FLEXIBLE PROTOCOL STACK FRAMEWORK: DESIGN, VALIDATION AND PERFORMANCE

Tim Farnham
Toshiba Research Europe Ltd, Bristol, UK. tim.farnham@toshiba-trel.com
Thorsten Schöler
Siemens Mobile, Munich, Germany. thorsten.scholer@siemens.com.

ABSTRACT

Multi-mode capabilities are becoming more important for mobile terminal and network devices and can exploit the flexibility provided by SDR technology. The reason for this is that the number of different radio access technologies is increasing with the growing popularity of Wireless Local Area Networks (WLANs) and increasing diversity of cellular radio access technologies with 2G, 2.5G, 3G and now even proposals for 4G air interfaces. Each of these standards is evolving at different rates with many optional enhancement features for providing different QoS, power efficiency and security properties, yielding a much higher software complexity than traditional single-mode protocol stack implementations. Therefore, a single inflexible protocol stack arrangement is no longer an attractive option for multi-mode support. A preferable solution is to allow protocol stack reconfiguration by software download (or peer-to-peer software exchange) to support upgrade, bug fixing and optimisation or customisation of the protocol stack to the context of the terminal whilst ensuring proper operation by thorough validation and supervision of software configurations. Furthermore, this paper investigates the performance of a proposed solution to this problem using platform and language independent support for generic interfaces between protocol stack components.

1. INTRODUCTION

Dynamic reconfiguration of protocol stack software raises a number of reliability and performance issues. These issues become more complex when protocol stack software from multiple vendors is supported on the same device. A solution to some of these problems is proposed in this paper and a performance assessment is conducted to determine its viability. The generic protocol stack architecture meets the requirements defined for the identified key scenarios. Specifically, it allows for the high level requirements of:

- Dynamic reconfiguration of protocol stacks during active communication sessions.
- Upgrading of protocol behaviour to support optional features.
- Adding extra functionality into existing protocol stacks or replacing complete stacks.
- Download of third party protocol stacks or protocol stack modules.
- Ensuring reliability and preventing rogue terminal behaviour.
- Optimisation of protocol stack behaviour depending on the context of the terminal.
- Operation of multiple stacks simultaneously.

Previous research performed on software radio architectures and proposed solutions (see [1] and [2]) have focused on the functionality required to perform reconfiguration between a set of well defined radio access technology standards. However, protocol stack implementation can be reconfigured in different ways, for instance to more efficiently utilise device resources, or to reduce power consumption or improve security. This has not been previously considered in the context of reconfigurable terminals to allow the exploitation the capabilities of heterogeneous execution environments.
A language and platform independent framework is proposed to meet these requirements, to allow the use of different operating systems, software development languages and execution environments in different combinations and optimised to the capabilities of individual device resources. It is further proposed that generic protocol components can be reconfigured to support different protocols (for instance different radio access technologies) with different optional or even proprietary features. Because of the new flexibility and the high reliability and security requirements imposed on reconfigurable terminals, a thorough validation scheme is proposed. The validation scheme utilises on the one hand, virtual prototyping for software configuration validation and on the other hand, assertion based monitoring of protocol stack execution to avoid rogue behaviour of the terminal.
This paper investigates the architecture implementation issues and the initial performance results. These performance results are obtained for different execution environments, (operating system and Java virtual machines) on different hardware platforms. In particular the framework allows the dynamic addition of extra functionality and optional features together with the customisation of active protocol stacks.

