Journal of Mobile Multimedia, Vol. 5, No.1 (2009) 081-112

ANALYSIS OF CONNECTIVITY AND SESSION MANAGEMENT FOR MOBILE PEER-TO-PEER APPLICATIONS

OTSO KASSINEN TIMO KOSKELA ERKKI HARJULA

JUKKA RIEKKI MIKA YLIANTTILA

University of Oulu, Oulu, Finland
{firstname.lastname}@ee.oulu.fi

Received May 23, 2007
Revised January 20, 2009

Mobile applications utilizing wireless networks are growing in popularity as increasingly capable terminals and advanced networking technologies emerge. In order to provide a seamless user experience, applications must be able to rely on an intelligent mobile middleware that hides the complexity of underlying technologies and allows developers to solve application-specific problems instead. A middleware should take care of generic networking functionality such as management of user communities, signalling for sessions, interaction with content-licensing services, and management of the terminal's networking resources. This paper focuses on two major components of a prototype peer-to-peer networking middleware: a solution for connectivity management and another for session management. First, the connectivity management solution is discussed. The solution formalizes cross-layer resource optimization and employs upgradeable state machines to make connectivity selections based on context data and user preference, aiming to always provide the best connection for different communications and keep the system extensible. Second, the session management solution is discussed. The solution enables installation of missing software dynamically on a terminal when another user proposes a mutual application session. This greatly increases users' possibilities to initiate sessions with each other. In this paper, design principles behind each of the novel solutions are studied, their prototype implementations are evaluated on the Symbian smartphone platform, and they are contrasted with existing technologies. A lightweight Session Initiation Protocol (SIP) stack has also been implemented as a component for the middleware. Future work concerning the connectivity and session management solutions includes evaluation of the technologies in more realistic settings than was possible within the work for this paper.

Key words: Mobile middleware, cross-layer optimization, state machines, content distribution
Communicated by: K. Liu & P.J.M. Havinga

1 Introduction

Mobile networking has established its position as part of the everyday lives of a substantial number of end-users. Networked data-intensive mobile applications, such as e-mail clients and World Wide Web browsers, are used by an increasing number of people and organizations. Thanks to the ongoing adoption of more and more capable network environments and terminals, mobile networking is no longer a mere curiosity. Mobile data networks are becoming a regularly used enabling technology for applications, paving the way for novel wireless services as well as more flexible access methods to the existing Internet services.

However, mobile networking definitely has not yet reached its full potential. It is supposed that advanced wireless network services will emerge. The increasing availability of high-speed wireless access networks is a major enabler, but that alone does not drive the development of meaningful services and a seamless user experience. Unanswered questions include what kind of networked services are feasible; how users will interact with each other in the context of the services, i.e. what is the social impact of having a highly connected smart terminal in one's pocket 24 hours a day; what kind of enabling technologies are needed to provide reusable functionality, on an adequately high level of abstraction, for application developers who strive to create new innovative services on the platforms they are given; how do mobile terminals collaborate in the networks and cope with the dynamic and heterogeneous environment of wireless networks; and, how to achieve all this without bothering the end-user with technical details and crashing systems.

Efforts of today's researcher community are needed to shape the conceptual and technological basis for the solutions that we will see in commercial use five to ten years from now. Dominant as well as weaker trends in the field of mobile networking must be observed in order to be able to steer the course of advancement; one of the strong trends visible today is the tendency to use the Internet Protocol (IP) to carry increasingly many types of communication.

This paper introduces two mobile middleware components implemented for an All-IP network environment. The paper is organized as follows. First, we introduce the conceptual framework behind our work and present the research questions to be answered. Then we extensively discuss the design and implementation of our connectivity and session management middleware components; also measurement results of their performance are provided. Finally, we discuss the results and related future work, and provide a conclusion of the paper's contribution.

2 Application Supernetworking

2.1 The Concept

The unifying purpose of the All-IP project work is to realize the envisioned paradigm of *application supernetworking* by defining the components of the framework and implementing prototype systems. This paper elaborates the concepts of session and connectivity management with respect to application supernetworking.

Application supernetworking, according to [1], is a natural continuation of today's mobile communication. As the world moves towards the use of faster communication technologies and more capable end-user devices, the concept can be fully realized and deployed in real-world solutions, benefiting end-users and commercial players by offering a more seamless user experience and a set of novel services, as well as potentially reducing the costs of networking. Application supernetworking is defined as a functional framework, embracing the following three key elements: 1) multisessions and/or rich calls; 2) plug-and-play interactions between sessions and applications; and 3) holistic connectivity management.

While not completely abandoning the client-server model, application supernetworking has an emphasis on *peer-to-peer* (P2P) networking. Here the word P2P refers to application-layer protocols and applications, not to be confused with lower-layer technologies sometimes labelled as P2P, such as routing in ad hoc radio networks. The application-layer logical networks established among peers are

called *overlays* and determined by the P2P protocol used [2]. Discovery of fellow users and sharing of resources are examples of the usages of P2P networks in application supernetworking.

Peer groups are essential for modelling the relationships between users in the network. A peer group is a collection of co-operating peers sharing the same interests and providing a common set of services to other members of the group; members are held together by shared interests [3]. The management of peer groups is supported in certain P2P protocols such as JXTA, and also Direct Connect++ (DC++) in which the concept of users on a hub equals the concept of peer groups.

The session-related concepts may need clarification due to the possibly differing interpretations of the terms by different people; we use the terms as they were defined in [1]. A *session* is an abstract entity that is created when a process starts communicating with another, for example, when a call is initiated between two voice-over-IP (VoIP) application instances. The communicating processes typically run on different terminals. Sessions can also be established between processes of the same device. A session is destroyed when the communication channel is no longer needed.

In a peer group of N users and M possible application modules, group communication can be described as an N-dimensional session space E, presented as a matrix. This session space E indicates which application modules, at the moment, are being used for sessions between specific users in the group. The width of the matrix is M in every dimension, and each value in it can be 1 to indicate an active session, or 0 otherwise. A 2-dimensional matrix (i.e., the case of $N = 2$ terminals) in its general form is shown below, containing $M \times M$ session-entries S_{ij} ($1 \le i \le M$, $1 \le j \le M$):

$$E_{example} = \begin{matrix} S_{11} & S_{12} & \cdots & S_{1M} \\ S_{21} & \ddots & & \\ \vdots & & \ddots & \vdots \\ S_{M1} & & \cdots & S_{MM} \end{matrix} \quad (1)$$

If an active session S_{ij} has the property $i = j$ (the communicating application modules on both terminals are the same), it is called a *homogeneous session*. On the other hand, if it has the property $i \neq j$, it is called a *heterogeneous session*. Homogeneous sessions are on the diagonal of the matrix; heterogeneous ones are outside of the diagonal. "Different modules" might also be different versions of the same module or different implementations of the same functionality. On some level, these modules *are able to* communicate, but they are not identical; hence, such a session is heterogeneous. However, radically different application modules (e.g., a VoIP module and a file-sharing module), cannot communicate in a meaningful way. Thus, if the modules are completely different, a session requires dynamic installation of the missing software on the terminals that lack the software; then the resulting session is homogeneous, if both communicating modules are identical after the installation.

A *multisession* enables having multiple sessions open concurrently. For instance, a user may have two videoconference sessions, one voice call, and one remote desktop session active simultaneously. When a multisession is active, the platform can manage the resources required by the sessions in a centralized manner, potentially attaining synergy benefits between them. In E, a multisession exists if there are more than one homogeneous sessions between two specific terminals.

The term *rich call* refers to the use of several types of media streams within one session, as opposed to the separate streams within a multisession. Rich calls are visible in E as ordinary sessions S_{ij}; the matrix does not differentiate between rich calls and non–rich call sessions.

A *supersession* is in question, when there are several sessions between two specific terminals in E, and at least one of those sessions is a heterogeneous session.

Application supernetworking also provides support for *inter-device session mobility* [4]. This means that a session, when certain triggering conditions are fulfilled, is decoupled from its current physical network interface and also from the terminal it runs on, and moved to another host without the need to tear down the session. An example case could be the migration of a video call, received on a mobile phone, to the user's more capable device such as a PC.

Also *intra-device session mobility* is possible: consider a case where a heterogeneous session has been activated, but the terminal is downloading a module's upgrade package in the background. The upgrade package contains the module version that is identical with the other party's version and thus enables a homogeneous session. When the identical module has been installed, the session can be seamlessly transferred from the non-identical module to the new one. The benefit of this is that the session could be initiated *already* when only having non-identical modules. Obviously, it must be decided, which party's terminal upgrades its module to conform the other party's one. Possible rules could be "the initiator's version prevails" or "the newest version prevails".

The Internet Engineering Task Force (IETF) backed, widely used Session Initiation Protocol (SIP), RFC 3261, is a candidate for the main communication protocol of supernetworked applications. As stated in [4], SIP provides a lightweight toolkit for session signalling between peers, as well as identifying users with SIP Uniform Resource Identifiers (SIP URIs). SIP supports a number of signalling primitives, such as event notification and redirection of connection attempts.

2.2 Plug-and-Play Application Platform Middleware

A mobile middleware solution, *Plug-and-Play Application Platform* (PnPAP), has been designed and implemented for the purpose of providing common services for supernetworked applications. It runs on top of Symbian OS for Nokia Series 60 smartphones. PnPAP comprises of a body of approximately 20,000 lines of C++ code.

The PnPAP middleware, first proposed in [5], was initially designed for the automatic selection of optimal P2P protocols and physical connectivities, exhibiting a relatively monolithic architecture. This section describes the status of PnPAP as it was just before augmenting its functionality with the solutions that are described later in sections 4 to 7.

The heart of the middleware is the PnPAP server, a central server process to which all the user-level applications automatically connect. Applications access PnPAP functionalities through the PnPAP Application Programming Interface (API). A client stub library, linked with the PnPAP-compliant applications, works as an access mediator between the application process and the PnPAP server: applications call the methods of the client library, which translates them to Symbian client-server calls. These calls are interpreted by the PnPAP server, which carries out the specified task.

Generic interfaces of software modules within PnPAP enable the use of functionalities on a higher level of abstraction. Let us consider the scenario that an application requests PnPAP to send a human-

readable instant message to a remote peer, using any combination of a P2P protocol and a physical connectivity that PnPAP finds suitable for the purpose. Of course, the currently selected protocol and connectivity will probably be used, but the application would not notice even if PnPAP dynamically decided to pick a new protocol or connectivity. Having an active P2P protocol (e.g. DC++) and physical connectivity (e.g. GPRS), PnPAP calls the protocol module's method that sends the instant message. As this interface is the same for all P2P protocols, PnPAP can call the methods in the same way regardless of which protocol implementation it selected as the best for the purpose.

