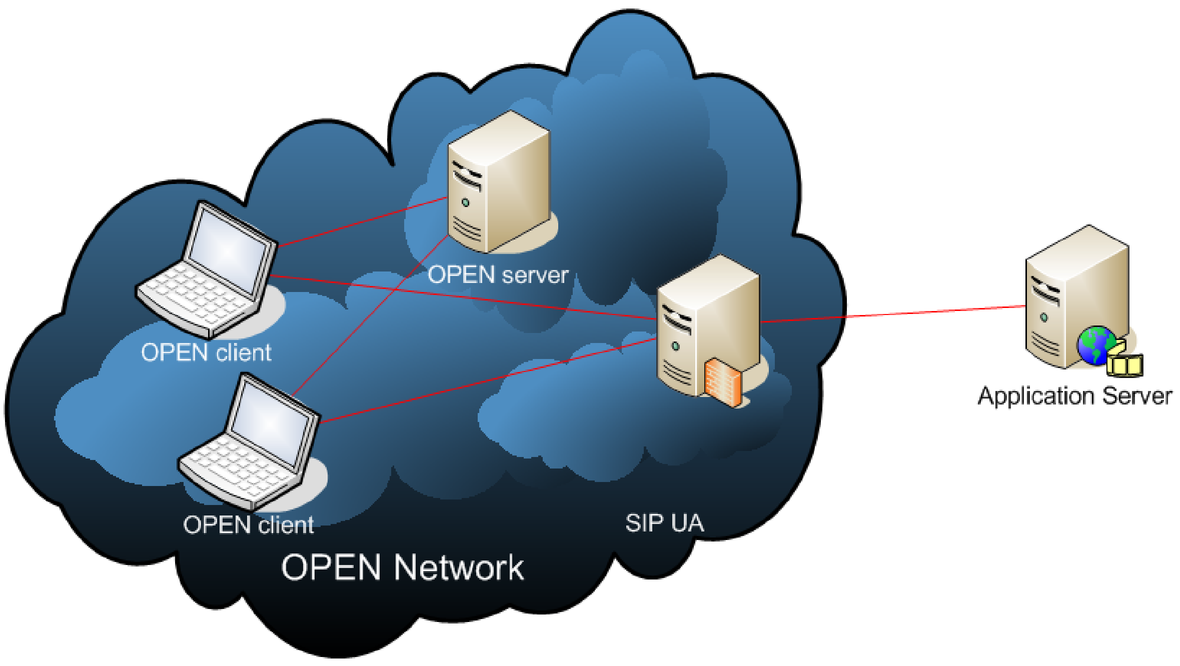


Figure 59: The figure shows how SIP can be used without application server support. A SIP UA will have to be placed at the border of the OPEN network to terminate the SIP signaling path.

The SIP UA can in principle be either a Back-to-Back UA or a gateway UA but for this purpose a gateway UA like used e.g. interacting with the PSTN network actually is what is needed in order to translate between the two types of networks. The SIP gateway effectively works as the MAP defined in the general mobility support function as shown in Figure 60.