2. FRAMEWORK

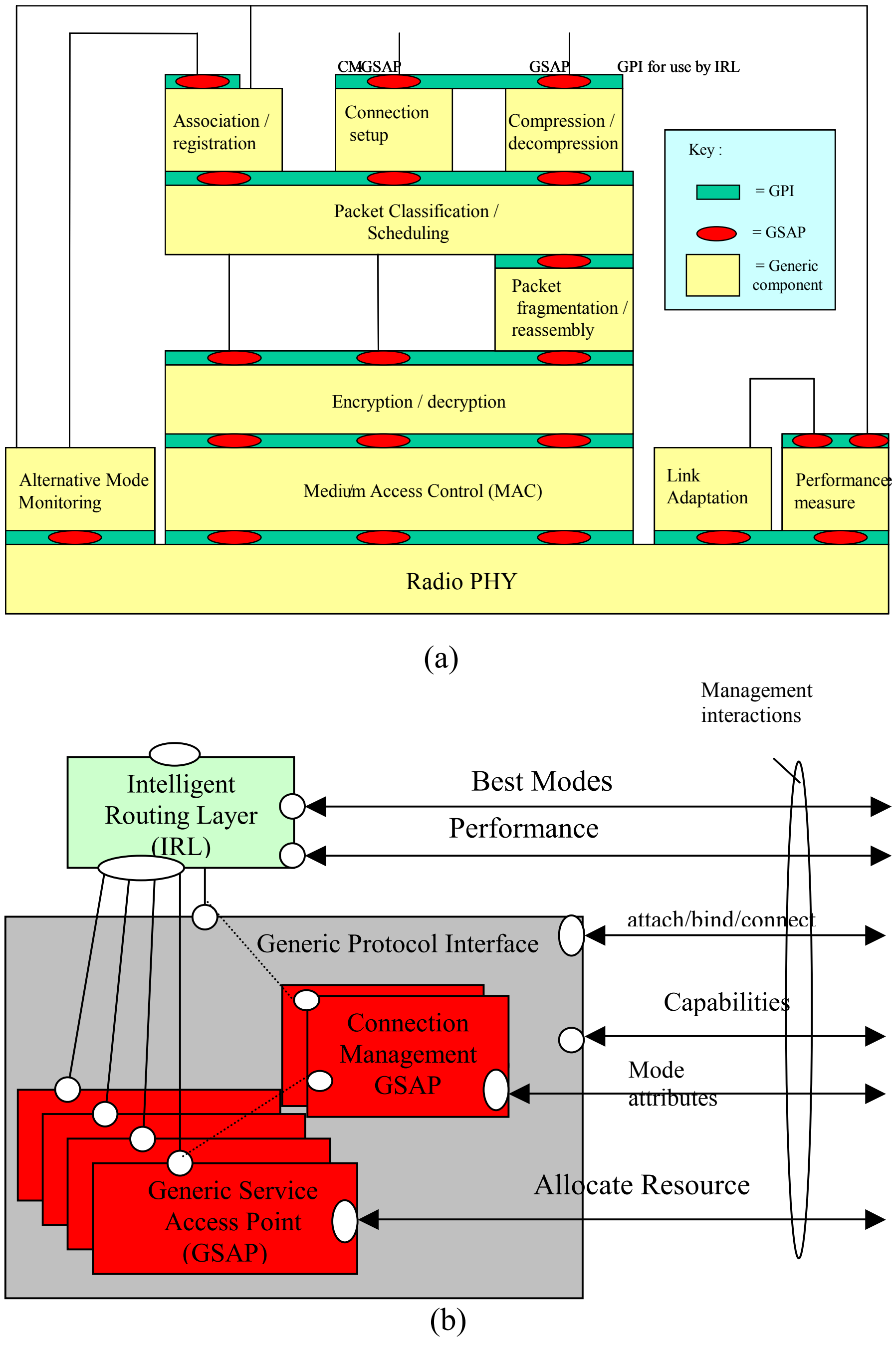
CMGSAP
GSAP
GPI for use by IRL
Association / registration
Connection setup
Compression / decompression
Key :
= GPI
= GSAP
= Generic component
Packet Classification / Scheduling
Packet fragmentation / reassembly
Encryption / decryption
Alternative Mode Monitoring
Medium Access Control (MAC)
Link Adaptation
Performance measure
Radio PHY
(a)
Management interactions
Intelligent Routing Layer (IRL)
Best Modes
Performance
attach/bind/connect
Generic Protocol Interface
Capabilities
Connection Management GSAP
Mode attributes
Allocate Resource
Generic Service Access Point (GSAP)
(b)