Figure 1 illustrates the basic setting, PnPAP with several different connectivities and P2P protocols to orchestrate for the benefit of the user's applications on the top. There are two terminals in the picture, and a session (supersession) is shown between some of their application modules.

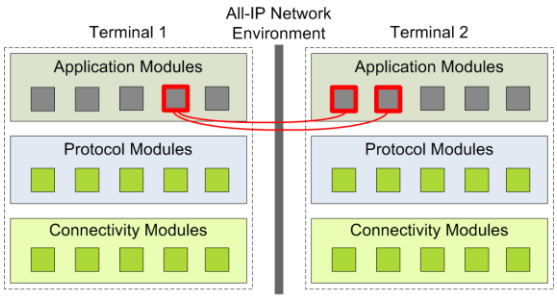

Figure 1 Modular interaction model of PnPAP.

Applications can query PnPAP whether a given functionality is present in the currently selected protocol, because all protocols do not support all communication primitives; the absence of peer group functionality in some P2P protocols is an example. The protocol module's interface seen by PnPAP supports querying the supported functionalities of the protocol represented by that module.

Sometimes a P2P protocol required for a specific purpose is unavailable on the terminal: it might happen that none of the existing protocol modules support a given functionality requested by the application. This is why *dynamic installation of P2P protocols* was implemented. PnPAP detects the application's need for the retrieval of a specific protocol implementation, and is able to retrieve the required protocol module from the network dynamically. The motivation for dynamic installation is that different P2P protocols are incompatible, and a given node might not have the same protocols installed as the other parties of the communication; with dynamic installation the nodes will have a common protocol to use. Deciding what protocol to install will of course require some signalling between the nodes; this can be accomplished, for example, with the PnPAP-to-PnPAP signalling network that is described later in this section.

However, the solution relies on the existence of a central server that provides protocol modules for the PnPAP nodes to download. PnPAP does not browse P2P file-sharing networks to find the missing protocol module from there, although that kind of solution would be clearly more flexible and less centralized than a server.

Traditional P2P applications usually have a single P2P protocol to use, and are thus restricted to use only the network associated with that protocol. As stated above, a PnPAP node has access to

multiple P2P networks, depending on the protocol modules that are available. As a corollary of this, PnPAP nodes see an overlay network that is composed of the networks accessible by individual protocols [5]. The set of peers that a specific PnPAP node can contact ($S_{overlay}$) is the union of the sets of peers that can be contacted using each individual protocol. If each protocol i of the N_{prot} protocols available can be used to contact a peer-set $S_{prot}(i)$, then

$$S_{overlay} = S_{prot}(1) \cup S_{prot}(2) \cup \cdots \cup S_{prot}(N_{prot}). \quad (2)$$

Thus, an application on top of PnPAP may be able to make connections with a far greater number of peers than a traditional P2P program. This also increases the probability to find a resource that is not available in all networks, e.g. a rare shared file. The PnPAP overlay solution is also significantly different from just having multiple different single-protocol applications in the terminal: with PnPAP, several P2P networks are accessible from one application; the centralized nature of PnPAP also helps save the resources of the mobile device [6].

While the aforementioned PnPAP overlay network enables the user to access several P2P networks from a single application, it could still happen that two PnPAP nodes do not have access to a common P2P network. Could this mean that those nodes cannot communicate with each other? Even dynamic installation of new P2P protocols would not help here, because the peers would not have knowledge of each other's existence in the first place.

However, there is a straightforward solution to the problem. All PnPAP nodes together form another logical network, where any node can contact any other node regardless of their installed P2P protocols. (It should also be noted that this network contains only PnPAP nodes, while the peer-set $S_{overlay}$ of an overlay network seen by a given PnPAP node contains also non-PnPAP nodes that happen to be using the same P2P protocols). The PnPAP-to-PnPAP network has been implemented on top of SIP; every PnPAP node runs a SIP stack. Every PnPAP node can be unambiguously identified using a SIP URI even when no identification schemes from other P2P protocols are present, and the SIP URI is used as the recipient address in PnPAP-to-PnPAP messaging. All the node-intercommunication messages are carried in bodies of SIP MESSAGE packets. The data records of the messages are encoded using a simple type-length-value scheme. The SIP messages are routed through the home network's SIP server, where a node registers itself when coming online. Over the PnPAP-to-PnPAP communication channel, the nodes are able to, among other things, share their context information within a peer group.

The work for implementing PnPAP has produced an important reusable software component: a lightweight SIP stack for Symbian OS. The stack provides a sub-set of the functionality described in RFC 3261. The stack was created for PnPAP, because there was no existing client-side Symbian SIP stack reliable and flexible enough to be used as part of our system. The SIP server used in our communication architecture was the open-source SIP Express Router (SER).

3 Problem Statement

3.1. Need for Enhanced Connectivity Management

Wireless networks are converging into a ubiquitous, seamlessly accessible data-transmission network [7]; however, this will not happen by itself, as the heterogeneity of mobile networks and services will

raise several certainly non-trivial issues to solve. Modern mobile terminals, such as smartphones, are equipped with multiple network connectivities that operate on several radio-frequency bands and have bitrates varying by several ten-folds, different requirements for external network infrastructure, and different operational ranges, among other things.

Despite the advances in radio-based communication technologies, the typical mobile terminal still has relatively restricted capabilities with regard to networking. Even if the mobile terminal is equipped with an interface to a high-speed connectivity, for example WLAN, the network is not necessarily available in all locations. When the user moves in and out of the coverage areas of different networks, the terminal should select the best interfaces to use without bothering the user.

Versatile physical connectivities are just the top of the iceberg, as the communicating end-systems feature also full-fledged IP stacks, different transport protocols such as TCP and UDP, and a plethora of standardized or proprietary application-level protocols ranging from simple messaging protocols to real-time multimedia. Several concurrently active system-level processes and end-user applications pose versatile requirements for the management of these entities. One noteworthy challenge will be the anticipated emergence of IP version 6 (IPv6). During the transition phase, IPv4 and IPv6 will be used concurrently and transition-aiding mechanisms are applied [8].

There has been academic research on different aspects of automated connectivity management. In the following paragraphs, a summary of some related publications is provided.

Vertical handoff, i.e. the handoff of connection between heterogeneous network interfaces, is extensively discussed in [9]. The work mainly concentrates on the analysis of handoff systems, mobility management, and optimization of the actual handoff event in a multi-network environment, not mechanisms for selecting the best connectivity to use.

A handoff decision system, employing user-adjustable policies for selecting the wireless network to use, is presented in [10]. The work includes the design of a performance-reporting scheme that estimates current network conditions; an agent gathers bandwidth-usage data at a base station and transmits it to the nodes in the coverage area. Goals of the system are to balance the bandwidth load across several networks that have comparable performance, and to minimize user interaction. A multi-parameter network cost function (a weighted sum of parameters normalized with logarithm functions) is periodically calculated.

In [11], partially by the same authors as [10], a similar system for triggering the use of heterogeneous wireless interfaces has been implemented. The work is built around the concept of physical overlay networks, i.e. a hierarchical structure of overlapping room-size, building-size, and wide-area networks; the more bandwidth, the less coverage. Availability of networks is detected by observing periodic beacon messages sent by base stations.

In [12], a decision model is proposed for determining both the target network and the moment to perform the vertical handoff. Again, user preferences and network properties are used as input for an algorithm. The authors claim that their weighted-sum score function for ranking network interfaces might be more sophisticated than the preliminary system presented in [10]; parameters are normalized without logarithms so that zero-valued parameters do not cause problems.

A generic function for selecting between different wireless networks is presented in [13]. A quality value for each network interface is calculated using a first-degree polynomial function, based on

several factors. It is pointed out that some factors, such as security, might not have a numerical representation, thus their applicability in the formula may be restricted.

The solution in [14] features an architecture that enables automatic connectivity selection with the network's assistance. New network-layer "assistant" nodes are defined, along with new functionality for some of the existing nodes in the current systems. The network informs the user's terminal about the best connectivity for the requested service, based on parameters such as QoS and current network conditions.

Intelligent selection of network interfaces is elaborated in [15]. Although not implemented, the proposed algorithms are designed to deduce the optimal connectivity at any time.

There has also been research on mobile connectivity management with cross-layer optimization. The work in [16] names two enabling layers for seamless mobile networking. The first one is the selection between different packet delivery methods such as regular IPv4 or mobile IPv4. The second one is the selection between different physical interfaces.

A more clearly cross-layer oriented system for network interface selection is presented in [17]. Decisions are made according to user preferences and information on the networks currently available; this information is gathered from link layer, IP layer, the service providers, and the applications or users that are being served. The solution features rule policies that can be created by the user based on different metrics. The system also supports simultaneous multi-access, i.e. traffic flows can be divided between different interfaces in parallel.

Another cross-layer approach is presented in [18], building on the concept of overlay networks as in [11]. The coverage of the overlay is the same as the coverages of the individual networks combined; its performance at a given location is that of the best network in range. Cross-layer optimization is used in several parts of the system: for example, transport-layer information is exploited to guide link-level retransmission and application-level content adaptation. Mobility issues such as handoff are handled by dedicated software agents, which reside within the network infrastructure but existing clients and servers do not need to be modified.

An issue closely related to switching of connectivities is IP mobility management, i.e. the policies that dictate how the terminal manages network handoffs in an IP-based wireless environment [19]. When switching between networks, it is desirable to take advantage of mobility support in the IP layer. Standardized solutions such as Mobile IP (MIP) and Host Identity Protocol (HIP) exist. MIP maintains a mapping between the changing care-of address and the non-changing home address of a node; a so-called home agent is aware of the node's current physical location, i.e. the care-of address. All protocol instances above the IP layer see only the home address [10]. Accordingly, many of the aforementioned connectivity management systems, for example [17] and [18], utilize MIP in the networking layer for mobility management.

HIP takes a slightly different approach to mobility. In HIP-based systems, the two basic functions of a traditional IP address are separated: host ID and host locator. The introduction of the new Host Identity namespace enables a mobile host to retain its unchanging identity in any inter-network handoff, while its IP address changes to reflect the new location. When using HIP, nodes can keep their ongoing TCP connections even if a vertical handoff occurs.

Despite the various connectivity management solutions discussed above, there exists no system capable of selecting combinations of arbitrary networking resources in a formal manner. The existing solutions have not provided a fully consistent and extensible model of the communicating end systems. Neither is there a system that uses dynamically upgradeable, state-machine controlled rule-bases for making connectivity-related decisions. A cross-layer framework accomplishing these tasks will be called *Holistic Connectivity* (HCon). The name is derived from one of the cornerstones of application supernetworking: holistic connectivity management. It is expected that the system employs an intelligent state-machine based decision engine, since applicable research results and software components are available from the ISG research group of the University of Oulu.

The research question for connectivity management is: *How to implement an efficient HCon system with state-machine based decision intelligence, accessible as a mobile middleware module?*

An answer to this question is provided in sections 4 and 5.

