Research on Cross-Layer Manipulation Model in TCP/IP Architecture

TCP/IPアーキテクチャにおける クロスレイヤ操作モデルに関する研究

July 2011

VU Truong Thanh

Research on Cross-Layer Manipulation Model in TCP/IP Architecture

TCP/IP アーキテクチャにおける クロスレイヤ操作モデルに関する研究

July 2011

Graduate School of Global Information and Telecommunication Studies
Waseda University

Information Network System II

VU Truong Thanh

ACKNOWLEDGEMENT

After 8 years at Waseda University, I have realized one thing – I could never have done any of this, particularly the research and writing that went into this dissertation, without the support and encouragement of a lot of people.

First of all, I would like to thank my advisor, Professor Dr. Yoshiyori Urano. I owe you so much. You've been my teacher, my mentor, my confidant, and a never-ending source of motivational support. You have given so much of yourself to help me succeed. You are instrumental in helping me fine-tune the dissertation and your guidance and advices have helped me get past all the self-doubting that inevitably crops up in the Ph.D course. If I do take the academic path, I only hope that I can be half the kind advisor that you have been to me. Whatever path I do take, I will be prepared because of you.

This work would also not have been possible without the support of Dr. Hidetoshi Yokota. Most of the hurdles in my research would have still laid there if it were not because of you. You have given me the confidence to continue my academic life. I cannot say how much I owe you for always being there for me at the crucial time of my research. Without your unending support, I never would have made it through this process or any of the tough times in my life.

Studying in GITS, Waseda University also gives me the chance to meet and learn from many distinguished professors with immense knowledge, great personality and big caring heart to students. I will never forget your comprehensive education and warm consideration given to me, which help me build and strengthen my academic, research and working capacity as well as my professional ethics.

I am lucky enough throughout most of the time in the graduate school, where I've been able to concentrate mostly on my research. This is due in a large part to the Research Associate program of Waseda University. In this stage of my life, I have the great luck of working under the guidance of Professor Hidenori Nakazato, Associate Professor Mutsumi Suganuma, together with the collaboration and support of my fellow RAs and GITS staff members, whose kindness has put me through the hardship of getting used to the working life in Japan.

I am also lucky to have a great group of acquaintance at Waseda University. This includes my friends in Urano Lab, my friends and colleagues in GITS, too numerous to name, especially I will remember Nameki-san, who always greets me with a great smile questioning whether there is anything that she may help. This, I believe, is one of the keys to getting through any program – having good friends to have fun with and complain to.

I would like express my sincere thanks to leaders of Posts and Telecommunications Institute of Technology in Vietnam, especially Dr. Hoang Minh, President of PTIT, who have always expressed their utmost supports to my studying in Japan.

I would also like to thank Dr. Nguyen Ai Viet, Director of Information Technology Institute of Vietnam National University, Hanoi for his encouragement and support for my research and publication in Vietnam.

And last, but not least, I would like to express my deepest gratitude to my big family in Vietnam. Especially, thank you mother and father for your endless spiritual and material support to me. I also would like to thank Ms. Nguyen Hong Mai and her family for taking care of my children my other matters at home while I was engaging in my study abroad. All of you have endured so much during my course without complaining a single word. Your love, encouragement and patience are my hidden source of energy to go on with my study. And I could not have done it without you and I would not have done it if it were not for you.

Thank you!

TABLE OF CONTENTS

SUMMARY

The conventional Internet architecture is designed in a static environment: wired-line with stable electronic signals and low-speed stationary desktops not suitable for complex networking processing. In this rather stable environment, there is no meaningful control of the networking subsystem (NS) by communication applications. This results in the layering principles of the TCP/IP protocol stack, in which a protocol in a certain layer keeps the functions and internal states to itself.

While the layering principle facilitates well the development of the TCP/IP protocol stack as well as simple communication services such as email, news group, World Wide Web pages and real-time IRC, it does not provide any support for flexible or reliable services, such as fault-intolerant session-based application for which service developers must rely on special and often very complex mechanisms due to the almost zero support from the networking stack. Things get even worse when less reliable wireless access technologies, together with them are nomadic-related issues, become widely available.

These changes in networking environment, together with advances in processing power even for handheld devices, signal that the conventional model of self-contained, status-hiding layering approach of the Internet needs to be revised to provide applications with more cross-layer information and control to better adapt to the developments of the Internet realm.

In general, there are two objectives of a cross-layer system:

(obj-a) To allow exchange of information and possibly commands among layers so that a protocol instance of a layer can harmonize its activities with the condition of other layers.

(obj-b) To allow for the safe update (modification) of a protocol's parameter so that the internal state of a protocol can be altered and adapts to the changes of external environment.

While (obj-a) can optimize the performance of a protocol in particular or the whole system in general, (obj-b) allows the protocol to change its working environment from one setting/configuration to another, which in many cases means the protocol has evolved to another protocol. However, because the settings/configurations can be different from session to session, (obj-b) requires the cross-layer system to be able to identify and access an individual instance of a protocol (in the case the protocol has several instances running in the networking subsystem).

The purpose of this research is to propose a new TCP/IP architecture called the *InterLay model* that can facilitate the sharing and manipulation of information across layers' borders in a general, comprehensive and secure manner to support both obj-a&b discussed above. Some examples on the applications and coverage for future development of the new architecture (in Chapter 5) show that it can provide more service flexibilities over existing approaches. Moreover, because the control is not limited to local entities, it also opens the possibility of better service coordination with external entities. In addition, the research proposes a methodology using the test questions to identify and categorize the type of parameters suitable for cross-layer manipulation.

Details of the organization of the dissertation are explained as follows:

Chapter 1: Introduction

This chapter introduces the history of the development of the conventional TCP/IP architecture, and how the principle of layering has facilitated the development of the Internet. This chapter also explores the recent changes to the Internet world, which makes the requirements of the layering principle too stringent. Finally, the objectives of the research are specified, which focuses on addressing the above mentioned problems.

Chapter 2: Limitations of Conventional TCP/IP Architecture and Related Works

This chapter first explores the limitations of the layering principle in today communication environment, which prevent high layers from synchronizing their operations with the condition of lower ones. Because of the development of new services and hardware, there are cases where these limitations are not preferable. The chapter then explores results and also limitations the existing works that try to overcome the inadequacy of the TCP/IP layering principle. Some of these works focus on making changes to the architecture to adapt to a new feature on a case-by-case basis. Others,

although of cross-layer information exchange approach, only support (obj -a) because they are not designed to identify an individual protocol instance or to support a secure way to access and alter the value of a parameter.

Chapter 3: InterLay Model for Cross-Layer Information Manipulation

This chapter provides the detailed design of the new InterLay model for TCP/IP architecture. The InterLay model will have to support both objectives explained above.

First, the reasons to use Object-Oriented (OO) Technology as the tool to analyze and design the new architecture are provided. The main advantage for using OO Technology in the InterLay model is that individual protocol instances are main players, implementing the protocol as an independent entity (i.e. object in OO Technology) makes it easier for the protocol instances to maintain their states as well as issue request to or react to request from other layers.

Next the detailed design of the InterLay entity which is consisted of three distinct functional groups, namely the Policy Element (PE), the Enforcer and the Informer, is explained. The PE is the checkpoint to authorize requests that can potentially affect the TCP/IP protocol stack (namely the update of paramter‘s value, executing a networking protocol procedures and registering for an event), as well as to authenticate the request from external entity. The Informer is in charge of returning the current value of the parameter, as well as informing the requester of the occurnence of a registered event. The Enforcer carries out the actual update or alternation of parameter value, as well as executing networking procedures. The InterLay model uses the Enforcer to support (obj-b) described above. The Enforcer also contains security measures to safeguard its actions.

This chapter also provides the specifications of new system calls that allow the user application to control the underlying NS. Finally, the interaction scenarios between InterLay and various entities are provided.

Chapter 4: Test Questions for Selection of Fine-Tunable Parameter List

This chapter explains the need to find all the right parameters which will assist the developers in the service and protocol development process which can save development time for cross-layer services. Test questions are defined and used as a method to find those parameters. The test questions are then applied to various protocols in each layers

of the TCP/IP protocol family. The parameters that are found are summarized for each layer.

One important aspect of the test question approach is that its methodology and results can be applied not only the InterLay but to any other systems that provide cross-layer control.

Chapter 5: Discussion and Analyses

This chapter first discusses the coverage of the InterLay model over the existing and potential future requirements toward the conventional TCP/IP architecture. It explores how InterLay supports mobility (including route optimization), fault-tolerance and insertion of SHIM Layer header. Advantages and benefits of the Interlay model are analyzed in comparison with related systems. The deployment strategies for the new architecture are also provided in two modes, namely disruptive and non-disruptive deployment. Some related issues on overhead in OO programming, performance and security are also discussed.

Chapter 6: Conclusions and Future Works

This chapter concludes the research, summarizing the major contributions of the research. One of the most notable contributions of the research is that the InterLay model is the only solution for cross-layer manipulation that supports the “write” operation of protocols’ parameters. As a result the InterLay model can support new features by just using the programming skill instead of requiring a new protocol to be developed. And as recommended in Request For Comment 1958: “Nothing gets standardised until there are multiple instances of running code”, the InterLay can be used in this sense to implement and monitor various aspect of a new feature, and the information obtained from this process can serve to speed up the development of the correspondent protocol. So the InterLay model can be used as a testbed to develop protocols for the TCP/IP architecture!

Future works to fulfill the potential of the proposed architecture are also suggested.

Lastly, some supplement information related to the operation of InterLay model as well as comparing its performance with and other proposals on maintaining TCP sessions over address changes are provided, analyzed and discussed in Appendix 1 and Appendix 2 respectively.