Figure 1 : Example Generic Protocol Stack (a) and Interface Detail (b)

A framework proposed to meet the requirements outlined above is based on a generic protocol interface (GPI) definition and generic protocol components as illustrated in the example layered protocol stack in Figure 1. The GPI supports generic service access points (GSAP) that can allow logical layer instances to be dynamically bound or rebound to form different protocol stack arrangements as required, with each protocol component running in the most appropriate execution environment (for instance processor, operating system, programmable logic or Java virtual machine) using the most appropriate programming language. The benefit of flexible communication co-processors have been investigated by other researchers [3] and potentially have many benefits for executing common protocol functionality (generic components). As also outlined in [3] a protocol state machine translator can be implemented within a co-processor to dynamically interpret high level protocol code written in the standard Specification and Description Language (SDL). However, even with such capability there is still likely to be much use made of protocol implementations in C, C++ and Java.

Simultaneous support for different radio access technology protocol combinations is provided using an intelligent routing layer (IRL) that can route packets (based on selection criteria and "best mode" indication) to and from the corresponding GSAP instances in a flexible manner. The IRL provides an open interface in order to allow selection of the most suitable modes by external mode negotiation and switching module (MNSM). The GPI itself supports binding and rebinding and security mechanisms to prevent unauthorized use of protocol components by malicious software.

It is assumed that the configuration of the protocol stacks is controlled by a configuration management module (CMM) as described in previous work [1]. The CMM is responsible for configuring and reconfiguring the protocol stacks to the correct configurations with the most appropriate combination. This is performed by binding and rebinding logical layer instance using the GSAP instance to identify the configuration and state associated with the particular binding. Therefore a unique GSAP and GPI instance naming convention must be used in each system.

In order to simplify the implementation and management it is assumed that the generic protocol components have a set of capabilities that can be retrieved to allow CMM to

determine what configurations are possible. This must be specific enough to indicate what optional algorithms and features are supported and conformance to which standards and versions and the estimated resource requirements for these different options. Proprietary extensions can also be supported in this framework by insertion of additional layers. For example a proprietary PPP based layer performing header compression and encryption could be placed on top of a standard GPRS compliant stack.
Finally, the special connection management GSAP (CM-GSAP) instances enable the setting up of connections in a generic manner. For example the QoS, security and other connection attributes could be specified in a technology neutral high level way. The CM-GSAP then handles mapping and translation to technology specific mechanisms using default attribute values stored in the CMM or MNSM (for example user profiles) as appropriate.
Currently, the proposed framework is being evaluated in terms of performance and implementation complexity in different scenarios.

3. VALIDATION SCHEME

The introduction of open Application Programming Interfaces (API) to mobile terminals has led to a number of security problems, most of them already tackled (see [7] and [8]). The validation scheme proposed for the flexible protocol stack architecture, basically follows the guard and assertion checking approaches [9] and additionally introduces system validation by virtual prototyping as new system security measure.

3.1. Guards and assertion checking

Small software units, so-called probes, will be integrated in the executing protocol stack software by the framework. These probes will secure the correctness of software execution for over-the-air protocol stack software download and network based validation. Software probes contain assertions or other means of fault detection to supervise proper software execution.
Normally, software probes will remain in code downloaded onto the target (resident probes). Using the virtual prototyping approach, software probes will be evaluated by the simulation environment and will not be present in the downloaded protocol stack code (non-resident probes). The use of non-resident probes will avoid the general drawback of using assertions: Slow-down of software execution.