3.2. Need for Enhanced Session Management

The current paradigm for the start-up of application-to-application sessions over a wireless network involves two or more instances of the same application, or instances of different applications understanding a common protocol, on different hosts in the network. (In this section, the word "application" means applications whose usage involves user-to-user sessions over the network). Session start-up is typically initiated by sending an invitation message over the network to the remote peer's terminal, which is listening to a specific port and is able to initiate a session using its local instance of the application. SIP is a widely used protocol for this kind of session signalling.

Now, let us examine the situation of a peer group where the peers have a heterogeneous base of applications installed, i.e. not many terminals have the same applications. As the different applications do not share a common language, nodes of the network are isolated in "islands" of heterogeneous application bases. Nodes with the same application installed are on the same island, but the others are not. Although the PnPAP-style approach of combining several protocols did remove barriers for the concurrent use of several general-purpose P2P networks, a similar approach obviously would not help with strongly application-specific protocols required by user-to-user games, for example.

Formally put, the isolated islands are not able to intercommunicate using the application (or, more precisely, the application protocol) A_i. Some terminals indeed have several applications, but in order to initiate a session between any two nodes (terminal devices) D_a and D_b, the terminal D_b must already have installed the same application A_i as the terminal D_a has. Otherwise establishing a mutual session using the application A_i will fail.

Let us define the "initiator" as the peer who tries to initiate a session with a remote peer, and the "receiver" as the mentioned remote peer who may accept or decline the proposed session. Furthermore, let there be $N_D > 1$ terminal devices in a peer group. Also, let d_i denote the number of terminals that have installed the application A_i. Then the probability that a given initiator with his installed A_i can initiate an A_i-based session with a given intended receiver is

$$P_s = \frac{d_i - 1}{N_D - 1}. \tag{3}$$

If N_D is large or modestly large, and the application is not installed in the majority of the terminals in the peer group (i.e. d_i is small), then the probability P_s is very low ($P_s << 1.0$). In other words, only few terminals – that is, few users – are able to initiate mutual application sessions in such a situation. Of course, there might be some applications that are popular, having been installed on almost every terminal of the peer group: most peers can initiate sessions using those applications. Still, any applications that have little coverage in the peer group would not benefit from this; they cannot be used to create sessions with most peers. This clearly diminishes the usage value experienced by a user who has installed, probably even paid for, a rare application. A simple example could be a chess game for two users to play over the network: how can you spontaneously play with someone, if no one else has installed the same game application?

The situation seems even more discouraging if the initiator wants to begin a multisession with the receiver, and the said multisession requires $n > 1$ applications ($A_1, A_2, \ldots, A_n$), all of which must be installed on the receiver's terminal. The probability to be able to initiate such a multisession is

$$P_{ms} = \frac{d_1 - 1}{N_D - 1} \cdot \frac{d_2 - 1}{N_D - 1} \cdot \ldots \cdot \frac{d_n - 1}{N_D - 1}. \quad (4)$$

If the multiplied values are each significantly less than 1.0, which is likely to be the case, then P_{ms} is very low: the initiator's attempts to start the multisession will very probably fail. It might even happen that the islands of the n required applications do not overlap at all within the peer group (resulting in $P_{ms} = 0$ for all nodes of the group).

The research question for session management is*: How to implement a session management solution that overcomes the problem of isolated islands, in a way that is both efficient and usable?* Naturally, the solution must also be able to perform other session management primitives besides the start-up of sessions, including the graceful termination of sessions and passing session parameters.

The problem of session management is tackled in sections 6 and 7.

4 Design of the Connectivity Management Solution

4.1. Objectives

As a response to the stated problem, the objective of the Holistic Connectivity (HCon) framework is to manage the wireless networking resources in a way that is optimal – the applications should be always provided the best combination of resources. *Holisticity* means the system's ability to make cross-layer connectivity decisions based on rich information that may involve, for example, network signal strengths, application-level semantics, and user preferences. The HCon framework must cover the practices required for pursuing this goal, although it is impossible to guarantee that decisions made by HCon are always optimal from the user's viewpoint. HCon has first been discussed in [20].

Another objective for the HCon framework's design is to introduce rigorous definitions for the concepts that are used for discussing the problem field. Those devoted to the optimization of connectivity management can discuss the topic within a solid frame of reference. A consistent view to the resources that are controlled may also facilitate the formal verification of networking systems.

HCon's approach to connectivity management must not mandate any specific set of networking resources to be controlled by the system. It might be tempting to explicitly define a list of cross-layer resources to use, as many existing connectivity management designs do [16] [17]. In sharp contrast, the HCon design should avoid the kind of logic that restricts functionality in the long run. It must allow introducing new resource types without the need to break its definition or to apply awkward, one-off patches. Otherwise, emerging networking concepts will cause problems in the future.

To determine what is the best networking behavior in a given situation, the applications or their users must be able to state their preferences. Good networking performance might often be synonymous with high data throughput, low transmission delay, or low packet loss. However, there are more factors to consider: for instance, the monetary cost of a service, the QoS of concurrent media streams, the choice of media codecs, and the security features in different protocols.

To allow for maximal adaptability to different conditions, including changing user preferences, HCon must employ dynamically upgradeable rule-bases. For the same goal, HCon must separate the decision logic of the rule-bases from the component that carries out the decisions (switches the networking resources). There must be no unnecessary couplings between the two components (as strong coupling makes the rule-bases harder to upgrade) and they must communicate over a well-defined interface and continue operation even if rule-bases change when applications are running.

Thus, an intelligent state machine (SM), containing the rule-base to use, is used as the brains of an HCon system. Any parameters for HCon decisions are the input for the SM, and its output is a connectivity decision when one is requested. State machines must be dynamically changeable. If the SM component is sufficiently decoupled from other parts of HCon, it also fulfils the requirement of separating decision logic from the "workhorse" component. SMs must be able to incorporate operational logic from various connectivity schemes; examples include algorithms from [15].

Mobility-supporting protocols such as the already mentioned MIP and HIP would probably provide significant added value to HCon. Upon a vertical handoff operation, it is desirable to have IP-level support for mobility in order to keep active connections alive and node addressing unbroken. Unfortunately, these technologies were deemed unfeasible to implement in reasonable time on our selected target platform. Thus, they were omitted from the first-phase HCon design, which mainly concentrates on the selection of local networking resources and not on IP mobility.

Proof-of-concept HCon is part of PnPAP middleware. Choosing SIP as the inter-PnPAP communication mediator alleviates the effects of lacking IP mobility. Admittedly, SIP cannot rehome an ongoing TCP connection. SIP however provides a certain degree of mobility support: a SIP URI identifies a user, and the changing IP address is the host's current location in the Internet.

4.2. Structure of HCon

The structure of a software-built system realizing HCon functionality is depicted in Figure 2. The user's numerous applications sit on top of the middleware that takes care of their networking needs. HCon is shown as part of PnPAP because that is how proof-of-concept HCon was implemented; a real-world HCon could be a stand-alone middleware solution, or part of the operating system.

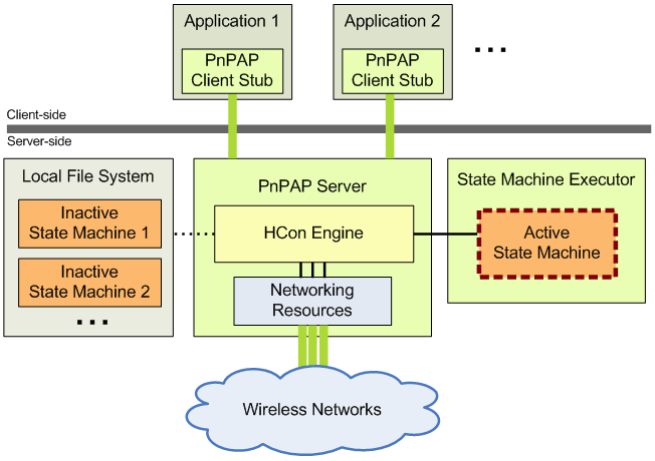

Figure 2 Structure of HCon.

The component that executes networking-related operations such as activating and de-activating protocols, but does not make the decisions when to do that, is called the *HCon engine*. The HCon engine is integrated into the PnPAP server. The HCon engine owns the protocol instances and most other networking resources; thus, the resources are bound to the PnPAP server process.

The *State Machine Executor* (SME) is responsible for running the state machines and can also change the SM to a new one if requested to do so. The SME process is separate from the PnPAP server in order to increase modularity. An interface between the HCon engine and the SME allows the two to exchange messages. Multiple different state machine descriptions, containing different rule-bases, can be stored in the local file system so that they can be taken into use when needed.

4.3. Formal View to the System under Control

In order to obtain a formal view over the system to be controlled, Holistic Connectivity treats the terminal's networking resources as a finite set of *entities*. An entity is an individually selectable, hardware- or software-based piece of technology for a specific networking-related task; a set of mutually interchangeable entities is an *entity class*. Examples of entity classes include "physical connectivities", "P2P protocols", "audio codecs", and "operator-provided service classes". Example entities in the class of "P2P protocols" could be FastTrack, DC++, and Gnutella.

An *entity stack* is the complete set of entities, which are selectable within the terminal. The entities in a stack are grouped by their entity classes, which are on top of each other; hence the name entity stack [20]. By breaking down the system to distinct functional components and managing them as a hierarchical stack, HCon attains an adequate level of controllability over the system. As was required, the design of HCon does not force the system to include any specific set of entity classes: HCon's entity-based approach allows for the inclusion of any new entity class.

The entity classes of an entity stack can be enumerated as $C_1 \ldots C_N$, where N is the total number of entity classes. In this notation, C_1 denotes the lowest-level entity class of the stack, typically the selectable physical connectivities, and C_N denotes the highest-level entity class. The number of entities within class C_i is n_i. Entity j from entity class C_i is referred to as $C_i(j)$ $(1 \leq j \leq n_i)$.

The notion of entities within a class being mutually interchangeable must be taken with a grain of salt. There are naturally some dependencies between the entities of different entity classes. For example, let us suppose that a system has entity classes for transport protocols and IP protocol stacks; selecting IPv6-based version of TCP as the transport protocol to go together with the IPv4 protocol in the network-layer would not make sense.

One might observe a similarity between the entity classes and the widely deployed network layer models. However, there might be several entity classes in one layer or one entity class could encompass several layers. HCon enriches the OSI and TCP/IP reference models by providing means for vertical control and by introducing the entity classes that can be added or removed.

In order to allow the data to flow from the applications to the physical radio network, a path for the bits must be established through the entity stack. An N-tuple containing one entity from each of the N entity classes is a *top-down connectivity policy* (TDCP). A TDCP represents the vertical path the bits take through the stack. When the HCon engine requests the state machine to select the best networking resources for the current situation, a TDCP is what the SM gives as its output. The details of the communication between HCon and the state machine are explained in section 4.4. Two example TDCPs through an example entity stack (N = 4 entity classes) are presented in Figure 3.