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

ADU	Application Data Unit
AE	Abstract Entity
AEPP	Abstract Entity Parameter Packet
AP	Access Point
API	Application Programming Interface
ASN.1	Abstract Syntax Notation One
CCID	Congestion Control Identifier
CCN	Content Centric Network
CEAL	Control Information Exchange between Arbitrary Layers
CLO	Cross-Layer Optimization
CN	Correspondent Node
COPS	Common Open Policy Service
DCCP	Datagram Congestion Control Protocol
DONA	Data-Oriented Network Architecture
ECN	Explicit Congestion Notification
ICMPv4/6	Internet Control Message Protocol version 4/6
IEEE	Institute of Electrical and Electronics Engineers
ILS	Inter-Layer System
IP	Internet Protocol
IPC	Inter Process Communications
IPSec	Internet Protocol Security

IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
MAC	Media Access Control
MH	Message Handler
MIB	Management Information Base
MIH	Media Independent Hanover
MIP	Mobile IP
MN	Mobile Node
MSS	Maximum Segment Size
NGN	Next Generation Network
NMS	Network Management System
NS	Networking Subsystem
OO	Object-Oriented
OS	Operating System
PCB	Protocol Control Block
PDU	Protocol Data Unit
PE	Policy Engine (of InterLay model)
PE	Protocol Entity (in CEAL)
PoA	Point-of-Attachment
QoS	Quality of Service
RFC	Request for Comments
RO	Route optimization
RTO	Retransmission Time Out
RTT	Round-Trip Time
SAP	Service Access Point

SCTP	Stream Control Transmission Protocol
SIB	Service Independent Building Blocks
SIP	Session Initiation Protocol
SMI	Structure of Management Information
SNMP	Simple Network Management Protocol
TCB	Transmission Control Block
TCP	Transmission Control Protocol
TOS	Type of Service
UDP	User Datagram Protocol

CHAPTER 1. INTRODUCTION

1.1. Research Background

At the inception of the Internet, the main concern was connectivity [1]. Therefore, much effort has been spent in designing a simple networking stack with the main objective of providing end-to-end data transfer capability.

Until the beginning of the last decade, the main access method to the Internet was wired-line with stable electronic signals, and types of end devices were mainly low-speed desktops which are stationary and not suitable for processing complex supplementary networking activities. This means that conditions of the underlying networking subsystem (NS) in end hosts are rather stable and there is no meaningful control of the NS for end user communication applications to carry out. As the kernel hides the underlying workings of the networking subsystem, communication application development is rather simple. The application just requests to open the connection and the rest are managed by the subsystem without the knowledge of the application. Since services were composed of non-real time and mainly static and low volume content, namely text and low resolution pictures which have no stringent requirements on the performance of the NS, there is no need for end user applications and different protocols within the NS to interact regarding the current status or adaptation to future changes of the network.

The above conditions result in the original design of the Internet using the layering principles, in which a protocol in a certain layer keeps its functions and internal states to itself, while basically providing only the service of sending/receiving Protocol Data Unit (PDU) to adjacent layers. Each protocol will try its best to perform its duty without relying on or finding the conditions of other layers.

The main advantage of this layering architecture is that it facilitates the incremental development and improvement of communication services, because it helps localize the

scope of change to a single layer, making it easier to find, try and implement improvements or corrections in each layer individually without the need to concern about the effects from/to other protocols. This makes the development of protocols and especially user applications simple, because all the applications have to do is to send the data to the networking subsystem to transmit to the other end, without concern for controlling or configuring the underlying NS.

This localization of changes and incremental developments were very important in the early development stage when there was almost no prior information on how the architecture would behave in different settings and environment. For example, as the network experiences various new types of communications services and networking technologies, the TCP congestion control algorithm has been refined many times but the protocols in other layers remained intact. Another example is when IPv4 experiences the depletion of address space and other limitations, the next generation IPv6 is developed and introduced without requiring any changes to both higher layers and lower layer.

However, over the last decade the Internet has undergone a great deal of change. A plethora of Wireless access technologies has been introduced and Wireless access is becoming the prevailing choice for last-mile access. Together with this development, some mobility-related problems introduced with Wireless access and mobile devices, such as the change of IP address and disruption due to handover, cannot be solved without modification to the conventional IP networking architecture. As discussed in various research works such as MIH in [8] or MIP in [9] and [10], in the new networking environment, a communication protocol of a layer now needs more support from other layers to better fulfill its tasks.

At the same time, with the introduction of various delay- and lost-sensitive applications like multimedia or gaming, applications now need to be able to customize and/or fine-tune lower layers to best suit their needs. On the other hand, mobile terminals now are mostly equipped with fast processors that can easily support more complex processing at the networking subsystem without negatively affecting the performance of end user applications. These developments of the Internet paradigm can be summarized in Table 1.

Attribute	*Conventional Internet*	*Advanced Internet*
Characteristics	Access technology, end devices, and services are fixed and/or static	Wireless and handheld devices create mobility-related issues. Also contents become more bandwidth-consuming and diversified.
End devices	Big, heavy and poor performance desktops and work stations	Small, light but powerful smart phones and tablets. They also carry multiple hi-res cameras, relatively hi-res and multi-touch screen, with gaming capabilities
Access technologies	Access technologies are wired cabling such as PPP over telephone line, leased line, Frame Relay, xDSL, and rather slow speed and very difficult to deploy.	Wireless technologies come in variety of choices such as Wireless LAN (Wi-Fi a/b/g/n), Wireless MAN (Wimax) with fast speed (54Mbps ~) and very easy to deploy.
Contents and Traffic	Contents are rather homogenous and static such as text and low-resolution images. The traffic for a session is low.	Contents come in heterogeneous media, with dynamic content and with very high volume traffic (HD videos or high-resolution images etc.)
Services	Mainly text- based informational services such as email, IRC, bulletin board etc…	Plethora of real-time interactive multimedia services such as online-video, high definition multiplayer interactive games …

Table I. The development of the Internet paradigm

While the original focus on connectivity of the conventional TCP/IP networking implementation facilitates very well the phenomenal development of the Internet, all of the developments in Table 1 indicate it is time to allow the modification of current and/or the creation of new TCP/IP architecture with more openness to cross-layer communication to meet quickly evolving communication demands and there have already existed necessary and sufficient conditions to realize this architecture in terms of service evolution, end-device performance and access technologies.

Moreover, the Coupling Principle [2] states that as things (in this context they are communication services) get larger, they often exhibit increased interdependence between components (here components are layers). As the TCP/IP architecture has already matured and been tested thoroughly (even IP next Generation, IPv6, is being rolled out in large scale), it is reasonable now to reduce the rigid requirements of the strict layering principle, and allowing layers to expose more internal data/states to other layers to provide user applications with more flexibility and customization.

1.2. Research Objectives

As explained later in chapter 2, in existing approaches whenever a new development requiring a new capability from the TCP/IP networking architecture, some solutions will be proposed to specifically address the problem, normally through making changes to the architecture. As these problems are carried out on a case-by-case basis, and not compatible with each other, namely the changes made for a problem cannot be used to solve later problems.

There are also some proposals on general cross-layer information exchange scheme to provide a broader range of service optimization and customization. In general, there are two objectives for a cross-layer system:

(obj-a) To allow exchange of information and possibly commands among layers so that a protocol instance of a layer can harmonize its activities with the condition of other layers.

(obj-b) To allow for the safe update (alternation) of a protocol's parameter so that the internal state of a protocol can be altered and adapts to the changes of external environment.

While (obj-a) can optimize the performance of a protocol in particular or the whole system in general, (obj-b) allows the protocol to change its working environment from one setting/configuration to another, which in many cases means the protocol has evolved to another protocol. However, because the settings/configurations can be different from session to session, (obj-b) requires that the cross-layer system to be able to identify and access an individual instance of a protocol (in the case the protocol has

several instances running in the networking subsystem). Because existing works are not designed to meet this requirement, they are not able to support (obj-b).

The purpose of this research is to propose a new TCP/IP networking architecture called the InterLay model that can facilitate the sharing and manipulation of information across layers' borders in a general, comprehensive and secure manner to support both obj-a&b discussed above. The architecture includes the core TCP/IP stack and a separate InterLay entity that provides the cross-layer manipulation capabilities to the various protocols in the TCP/IP networking architecture. To implement the Interlay, the OO programming will be used as designing tool. In addition, a special methodology (namely using test questions) will be used to define right parameters to be exposed for cross-layer manipulation.

In the new model, cross-layer manipulation of protocols' information will have the following characteristics:

- Allowing the access to and modification of not only control information (i.e. networking parameters) but also the actions of the networking protocols. The new model has to be able to access an individual instance of a protocol and alter its parameters without affecting other instances to support (obj-b).
- The provision of these manipulations is not limited to local entities but also available to external servers
- There will be many check points to apply authentication, authorization, and integrity testing procedures to provide more secure and reliable operations.

With such characteristics, the new model helps overcome some limitations and shortcomings of the conventional TCP/IP implementation and those that of existing works on a case-by-case basis, and would bring about some extra benefits including:

- Supporting faster service development by covering new developments in a single model
- Allowing the "intelligent use" of underlying network status and functions by the end-user applications.

- Its principles and methodologies are general enough to be extended to other networking models that use layering approach.

1.3. Structure of the Dissertation

This dissertation consists of six chapters: Chapter 1 introduces the research background and objectives; Chapter 2 describes the current limitation of the conventional TCP/IP and related works to address the problems; Chapter 3 discusses in details the InterLay entities; Chapter 4 explains the methodology to find the parameters to be exposed to external protocols; Chapter 5 discusses and analyses various aspects of the proposed architecture, and Chapter 6 concludes the research by summarizing main contributions of the dissertation and suggesting some future works to further develop the scheme.

Details of the organization of the dissertation are explained as follows:

Chapter 1: Introduction

This chapter introduces the history of the development of the conventional TCP/IP architecture, and how the principle of layering has facilitated the development of the Internet. This chapter also explores the recent changes to the Internet world, which makes the requirements of the layering principle too stringent. Finally, the objectives of the research are specified, which focuses on addressing the above mentioned problems.