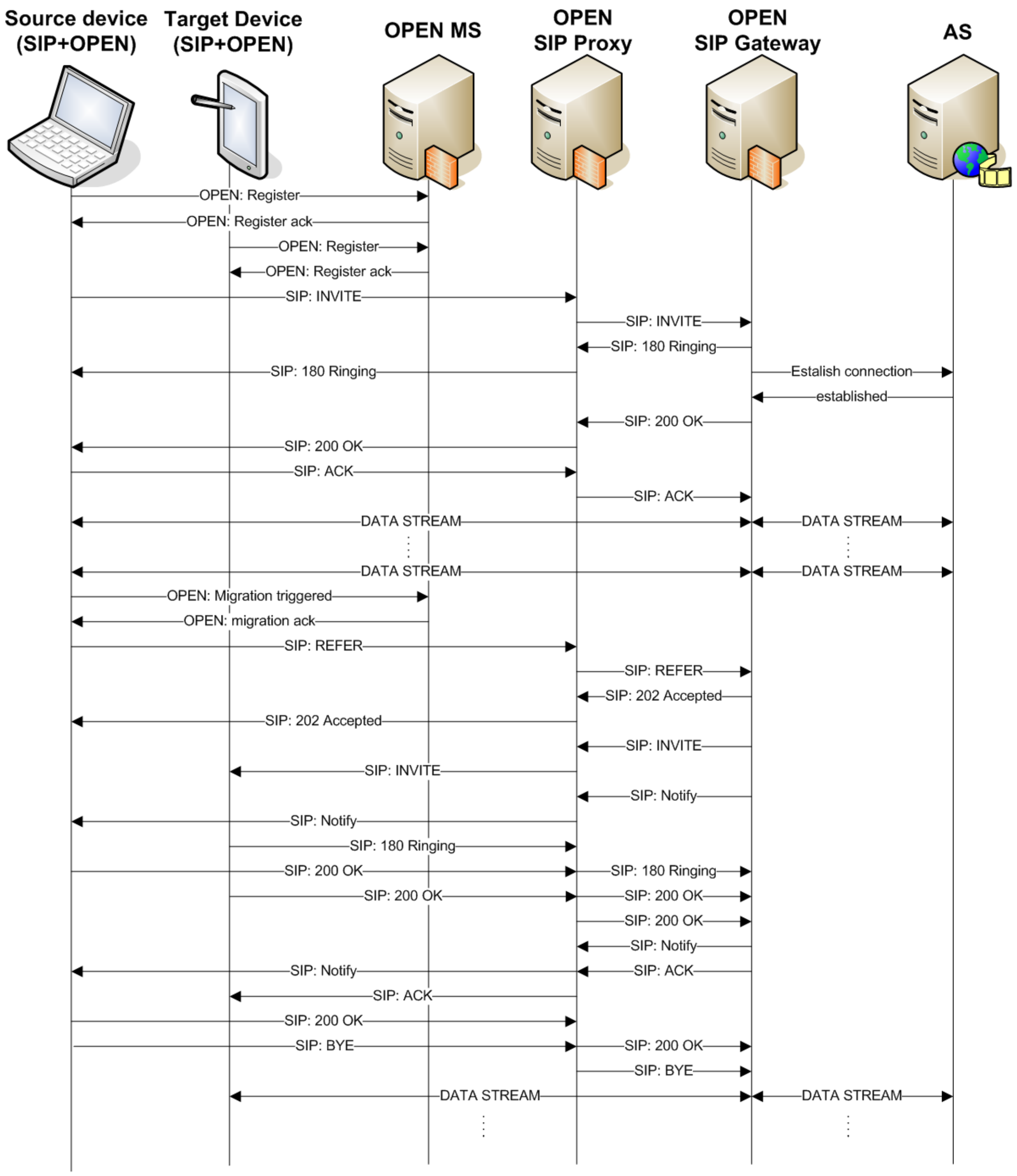


Figure 60: Migration without AS SIP support using SIP gateway

C. PERFORMANCE MONITORING

The performance monitoring module is an OPEN middleware solution to provide information about the performance of the platform to other modules within the OPEN platform. The reported performance could be of many types e.g. CPU utilization, memory usage etc, but in this initial work only the performance of the network is considered. The collected information about network performance is viewed as context information and it is passed to Context Management Framework (CMF). This information is stored at the CMF. Currently the network performance information is only accessed by the trigger management module in order to take the network performance into account when evaluating potential candidates for future migrations.

The importance of having information about the network performance can be illustrated by the following example. Let's imagine that we have two terminal devices connected through different access networks to the AS on the Internet. One of the networks becomes overloaded while the other one has more available bandwidth and better delay characteristics. Using this information, service migration can be initiated when a critical degradation in the network condition is detected.

The task of Performance monitoring module is to measure the most representative parameters in every end to end path. The metrics used to describe the performance of a network are typically time-varying due to e.g. changes in cross-traffic or channel conditions. To have useful observations, the measurements can be performed by Performance monitoring module on demand (reactively) by a request from CFM or in a proactive manner to have measurements already available when a request comes. In the proactive approach the frequency of updates can be made adaptive depending on how static the network is. The choice between a reactive or proactive method reflects a trade-off between the overhead associated with the measurements and the accuracy of the obtained information.

C.1 METRICS

The following metrics are usually used to describe network characteristics:

DELAY-RELATED MATRICS

Round-trip time: The network RTT is the time taken for a packet to traverse the network from the source to the destination and back. RTT can be easily measured using active monitoring tools like for example ping. However, it is also possible to measure RTT using solely passive monitoring techniques. One such technique is based on monitoring the TCP connections that pass through a link [Jiang02]. RTT can be estimated more accurately based on the time difference between the SYN and ACK packets exchanged during the three-way handshake of a TCP connection.