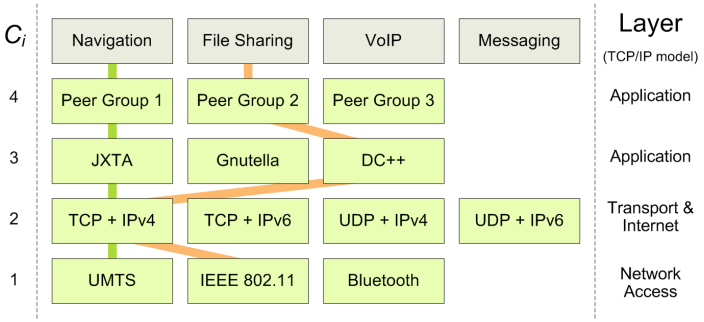

Figure 3 An example entity stack with two example TDCPs.

4.4. Decision Control using State Machines

The SME follows a state-machine execution architecture called SteMach. The architecture has two functional components: an event queue, and a storage module containing the current active SM.

All communication to and from the SME is asynchronous. However, the order of the arrivals of events is preserved in the event queue so the SME always reacts to asynchronous events in the correct order. The incoming events are safely stored in the queue if not enough main memory is available for handling them; thus the SME is able to recover from a temporary low-memory condition in the terminal. Only incoming events are queued. All outgoing events (in HCon's case, TDCP decisions) are simply sent out of the SME and forgotten.

Networking resources have logical states: for example, different observable signal levels of a physical interface, the current activities of a communication protocol, any context information, or the current user preferences. All these pieces of information are stored in the SM's internal states and treated as parameters for the TDCP selections. The current state of the SM represents the last known

combination of parameters. State information is the foundation of decision rules. A rule "if *a* and *b* then *c*" can be represented when *a* is an active state, *b* is an event, and *c* is a transition to a new state.

The state machine variety applied in HCon allows the SM to be in multiple concurrent states. This is beneficial for storing multiple pieces of state information about different parameters. Using a traditional single-active-state SM format would obviously explode the number of states required and would render the SM inconveniently large and also difficult to comprehend for its developers.

Complexity is also managed by applying nested states, which are essentially a special case of concurrent states. A given type of context information – for which the SM contains a number of states – can be divided into several sub-contexts, each of which is represented by a sub-SM. An example on how parameters are combined to make TDCP decisions is presented in section 4.5.

SMs may contain additional code sections that help make the TDCP decisions based on the current states. These decision scripts are defined to be executed at the entry or exit of a given state. They are often associated with a state that absorbs decision-request events from the HCon engine.

When the HCon engine sends a serialized SM specification to the SME, the SME loads the SM to a storage module and starts it from its initial state. The Notation 3 (N3) description language is the SM specifications' serialization format. The format is capable of capturing the structure of arbitrarily complex state-hierarchies and the allowed transitions. SMs can be created using a visual editor.

The HCon system is adjusted to comply with the user's expectations by employing new SMs. SMs must be created by the operator or other party with the skills and business interests to provide a good networking experience to the users. There could be a selection of ready-made SMs for the user's convenience. Users can also ignore the fact that their terminals are controlled by SMs. SM upgrades could even be issued over-the-air (OTA), with little or no user interaction.

To enable dynamic upgrading of HCon rule-bases, SMs are replaceable on-the-fly; this does not require any kind of re-compiling of the HCon engine or the SME. A replacement could occur if, for example, the operator issues an OTA upgrade. Whatever the reason and mechanism for the delivery of a new SM, the HCon engine notifies the SME about the replacement so the SME can gracefully terminate the current SM and start executing the new SM. No applications need to be terminated.

The HCon engine, as part of PnPAP, accesses the state machine services by calling the methods of the Property Service Interface 2 (PSII). PSII is a generic interface for exchanging information about so-called properties between two processes. A stub library, linked with PnPAP, implements the interface and takes care of the low-level IPC over a local-host UDP socket connection.

A property is identified with a name, for example "ev_WlanUp". A property's value can be any string of bytes. All messaging (parameters, decision requests, and decision responses) between the HCon engine and the SME is defined in terms of sending property values over PSII.

The performance of the Python-based SME has been evaluated in a Linux environment. The test system consisted of two virtual Linux nodes running in User-Mode Linux sandboxes with 128MB of memory. The sandboxes were running on an AMD Athlon XP 2800+ host with Linux 2.6.11-6mdksmp. The IP router SM, run on the host system, routed ICMP pings from one virtual node to the other. The transmission interval parameter was varied from 0.01 to 0.2 seconds and the packet size was 1428 bytes. With packet intervals over 0.06 s, round-trip times stayed below 100 ms and packet loss

around 5%. Performance started to drop clearly when packet interval approached 0.04 s. With an interval of 0.03 s, the round-trip time was about 300 ms and packet loss was about 80%.

4.5. Simple State Machine Example

An example state machine is presented in Figure 4. The SM switches physical connectivities (3G, WLAN) for VoIP application usage, according to the following kind of user-defined policy. The first priority is to maintain a high-quality connection at all times; the second priority is to do this with a low monetary cost. The decisions depend on the availability of the different access networks.

In the model, 3G is always high-quality. WLAN can be low- or high-quality, depending on link-layer information. The SM tries to get out of a low-quality, cheap WLAN state, regardless of costs. It also tries to get out of a high-quality, expensive 3G state into a high-quality, cheap WLAN state.

The SM is composed of multiple sub-SMs. The connectivity-specific SMs "3G" and "WLAN" do not know about each other. They are synchronized by the "VOIP" SM that communicates with them by using events; this modularity reduces complexity and enhances the system's extensibility.

The failure transitions are shown just to clarify that the SM can also be used for managing exceptions; the actual recovery mechanisms are not provided. It is possible that if a failure occurs, a counting timer can be started to trigger re-try *n* times. Failures are handled with the same primitives (states, transitions, and events) as other situations. Moreover, realized failures can be stored in the SM parameters; the failure is part of the device's context information, affecting the later decisions.

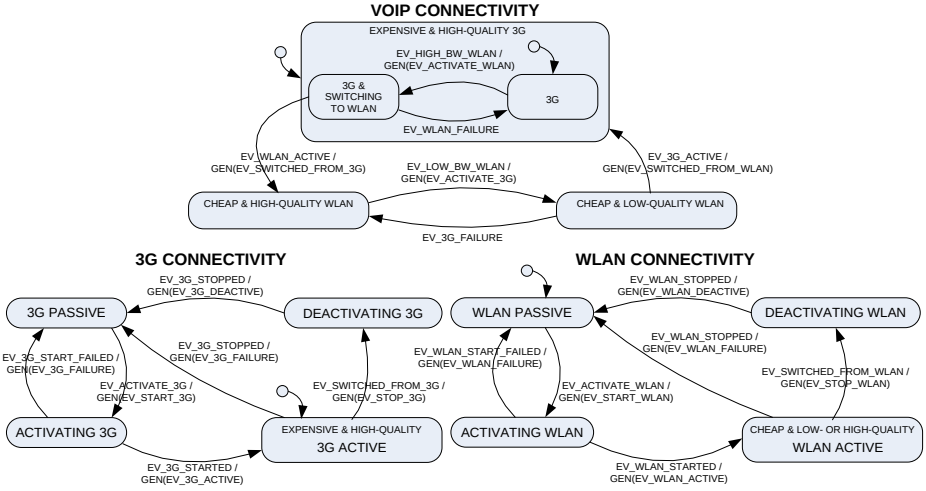

Figure 4 An example state machine for switching physical connectivities.

To simplify the SM, it is supposed that the handoff of the traffic from one access network to another does not require stateful handling. It should also be noted that when switching from low-quality WLAN to 3G, the SM enters a loop, where it repeatedly tries to activate 3G – without delay between the tries – until 3G activates. This is feasible, if we suppose that 3G is widely available and failures with 3G are short-lived; of course, a different re-try policy could be implemented.

Cross-layer information is utilized in this SM. The events concerning the detected quality of WLAN are based on information gathered at the link level, while all decisions are made in a unified manner by the SM that resides in the application layer (more specifically, in the middleware).

Regarding the potential of multi-entity-class operations for this VoIP-call-optimizing SM, a possible functionality that could be added is the dynamic switching of IP protocol versions. If the user enters a network where IPv6 support is detected, the device's IPv6 stack could be taken into use in order to derive benefit from IPv6's (future) packet prioritization support for real-time traffic.

5 Implementation of the Connectivity Management Solution

5.1. Implementation Environment

The target platform for the mobile software were state-of-the-art Nokia Series 60 smartphones (models 6680, 6630) running Symbian OS 2nd Edition, FP2. The Octopus testbed network in the city of Oulu provided us mobile subscriptions with fixed public IP addresses, freeing the us from the concerns of Network Address Translation (NAT) traversal and expiration of address lease periods.

Physically, the binary implementations of protocols and connectivity plugins for PnPAP are Symbian polymorphic Dynamically Linked Libraries (polymorphic DLLs) with entity-class-specific interfaces. Modules, which correspond to the different entities, are loaded and unloaded at run-time.

5.2. Implementation

What entities a given HCon implementation can control, depends on what entities have been implemented as PnPAP-compatible software modules. In our implementation, there are two entity classes in the system (N = 2), namely, physical connectivities and P2P protocols; of course, a real-world HCon implementation would be more complex, but these two entity classes are the most essential from the viewpoint of our P2P system and demonstrate clearly the operation of HCon. Each of the entity classes contains two entities. The physical connectivities available are GPRS and Bluetooth. The P2P protocols available are DC++ and the Very Simple Protocol (VSP). VSP is a proprietary protocol implemented only for demonstrating the functionalities of PnPAP.

The prototype SM takes 35 KB in the N3 format. It is important to note that this SM is different from the example SM that was presented in section 4.5.

The prototype SM supports the following context types: user's speed (bitrate) preference; user's monetary cost preference; application's latency preference; application's bandwidth preference; application's type (streaming, P2P, or chat messages); application's preference for transport type (stateful or stateless); application's battery usage. The HCon engine initializes the SM with the relevant context parameters, which could indicate, for example, that the application is P2P-oriented.

5.3. Delay Measurements

Delay times related to the SM-based decision-making procedure were measured when we evaluated the implemented HCon system. A measurement series would consist of 20 measurement iterations. Each of the iterations would consist of the following steps.

1. The SME is started manually so that it is ready to accept incoming requests.

2. The NaviP2P application (i.e., navigation with peer group members on the map) is started manually. Implicitly this causes also PnPAP to start to run and set up a socket connection to the SME. The HCon engine sends a SM-initialization message (context parameters) to the SME.