Chapter 2: Limitations of Conventional TCP/IP Architecture and Related Works

This chapter first explores the limitations of the layering principle in today communication environment, which prevent high layers from synchronizing their operations with the condition of lower ones. Because of the development of new services and hardware, there are cases where these limitations are not preferable. The chapter then explores results and also limitations the existing works that try to overcome the inadequacy of the TCP/IP layering principle. Some of these works focus on making changes to the architecture to adapt to a new feature on a case-by-case basis. Others, although of cross-layer information exchange approach, only support (obj -a) because they are not designed to identify an individual protocol instance or to support a secure way to access and alter the value of a parameter.

Chapter 3: InterLay Model for Cross-Layer Information Manipulation

This chapter provides the detailed design of the new InterLay model for TCP/IP architecture. The InterLay model will have to support both objectives explained above.

First, the reasons to use Object-Oriented (OO) Technology as the tool to analyze and design the new architecture are provided. The main advantage for using OO Technology in the InterLay model is that individual protocol instances are main players, implementing the protocol as an independent entity (i.e. object in OO) make it easier for the protocol instances to maintain their states as well as issue request to or react to request from other layers.

Next the detailed design of the InterLay entity which is consisted of three distinct functional groups, namely the Policy Engine (PE), the Enforcer and the Informer, is explained. The PE is the central point for receiving commands on the local networking parameters (including looking-up or updating the value) and networking procedures. The Enforcer carries out the actual update or alternation of parameter value, as well as executing networking procedures. The InterLay model uses the Enforcer to support (obj-b) described above. The PE and the Enforcer contain security measures to safeguard the update of the parameters. Finally, the Informer is in charge of informing the requester of the value of the parameter (i.e. returns the result for the look-up request), as well as informing the requester of network events.

This chapter also provides the specifications of new system calls that allow the user application to control the underlying NS. Finally, the interaction scenarios between InterLay and various entities are provided.

Chapter 4: Test Questions for Selection of Fine-Tunable Parameter List

This chapter explains the need to find all the right parameters which will assist the developers in the service and protocol development process which can save development time for cross-layer services. Test questions are defined and used as a method to find those parameters. The test questions are then applied to various protocols in each layers of the TCP/IP protocol family. The parameters that are found are summarized for each layer.

One important aspect of the test question approach is that its methodology and results can be applied not only the InterLay but to any other systems that provide cross-layer control.

Chapter 5: Discussion and Analyses

This chapter first discusses the coverage of the InterLay model over the existing and potential future requirements toward the conventional TCP/IP architecture. It explores how InterLay supports mobility (including route optimization), fault-tolerance and insertion of SHIM Layer header. Advantages and benefits of the Interlay model are analyzed in comparison with related systems. The deployment strategies for the new architecture are also provided in two modes, namely disruptive and non-disruptive deployment. Some related issues on overhead in OO programming, performance and security are also discussed.

Chapter 6: Conclusions and Future Works

This chapter concludes the research, summarizing the major contributions of the research. One of the most notable contributions of the research is that the InterLay model is the only solution for cross-layer manipulation that supports the “write” operation of protocols’ parameters. As a result the InterLay model can support new features by just using the programming skill instead of requiring a new protocol to be developed. And as recommended in Request For Comment 1958: “Nothing gets standardised until there are multiple instances of running code”, the InterLay can be used in this sense to implement and monitor various aspect of a new feature, and the information obtained from this process can serve to speed up the development of the correspondent protocol. So the InterLay model can be used as a testbed to develop protocols for the TCP/IP architecture!

Future works to fulfill the potential of the proposed architecture are also suggested.

Lastly, some supplement information related to the operation of InterLay model as well as comparing its performance with and other proposals on maintaining TCP sessions over address changes are provided, analyzed and discussed in Appendix 1 and Appendix 2 respectively.

CHAPTER 2. LIMITATIONS OF CONVENTIONAL TCP/IP ARCHITECTURE AND RELATED WORKS

2.1. Limitations of conventional TCP/IP architecture

The ARPAnet (predecessor of the Internet) began with just one core protocol, the TCP (which stands for Transmission Control Program), which was formally described in Request for Comments 675 in 1974. In 1977, it was proposed that TCP should be further divided in a layered and modular way into two protocols, one serves as host level end to end transport protocol (the TCP layer), and the other for routing packets through the network to the destination (the IP layer) [5]. The result was the creation of the TCP/IP architecture using layering principles [1].

The disadvantage of layer enclosure is that except the Protocol Data Unit (PDU), higher layers have virtually no status information from lower layers therefore they have to accept general assumption that lower layers are doing their jobs well, without knowing *how well* the lower layers are doing their job, or whether any critical changes have happened to the lower layers. This prevents the higher layers from choosing the operation mode that is most appropriate to the current condition, which hinders the development of more flexible, optimized and customizable applications.

The conventional model supports well simple services such as email, news group, World Wide Web pages or real-time IRC and multimedia. However, for flexible or reliable services, such as fault-intolerant session-based applications, service developers must rely on special and often very complex mechanism [13][16][25][26] due to the almost zero support from the networking stack. Things get even worse when less reliable wireless access technologies, together with it are nomadic and packet loss-related issues, become widely available. The TCP/IP architecture was originally built on the assumption that the terminal's access point to the network was static, with stable electrical signals,

therefore IP addresses are used for both routing and identification, and loss is due mostly to congestion.

For example, in Mobile IP, the cross-layer information of a prominent L2 handoff from the Data-Link layer helps IP layer to prepare for the handoff to a new Foreign Agent in advance, so that the handoff process is faster and possibly seamless [6]. Another example is that due to data-hiding, buffer size between layers were not synchronized, therefore small chunks of data were being written to the TCP buffer, which led to sub-optimal performance [7]. If there is a way for the application to be informed that the TCP layer is constantly experiencing congestion, it can cooperate by reducing its transmission rate for example by using a slower codec.

On the other hand, the lower layer such as the Data Link layer can make a better decision when performing a handover if it knows the preferences of the above applications. However, in the current TCP/IP architecture, there is no means for higher layers to influence the operations of lower layers.

Moreover, the encapsulation of information among different layers also leads to redundancy of information and operation. An obvious example is the proposal of the IPv4 header compression technique [36] [37]. This shows that the conventional of separation among layers create a large ratio of overhead in some popular applications. Also, because of the separation between IPv4 and Transport layer, the checksum operation of IP header and Transport header is done separately, which can be a big overhead. The same line of optimization techniques are [45] [46].

The above cases indicate that it might be beneficial for higher layers to obtain and control information from lower layers, to optimize as well as to provide seamless operations to end-user communication services.

2.2. Related works

There are two way to copes with the changes explained above to the Internet. The first approach, called the evolutionary approach, tries to solve new issues by modifying a certain protocol or added a new protocol, while keeping the original TCP/IP architecture intact. The second approach, called the clean-slate approach, tries to develop a new

architecture that take into account all the changes in networking environment, to replace the current TCP/IP protocol stack.

We will look into each approach in the following subsections.

2.2.1. Evolutionary approach to address issues of the TCP/IP architecture

Cross-layer exchange of protocol data has been extensively researched to optimize the utility function of end to end throughput in ad-hoc network [19] [20] or optimize the exchange of information and conserving energy in sensor network [21] [22].

Ad-hoc network related approach generally combines information from different layers to coordinate the transmission among peers to maximize the utility function for all participants. For example in [19], the authors propose a practical cross-layer optimization (CLO) design that take into consideration some components namely source rate control, hop-by-hop flow control, MAC scheduling and prioritization, link-aware and congestion-aware routing to maximize the utility function of the whole network close to theoretical level.

An optimization agent is proposed in [21] in Wireless sensor network, as a medium for layers to communicate. It contains a database to store essential information about the network condition such as node identification number, hop count, energy level, link status. The information will be accessible and used by protocols in all layers to optimize its operation for parameters such as transmit power, coding rate or data rate transmission to suit a specific application. The research in [22] proposes a new sensor network architecture called X-Lisa, which standardizes cross-layer information-sharing and organizes the information shared between layers. In X-Lisa, protocols are provided with status of active queries in the network so that they can adapt their behaviors accordingly, which improves the overall performance.

In [23] the authors investigate the combination of APP-MAC-PHY layers to find optimal modulation scheme for multimedia data, as well as to optimize power consumption.

The research in [52] uses the information that is conventionally closed to Layer 2 to provide application with more information on the condition of the access link, so that

user can have a better service experience. However the limitation of [52] is that it utilizes only the support from Data Link layer, and it is designed for use with SIP services.

Traditionally the TCP layer does not change its static information (e.g. the address) during its operation. However, the research in [108] shows that by letting an external entity change the configuration of the TCP protocol (in this case the IP address) it can overcome mobility constrains that normally would tear down the connection.

Apart from case-by-case solutions, there are also some proposals on using a cross-layer information exchange approach as a general solution to support a wide range of service optimization through learning and harmonizing among protocols at different layers.

Media Independent Hanover (MIH) architecture [106] is being developed by the IEEE 802.21 Working Group to enable the smooth handover between IEEE 802 technologies and other access technologies by introducing new function entities to allow higher layers to interact with layer 2 (namely IEEE 802 access technologies) during handover process.

One problem with MIH is that it supports only the interaction between higher layers and layer 2, not among higher layers. Therefore in [107] the authors propose the Control Information Exchange between Arbitrary Layers (CEAL) designed to provide similar cross-layer functionalities to that of Media Independent Handover, but not confining to layer 2 but extending to other layers as well.

The common weakness of MIH and CEAL is that they can not support the alternation of protocol's parameters because they are not designed to identify an individual protocol instance, or to support a secure way to access and alter the value of a parameter, so they can support only (obj-a) of a cross-layer system described in section 1.2 of chapter 1.

We can see that the existing researches using the evolutionary approach mostly examine the benefit of cross-layer design from the performance aspect by asking different layers to adapt themselves according to current status of the network. With a different broader perspective, the research in this paper focuses on providing and manipulation of information from lower layers so that applications not only be able to adapt their performance according to lower layers status, but they can also (i) make more choices beside performance (such as arbitration decision route optimization as explained

in section 5.1 below) and (ii) be able to support (obj-b) described in section 1.2 to adapt better with future changes (for example, in the MobiSocket proposal [18], if the approach of this research were adopted in the beginning of the TCP/IP architecture, then it would be feasible to maintaining of TCP session across IP address changes without difficulty).