One-Way Delay: The one-way delay is the time taken for a packet to traverse the path from the source to the destination. The asymmetric routing that commonly occurs within the Internet makes this metric important for some applications. The one-way delay can be measured using two passive monitors located at the source and destination network domains. When the same packet passes through both monitors, the one-way delay can be measured from the difference in the time each monitor observed the packet. For such measurements, the clocks of the monitors have to be synchronized (using Network Time Protocol NTP or Global Positioning System GPS, depending on the required accuracy).

Jitter: Jitter is the variation in the one-way delay of successive packets. Jitter is particularly important for real-time applications, since it predetermines the sizes of the relevant stream buffers.

PACKET LOSS-RELATED METRICS

Packet Loss Ratio: Packet loss occurs when correctly transmitted packets from a source never arrive at the intended destination. Packets can be lost due to e.g. congestion at the queue of some router, due to routing system problems or due to poor network conditions that may result to damages in the datagram. The packet loss ratio is a very important metric since it affects data throughput performance and overall end-to-end quality [Michael02].

BANDWIDTH-RELATED METRICS

Capacity: Capacity is considered here as the maximum possible bandwidth a link or path can deliver. The link with the minimum transmission rate determines the capacity of the path.

Available bandwidth: AB is the maximum unused bandwidth at a link or path. The link with the minimum non-utilized capacity limits the available BW. The available bandwidth of a link relates to the unused or spare capacity of the link during a certain time period. One should note that the capacity of a link depends on the underlying transmission technology and propagation medium. However, the available bandwidth of a link additionally depends on the traffic load at that link, and is typically a time-varying metric. At any specific instant in time, a link is either transmitting a packet at full link capacity or idle, so the instantaneous utilization of a link can only be either 0 or 1. Thus, any meaningful definition of available bandwidth requires time averaging of the instantaneous utilization over the time interval of interest. The average utilization $\bar{u}(t - \tau, t)$ for a time period $(t - \tau, t)$ is given by

$$\bar{u}(t-\tau,t) = \frac{1}{\tau}\int_{t-\tau}^{t} u(x)dx$$

where u(x) is the instantaneous available bandwidth of the link at time x. We refer to time length τ as the averaging timescale of the available bandwidth.

From OPEN project perspective, we will focus first of all on RTT and available bandwidth estimation as these parameters provide information about the current status of a network. Answering a question if a service can be migrated to another network, we are not interested in capacity since a network with large capacity might be overloaded, and instead we obtain measurements for available bandwidth.

C.2 ACTIVE AND PASSIVE MONITORING STRATEGIES

A common classification scheme that is often used when dealing with network monitoring distinguishes between active and passive monitoring techniques. A monitoring tool is classified as active if it induces traffic into a network, otherwise it is classified as passive.

C.2.1 ACTIVE MONITORING TOOLS

The active approach relies on the capability to inject test packets into the network or send packets to servers and applications, following them and measuring service obtained from the network. As such it does create extra traffic, and the traffic or its parameters are artificial. This can present a problem and affect the obtained results especially when the traffic volume is significant. On the other hand, active approach provides explicit control on the generation of packets for measurement scenarios. This includes control of the traffic generation at the application layer (packet sizes, packet frequency, type of traffic, path) and when to schedule it.

C.2.2 PASSIVE MONITORING TOOLS

The passive approach uses devices to watch the traffic as it passes by- These devices can be special purpose devices such as a sniffer, or they can be built into other devices such as routers, switches or end node hosts. Examples of such built in techniques include Remote Monitoring (ROMN), Simple Network Monitoring Protocol (SNMP) and netflow capable devices. The passive monitoring devices are polled periodically and information is collected (in the case of SNMP devices the data is extract from Management Information Databases (MIB) to assess network performance and status.

The passive approach does not increase the traffic on the network for the measurements and it measures the real traffic. The polling required to collect the data from the passive measurements does however generate network traffic. The amount of data gathered can be substantial especially if one is doing flow analysis or trying to capture information on all the packets.

Since the passive approach may require viewing all the packets on the network, there can be privacy or security issues about how to access/protect the data gathered.