3. SIP username and password for the PnPAP network are entered manually in NaviP2P. Then the HCon engine sends one decision request to the SME, and, after a while, receives the response.

4. NaviP2P is exited and also the SME is shut down. The terminal is rebooted, and the system is ready for the next measurement iteration.

Each iteration produces five timestamps. Delays for different actions can be calculated from the time intervals between the timestamps recorded in the logs. The following values can be calculated:

- $T_{sm\text{-}init}$ = delay of the SM initialization phase;
- $T_{conn\text{-}decision}$ = delay between the decision request and the connectivity-decision response;
- $T_{prot\text{-}decision}$ = delay between the decision request and the protocol-decision response.

Before beginning, there was one factor to consider regarding the SME's performance: Python programs can be human-readable scripts or compiled byte-code. Compiled byte-code loads faster than the source script, because the non-compiled script must be interpreted into byte-code each time it is run; there is however no difference in the actual execution speed between interpreted and compiled code, because the compiled file is just a re-usable result of the same interpretation process, as stated in the documentation of Python. We performed two measurement series: one with the non-compiled Python codes and one with the compiled codes. The codes were otherwise identical.

Hypothetical delays were stated before carrying out the measurements in order to enable comparison of the actual performance and our assumptions. The hypothetical average delays were as follows: $T_{sm\text{-}init}$ = 0.50 s; $T_{conn\text{-}decision}$ = 0.20 s; $T_{prot\text{-}decision}$ = 0.40 s (i.e. there would be 0.20 s between receiving the connectivity-decision and the protocol-decision responses: 0.20 s + 0.20 s = 0.40 s).

The measured delay values for the non-compiled (interpreted) and compiled SME codes are presented in Figure 5 and Figure 6, respectively.

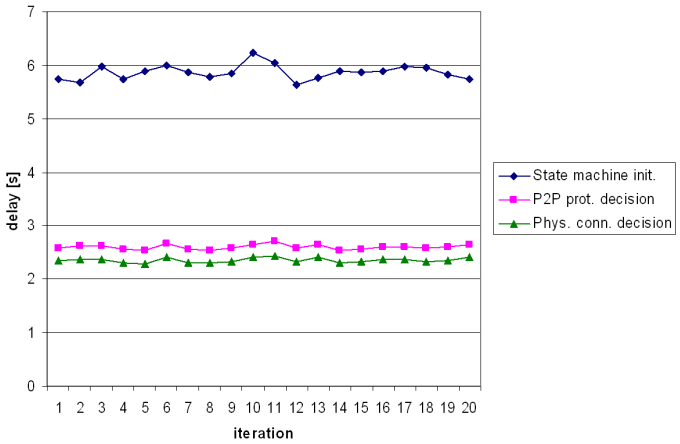

Figure 5 HCon delay measurements: non-compiled Python code.

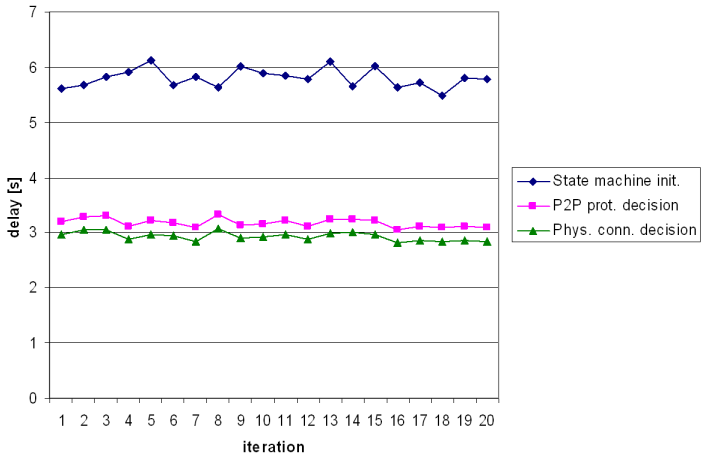

Figure 6 HCon delay measurements: compiled Python code.

With the non-compiled codes, the average measured $T_{sm\text{-}init}$ was 5.87 s (std. dev.: 0.14 s). The average measured $T_{conn\ decision}$ was 2.36 s (std. dev.: 0.05 s) and the average $T_{prot\ decision}$ was 2.60 s (std. dev.: 0.05 s). Between receiving the connectivity-decision and the protocol-decision responses, the average time interval was 0.24 s.

With the compiled codes, the average measured $T_{sm\text{-}init}$ was 5.80 s (std. dev.: 0.17 s). The average $T_{conn\ decision}$ was 2.93 s (std. dev.: 0.08 s) and $T_{prot\ decision}$ was 3.18 s (std. dev.: 0.08 s).

An anomaly is detected when one compares the two measurement series: for some reason, the compiled byte-code seems to introduce more delay than the non-compiled scripts. This was not expected at all. The compiled codes should yield a shorter total delay (loading and execution delay components combined), as they load faster than interpreted code. There is a speculative explanation: we compiled the codes on a desktop computer, and although Python byte-code should be cross-platform compatible at least between instances of the same Python version, there could be some differences between the two environments, and this would affect some component of the total delay.

In both measurement series, the measured decision-making delays were clearly higher than the hypothetical values, but not excessively so. The measured level of performance is satisfying for a proof-of-concept solution. The implementations of the SME or the PSII library should be further optimized. The measured delays of the HCon implementation are high, because there is a great amount of overhead from the SME's complexity.

Of course, if HCon is to be applied to mobile networking in the real world, the system will probably be implemented in native binary code, preferably as one single process in order to eliminate also IPC-related delays. More performance metrics should be evaluated with this more capable implementation. It is clear from the measured delays that the current implementation is not ready for large-scale deployment. The purpose of the evaluation was, however, only to create a functional architecture for decision-making and not to optimize the performance.

Finally it must be noted that the evaluated HCon system lacked a reasonably large variety of physical connectivities and P2P protocols. This obviously made the TDCP decisions easier; the

evaluation was not carried out with a system of realistic complexity. The evaluation results however provide suggestive information about the performance of a SM-based connectivity management system on a state-of-the-art deployment platform.

In order to estimate the potential real-life impact of TDCP decision on specific networking metrics, we can consider an example case where a P2P protocol supporting only point-to-point downloads is changed to one that is able to download from multiple sources, removing the bottleneck of remote-node uplink capacity; this would result in increased download speed. In another example case, the monetary cost of file transfer is decreased when HCon detects that two terminals are close to each other and changes the connectivity from a cellular network to Bluetooth.

It would be interesting to compare other connectivity-management solutions in the literature to our results. While no directly comparable results to our entity-class model were found, we provide a brief high-level comparison to selected approaches that are based on the "Always Best Connected" ideology. In [21], access network selection is modelled as a variant of the bin packing problem and specific algorithms for it are evaluated with simulations; while the algorithmic base is different from ours, there is an architectural similarity, the intention to deploy connectivity management as part of a mobile middleware. A modular system, with selection procedures specified in pseudo-code, is presented; some differences to our approach are the specified level of detail for input information and the design of the components used for parameter-gathering and decision-making. In [22], an object-oriented framework for encapsulating connectivity management is proposed, along with a problem formulation based on the knapsack problem, reminiscent of the formulation in [21]. The architecture in [22] is not completely unlike ours, as it features a freely adaptable model for context parameters, and state machines are used for modelling application and network behaviour. Decision-making itself, however, is not provided by state machines. The overviewed solutions, despite some cross-layer considerations, have no emphasis on problems such as P2P protocol selection.

6 Design of the Session Management Solution

6.1. Connecting the Isolated Application Islands

As indicated in the problem statement, a key feature of the session management solution is the initiator's ability to start sessions over the network with any other user, using his installed application A_i. This must happen in such a way that the receiver does not need to have the same application A_i on his terminal at the moment when the session initiation is attempted. The probability (P_s) to be able to initiate A_i-sessions with a specific peer should be near 1.0, provided that the session management solution is deployed on all terminals of the current peer group.

Our session management solution overcomes the problem of isolated application islands by providing a means to dynamically install and launch the application on the receiver's terminal if that application is not yet installed. The solution also liberates users from thinking about these installations before the need for installation is detected.

From the initiator's point of view, a session can be proposed *without knowing* does the receiver already have the application or not. Of course, the initiator might also know about the application-installation status of the receiver (because, for example, they have talked with each other about their applications, or because the status is automatically transferred over the communication channel

between middleware instances), but the solution enables him to try initiating a session *even if* no such information is available. When the receiver gets the session proposal (but does not have the application yet), he is proactively informed about the ability to install the application and can accept or deny the installation. If the receiver accepts, the installation package is retrieved from the network without his effort; then the application is installed and a session with the initiator is instantly activated. As a consequence of all this, any user can try to initiate sessions with any member of his peer group, using any of his applications that support user-to-user sessions.

The solution overcomes the problem of isolated application islands, as any application A_i can be used to initiate a session with any user; there is no more a dependency to the receiver's already installed set of applications. The means to install and launch dynamically the application on the receiver terminals can re-use existing system functionalities: for installation, the generic application installer can be used, and the installed software can be launched by the middleware as easily as any locally installed application, of course with a suitable way to provide the parameters for the session.

From the receiver's point of view, the decision to install the application is made as effortless as possible. The initiator, in turn, has the freedom to propose an application session to any of his friends – without the need to think does the receiver have the application already or not.

Automatic start-up of an application session, immediately after the installation, is a significant part of the functionality of the solution. After all, a session is what the initiator proposed in the first place, and the session is also presented to the receiver as a consequence of accepting the proposal. Correct parameters for the to-be-launched session are automatically passed to the application at the receiver's end, so the session can be correctly established. Had the application been already installed on the receiver's terminal, the parameters could be used to initiate a session right away; of course, the receiver has the power to accept or deny the session also in that case.

It is possible that several users try to initiate sessions simultaneously with the same receiver. This can be handled by a suitable policy: the solution can be configured to reject new session proposals if one is already in progress, or to remember the proposals and provide the receiver the option to "call back" the initiators when the current session is finished.

Naturally, the system should not install anything without the user's consent. The receiving user must be given a clear indication about his ability to accept or deny the installation, about the identity of the initiator peer, and about the consequences of accepting the proposal: accepting will, among other things, result in a session with the initiator. It must also be indicated whether a payment is involved or not. The session proposal indication should be in the form of a visual message on the screen, preferably accompanied with a sound alert – much like an ordinary incoming voice call is indicated by means of audiovisual signals that capture the user's attention. Blocking of incoming session proposals could be implemented based on the receiver's presence status, e.g. "busy".

The exact wording of the message may differ, and the phone screen size is a limiting factor, but the message conveyed to the receiver should be essentially this: "<Peername> is proposing a session using the application <Application>. If you permit installation of the application, a session of <Application> will begin immediately between you and <Peername>. Price of installation: <Price>."