3.2. Virtual prototyping

A modern way of evaluating system behaviour is by building a virtual prototype. A virtual prototype shows the same functional behaviour and timing as a real hardware-prototype but being completely simulated by a simulation tool. A virtual prototype contains, besides the simulation of the target device, a simulation of the system's environment.
An environment for functional and timing-accurate simulation of multiple system domains is ClearSim MD. Current development of the ClearSim MD tool has extended the simulation model with the ability to insert assertions [10].

3.3. Protocol stack software validation

The proposed validation scheme consists of three stages:

- Network-based off-line validation,
- Terminal-based on-line validation and
- Run-time validation (also terminal-based)

Off-line validation uses extensive system simulation for validation of the collaboration of the mobile terminal hardware and the installable protocol stack software and collaboration of the various software components itself. For that, a virtual prototype of the mobile terminal is designed and simulated in connection with simulation modules, which simulate network and user behaviour.
Terminal-based on-line validation cannot fall back on system simulation due to resource constraints and security reasons, thus being limited to simpler checks. The terminal validates to-be-installed protocol stack software configurations according to predefined rules and tries to identify suspicious software configurations. The rules may contain plausibility checks, syntactic and semantic software configuration checks.
After being validated by on-line validation, the necessary software will be downloaded and subsequently the requested protocol stack will be configured and executed on the terminal.
During execution, the mobile terminal is able to supervise protocol stack behaviour with code-resist assertions and check-points. Such assertions may i.e. check for valid content of communication messages as well as for compliance to certain threshold conditions. This is called run-time validation.

4. IMPLEMENTATION

To enable the framework to be execution environment and programming language independent, the GPI and GSAP interfaces must be able to support many different heterogeneous mixes of environment and language in a secure manner (i.e. with sufficient reliability and integrity). Solutions that appear promising, while still being lightweight enough to operate on resource constrained terminals, are asynchronous messaging mechanisms based on flexible and extensible signal data formats. These approaches are often used when modelling protocol stacks to abstract away from implementation detail and therefore can most easily cater for heterogeneous environments and languages. This approach to modelling does not specify how the message passing is performed only the format of the signal data and the destination logical entity. In the proposed framework the logical entities are identified by GSAP and GPI instances which are uniquely identifiable within a system.

Name	Type
PC Linux	1GHz Intel P3 PC with Linux 2.5.54
PC Windows	1GHz Intel P3 PC with Windows 2000
SA Linux	Intel StrongARM 200MHz with Linux 2.4.18
SA OSE	Intel StrongARM 200Mz with Enea OSE Delta RTOS
PXA Linux	Intel PXA 250 400MHz with Linux 2.4.18
PXA WinCE	Intel PXA 250 400MHz with Pocket PC 2002

Table 1 : Platforms Considered

4.1. Execution Environments

The execution environments considered within this paper are Java virtual Machines (JVM), Java interpreter and compiled native code, each environment supporting native threads and multiple processes. It is important to point out that the proposed framework is platform independent and so consideration is given to different platforms and a range of different platforms have been used to benchmark performance of different methods of implementing the proposed framework (see Table 1).

The JVM execution environment has many inbuilt security mechanisms and allows the setting of permissions for access to various resources in a relatively fine grain manner. Including runtime permissions (for example ability to stop a thread) and network and file system access permissions etc.. This has many benefits in terms of implementing protocol modules that should only be allowed access to certain resources and a set of other protocol modules. The Java language and JVM environments also support dynamic module loading and software module extensibility by inheritance. These are also useful when considering protocol stack customisation, for example adding new features to existing protocol stacks. However, this approach is highly language specific and not as appropriate for heterogeneous execution environments and programming languages.

4.2. GPI and GSAP

As previously mentioned messaging based interfaces, with dynamically extensible signal data format, are assumed most appropriate to allow execution environment and language independence. Further to this logical layer instances accessed via the GPI and GSAP can be executing in different execution environments to provide the level of protection required or because performance can be enhanced. Different execution environments may imply the use of different operating system threads or processes or could also mean the use of different physical processors. The use of different processors was not considered in the benchmark performance evaluation.