C.2.3 ACTIVE VS PASSIVE COMPARISON

Passive monitoring is more appropriate for monitoring gross connectivity metrics like link throughput; it is also needed for accounting purposes. Passive monitoring techniques analyze network traffic by capturing and examining individual packets passing through the monitored link, allowing for fine-grained operations, such as deep packet inspection. The main benefit of passive monitoring approaches, compared to active monitoring, is its non-intrusive nature, see Table 7. Active network monitoring techniques incur an unavoidable network overhead due to the injected probe packets, which compete with user traffic. In contrast, passive network monitoring techniques passively observe the current traffic of the monitored link, without introducing any network overhead.

	Pros	Cons
Active	Control traffic generation Can be used when information is needed	Overhead Not instant (probes needed to get a measurement)
Passive	Instant (when available) No overhead	Security/privacy issues Requires network activity

Table 7: Active versus Passive monitoring tools.

C.3 TAXONOMY OF PUBLIC NETWORK MONITORING TECHNIQUES AND TOOLS

This section summarizes some of the available active and passive performance measurement tools. They have been classified according the parameter they measure shown in Table 8.

Metric	Active	Passive
RTT	**Ping**	**TCP Stack Kernel access**, tcptrace, 3-way handshake, TSTAT
Jitter	**Pathload**	**TCP Stack Kernel access**, tcptrace
Packet Loss	**Ping, IPerf**	**TCP Stack Kernel access**, tcptrace
Available BW / Throughput	IPerf, Cprobe, **Pathload**, Pathchirp, IGI	**cwnd**, tcptrace

Table 8: List of parameters and relevant measurement methodologies.

C.3.1 PING

Ping is a computer network tool used to test whether a particular host is reachable across an IP network; it is also used as a latency test. It works by sending ICMP “echo request” packets to the target host and listening for ICMP “echo response” replies. Ping measures the round-trip time and records any packet loss, and prints when finished a statistical summary of the echo response packets received, the minimum, mean, max and in some versions the standard deviation of the round trip time.

C.3.2 PATHLOAD

Pathload [Jain02] is a tool that can estimate the available bandwidth of network paths. It is based in Self Loading Periodic Streams (SloPS). SLoPS is a recent measurement methodology for measuring end-to-end available bandwidth. The source sends a number K ≈ 100 of equal-sized packets (a periodic packet stream) to the receiver at a certain rate R. The methodology involves monitoring variations in the one-way delays (jitter) of the probing packets. If the stream rate R is greater than the path's available bandwidth A, the stream will cause a short-term overload in the queue of the tight link. One-way delays of the probing packets will keep increasing as each packet of the stream queues up at the tight link. On the other hand, if the stream rate R is lower than the available bandwidth A, the probing packets will go through the path without causing increasing backlog at the tight link, and their one-way delays will not increase. In order for Pathload to calculate one-way delays the clocks of the two endpoints needs to be synchronized.

C.3.3 TCPTRACE

tcptrace is a tool written by Shawn Ostermann at Ohio University, for analysis of TCP dump files. It can take as input the files produced by several popular packet-capture programs, including tcpdump, snoop, etherpeek, HP Net Metrix, and WinDump. *tcptrace* can produce several different types of output containing information on each connection seen, such as elapsed time, bytes and segments sent and recieved, retransmissions, round trip times, window advertisements, throughput, and more. It can also produce a number of graphs for further analysis [Ostermann09].

C.3.4 TCP STACK KERNEL

The Linux kernel has internal data structures that keep track of active TCP connections and their parameters. These data structured are used by the kernel to keep track of the information needed to make the TCP algorithms work for e.g. TCP flow control. Inside an application that owns a network socket with a live TCP connection, we can always request the current TCP parameters. The man pages of `getsockopt()` and `tcp` tell us how to do this - calling `getsockopt()` with the `TCP_INFO` option fills a memory structure with information described in `struct tcp_info`, which is defined in `/usr/include/netinet/tcp.h`. [Pfeiffer].

D. QUALITY OF SERVICE

This Section of the appendix investigates the possibility of implementing a reliable and effective Quality of Service (QoS) solution to an OPEN based migration network. A brief explanation of different QoS solutions is presented in order to determine the most suitable solution. QoS can be implemented in OPEN environment to enhance and guarantee traffic to and from any of the OPEN server modules in case of network congestion. QoS in OPEN is used as a measure of assurance for a more reliable migration process in case of network congestion.