This session management solution has been named Agile Content Push Control (ACPC), and it was first proposed in [3]. The name derives from the fact that the system, in addition to managing sessions, factually performs a user-approved content push when installing the missing software.

Depending on the situation of the receiver, one of three things can happen when an ACPC-based session proposal is sent to him. 1) The receiver has the application already running. After the receiver accepts the session proposal, initiating a session is trivial. The resulting session is homogeneous, if the applications are identical; it is heterogeneous, if the applications are non-identical but can nonetheless communicate. 2) The receiver has the application installed, but not running. After the receiver accepts the session proposal, ACPC must dynamically launch the application and initiate a session. 3) The receiver has not installed the application. After the receiver accepts the session proposal, ACPC must dynamically download the application, launch the installer, and finally launch the application and initiate a session. This is the novel part of the solution. The resulting session is homogeneous, since both parties have identical applications after the installation.

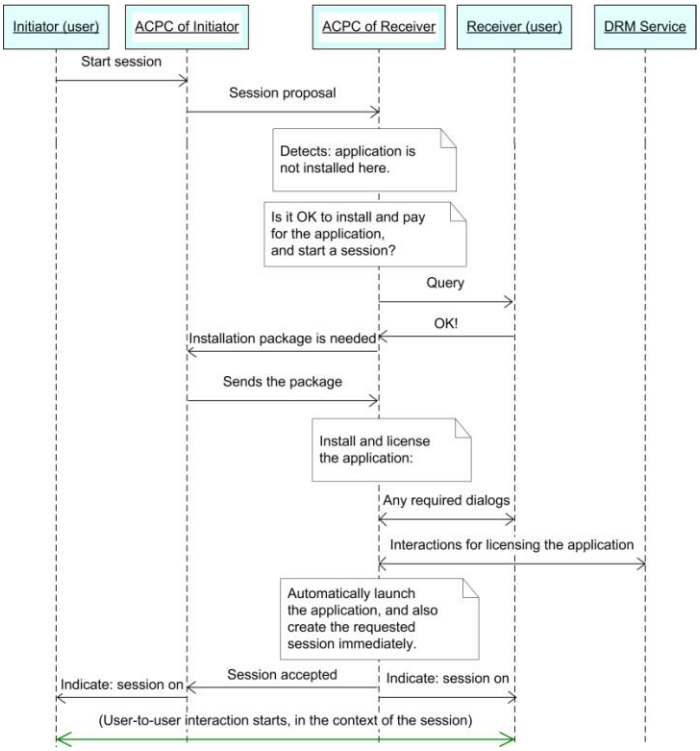

Figure 7 Sequence diagram for ACPC session start-up.

In situation 3, the installation package of the application is retrieved from a *content repository*, which is automatically selected by ACPC and is one of the following: 1) a content server; 2) a P2P storage network; or 3) the disk of the initiator peer's terminal. The users do not necessarily know about these options at all; they just see that a file is automatically downloaded from somewhere.

One possible sequence of ACPC action is visible in Figure 7. The sequence depicts a situation where the content repository is the initiator's disk and the session is created successfully.

The typical usage situation of ACPC cannot be explicitly defined, since it is dependent on the application and the session type to be initiated. When a game or other interactive session initiation is concerned, we argue that it is likely to happen while having dead time, e.g. travelling by bus. However, the user does not know if any of his friends is having dead time at the same exact moment (if not revealed by their presence information), which emphasizes the prevalence of physical presence e.g. in the gaming situation. This limitation would shrink the potential user group only to a minority if the user could not easily retrieve the information about his friends' willingness to have e.g. a game session with him. Thus, the users are able to bring forth their willingness (for interactive sessions such as games) that is shared by PnPAP among other context information.

To guarantee a fluent user experience, ACPC must include functionality to react to exceptional situations such as the interruption of installation on the receiver terminal. The middleware must observe these exceptions and indicate them also to the initiator with an appropriate level of detail; the initiator must never be left in a state of uninformed waiting.

ACPC can also be utilized for disseminating non-interactive data objects such as video clips and still pictures. This usage situation is less tied to time and physical presence than a session proposal. ACPC may take care of finding the appropriate application for consuming the received media; if needed, an automatic application search is performed in the available P2P networks or content servers. As with traditional media delivery solutions (such as MMS), the ACPC user may react to the received media offering right away, later on, or ignore it completely.

It is intended that ACPC be implemented as a mobile middleware, thus application developers can easily access its functionality. Since session management closely interlocks with other networking functionality, we deemed it best to incorporate ACPC into the design of the PnPAP middleware. Application developers see ACPC functions as part of PnPAP. This integration approach also reduces the complexity of the run-time environment of supernetworked applications, because the programs only need to communicate with one piece of middleware.

A popular mobile content-push solution is WAP push. It is based on text messages that trigger a WAP download when opened. Comparison between ACPC and WAP push shows that WAP push emphasizes the delivery of a data object, while ACPC emphasizes the assisted setup of a session and close integration with the supported applications. Although ACPC is designed to be IP-based, it could also utilize WAP push for the content delivery step, if needed. Another existing content push technology is OMA DS push; it is used for synchronizing personal information between a phone and a server, and is not suitable for ACPC-like user-to-user pushes. When ACPC is compared within the category of *content delivery* solutions (not just push), it seems to bring certain advantages over the common installation methods of mobile software: text-message based subscriptions and WWW downloads require explicit action and initiative from the installing user, whereas ACPC does not.

6.2. Flexible Management of Versions and Extensions

Despite its benefits, the ability to automatically propagate copies of just a single installation package (one per each application A_i) to any peer in the peer group is hardly enough for raising P_s to 1.0. The

problem of isolated application islands is further aggravated by the fact that today's mobile terminal base is very diverse. The same application-installation package that works on the initiator's device might not work on the receiver's different kind of device.

Terminals by different vendors are seldom software-compatible, and also devices by the same manufacturer often suffer from breaks in binary or source code compatibility between older and newer products, or between parallel differentiated product lines. Concerning the heterogeneous terminal base, it is desirable that ACPC can proactively alleviate the problems of incompatibility. This is possible, if different installation packages of A_i for different platforms are available in the content repositories. When the receiver's terminal starts to download the application installation package, it should automatically discover the package that is compatible with the local platform.

Of course, there is still the issue of multiple application versions: different versions of the same application might not be compatible. Version information should be attached to the application packages in order to provide a fix to the problem. When a receiver has, for example, an older version of the application than the initiator has, he could be given the possibility to dynamically upgrade to the more recent version that is available in the content repository, and, when the new version has been installed, ACPC initiates a session using the upgraded application.

As required in the problem statement, the probability to be able to initiate multisessions (P_{ms}) can also be increased by using ACPC. This is naturally done so that ACPC checks which of the multiple required applications have been already installed, and dynamically installs those applications that are missing, just like ACPC would do with single-application session proposals. Moreover, extensions and patches to ACPC-supporting applications could be distributed in a similar manner as the applications themselves. This could include the distribution of bug fixes and data files such as game levels. Again, these installations would be proposed to the user upon session initiation.

6.3. Security Aspects

Security is clearly an important requirement for a solution that enables any kind of installation of executable content, especially when dealing with push-type installation of applications.

In the ACPC solution, only the members of the same peer group are able to make session proposals that could lead to the download and installation of an application. While peer-group based protection alone is not a sufficient countermeasure for dangerous pushes, at least it prevents pushes from any malicious user who is not a member of the group. The session proposals must be cryptographically signed with the initiator's identity to protect users from fake-identity pushers.

Similarly, the installation packages should be digitally signed by their vendors in order to prevent the rigging of their applications with malicious code. Thus, ACPC should include cryptographic functionality that forces the installation packages to be digitally signed and thus of verifiable origin. ACPC should never open non-signed packages during a content push.

Security hazards related to ACPC must be taken seriously. For example, a simple digital signature is not effective to detect attacks from compromised peers, if the signature is used for verifying that a given peer (in the user community) has created a message. However, if the signature is of a trusted operator's server (and never of a peer in the user community), security is significantly stronger: the software installation package may not be tampered with, if the server is located in a known, trusted

party's premises and kept secure. Deceptive techniques like malware, Trojans or phishing are hard to detect by common users in practice. For maximal security, ACPC could only use centralized content repositories. It is also interesting to note that, for instance, Linux software is routinely installed from remote software repositories that are generally trusted by the users.

6.4. Potential Positive Implications

Commercially, ACPC appears attractive at least from the viewpoint of content providers and mobile network operators, since the session initiations and the transmission of actual data create revenue through both the increased data traffic and content purchasing. When examining the mobile content business more closely, ACPC can also be seen as a functional tool for the superdistribution of commercial content. In superdistribution, the users distribute the content in their social communities legally, provided that the rights for the content are always acquired separately for the each new user [23]. We argue that the threshold for licensing new content (now including both applications and static content) is lower when recommended by a friend than a literally faceless commercial party.

In traditional superdistribution, the pusher must think: "I'll send this file." In ACPC however, the initiator's actual purpose is *not* the content push – he is just proposing a session to his friend. If a content push happens, it is a side effect of session proposal. From the pusher's point of view, the triggering of a content push is implicit when using ACPC, whereas traditional superdistribution requires explicit actions for sending the content. From the receiver's point of view, the ACPC receiver is not just "accepting a file". The message on the screen is primarily a proposal for a session, and the software installation is only a requirement for being able to initiate that session.

If ACPC provides the possibility to create plugin modules that augment its functionality (for example, enabling the middleware to handle new kind of content push situations), developers can easily introduce their own new modules and enhance the functionality of ACPC. These software updates can be distributed in the network and ACPC can be updated like any other software.

Though ACPC is aiming to minimize the effort from the end-users, the users should always be aware of the system's context, and at their will, take part to technical decision-making. This approach is called seamfulness and presented in [24]. When the users are also aware of the technical boundaries of the system, they have a chance to create and develop new kind of uses for ACPC.

6.5. Potential Problems

Technologically, ACPC enables session initiation with any friend in a peer group, but if the user (as the initiator) does not understand this, he might never get the spontaneous idea to propose a session to his friend. Thus, the lack of an appropriate mindset among the users could diminish the use of ACPC. Fortunately, ACPC-supporting applications would probably have menus with clear indications that session proposal to a friend is possible. For example, a game could have an option called "play with a friend". Names of users to contact in the current peer group would be listed under that menu. This would probably urge the user to try initiating a user-to-user session.

As with most middleware solutions, any applications on top of ACPC must be aware of the underlying middleware and the API. This could be an obstacle for the widespread commercial adoption of ACPC, if application developers are not eager to add ACPC support to their products.

Applications must be written so that they explicitly support ACPC, and also existing applications must be modified when ACPC support is added to them. A hen-and-egg problem can be recognized: if the middleware is not installed in a large number of terminals, application developers might not be interested in ACPC – and if there are no ACPC-supporting applications, it might be difficult to justify the integration of ACPC middleware into mobile terminals.