The thread and process functionality differs between operating systems and support for different thread and process messaging security options and performance can vary widely. Some operating systems have inbuilt thread and process messaging mechanisms. For example the Linux OS supports System V based inter-process messaging.

Important considerations for supporting the proposed framework are the association of message queues with GPI and GSAP instances. The location and protection of message queues within memory and access control to these queues in an efficient manner. Clearly, operating system kernels with inbuilt mechanisms to support these features will likely outperform add-on solutions. However, in-built mechanisms are often not sufficient to support every possible arrangement efficiently and securely.

The proposed thread and process architecture solution to support protocol stack reconfiguration is to associate each GPI and GSAP instance with a persistent independent input message queue. In this manner, the logical layer instance accessed via the corresponding GPI and GSAP instances can be executed within the same thread, in the same or indeed different execution environments, or in different threads or even different processes.

Operating logical layer instances in different threads and processes has the benefit of being able to provide different performance (by prioritisation) and operating system based security mechanisms to provide protection between different instances. However, the penalty is in the increased messaging latency between threads and processes.

4.3. Secure Messaging

The key features of the messaging solutions are that the identity and authority of the destination and logical layer instance using a particular GPI oe GSAP can be established and the delivery mechanism can continue when the protocol stack is being reconfigured. This necessitates that state and security related information (such as authorised users) of individual GPI and GSAP instances be maintained in a persistent manner. It also implies that messages are persistent when the sending or receiving instances are temporarily not available. Further to this the GSAP access control mechanisms must allow for the handover of ownership from one logical layer instance to another for protocol stack upgrading. This allows individual logical layer instances to be upgraded in addition to complete protocol stack layers by reassigning an existing GSAP to a new logical layer instance.

It is proposed that messaging security should have the following mechanisms to allow dynamic protocol stack upgrade and customisation:

- Unique message queue identification
- Thread safe message queue access
- Message validation