2.2.2. Clean-slate design approach for the Internet

The Internet was designed with only the telephone network as a reference. Therefore the communication model in TCP/IP architecture is a conversational session between locations, i.e. the user has to know and connect to the storage location in order to retrieve the desire information content.

However, as the Internet evolves, techniques such as caching allows for a content to be available at several places, and the location known to the user might not be the closest or easiest to retrieve.

There are already some researches on how to find the most appropriate location for a resource.

Most existing works solve the problem by proposing to modify or establish an alternation of DNS. Data-Oriented Network Architecture (DONA) [109] involves a clean-slate redesign of Internet naming and name resolution. It replaces DNS names with flat, self-certifying names and a name-based anycast primitive above the IP layer. The content in DONA must first be published, or registered, with a tree of trusted resolution handlers (RHs) to enable retrieval, which provide the next hop information for the content. The requester then uses normal IP connection to retrieve the content.

A second approach is Content Centric Network [110], which is meant to replace TCP/IP as the transportation mechanism for the Internet. In Content Centric Network, content chunks are used as the basic information exchange unit, equal to the IP data packet in TCP/IP networking. When a certain content is first requested and returned to the requester, a copy of it is also temporarily stored at the intermediate node (i.e. router), and if another request for the same content arrives at the intermediate node while the copy is still cached, the copy is returned to the requester without having to traverse back all the way to the content originator. This mechanism can save a lot of network traffic for

content that is popular. And while CCN works well for non-real-time content, it can also support real-time conversational applications [111].

With regard to the research on cross-layer manipulation of TCP/IP architecture, the solutions of the first approach (namely DNS-based) still use TCP/IP to exchange information, therefore the research in this dissertation is still beneficial for these researches.

On the other hand, whether the second approach (i.e. CCN) will eventually replace IP networking is still an open question, because IP is still doing its job well. Moreover, even if the networking environments changes to the point that IP must be replaced by CCN, the sheer size of the infrastructure and application base will make the transition a very lengthy process. And because a new architecture does not necessarily suspend researches on existing architecture if the deployment of the latter is large, during the transitional process to CCN, the IP networking architecture still needs to be supported when new needs arise. On example of the old architecture needs to be supported during the transition to a new architecture is that even though IPv6 has been standardized since 1998, but until 2003 a RFC [31] is still being established to solve the problem that SIP experience that is specific to IPv4 only.

The discussion in [116] suggests that clean slate and evolutionary research are not at odds with each other, but clean slate networking research can help guide the evolution of the Internet. However, we would like to stress the reverse is also true, namely by studying the evolution process of the current Internet architecture, the future clean slate architecture can avoid problems that might arise in the future. As a more concrete example, as CCN also has a layering architecture, the InterLay can provide some useful insight on cross-layer to CCN, such as how to cope with link layer disconnections.

Therefore, the research in this dissertation will be useful for Internet networking for a long time to come.

CHAPTER 3. INTER-LAY MODEL FOR CROSS-LAYER INFORMATION MANIPULATION

In this part, we discuss the proposal of a new model for cross-layer TCP/IP layering architecture called *InterLay*. In InterLay model, protocol instances from lower layers will reveal selected internal information to higher layers, either adjacent or several layers away. The networking stack can also interact with external entities to receive external instructions in the form of policy exchange.

As the openness of network layers is unavoidable, it would be more beneficial to communication service developers if there is a way to provide cross-layer manipulation in a safe and protected way. The InterLay is proposed to fulfill this requirement, with several layers of protection (namely through the InterLay object and the intrinsic protection through the *get()/set()* method of object oriented technology) to eliminate the risk associated with opening the internal mechanism of the network protocols.

As a practical guideline for real-world development of the networking stack, the discussion of the new architecture will be based on the implementation of TCP/IP networking stack for Linux, as Unix-like systems are becoming more and more popular especially to mobile devices, and the fact that modern OSs are similar in capabilities so it can be extended easily to other platforms.

Moreover, the new architecture will be analyzed and designed using object oriented technology, and the reasons for this selection are explained in section 3.1 below.

3.1. Selection of object oriented design for the cross-layer architecture

Currently the TCP/IP networking stack is implemented in structured procedural programming fashion, where the operations are carried out in sequential procedures. Comparing to Object Oriented (OO) programming, procedural programming has smaller executable code and higher performance due to no overhead from object invocation.

Because accessing the network parameters has the potential of destabilizing the networking subsystems, the new model should be designed so that the possibility that network parameters of a communication protocol be mistakenly altered by other entities should be avoided as much as possible. However, keeping the existing procedural programming model, and adding codes to expose a layer's internal parameters will be risky as variables representing networking parameters are accessed directly, there is no way to guarantee the integrity of networking parameters even in a read-only procedure with structured programming, therefore unwarranted changes to the internal states of the networking subsystem might happen with unpredictable consequences.

If OO Programming is used then the *get()* method for accessing the attributes associated with the networking parameter will natively allow the exposure of internal data of a layer without the danger of (mistakenly) altering the attributes. Another aspect is that with conventional procedural programming, whenever the parameter is accessed or updated, the codes for basic protection mechanism (for authorization, integrity check etc.) will have to be repeated; while in OO programming, all of these basic protections are inserted only once in the *set()* and *get()* method for the parameter, and whenever the parameter is accessed or updated, all of these protection mechanism will be automatically applied.

So implementing the architecture in OO programming not only reduces the complexity of the implementation, workload and potential errors, but it also has the potential to reduce the size of the executable code.

There are also some other advantages in applying OO paradigm to the new cross-layer communication model as follows.

- The foremost advantage of implementing the InterLay model in OO is that because the model allows a protocol in a certain layer to interact with other protocols in different layers, implementing the protocol as an independent entity (i.e. object in OO) make it easier for the protocol instances to maintain their states as well as issue request to or react to request from other layers.
- The layering approach and OO technology have the same principles, namely self-contained internal attributes, interactions using pre-defined interface, the

modification of one entity does not affect other existing entities. As the data/operation of networking protocols are extensively analyzed and documented, converting TCP/IP layer to object should be straightforward.

- The new InterLay cross-layer manipulation architecture concentrates on data (namely networking parameters of networking protocols) to be provided cross-layer. This data-centric purpose is obviously compatible with OO Programming. Moreover, the data requires utmost discretion when accessed and modified, and this is already taken into account by OO Technology.
- The cross-layer model is dynamic, and is expected to be updated when new features or capabilities become available. To include a new feature, it is much easier to add an extra attribute or a method to a protocol class in OO Programming than to find the right place and right mechanism in procedural programming.
- For look-up (i.e. read-only) operations of network parameters' value, the OO programming ensures the possibility of inadvertently alternating the parameter to be reduced to zero, while for update (i.e. write) operations of network parameters' value, it provides as many protection and authorization layers as possible. So the utmost discretion required by data when accessed and modified has been already taken into account by OO Technology.
- The development process for OO programming is also smoother because protocols in a layer share many similar characteristics, therefore once the base class of a layer has been developed, the development of its derivative protocols can also be smoother via inheritance and polymorphism.
- As new protocols are being constantly introduced into TCP/IP to accommodate new communication requirements, the ability of OO to reuse common codes with inheritance and polymorphism will makes it easier when realizing these new protocols into real codes. For example, a common class for reliable transport layer protocol with all the common virtual methods (such as *bind(), listen(), accept(), connect()* … with the *connect()* method containing the virtual *hand-shake()* method) can be used as a template to develop the TCP and the newer

SCTP (Stream Control Transmission Protocol) of which the original methods will be overridden with the correct input parameter using polymorphism. This process can speed up the implementation process of new protocols.

- Moreover, as new protocols and features are constantly introduced, the networking subsystem will have to be actively and continuously developed and maintained for a very long period of time. The advantage of modulation of OO Programming creates a better documentation repository and makes the process of passing codes among programmers more smoothly.
- Because the query and update activities are carried out independently among classes, we can place each entity (in Figure 1 below) in a separate thread; and as multi-processor CPUs are popular nowadays, each object can be executed in a separate processor which will improve the overall performance.
- By using OO design translation tools, as well as consulting existing OO reference framework for protocols such as one described in [9], the implementation of this new layering architecture would be made easier.

For the above arguments, we will use OO technology in the analysis and design of the InterLay cross-layer enabled networking architecture.

3.2. The Cross-Layer communication model for TCP/IP networking architecture

The new model is composed of two parts: the InterLay object handling the manipulation of parameters across layers, and the conventional TCP/IP protocol stack handling the actual sending/receiving of data.

The overall system of the new TCP/IP architecture is depicted in Figure 1. The model also needs some supplement entities such as buffers for PDUs which are not depicted in the figure for simplification.