Also, if the degree of ACPC penetration in the mass of mobile-terminals is low, those users who have ACPC do not have enough peers to initiate sessions with. If the initiator has ACPC but the receiver does not, session proposals will fail. As it is crucial that ACPC itself has enough penetration among the mobile user community, it should probably be an operator-promoted service or even integrated into the handset's operating system. If ACPC is to be utilized in the real world, operators and large content suppliers have a central role in helping ACPC to pervade among the mobile users.

7 Implementation of the Session Management Solution

7.1. Scope

When implementing ACPC began, it was necessary to define which parts of the functionality to realize. Many of the advanced features could be left unimplemented. For instance, the creation of different application binaries for several types of terminals, and the related automatic search capabilities for finding the installation package for a specific terminal type, were not implemented.

Applications access ACPC functionality via the *ACPC Application Programmer's Interface* (API). The ACPC API is part of the PnPAP API and provides the following session-related functionalities: start (propose) a session to a remote user; accept a session being proposed by a remote user; reject a session being proposed by a remote user; end an active session.

All these actions return asynchronously; they involve SIP-based communication with a remote ACPC middleware instance over the network. When an asynchronous ACPC action returns a status value or an ACPC message from a remote peer requires the attention of the application, the ACPC-to-application messaging is accomplished by using the methods of the *ACPC Callback Interface*. How these events are indicated to the local user, is application-specific.

7.2. An Application with ACPC Support

For the evaluation of the ACPC middleware module, an ACPC-supporting application was needed. Fortunately, we already had an application that involved user-to-user sessions and was therefore an excellent application for demonstrating ACPC: the Real-Time VoIP application, which is a supernetworked P2P voice-call application built on top of PnPAP.

Real-Time VoIP was modified so that its VoIP functionality remains intact, but its session management component is aware of ACPC. Essentially this means that 1) all session-related signalling actions are handled using the ACPC API; and 2) the application is able to start a session automatically on the receiver's terminal, when it has been installed using ACPC's dynamic download functionality. Session parameters for Real-Time VoIP application include the voice codec to use, the sample rate, sample size in bits, packet size, and the initiator's SIP URI.

The voice codec used in the test runs was Adaptive Multi-Rate Narrow Band (AMR-NB), at 4.75kbit/s (mode 0), with 13 bytes of audio data and a 4- or 8-byte header per frame, being able to compress the 16-bit 8000Hz PCM voice samples for half-duplex speech transmission over GPRS.

7.3. Delay Measurements

Quantitative information about the performance of the ACPC implementation was gathered by measuring the different delays generated during the basic flow. This section describes the measurement setup and presents the measurements' results.

It is important to notice that during the basic flow, both *technology-originated delay* (TOD) and *user-originated delay* (UOD) are generated. TOD is any delay caused by technical factors, such as the time required for downloading a file. On the contrary, UOD is any delay caused by the user's human slowness: a user might cause UOD for example when he reads a dialog on the screen.

This distinction between TOD and UOD was not needed when we measured HCon delays, because all the HCon delays were TOD. In ACPC-based content push, however, both types are present, and UOD could even be the dominant type of delay. Obviously, the amount of UOD may vary greatly, depending on the situation and the user's level of ACPC experience. The total delay (all TOD + all UOD) should be as low as possible, especially for the convenience of the initiator.

In the measured ACPC scenario, the pushed software was our Real-Time VoIP application. Two phones were involved in the measurements, but there was only one operating person, acting as both the initiator and the receiver. This person was an experienced ACPC user. Below, the terminals of the initiator and the receiver are called the "initiator-phone" and the "receiver-phone", respectively.

One measurement series of 20 iterations was conducted. Each iteration consisted of the following steps that reflect the course of the basic flow.

1. Real-Time VoIP is manually started on the initiator-phone, and NaviP2P (our navigation application) is manually started on the receiver-phone. (NaviP2P is used, because it starts an instance of PnPAP. ACPC is part of PnPAP and it must be listening to incoming session proposals.)

2. On both phones, the same peer group is manually joined to.

3. On the initiator-phone, the receiver peer's name (SIP URI) is manually picked from the peer list, and a voice session proposal is selected to be sent from the options menu.

4. When the session proposal (the "download or not" query) is shown on the receiver-phone, "Yes" is manually selected to accept the proposal. Downloading of the installation package starts.

5. When ACPC has opened the installation package that was downloaded, the installation dialog is manually gone through on the receiver-phone. When the installation dialog is over, ACPC automatically launches the new application that was installed.

6. The receiver's auto-launched application builds its session state, based on the session parameters ACPC gave to it. Then it sends a session-accept message to the initiator's application.

7. The session starts; the operating person exits the application on both phones. Real-Time VoIP is removed from the receiver-phone, thus it can be installed again during the next iteration.

Each iteration produces seven timestamps. From them, the following delays can be calculated:

• T_{total} = total delay seen by the initiator, between the moment he sends a session proposal and the moment when the actual session initiates.

• $T_{signalling}$ = TOD from all node-to-node signalling, i.e. transmitting ACPC messages over the network and processing them in PnPAP.

• T_{query} = UOD before the receiver selects "Yes" in the download query.

• $T_{download}$ = TOD when the installation package is downloaded.

• $T_{install}$ = UOD and TOD when the installation dialog is performed and when the application is auto-launched after the installation dialog.

• $T_{final\ accept}$ = TOD between the application launch and the sending of the final session-accept message to the initiator. This is basically synonymous with "the time required for building the session-state in the receiver's application after it has been launched".

As the experienced user in this pre-determined usage scenario does not spend time reading the dialogs and thinking what to do, the UOD is near the best possible case, i.e. the minimum UOD. The physical connectivity, however, was GPRS; thus, the TOD represents almost a worst-case scenario. Again, we stated hypothetical delays before the measurements. The hypothetical average delays were as follows: T_{total} = 39.0 s (sum of all other delays); $T_{signalling}$ = 3.0 s; T_{query} = 2.0 s; $T_{download}$ = 24.0 s; $T_{install}$ = 5.0 s; $T_{final\ accept}$ = 5.0 s.

The results from the actual measurements are presented in Figure 8.

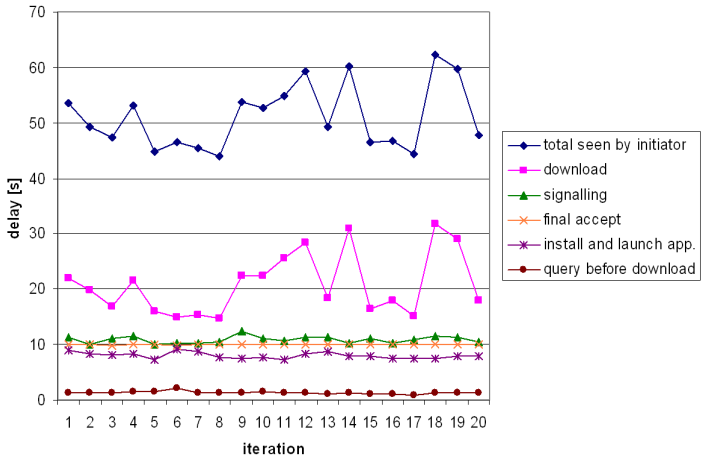

Figure 8 ACPC delay measurements.

The average measured T_{total} is 51.0 s (std. dev.: 5.8 s). This average total delay is a little higher than the corresponding hypothetical value (39.0 s). Downloading ($T_{download}$) took approximately the expected amount of time, but the ACPC-to-ACPC signalling delay ($T_{signalling}$) and session setup delay at the receiver ($T_{final\ accept}$) were higher than expected. The signalling delays are explained by network roundtrip time and the internal processing of the middleware and applications; they took more time than was expected. In real superdistribution, there would also be delay from DRM service access.

7.4. Recognized Issues on Usability

During testing it was understood that the initiator might start to feel bored during a content push. The receiver's attention is occupied when he sees the dialogs and other push-related action on his terminal's screen, but the initiator may get bored if he just passively waits for things to get done by the remote receiver. If the receiver's slowness introduces a lot of UOD, the initiator's waiting time could be several minutes. The waiting time could be made more pleasant by providing some kind of waiting-time activity, such as a mini-game to play or an animation to watch, for the initiator.

The content push situation also highlighted the importance of trial-period functionality for commercial content. Indeed, it would feel somewhat unnerving to pay for a new application before trying it out. A free evaluation period for applications is an essential feature for commercial ACPC.

8 Discussion and Future Work

The efforts behind this paper focused on creating clean abstract designs of the two components and then implementing a relevant sub-set of the designed functionalities. The main goals of the work were attained: the systems realized are functional, even though the measured delay times reminded us of the fact that the performance of proof-of-concept solutions is often far from optimal.

Future work with HCon includes its evaluation with a greater number of versatile entities. A greater number of P2P protocols, such as JXTA, Gnutella, and FastTrack could be implemented as changeable modules for HCon-based protocol management. The aforementioned P2P protocols were studied in the beginning phase of the All-IP project, when the key decisions about technologies to implement were made. Only DC++ and VSP were actually implemented for PnPAP.

The issue of IP mobility has not been emphasized in the work for HCon yet. The design of HCon does not feature any specific mechanisms for IP mobility management as we have concentrated on realizing the proof-of-concept decision-making framework. In the future, HCon should be optimized with regard to mobility management. The anticipated triumph of IPv6, also in the mobile realm, makes it desirable to add MIPv6 support into HCon so that the networking resources can take advantage of IP mobility in vertical handoffs.

Evaluation of HCon showed that the implemented system was reasonably efficient, although more optimization could be done. Future work with HCon may include implementing the system from scratch on a different platform, using different tools and a more effective approach to implementation. It would be interesting to compare HCon's decision-making performance with earlier solutions. However, none of the related publications discussed in section 3.1 provided any kind of measurements of decision-making delays. Vertical handoff delays were often discussed, but they are an entirely different thing from the delays of the decision-making logic itself.

In the future, specialized responsibilities for HCon could also be studied in detail. For instance, HCon-based connectivity management could be used to control power-consumption on the mobile platform. Wireless network access, the very thing controlled by HCon, is a major power drain in phones. Research on battery saving [25] suggests that significant energy savings can be attained by dynamically adjusting the fidelity of media rendition in applications. In a similar fashion, HCon could observe the entity stack for parameters subject to optimization in order to save the battery. Battery saving is just one example of the special responsibilities of HCon requiring further study.

HCon could be adapted to concurrent multi-interface data transmission for the needs of very bandwidth-intensive applications; this would involve several TDCPs for a single logical channel of communication. Stream Control Transmission Protocol (SCTP) is a protocol intended to replace TCP for real-time communications, and M2-SCTP [26] is a proposed mobile SCTP-variant that uses more than one connectivity concurrently to enable higher aggregate bitrate. Applying M2-SCTP in HCon could provide higher-capacity connections for the applications on top of HCon.