D.1 WHAT IS QOS?

Quality of Services in the networking terms means the ability of a network element where that element can be an application, host or router to have a degree of guarantee that its network traffic and service requirements can be satisfied. Figure 61 show a list of protocols and technologies that may be implemented over different physical supports and the list of possible QoS solutions and algorithms that may be implemented.

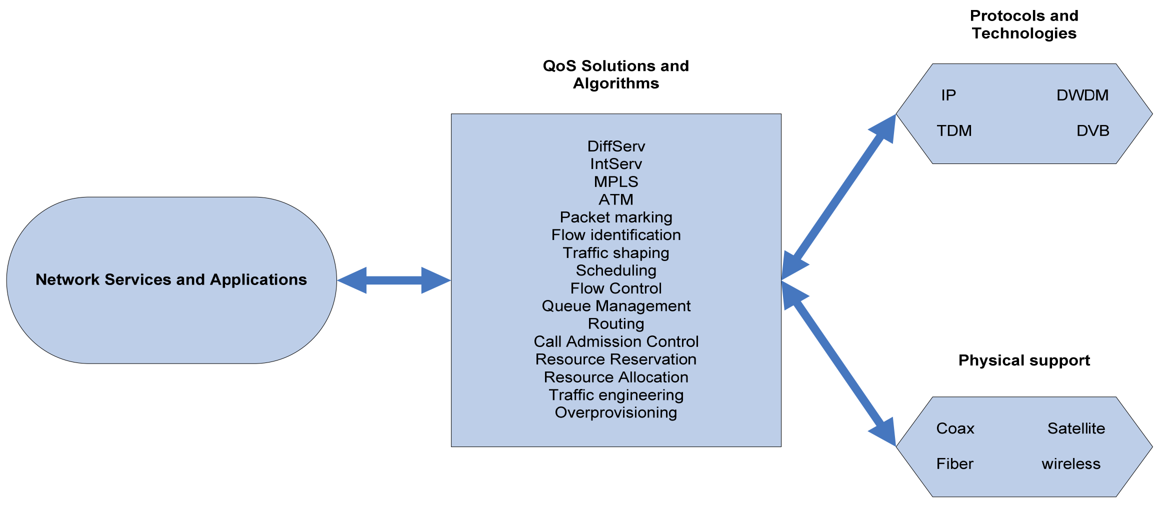


Figure 61: QoS Solutions and Algorithms [Marchese07].

Intrinsic QoS is a term that is referred to the type of QoS that are directly implemented and provided in the network itself. It is therefore interesting to see the impact of implementing a type of intrinsic QoS in the network architecture of the OPEN environment. In general QoS are implanted into networks to aid the applications and services that require a certain degree of level assurance from the network, an example on of such applications and services can be basic services for information transfer for backbones and access networks, assured database, protection of vulnerable network packets... etc.

There are several types of intrinsic QoS oriented technologies such as ATM, MPLS, QoS-IPv4 and QoS-IPv6. Since the OPEN network environment is IP based it is therefore interesting to look into implementing an IPv4 based QoS solution.

D.2 IPV4 QOS SOLUTIONS

Native IP on its own provides a connectionless best-effort service i.e. the packets received by a user depend on the load of the network. IPv4 is able to mark traffic in two ways:

1. Integrated Services (IntServ)
2. Differentiated Services (DiffServ)

D.2.1 INTEGRATED SERVICES

The Intserv QoS solution is based on the concept of single flow identification, a flow is indentified in Intserv by a 5-Tuple IP Source address, IP destination address and Protocol which are found in the IP header and TCP/UDP source port and TCP/UDP destination port which are found in the TCP/UDP header as can be seen in Figure 62.