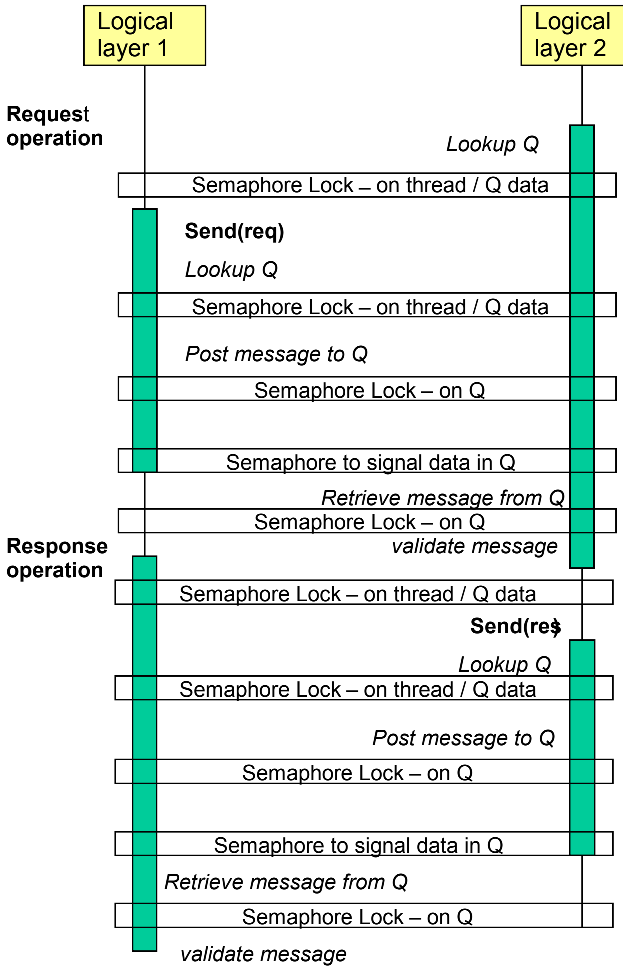

Figure 1 : Request Response Benchmark

Firstly, the message queue identification corresponding to a GSAP instance is based on a protected (lookup, add and delete) mechanisms that accesses a repository of GPI and GSAP instance status information. This allows the current message queue for a particular GPI or GSAP instance to be obtained. GPI and GSAP instances can then be assigned to different logical layer instances as they are upgraded.
Next, the message queue access is also a protected mechanism. This prevents simultaneous access to the queues and can enable the authority of the logical layer instance attempting to access a GPI or GSAP to be validated. Logical layer instances can then have exclusive access to individual GPI or GSAP message queues.
Finally, a validation is performed on actual messages within the queue to ensure that the posted messages are from an authorised source and of the correct format.

The above message security mechanisms present a certain additional overhead in ensuring reliability and integrity within dynamic protocol stack configuration. Therefore more efficient solutions have been explored that can be used in situations where this high level of protection is not required or can be provided by other means.

5. BENCHMARKING

Tests were performed to determine the performance of the proposed architecture mainly in terms of latency to perform messaging interactions across GPI and GSAP boundaries. The basic test involves measuring the round trip time for an information request operation over a GPI or GSAP interface boundary. This operation simply sends a request message and waits for a response message. The test is repeated a number of times in order to be able to measure the latency of the individual (round trip) request operation.
The comparisons of latency are performed for four different scenarios which are shown in Figure 1 and described below.

5.1. Five step process

The five step process involves five semaphore operations:
Lookup lock - The sending and receiving threads first perform a semaphore protected lookup operation to determine the message queue for the recipient GPI or GSAP instance. This ensures that GPI or GSAP cannot be added or deleted during access to this data repository.
Queue lock (send) - The second step is the semaphore locking of the message queue to ensure that no other thread can access the queue while a message is being added.
Signal - Another semaphore operation to signal the message has been added to the queue.
Queue lock (retrieve) - The final step is the semaphore locking of the queue to retrieve the message.

5.2. Three step process

The three step process eliminates the need to perform queue locking by having an atomic mechanism to add and delete messages from the queues. This means that message can be added and removed without locking.

5.3. Single step process

The single step process removes the need to perform a lookup lock because the GPI and GSAP information is held in shared memory with all threads and processes having read access to the queue information and the queues themselves held in a shared memory segment. The addition and deletion of GPI and GSAP queues is performed in an atomic operation. An advantage of holding all messages in shared memory is that it reduces the amount of data copying that needs to be performed to retrieve queue information.

5.4. No Context Switch

This approach uses no semaphore operations, as it is assumed that the sending and receiving is performed in the