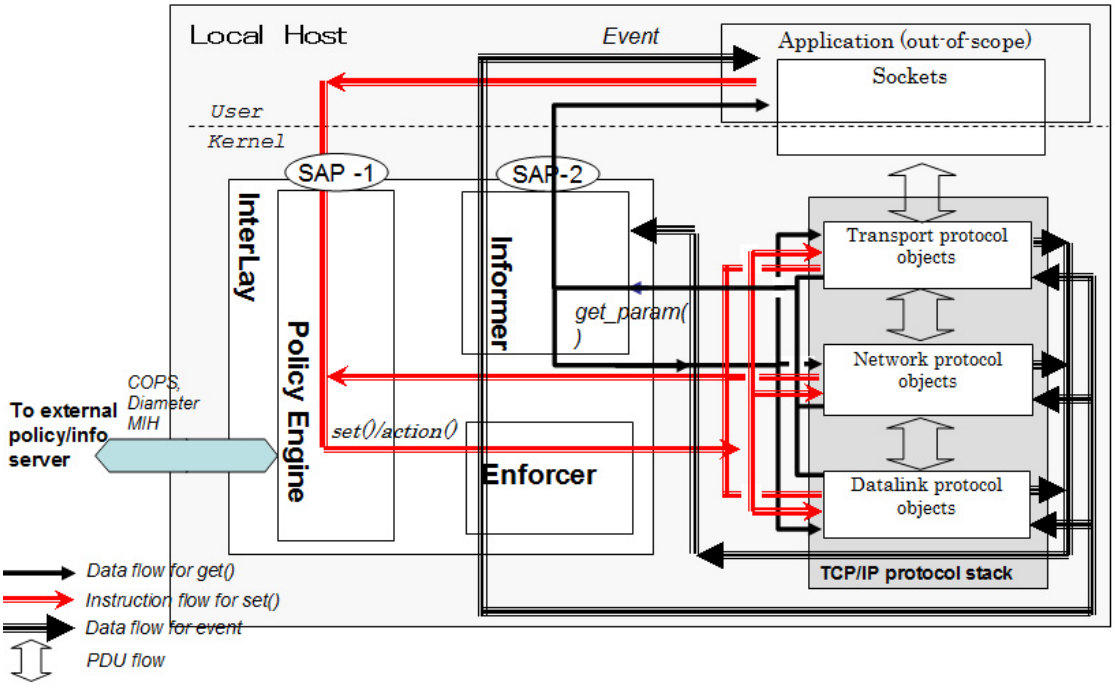

Figure 1. The new TCP/IP architecture

3.3. The conventional TCP/IP stack

In the InterLay model, the conventional TCP/IP protocol stack fulfills the fundamental job of the networking subsystem, namely sending user application data to remote host, and receiving data from remote host to user application.

When implementing in OO programming, each layer will be represented by a generic virtual class with all the basic parameters and functionalities of that layer.

The networking parameters of that layer will be implemented as the corresponding attributes of the class. In the InterLay model, these networking parameters are called real-time parameters (more details on real-time and event parameters are discussed in section 3.5). Note that while the class contains all parameters of the protocol, in this research we only concern the parameters that should be revealed across layers.

The functionalities performed by the layer become the methods for its corresponding class. The methods of a layer class either contain some generic processing that is common for all protocols of that layer (and will be reused when an actual protocol class is developed), or it might only be the place holder (i.e. implemented as virtual functions) to be replaced by an inheriting class representing a specific protocol.

To enable reusing of code, the actions performed by these functionalities would be decomposed as much as possible into atomic components, and a certain functionality of the layer can be constructed by making use one of these components.

The class that represents a protocol of a layer may inherit directly the generic class representing the layer, or inherit the corresponding layer via a hierarchy, with additional attributes and methods corresponding to parameters and functionalities specific to the protocols. In the case of hierarchical inheritance, an intermediate common class that contains the shared common major properties of the protocols of the same family, and a specific protocol will inherit the layer class via this intermediate class. For example, the IEEE 802.11 a, g family may be mapped into an immediate class called IEEE80211, with all the common attributes, except, for example, speed, frequency, modulation techniques etc.

For Data Link layer, the object might be designed as a wrapper object of the device driver. For the application layer, as the applications interact with the networking subsystems via the socket, the application layer of the TCP/IP stack will be represented by the socket class.

In the InterLay model, real-time parameters of a protocol that meet the requirements explained in chapter 4 will be exposed to be queried or updated by other entities. An extra *get()* method is implemented for the parameter that is opened for querying the value, while the *set()* method is implemented only if the parameter is opened for updating by other entities. The *set()* and *get()* method will be implemented in the class that represents the protocol, not in the class that represents the layer.

Apart from real-time parameters, certain functionalities of a protocol will be exposed via the InterLay to other entities, by letting these entities to invoke the methods corresponding to those functionalities. The methods that are exposed to other entities are called *action()* methods (more detail discussion of *action()* method is discussed in section 3.6).

3.4. On the extra functionalities of the TCP/IP stack

When implementing in OO programming, the existing procedural code of the networking subsystem (i.e. the TCP/IP stack) can be reused, for example the TCB

(Transmission Control Block) can be reused as the main structure to construct the TCP protocol class, and code and algorithms which perform the processing of TCP protocol can be reused in the *action()* method of the TCP protocol class.

In addition to the above basic functionalities, comparing with the conventional TCP/IP stack, the TCP/IP stack of the InterLay model should be equipped with the following two extra groups of functionalities:

- Group A relates to exposing intrinsic capabilities of the protocols of the TCP/IP stack, including functionalities for querying networking parameters or actions that the protocols will perform according to standardization documents of the protocols.
- Group B provides extra functionalities that are not required or specified by the standardization documents. These functionalities will allow more flexibility and customization for communication services.

Specifically, Group A includes the extra functionalities as follows:

- An extra *get()* method for each real-time parameter, as well as a *set()* method if the parameter is allowed to be updated by other entities.
- In the InterLay model, the lower layers (i.e. Layer 2 to Layer 4) can also make requests to the InterLay to learn and manipulate the status of real-time parameters and events or to execute *action()* methods. As the original documentations of the TCP/IP protocol stack do not dictate these new functionalities, if the protocols from lower layers want to utilize InterLay to manipulate protocols from other layers, they have to be equipped with extra codes to explicitly make use of InterLay.

For Group B, because the purpose of the InterLay model is to manipulate the operation of the TCP/IP stack for service flexibility and customization, the TCP/IP stack should also provide the InterLay with more control in terms of management aspects of TCP/IP protocols objects themselves, or their main task (i.e. the sending/receiving of data), including:

- The ability to protect or destroy protocol objects arbitrarily by other entities. This feature is used, for example, to protect TCP session from automatically being terminated in order to support TCP mobility as described later in section 5.1.2.
- The ability to start or stop processing incoming and outgoing data for an individual session at each layer, namely at the IP object or Transport Layer object (such as TCP or UDP). This provides the ability to *freeze* a Transport Layer which is important when a connection is being restored, transferred or handed-off, as described in section 5.1 and 5.2.
- A similar ability to the above is the ability to accept or reject certain types of PDU. This can be used to provide security-related functionality.
- The ability to append/prepend extra header before or after a certain PDU.

Moreover, because InterLay allows more flexible control of the TCP/IP stack, in order to make better use of the InterLay model, Group B also includes the ability for other entities to make choices whenever alternations are available in TCP/IP stack. For example, many congestion control algorithms have been developed for TCP, each of which is suitable for a certain type of network link, such as algorithms for wireless access technologies [75] or satellite link [76-77]. However, in current TCP/IP implementation the congestion control is predefined and fixed, and the user application cannot choose the most suitable algorithm for its TCP session. If the TCP protocol object provides an *action()* method that allows the user application to choose the algorithm (for example a method with the name of *cc_select(algorithm to use)*) then better performance of TCP session might be achieved. The implementation of such an example can be done either by implementers of the TCP/IP stack in anticipating of its usefulness in an ad hoc manner (i.e. without specification from standard documentation), or officially by altering the concerned standard documentation for the protocol.

All of these functionalities will be used to provide some new customization and flexibilities for TCP/IP networking via the InterLay model as explained in section 5.1.

3.5. Real-time vs. event parameters

The InterLay model defines and separates two types of information regarding the condition/status of a protocol that are of interest to other partners: real-time and event.

3.5.1. Real-time Parameters

Real-time parameter corresponds to parameters of a networking protocol, and in this research we will concern only those of which immediate value is of importance (i.e. should be read or updated by other entities).

As stated in 3.3, this kind of network parameter for a layer (and its specific protocols) will be implemented as a corresponding attribute of the corresponding class; therefore in this research attribute and network parameter are exchangeable: attribute is the representation of the network parameter of a protocol in the corresponding protocol class. The attribute value will be modified and retrieved using the associated set() and get() methods. The attribute stores the value of the corresponding network parameter that is used during the operation of the concerned protocol. For example the TCP class will contain, among others, a smoothed Round-Trip Time (RTT) (called *rTTT*) attribute that stores the current value of the smoothed RTT value.

Real-time parameter can either be static or dynamic. An example of static real-time parameter is the IP address or port number of the source or destination of a connection and in normal condition the value of this type of parameter stays unchanged. Dynamic real-time parameter can be, for example, the current sequence number of a TCP connection, where the value of such parameter will be updated according to the progress of the communication session. However the application developer should bear in mind that for fast changing dynamic real-time parameter, the value may already be obsolete when it reaches the requester, so that value is true only sometime between the time of sending the request and the time receiving the reply, but not the value of the parameter at the time the reply reaches the requester.

3.5.2. Event Parameters

In contrary to real-time, event parameters can not be really be read at an arbitrary time, and normally do not represent a specific value. Rather, it indicates that a critical point has

been reached inside the protocol, which will potentially change the behavior of the protocol and when other entities are informed of the event they can react accordingly.

While the real-time parameter exists for a certain period of time (normally for the entire lifetime of the owner protocol), an event will appear only at a specific point of time, and normally it will cause a series of reactions at the owner protocol, and these reactions can be implemented as *action()* method of the protocol class.

Even though it does not represent a value but rather the start of a reaction process, an event is still classified as a parameter in the InterLay model, because it can also be requested by and notified to other networking protocol objects or applications just like real-time parameter.

An event can be associated with a real-time network parameter, but in this case the importance is not in the immediate value, but whether that value has crossed above or below a certain value which in general will generate an event in real world. The concerned network parameter will be represented by two entities in the corresponding protocol class: one is the real-time attribute that stores the immediate numeric value, and the second is the event that is generated when the value is over/under a certain threshold.

One example of this kind of event is the Retransmission Time Out (RTO) event, which is based on the absolute value of the round-trip time (RTT) of a packet. The intermediate value of an RTT is not important even if it varies greatly from packet to packet. The absolute value of the RTT is taken into consideration only if it is greater than that of RTO. In this case, the TCP protocol will start the RTO loss recovery by setting the congestion window (`cwnd`) to 1. But even when the timeout event happens, the exact value of the absolute RTT is of no importance, but what's important is how the system reacts to such event.