For ACPC, session mobility support can be seen as an important future extension. The design work behind application supernetworking has not yet precisely decided how to implement session mobility. A generic session-mobility solution must be provided to be used by versatile applications; this functionality could be incorporated into ACPC that is currently the session-management component for supernetworked applications. Of course, a collaborating middleware solution must be deployed on any device that is intended to be a party in session handoffs.

Currently the two novel components, HCon and ACPC, are not optimally orchestrated to perform as a whole. For example, HCon is not aware of the special needs of an ACPC-based download. HCon and ACPC should be more closely integrated together. In order to realize the application supernetworking paradigm, its components must collaborate in a truly seamless manner. This speaks in favor of a single-middleware solution, rather than a disparate array of single-purpose middleware programs with limited awareness of each other and no common operational logic.

Hybrid cases for ACPC and HCon are an additional justification for integrating both solutions more tightly into a single middleware. For example, consider PnPAP's dynamic protocol installation capability. Integrating this ACPC-style functionality with HCon would enable the seamless acquisition of new protocol implementations, when the SM-based decision procedure indicates that a specific protocol is optimal for the TDCP but the protocol has not been installed yet. This could be specified as a hybrid functionality of ACPC and HCon. This functionality adds freedom to TDCP decisions: a larger set of TDCPs is available when dynamic installation of new protocols is possible. Optimization of TDCPs for multisessions and rich calls is another hybrid case for ACPC and HCon.

Collaboration between HCon and ACPC could be achieved by a layered solution, where ACPC is on top of HCon and, knowing the most about the session details, is able to translate that information into parameters for HCon. Changes in session state or expected data transmission patterns would trigger HCon-level decisions about the networking resources for different sessions: for example, when a jitter-tolerant file-sharing session changes into a VoIP session, HCon might abandon a low-quality WLAN connectivity currently in use and switch to a low-jitter 3G connection.

Future work with HCon especially involves the consideration of terminal mobility support and the creation of more capable state machines. Future work with ACPC must more thoroughly deal with the security of software installations and the details of the user experience. In addition, more efficient, practical implementation and realistic evaluation are needed for both, HCon and ACPC.

Disruption points, such as the emergence of a new set of technologies, could give a foothold to innovative solutions like ACPC and HCon, contributing to their large-scale adoption. It is easier to take new solutions into use when old paradigms are being abandoned. When the revolution starts, the possibilities provided by intelligent mobile middleware solutions will hopefully be recognized.

9 Conclusions

This paper discussed the design and implementation of two solutions, one for connectivity management and the other for session management, which are building blocks for mobile systems that realize the application supernetworking paradigm. First, an overview of the current and future trends of mobile networking was provided. Then the conceptual framework behind this paper was explained. The state-of-the-art of the problem area was presented, and the research questions to be answered in this paper were stated.

The design of the connectivity management solution, Holistic Connectivity, was presented. It was required that the solution would be implemented as a mobile middleware that manages the networking resources used by applications. The design featured cross-layer decisions controlled by replaceable state machines that contain the decision rule-bases. Context information and user preferences can be used as parameters for the connectivity management decisions.

After its design, a proof-of-concept implementation of Holistic Connectivity was presented. The technical choices made during the implementation work were discussed and the performance of the system was evaluated with delay measurements.

Then, the design of the session management solution, ACPC, was discussed. The solution enables the semi-automatic installation of missing software on the target terminal of a session request. If the application required for a session has not been installed on the receiver's terminal, it can be dynamically downloaded and installed; the need to perform this installation is proactively detected by the ACPC middleware. Application installation is thus seen just as a side effect of session initiation. Apparently ACPC could enable efficient commercial user-to-user content push for applications, because the solution makes content-sending implicit and motivates content-accepting.

A working implementation of ACPC was then presented, as part of the PnPAP middleware. Delay measurements were conducted to obtain suggestive information about the technology-originated and user-originated delays during a content push.

Finally, the results and experiences from the work were summarized and the implications of the two novel technologies were discussed along with the related future challenges.

Acknowledgements

The acknowledgements are due to the Application Supernetworking project teams at the MediaTeam Oulu and Intelligent Systems research groups, University of Oulu, Finland. The work has also been supported by the ExpeShare and Decicom research projects. The authors would like to thank TEKES (Finnish Funding Agency for Technology and Innovation), Nokia, TeliaSonera, Elektrobit and Serv-It for supporting the project financially. Special acknowledgements are given to Mr. Juha Ylipekkala and Mr. Mikko Polojärvi for their technical assistance.

References

1. Ala-Kurikka, J., Ylianttila, M., Harjula, E. and Kassinen, O., Empirical aspects on implementing application supernetworking. in Proceedings of Nordic Radio Symposium, (Oulu, Finland, 2004).

2. Androutsellis-Theotokis, S. and Spinellis, D. A Survey of Peer-to-Peer Content Distribution Technologies. Journal of ACM Computing Surveys, 36 (4). 335-371. 2004.
3. Kassinen, O., Koskela, T., Harjula, E., Pohjanen, P., Ala-Kurikka, P. and Ylianttila, M., Group-based content push with dynamic session startup. in Proceedings of 4th International Conference on Mobile and Ubiquitous Multimedia, (Christchurch, New Zealand, 2005).
4. Howie, D., Ylianttila, M., Harjula, E. and Sauvola, J., State-of-the-art SIP for mobile application supernetworking. in Proceedings of Nordic Radio Symposium, (Oulu, Finland, 2004).
5. Harjula, E., Ylianttila, M., Ala-Kurikka, J., Riekki, J. and Sauvola, J., Plug-and-Play Application Platform: Towards mobile peer-to-peer. in Proceedings of 3rd International Conference on Mobile and Ubiquitous Multimedia, (College Park, MD, U.S.A., 2004).
6. Ala-Kurikka, J., Ohtonen, J., Harjula, E. and Ylianttila, M., Improving multiple mobile application interaction with unified session management. in Proceedings of 17th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, (Helsinki, Finland, 2006).
7. Gazis, V., Houssos, N., Alonistioti, A. and Merakos, L., Evolving perspectives of 4th generation mobile communication systems. in Proceedings of 13th International Symposium on Personal, Indoor and Mobile Communications, (Coimbra, Portugal, 2002).
8. Grosse, E. and Lakshman, Y.N. Network Processors Applied to IPv4/IPv6 Transition. IEEE Network, 17 (4). 35-39. 2003.
9. Ylianttila, M. Vertical Handoff and Mobility – System Architecture and Transition Analysis. Doctoral thesis. University of Oulu, Department of Electrical and Information Engineering, Oulu, Finland, 2005.
10. Wang, H.J., Katz, R.H. and Giese, J., Policy-enabled handoffs across heterogeneous wireless networks. in Proceedings of 2nd IEEE Workshop on Mobile Computing Systems and Applications, (New Orleans, LA, U.S.A., 1999).
11. Stemm, M. and Katz, R.H. Vertical Handoffs in Wireless Overlay Networks. ACM/Baltzer Mobile Networks and Applications, 3 (4). 335-350. 1998.
12. Chen, L.-J., Sun, T., Chen, B., Rajendran, V. and Gerla, M., A smart decision model for vertical handoff. in Proceedings of 4th ANWIRE International Workshop on Wireless Internet and Reconfigurability, (Athens, Greece, 2004).
13. Hasswa, X.A., Nasser, N. and Hassanein, H., Generic vertical handoff decision function for heterogeneous wireless networks. in Proceedings of 2nd IFIP International Conference on Wireless and Optical Communications Networks, (Dubai, United Arab Emirates, 2005).
14. Bari, F. and Leung, V.C.M., Service delivery over heterogeneous wireless systems: network selection aspects. in Proceedings of ACM International Wireless Communications and Mobile Computing Conference, (Vancouver, Canada, 2006).
15. Sun, J., Riekki, J., Sauvola, J. and Jurmu, M. Policy Mechanism and Evaluation Algorithm for Connectivity Management Adaptability. International Journal of Pervasive Computing and Communications, 3 (1). 57-81. 2007.
16. Zhao, X., Castelluccia, C. and Baker, M. Flexible Network Support for Mobile Hosts. ACM/Baltzer Mobile Networks and Applications, 6 (2). 137-149. 2001.
17. Brewer, E.A., Katz, R.H., Chawathe, Y., Gribble, S.D., Hodes, T., Nguyen, G., Stemm, M., Henderson, T., Amir, E., Balakrishnan, H., Fox, A., Padmanabhan, V.N. and Seshan, S. A Network Architecture for Heterogeneous Mobile Computing. IEEE Personal Communications Magazine, 5 (5). 8-24. 1998.
18. Ylitalo, J., Jokikyyny, T., Kauppinen, T., Tuominen, A.J. and Laine, J., Dynamic network interface selection in multihomed mobile hosts. in Proceedings of 36th Annual Hawaii International Conference on System Sciences, (Waikoloa, HW, U.S.A., 2003).
19. Puttonen, J. Mobility Management in Wireless Networks. Doctoral thesis. University of Jyväskylä, Department of Mathematical Information Technology, Jyväskylä, Finland, 2006.

20. Kassinen, O., Ylianttila, M., Sun, J. and Ala-Kurikka, J., Top-down connectivity policy framework for mobile peer-to-peer applications. in Proceedings of 4th Annual IEEE Consumer Communications and Networking Conference, (Las Vegas, NV, U.S.A., 2007).
21. Xing, B. and Venkatasubramanian, N., Multi-constraint dynamic access selection in always best connected networks. in Proceedings of 2nd Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services, (San Diego, CA, U.S.A., 2005).
22. Gazis, V., Alonistioti, N., and Merakos, L. Toward a Generic "Always Best Connected" Capability in Integrated WLAN/UMTS Cellular Mobile Networks (and beyond). IEEE Wireless Communications, 12 (3). 20-29. 2005.
23. Löytynoja, M., Seppänen, T. and Cvejic, N., Experimental DRM architecture using watermarking and PKI. in Proceedings of 1st International Mobile IPR Workshop: Rights Management of Information Products on the Mobile Internet, (Helsinki, Finland, 2003).
24. Chalmers, M., Dieberger, A., Höök, K. and Rudström, Å. Social Navigation and Seamful Design. Cognitive Studies: Bulletin of the Japanese Cognitive Science Society 11 (3). 1-11. 2004.
25. Flinn, J. and Satyanarayanan, M., Energy-aware adaptation for mobile applications. in Proceedings of 17th ACM Symposium on Operating Systems Principles, (Kiawah Island, SC, U.S.A., 1999).
26. Huang, C.-M. and Tsai, C.-H. The Handover Control Mechanism for Multi-path Transmission using Stream Control Transmission Protocol (SCTP). Computer Communications, 30 (17). 3239-3256. 2007.