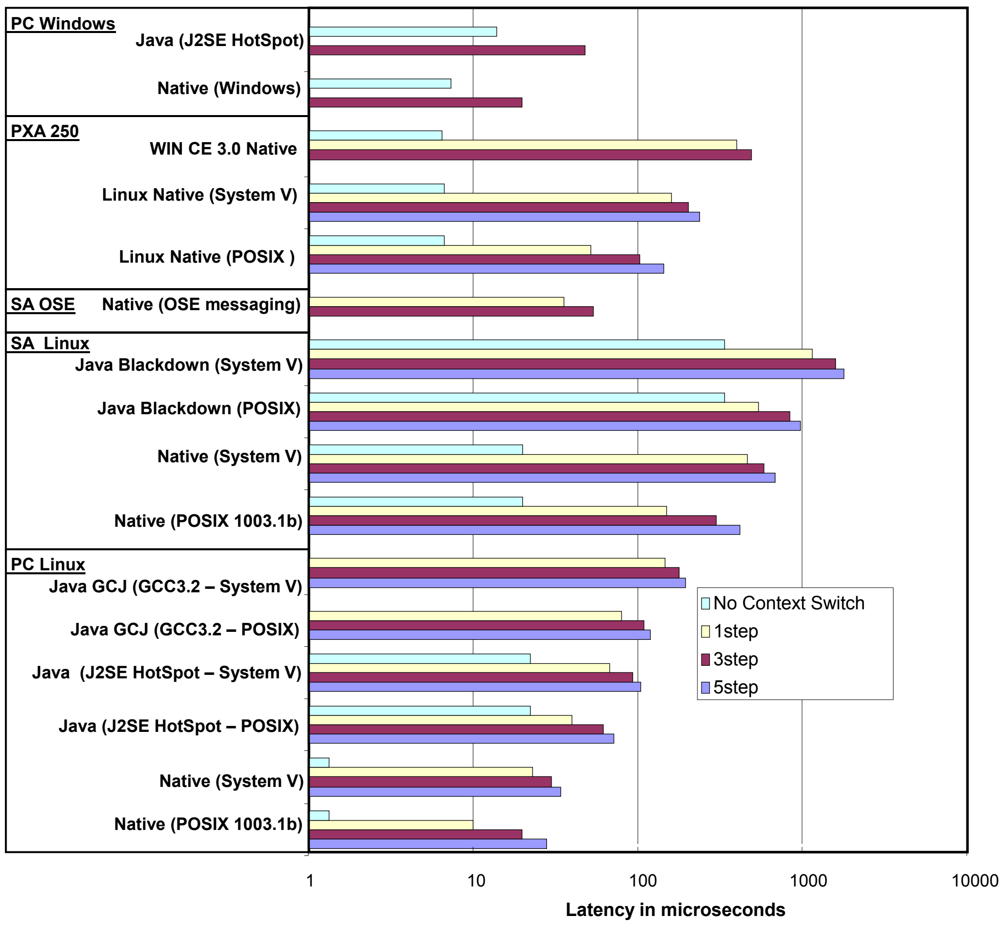

Figure 4 : Latency for Request / Response

same thread and therefore no context switch is performed. However, a look-up is still performed to post the message to the appropriate queue and dynamically obtain the necessary function call to invoke. It is functionally the same as the thread and process messaging to enable secure dynamic reconfiguration and so message validation is still performed.

5.5. Results

The results show that in the tests the PC is consistently around 15 times faster than the Strong ARM device running the Linux operating system. It is also possible to see that the System V semaphores supporting messaging between separate processes incur an additional 60 to 70% latency over POSIX thread semaphores. However, the use of an operating system with built-in support for efficient secure messaging (Enea OSE) has significant advantages being at least four times faster than the Linux based solution and only around 3 times the latency of the much more capable PC platform running Linux.

It is also notable that the JVM implementation of the Strong ARM platform consumes 3Mbytes of memory (excluding shared memory) for a logical layer instance supporting the GPI with 9 threads (4 of which are JVM related administrative threads). This is compared to the 630kBytes required for the native approach.

4.6. Observations

The results indicate that there are significant benefits if the GPI and GSAP interface boundary occurs within the same thread or process (no context switch case) to avoid context switches. However, there are benefits of using different threads and processes optimised to provide different performances and security, particularly when multiple processor and mixed protocol software vendor environments are used. For example, any misbehaviour within a logical layer instance thread can be relatively quickly identified and the GSAP message queues blocked and the thread suspended. However, if a GSAP boundary is within a thread it may be difficult to determine the logical layer instance causing a particular problem (such as memory leaks) and suspend (or block) the correct GSAP and extract the offending protocol stack software component.

A context switch is not necessarily required when using different environments under the control of the same operating system, however, the performance benefits vary considerably between the platforms considered. For example, performing a I step based context switch between threads on the Linux based PC platform is not much longer than with no context switch on the PXA 250 and PC Windows platforms (and in fact less that on the SA Linux platform).