Event parameters can also be received externally, such as the Explicit Congestion Notification (ECN) received from the communication peer, which indicates that the network experiences congestion in the *Tx* direction. The event might also come from policy exchange with external entities on the network.

3.5.3. Implementation of real-time and event parameters

The type of a parameter (i.e. real-time or event) will decide how it is implemented in the cross-layer communication model.

For real-time parameter, the value of real-time parameter is that of the corresponding attribute of the protocol class, namely the returned value of the associated *get()* method. Because a read-only operation does not alter the status of the networking subsystem, when a look-up request for the current value of the parameter is made to the InterLay object, it will return the direct reading of the parameter (or in some cases through some conversions) without the need for any authorization.

On the other hand, updating the value of a network parameter is potentially dangerous to the stability of the networking subsystem, and may cause troubles to it if the update is not implemented correctly. Therefore, when the InterLay object receives a request to fine-tune (i.e. modify) a certain parameter (i.e. attribute), it will have to perform all necessary permission and integrity checks to guarantee that the update will not negatively affect the networking subsystem or other processes that are relying on the networking subsystem. Moreover, for parameter of which value is obtained through probation or negotiation such as MSS (Maximum Segment Size), its value should not be modified directly but should be carried out through a suitable re-execution of the probation/negotiation process.

The procedure to interact with event parameters will be completely different from that of real-time. Firstly, because the event does not correspond to a real value, it is not possible to issue a read or write operation on the parameter (i.e. the parameter is not associated with the *set()* and *get()* method). To the networking subsystem the occurrence of an event is equal to the execution of the series of reactions associated with that event.

Secondly, because the exact moment that an event occurs cannot be predicted beforehand, protocol objects from other layers cannot request to “get” it at an arbitrary time. Instead, it has to register in advance to be notified of the event, and the registration might be associated with a certain lifetime which is the period of time that the requester is interested in the event. As a result, the event parameter in the InterLay object is implemented as a notification list, and the list stores a *handler* for each and every

requester that registers for the event (more on the operation of the notification list is discussed in subsection 3.7.4). The handler might be a means to inform the requester of the occurrence of the event, or the actions that the requester wants the InterLay object to perform when the event takes place.

And how the requesting entity is informed of the event will depend on whether the requester resides in the user space or kernel space. This will be further discussed in subsection 3.7.4 and 3.9.4.

The InterLay object also contains a list of predefined reactions, and the entity that registers for an event can request the InterLay object to perform some of these reactions at the time of the occurrence of the event.

In the InterLay model, each parameter (both real-time and event) will be given a unique predefined ID (for example of data type DWORD), and the InterLay object will use this ID in the incoming requests to identify the concerned parameter.

3.6. The action() methods

As explained in 3.3, a protocol class will contain many methods. In this research, we divide them into two categories: the *auxiliary* methods and *action()* methods.

The *auxiliary* methods do not directly fulfill the functionalities of protocol as specified by standard documents for that protocol.

On the contrary, *action()* methods are those that represent the actions or procedures that a protocol performs to fulfill its duty as specified in standard documents. For example the Mobile IP class contains, among others, an *RO()* method that once called, it will perform route optimization procedure to a given destination. The TCP class contains, among others, the *timeout_recovery()* method that once called, it will set the `cwnd` to 1 MSS and change the congestion control state to slow-start.

In the InterLay model, apart from the above real-time and event parameters, the *action()* methods can also be invoked by external entities as well. In InterLay, in addition to the *action()* methods that represent activities as documented for the protocol, a protocol class will also include a new type of *action()* methods that manipulate the PDUs in each layer, including changing the protocol headers before it is sent to the next

protocol entity in the data flow, or prepend/append certain information to the PDU. These methods are beneficial in the case the applications want to insert or remove a SHIM header into the data packet.

And because the *action()* method also has the potential to affect the operation of the system, their invocation will be discussed in more details in section 3.9 together with the *set()* method.

The *action()* method should be divided into atomic actions. Because of the philosophy of InterLay that application has the most knowledge about the needs of itself, an action() method should not call others when carrying out it duties, but such service logic will be carried out by the calling application. For example, in the service scenario described in section 5.1.2.1, informing of IP address change to the Corresponding Node is not called by the *set_PCB()* function of the TCP/IP stack but by the application at the Mobile Node.

In the InterLay model, each *action()* method will also be given a unique predefined ID (and for example of data type DWORD), and the InterLay object will use this ID in the incoming requests to identify the concerned *action()* method. The name domain for the ID of the *action()* method is different from that of the parameter, therefore an *action()* method and a parameter can be assigned the same ID.

3.7. The InterLay object

In the proposed model, all activities related to cross-layer communications will be handled by the object of the InterLay class. Through the InterLay object, protocol objects of any higher layers (i.e. not limited to the Application layer) or external servers can manipulate certain parameters and actions of protocol objects from lower layers.

3.7.1. Composition/structure of the InterLay entity

The InterLay should support the following requests toward the TCP/IP stack

(i). Query of value of real-time parameter

(ii). Updating value

(iii). Executing an action

(iv). Notification of event

As the action of (i) is of read-only type, there is no security risk associated with this type of action, and therefore in general it does not require any authorization.

On the other hand, as the action of (ii) alters the internal state of the TCP/IP stack, it has the potential of negatively affecting the working of the networking subsystem. Therefore there should be some authorization and integrity checking mechanisms to guarantee the safe operation of (ii).

The action of (iii) requires the TCP/IP stack to perform certain actions of a protocol, and these actions also have the potential to alter the state of the networking subsystem. For example, if the *RO()* action method of the IP protocol object is invoked, it might change how the packets for a certain IP session are routed in the Internet. Therefore similar to (ii) the action of (iii) also requires extra authorization, integrity checking and any other extra security measures deemed suitable.

Finally, the action of (iv) at first glance is similar to that of (i) (i.e. read-only action) however due to the fact that the notification of an event might be associated with extra processing, making it similar to the case of (iii), therefore these requests also need to be authorized like the case of (ii) and (iv).

From the above discussion, we can divide the functionalities of InterLay into three functional groups:

- Request handling functional group: this group receives request and performs necessary authorization activities upon receiving the request.
- Information gathering and provisioning: this group gathers information (value of real-time parameter or occurrence of event) and then returns the information in appropriate form to the requesters (i.e. the value of the real-time parameter or the notification/reaction of the event).
- Enforcing of value update and *action()* method execution: this group updates the value of real-time parameter and execute an *action()* method of a protocol.

In the InterLay model, these functional groups are implemented in the following three entities respectively: the Policy Engine (PE), the Informer and the Enforcer, as shown in

Figure 2. These three functional entities can either be implemented as a single object (i.e. in this case the InterLay is a real object) or as three separate and independent objects (i.e. in this case the InterLay is a nominal entity, representing the three objects: the PE object, the Enforcer object and the Informer object). The advantage of implementing as a single object is that the methods inside the InterLay object can invoke each other directly, without the overhead of locating the correct object as in the case of implementing as three separate objects. However, implementing as three distinctive objects helps localize the changes made to each entity and simplify the modeling process of the InterLay, and it will also simplify the implementation of threading for each functional groups, which can improve performance when each entity is programmed (using multi-threading techniques) to run on a single processor in the case the InterLay model is implemented in a multi-processors device (this is a very realistic assumption in the near future since currently quad-core CPU are being used even in portable devices). Apart from the advantage of easier multi-threading programming, dividing the functionalities of InterLay into groups can respond to requests from various sources in a unified and consistent manner.

As explained above, the Policy Engine (PE) functional group serves as the central point of collecting and authorizing requests from other local and external entities. Therefore the PE will contain the necessary authorization mechanisms that can be applied to different types of requests from various sources. The InterLay model proposes one authorization mechanism based on priority, which will be discussed in 3.7.2.

The requests that need to be authorized (namely those of type (ii), (iii) and (iv) above) are sent to the PE. For the requests to update real-time parameter or execute *action()* methods, after being authorized at the PE they are forwarded to the Enforcer which will update the value for the parameter, or executing the *action()* accordingly. The Enforcer may again validate the request, such as checking whether the new value is of valid type and range. On the other hand, requests to register for event are sent to Informer after being authorized, and the Informer, which serves as the central point for collecting data regarding real-time parameters and events, will respond to the registration request by acting accordingly whenever the concerned event takes place.

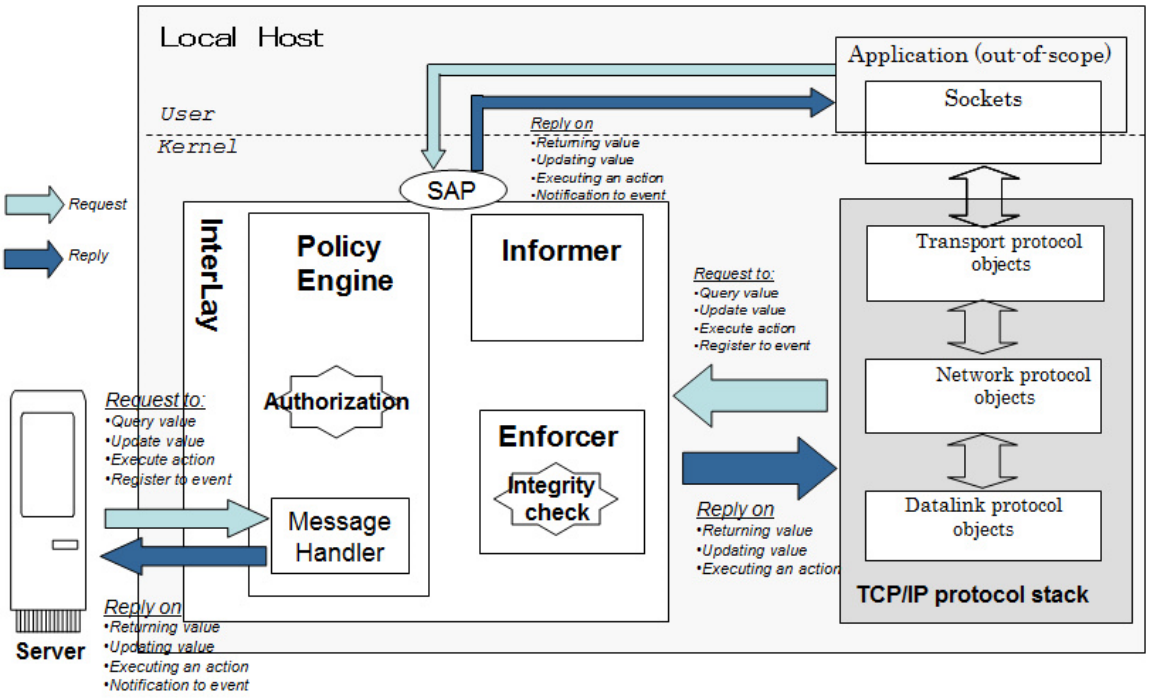

Figure 2. The InterLay Object

As all other requests go through the PE, it would look more logical for the PE to also handle requests to query value of real-time parameter, as it would give developers a more consistent requesting pattern. However, letting the Informer to serve this type of request directly has the following advantages:

- Firstly because the request for value is a read-only activity, it does not pose any security risk. Therefore it does not require the security checking specialty of the PE.
- Secondly, because queries for value of real-time parameters from local system (i.e. user applications and objects from the TCP/IP stack) are expected to have the highest rate, letting the Informer handling this type of request would share the load between the PE and the Informer, which might prove useful in the case both objects are implemented as individual objects and are running on different processors (the computing trend shows that in the near future multiprocessor end devices will be ubiquitous).

- Thirdly, in the case the request is handled by the PE, it is just an extra function call to the corresponding method of the Informer, therefore if due to possible future extension there is a need to transfer the role of receiving this type of local query operation between the Informer and the PE, it could be done trivially. Also, calling the query request through the PE introduces an extra function call, which clearly is an overhead.

In this research, the request for query of value of real-time parameter (i.e. the action of (i) above) is handled by the Informer due to the above advantages.

Finally, the recipients of the requests, namely the PE and the Informer, should be equipped with a rate control mechanism to protect the performance of the TCP/IP networking stack from being abused with excessive requests.

The three functional entities will be discussed in more details in the following sections. In this discussion, it is assumed that the PE, the Informer and the Enforcer are implemented as separated objects.

3.7.2. The Policy Engine

In more details, the Policy Engine (PE) performs the following functions:

- Receiving instructions/requests, providing the service of accessing and manipulating of internal parameters and actions of the local networking subsystem.
- Authorizing the requests from user applications, using the priority mechanism (described later) as well as authenticating requests that it receives from external servers.
- Dispatching requests to appropriate handling entities, namely the Informer and Enforcer.
- Returning the results of the request.

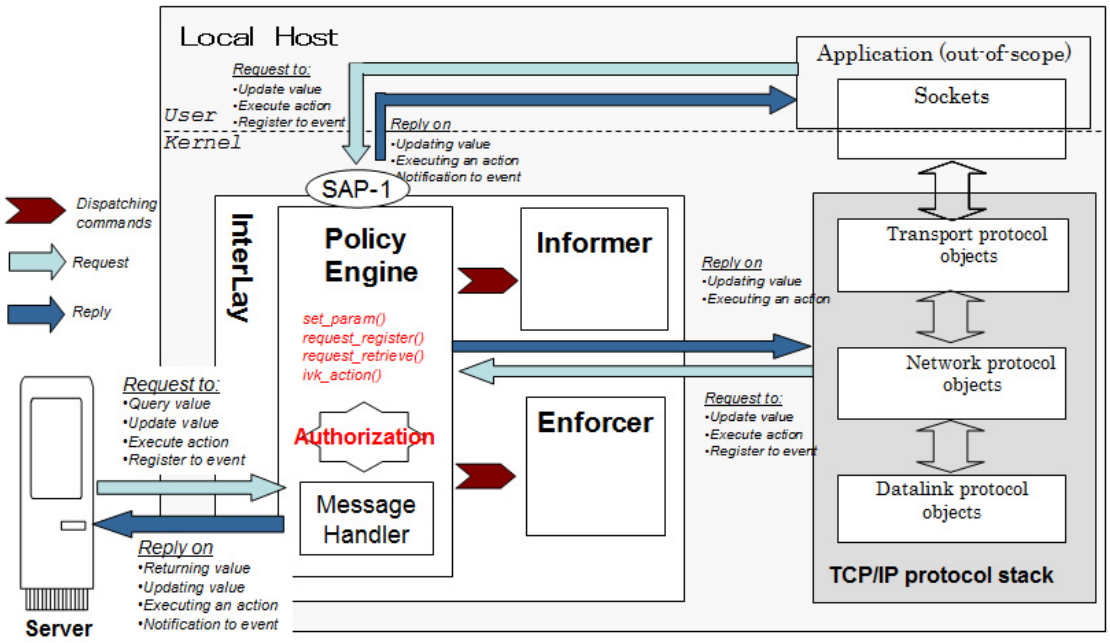

Figure 3. The Policy Engine

The PE accepts the requests from the follow three types of entities: end-user applications (in the user space), lower layer protocol objects (layer 2 to layer 4, in the kernel space), and external policy servers (outside the local system).

The PE contains four helper methods to deal with requests, namely (i) the *set_param()* method to process update requests for real-time parameters, (ii) the *request_retrieve()* method to process look-up requests for values of real-time parameters, (iii) the *request_register()* method to process registration requests for event parameters, and (iv) the *ivk_action()* method to process the requests to execute an *action()* method. These methods will either be invoked directly by entities residing on the same kernel space (including the Message Handler which serves external systems), or be mapped to system calls used by entities residing in the user space.

These methods might be overloaded with OO techniques to serve the different types of requesters with different types of input arguments. The PE may also contain primitive methods, which are basic building blocks that perform supplementary actions to assist in the processing of incoming requests. These primitive methods may perform, for example, integrity or authentication checks.

As explained previously, the Policy Engine can also be equipped with a rate-control mechanism to monitor and limit the request rate for a certain parameter from a requester (i.e. rate of requesting the *get()/set()* for that parameter), as well as the overall number of requests sent to the InterLay object from that requester to a reasonable number in a certain period. This number can be decided based on the nature of the parameter, for example the upper limit for fast changing parameter can be set higher for slow changing parameter, or the limit is higher when the network activities is low than when the activities is high. Any request that arrives after the rate reaches the limit will be discarded by the PE. This will help to prevent the InterLay object from being abused by the user applications and by the networking protocol objects themselves. This is a safeguard mechanism to protect the performance of the new cross-layer enabled TCP/IP architecture.

Within the PE, the processing of requests from local systems is different from external ones, and will be discussed in details as follows.

3.7.2.1. PE and the external systems

A) The Message Handler

The PE relies on a Message Handler to communicate with external entities for supplement policies and information using standard protocols (such as Diameter, COPS, MIH etc.).

Because these protocols are running in the application level, while the Message Handler (MH) resides inside the PE (and thus it will run in the kernel space), there are two solutions for the MH to receive the requests from external systems, which are:

- The first solution is to tweak a special version of TCP/UDP socket that runs directly in the kernel. In this case the user provides the necessary authentication information (namely the address of the server and the user account information used to access that server) through some interface provided by the OS. However, the development of such socket might be cumbersome.

- The second solution is to implement a policy exchange application in the user space, and the Message Handler will exchange these messages with the user-

space application via NetLink [35] [39] (see Appendix I for a discussion of how to use NetLink for kernel-user space communication).

The external systems will exchange requests with the MH to fulfill the following tasks:

- retrieving value of real-time parameter
- updating value of real-time parameter
- registering for an event
- invoking an *action()* method
- exchanging of policy to authorize actions
- providing supporting data to the networking subsystem

When the MH receives requests from external system, first it has to authenticate the request. Normally the external system would want to learn about the conditions of the local system within the context of some communication services. In this case, the client application (corresponding to the communication service) residing at the local system would maintain some kind of share-secrets with the external system. The client application and the MH can then cooperatively authenticate the request from the external system in one of the following two scenarios:

Scenario #1: The client informs the MH (through some pre-defined API) regarding the information to authenticate the request from the external system (address, share secrets, authentication mechanisms etc.) and the MH will performs the authentication accordingly. In this scenario, the priority of the request is equal to that of the MH, or is assigned explicitly by the user through a system interface depending on the origin of the request (priority is explained later in this section).

Scenario #2: The client application authenticates the request by itself. In this case the MH and the client application are preconfigured with two-way APIs so that the MH can pass the information related to the request to the application and receive the decision of the application on whether the request is authentic or not. In this scenario, the priority of the request is equal to that of the client application.

In these scenarios, the roles in the exchange of authentication protocol are reversed: the external system (normally a server) acts as the client or the originator of the request, while the client application/MH acts as the server of the request.

The advantage of the first scenario is that the MH does not have to invoke the client application every time a request arrives at the MH, which saves some processing overhead. Moreover, only the first scenario allows for the remote monitoring of local status without the need for an agent at the local system. The advantage of the second scenario is that the client application can reuse any existing security associations with the server to authenticate the request. However, the second scenario can be substitute by letting the server to communicate the request directly with the client application, and the client application will relay the request to the InterLay. The drawback is that in this case the client application has to both authenticate and relay the result to the external server, which will increase the workload for the application developer.

If the request passes the authentication check, the MH will parse it to get the requested action, then map it to one of the four helper methods described above, and subsequently invoke either the Informer or the Enforcer to fulfill the request. The detail operations regarding each type of actions are further explained in the following sections.

B) Querying value of real-time parameter

The external system can make a request for information on a real-time parameter of the networking subsystem, or issue requests to modify the parameters. The format of the request and reply message will depend on the communication protocol being used, but it must contain the ID of the concerned parameter and in the case of an update request it should also include the new value to be assigned.