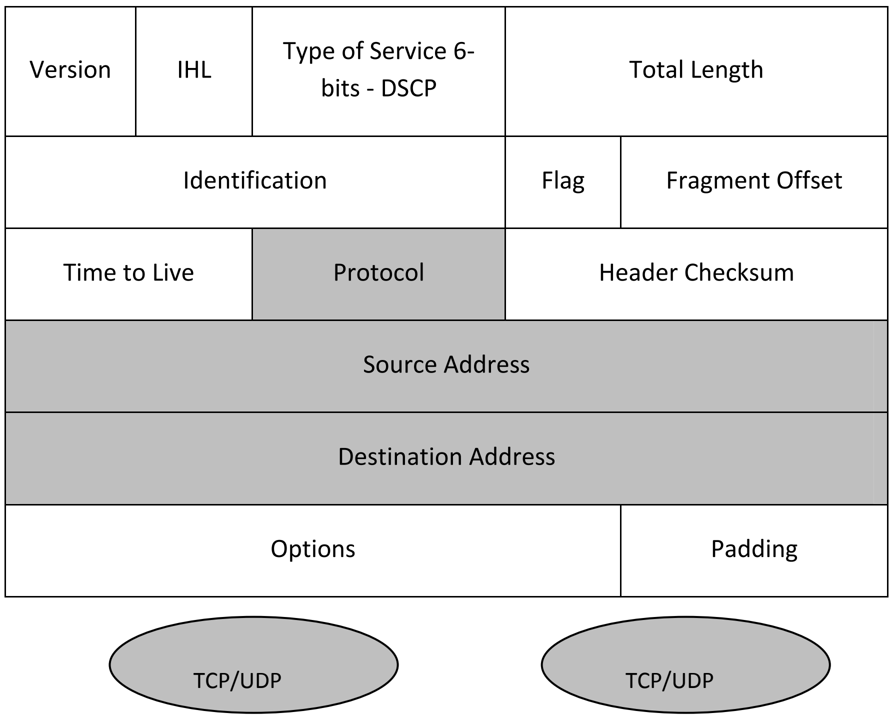


Figure 62: IPv4 Single Flow identifiers [Marchese07].

After identifying a flow QoS can be achieved by the appropriate tuning of resource reservation, admission control, packet-scheduling and buffer management.

The Intserv approach defines service classes the first is Guaranteed Service (GS) and the second is Controlled-Load Service (CLS). GS is defined in [RFC2212] and as it states it assures that IP packets arrive within a specific maximum delivery time and that the IP packets are not discarded according to congestion.

D.2.2 DIFFERENTIATED SERVICES

The Diffserv QoS [RFC2475] attempts to deal with the scalability problem that is found in Intserv QoS. Diffserv makes use of the 8-bits Type of Service (ToS) field that is found in the IPv4 header. The first 6 bits in the ToS header are used to define the Differentiated Services Code Point (DSCP) field as shown in Figure 63.

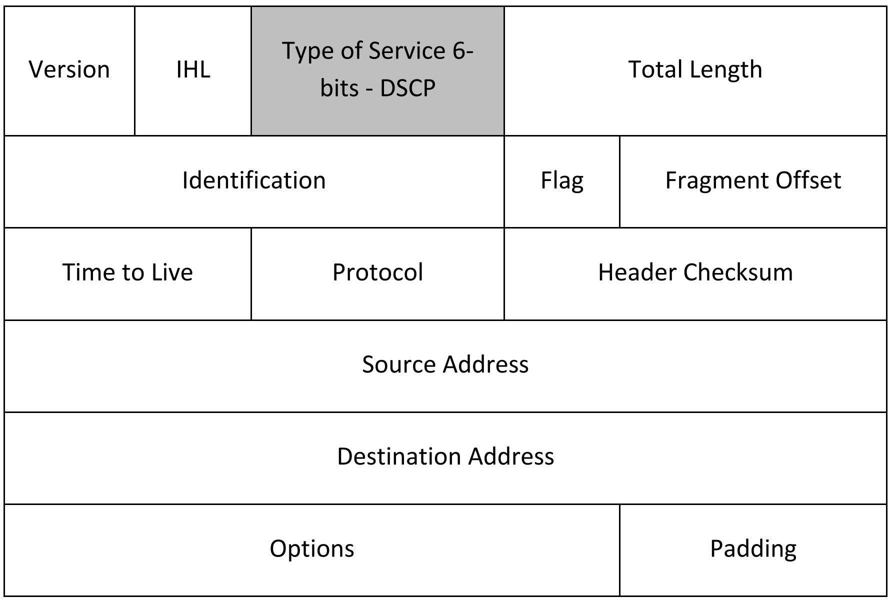


Figure 63: IPv4 packet header and group of flows identifier [Marchese07].