6. CONCLUSIONS

This paper has presented a framework for supporting flexible reconfigurable protocol stacks using secure and efficient asynchronous messaging in heterogeneous execution environments. In addition the performance results, in terms of latency for component interactions, have been presented for different platforms. The specific advantages of the proposed framework over other approaches (such as documented in [5] and [6]) are that it enables execution environment and programming language independence, (exploiting the full benefits of the device capabilities), without being restricted to using a single environment (such as a JVM, operating system or processor) or computational model (such as thread per message). At the same time the framework is lightweight and so it does not impact heavily on device requirements. The combination of off-line, on-line and run-time validation, combined with generic interaction reliability and integrity mechanisms are appropriate for the dynamic reconfiguration of protocol stacks on resource constrained devices.

The framework has limitations particularly when high performance is required for synchronous interactions with trusted software particularly within common execution environments. Currently, there is also ongoing research into high performance component technologies for scientific computing that supports multiple languages, execution environments and exploits the benefits of parallelism [4]. However, many of these approaches are too heavyweight for the often resource constrained terminal device. The proposed approach can, if necessary, be combined with these high performance component technologies and utilise a suitable common Interface Definition Language (IDL).

The performance observed in tests on different platforms indicates that the framework can be successfully applied in many scenarios to allow dynamic protocol stack reconfiguration while making use of heterogeneous execution environments and mixtures of programming languages within protocol stacks. Using such a framework could ultimately open up a new innovative era of third party protocol stack downloading to mobile devices akin to the application downloading that is common place for PC users today. This can allow network operators and users enjoy a new freedom of communication protocol customisation and optimisation providing enhanced performance, battery power and radio resource utilisation efficiency and security.

7. REFERENCES

[1] M Beach et al. “The European Project TRUST – reconfigurable terminals and supporting networks”; Annales des telecommunications, July / August 2002

[2] Christian Bonnet et al.; “An all-IP software radio architecture under RTLinux”; Annales des telecommunications, July / August 2002

[3] Schiller, J.H.; “A flexible co-processor for high-performance communication support“; GLOBECOM '95., IEEE , Volume: 2 , 13-17 Nov. 1995

[4] Armstrong, R.; et al.; “Toward a common component architecture for high-performance scientific computing”; The Eighth International Symposium on High Performance Distributed Computing,., 3-6 Aug. 1999

[5] Moessner, K.; Vahid, S.; Tafazolli, R.; “Terminal reconfigureability-the OPTiMA framework”; 3G Mobile Communication Technologies, 2001. (Conf. Publ. No. 477) , 26-28 March 2001

[6] Philip Eyermann, “Maturing the Software Communications Architecture for JTRS”; IEEE MILCOM 2001. Volume: 1 , 28-31 Oct. 2001

[7] 3rd generation partnership project (3GPP): Mobile execution environment (MExE), Service description. Technical report stage 1, release 4, TS 22.057 V4.0.0. October 2000.

[8] 3rd generation partnership project (3GPP): Mobile execution environment (MExE), Functional description. Technical report stage 2, release 4, TS 23.057 V4.1.0. March 2001.

[9] Naumovic, G. and Memon, N.: “Preventing piracy, reverse engineering, and tampering”. IEEE computer magazine. 2003.

[10] Oodes T. and Müller-Schloer, C; “UML-basierter Systementwurf sicherheitskritischer, heterogener Systeme“. 2002.

ACKNOWLEDGEMENTS

This work has been performed in the framework of the IST project IST-2001-34091 SCOUT, which is partly funded by the European Union. The authors would like to acknowledge the contributions of their colleagues from Siemens AG, France Télécom – R&D, Centre Suisse d’Electronique et de Microtechnique S.A., King’s College London, Motorola SA, Panasonic European Laboratories GmbH, Regulierungsbehörde für Telekommunikation und Post, Telefonica Investigacion Y Desarrollo S.A.Unipersonal, Toshiba Research Europe Ltd., TTI Norte S.L., University of Bristol, University of Southampton, University of Portsmouth, Siemens ICN S.p.A, 3G.com Technologies Ltd, Motorola Ltd, DoCoMo Communications Laboratories Europe GmbH