The 6-bit DSCP field indicates how the packets are to be forwarded within the DiffServ domain of different network providers. This behavior is called the Per Hop Behavior (PHB) and it is predefined by the network provider. The DiffServ QoS solution works differently from Intserv QoS solution in the sense that it does not identify each single user flow that is traversing the network but it classifies and aggregates the network traffic into different traffic classes as seen in Figure 64, these traffic classes are distinguished form each other through the DSCP field.

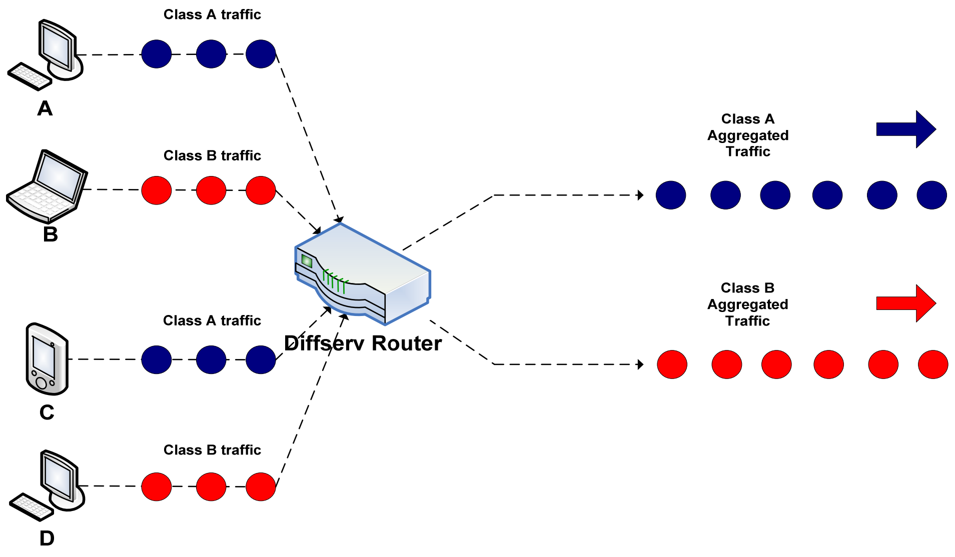


Figure 64: Diffserv aggregation traffic behavior [Marchese07]

The marking and assigning of packets is performed at the networks edge routers based on the network providers predefined policy. Within the Diffserv domain the network's core routers manage packets according to the value found in the DSCP field. The class selector PHB that can be marked in the DSCP field offers three types of forwarding priorities:

1. Expedited Forwarding (EF): This type of forwarding is characterized by a minimum configurable service rate, independent of the other aggregates within the router, and oriented to low delay and low loss services [Marchese07].

2. Assured Forwarding (AF): This type of forwarding is defined and recommended in [RFC2597] for 4 independent classes (AF1, AF2, AF3, AF4), within each of the classes traffic is differentiated into 3 drop precedence categories (low, medium, high). The packets marked with the highest drop precedence are dropped with lower probability than those that are marked with lower drop precedence [Marchese07].

3. Best Effort (BE): This type of forwarding does not provide any performance guarantee and does not define any QoS level [Marchese07].

D.2.3 INTSERV VS DIFFSERV

The advantages and disadvantages of both of the QOS-IPv4 solutions are presented in Table 9.

Feature	**Intserv**	**Diffserv**
QoS assurance	Per-flow	Per-aggregate

QoS assurance range	End-to-end	Diffserv domain
Resource reservation	Controlled by application	Configured at Edge router
Scalability	Limited by no. flows	Limited by no. of classes
QoS Services	GS, CLS, best effort	EF, AF and best effort
Complexity	High	Low
Availabilty	Yes	Yes

Table 9: Overview of differences between Intserv and Diffserv

Due to its low complexity and higher scalability Diffserv is therefore considered over Intserv to be implemented as a QoS solution for an OPEN environment.

D.3 DIFFSERV IN OPEN ENVIRONMENT

How fast and efficient a migration process is executed is a main concern to the network user. QoS is intended to be used in OPEN to assure more accurate and faster migration in case of network congestion. To be able to implement a Diffserv solution that assures migration during congestion, first the migration network traffic should be classified. A basic Policy scenario to determine how Diffserv affects migratory networks could be the identification and prioritization of migration control traffic that is sent and received by the orchestrator in the OPEN migration server. The different migration traffic can be seen in Figure 4, as the figure shows there is traffic to and from the OPEN migration server, OPEN mobile anchor point and source and target client devices.