Because it is difficult for the external system to single out a single Transport layer session in the local system, the exchange between the external servers and the PE will mostly concern the parameters of the Network layer and the Data Link layer. However, in the case the external system needs to indicate a Transport layer session, it can use the address tuple (i.e. the source and destination IP address and ports).

When the request is to retrieve the value of a parameter, the PE will call the *get_param()* function of the Informer (described later in subsection 3.7.4) with the ID of

the parameter as the input, and forward the returned value by the Informer back to the external system.

C) Updating value of real-time parameter

When the request is to update the value of a real-time parameter, the PE first compares the priority of the requester with the priority of the parameter. The update request is only authorized if the priority of the requester is equal or higher than that of the parameter. The PE might perform any other authorization actions deemed necessary for the request. If the request is authorized, then the PE will call the *update()* method of the Enforcer (described later in 3.7.3), with the ID of the real-time parameter and the new value to be updated as the input, and the Enforcer will call the appropriate *update* method for the parameter. The priority of the request and the parameter is discussed in section 3.7.2.3 below.

D) Invoking an action() method

When the external system wants to execute a specific *action()* method, it sends a request to the Message Handler an invocation request with the ID of the *action()* method. The PE first compares the priority of the requester with the priority of the *action()* method. The execution request is only authorized if the priority of the requester is equal or higher than that of the *action()* method. The PE might perform any other authorization actions deemed necessary for the request.

If the request fails the authorization, it will be discarded. Otherwise, the PE will call the *execute()* method of the Enforcer with the ID of the *action()* method and optionally the input arguments for the *action()* method to carry out the execution. The priority of the request and the parameter is discussed in section 3.7.2.3 below.

E) Registering for an event

For event parameter, the external system will send to the Message Handler a registration request message together with the ID of the event. The message might optionally contain some extra actions that it wishes the InterLay object to perform at the time of the occurrence of the event. These extra actions are predefined in the Informer. The PE will perform any authorization actions deemed necessary. If the request is

authorized the PE will inform the Informer of the registration by calling the *event_registration()* method (described later in section 3.7.4), with the following input arguments:

- The ID of the event
- A string represents the address of the external system
- The handler for the requester in the form of the pointer to the callback function *send_event()* from the Message Handler. The *send_event()* function basically contains the codes to notify the external system of the event.
- The ID of any extra predefined actions that the external system wants the local system to perform at the time the event occurs.
- A value that indicates to the Informer that this request comes from the external system.

When the event takes place, the Informer will invoke the *send_event()*, using the event ID as well as the address string to correctly inform the external system of the occurrence of the event ID.

F) Requesting information from the networking subsystem to the external system

The protocol object in the networking subsystem of the local system can use the PE (namely the services of MH) to query the information from external systems to better fulfill its job. In this case the external server will be given a special parameter ID, and the PE will just act as an intermediate agent, sending the request from the requesting protocol object to the external system and forward the reply to the requesting protocol object.

The request and reply messages contain the information that informs the external system of what information the protocol object of the local system is interested in. In specific, the request message normally contains the parameter ID of the external server (that will be used by the PE to find the actual address) and the indication of the type of information that is requested. The reply message contains the reference to the request message so that the receiver knows what information is being returned. Moreover, the InterLay should be able to find the right local protocol object to send the reply message

to. However, this is an extra functionality that is out of scope of this study and is left for further study, but this functionality can be used by the local networking subsystem to request and execute externally received scripts.

3.7.2.2. PE and the local system

The local system includes the OS and the applications running above it. Because end-user applications (in the user space) reside in different memory space from the InterLay object, it will interact with PE through the system calls of the Service Access Point (namely the SAP-1), and these system calls will be mapped to the four helper methods (the system calls will be discussed later in 3.9.)

On the other hand, because Transport and Lower Layer protocol objects reside in the same kernel space as the PE, they can issue requests directly to the four helper methods of the PE.

PE receives requests from the application and lower layer protocols to (i) update value of real-time parameter, (ii) register for an event, and (iii) invoke an *action()* method, and these operations will be discussed in more details as follows.

A) Updating value of real-time parameter

When the PE receives a request to change value of a real-time parameter, it first authorizes the request using the priority mechanism by comparing the priority of the requester with the priority of the parameter. The update request is only authorized if the priority of the requester is equal or higher than that of the parameter. In this case the PE will call the *update()* method of the Enforcer, with the ID of the real-time parameter and the new value to be updated as inputs, and the Enforcer will call the appropriate *set()* method for the parameter.

Otherwise, if the request is not authorized it is discarded.

B) Registering for an event

When the PE receives the registration request for an event with its ID, it will perform any authorization actions deemed necessary. If the request is authorized, the PE will inform the Informer by invoking the *event_registration()* method to register the requester to the event notification list.

For the request from a user application, the PE includes in the call to the *event_registration()* method with the following input arguments:

- The ID of the event
- Any information that helps locating the right owner of the event (optional)
- The process ID of the application
- The ID of any extra predefined actions that the external server wants the local system to perform at the time the event occurs.
- A value that indicates to the Enforcer that this request comes from user application.

If the request comes from the networking subsystem, the PE includes in the call to the *event_registration()*method with the following input arguments:

- The ID of the event
- Any information that helps locating the right owner object of the event (optional)
- The pointer to the callback function that handles the event
- The ID of any extra predefined actions that the external server wants the local system to perform at the time the event occurs.
- A value that indicates to the Enforcer that this request comes from networking subsystem.

C) Invoking an action() method

When the PE receives a request from the application or lower layer protocols to execute a specific *action()* method with the ID of the *action()* method, the PE first compares the priority of the requester with the priority of the *action()* method. The execution request is only authorized if the priority of the requester is equal or higher than that of the *action()* method. The PE might perform any other authorization actions deemed necessary for the request.

If the request is authorized, then the PE will call the *execute()* method of the Enforcer (described later in 3.7.3, with the ID of the *action()* method and optionally the input

arguments for the *action()* method to carry out the execution. The priority of the request and the parameter is discussed in section 3.7.2.3 below.

Otherwise, if the request is not authorized, the request is discarded.

3.7.2.3. Priority of requests for updating real-time parameter or invoking *action()* method

The PE contains the priority values of all requesting parties, as well as for each real time parameter and *action()* method.

For real-time parameters, each is associated with a certain priority. A requesting network protocol object of the local networking subsystem will also be assigned with a certain priority. When the networking subsystem is initialized, the parameters and networking protocol objects are assigned a default priority.

Excluding the application layer, in general higher layer protocol has higher priority than lower one because it has more comprehensive view of the current condition. For an object in the application layer, its priority is highest for parameter that is dedicated to itself (i.e. parameters from Layer 4 (L4) protocol object which are created by the application) but it has a default lowest priority for parameters of Layer 2 (L2) and Layer 3 (L3). On the other hand, the priority of requests coming from external policy server can be set on a case-by-case basis.

An update request for the value of a parameter is served only if the priority of the requester is higher than or equal to the current priority of the parameter, and requests from requesters with equal priority will be executed in chronological order of arrival (i.e. late request overrides earlier one). As higher priority overrides lower ones, the kernel can protect a certain parameter from being altered by setting the parameter priority to the exclusive (highest) priority with infinite lifetime.

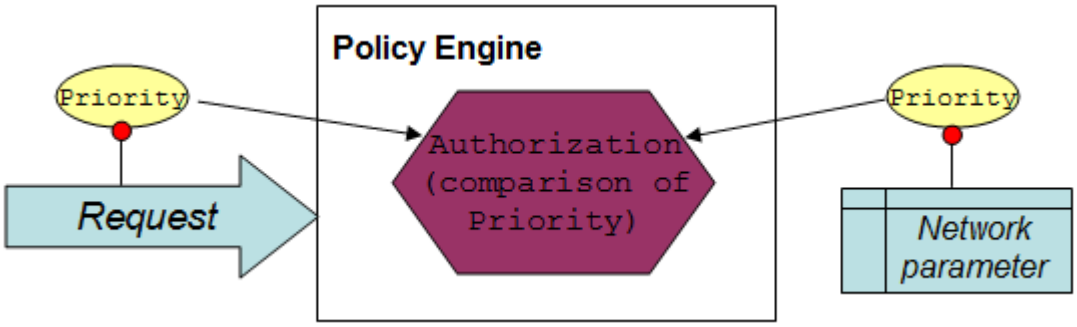

Figure 4. The operation of priority mechanism

The priority of a parameter is set to equal to that of the last accepted request (however a request is given a certain lifetime, and after that period the request's veto right on the parameter is obsolete, and the priority of the parameter returns to that of the default value).

For *action()* method, each method is also associated with a priority. Each networking protocol object and application is also given a priority, normally the default priority. The priority of requests coming from external policy server can be set on a case-by-case basis. An *action()* method is executed if the priority of the requester is higher than that of the concerned action() method. The kernel can prevent a certain parameter from being executed by setting the method's priority to the exclusive (highest) priority with infinite lifetime.

The networking subsystems may provide an interface for the users to explicitly set the priority of a certain application or protocol object.

3.7.2.4. Other extra security checks

Apart from priority check for the update of a parameter, the PE might also perform any other necessary permission and integrity checks before calling the *update()* method of the Enforcer to update the value of a network parameter. Normally the check aims at guaranteeing that the update will not negatively affect the networking subsystem or other communication applications. One example is to check whether the requester of an event registration is eligible for the extra reactions included in the registration message for an event.

3.7.3. The Enforcer

The Enforcer performs two type of actions, one is to update real-time parameters and the other is to execute *action()* methods.

The Enforcer may also contain other primitive methods which are basic building blocks that perform supplementary actions that are common for all *set()* or *action()* methods, such as general integrity check or authorization.

The InterLay model uses the Enforcer to support (obj-b) described in section 1.2 of chapter 1.

3.7.3.1. Updating real-time parameter

The Enforcer performs actual changes of real-time parameters as requested by the PE. It contains an *update()* method that receives the ID of the parameter and the new value to be updated as input arguments (see Figure 5.)

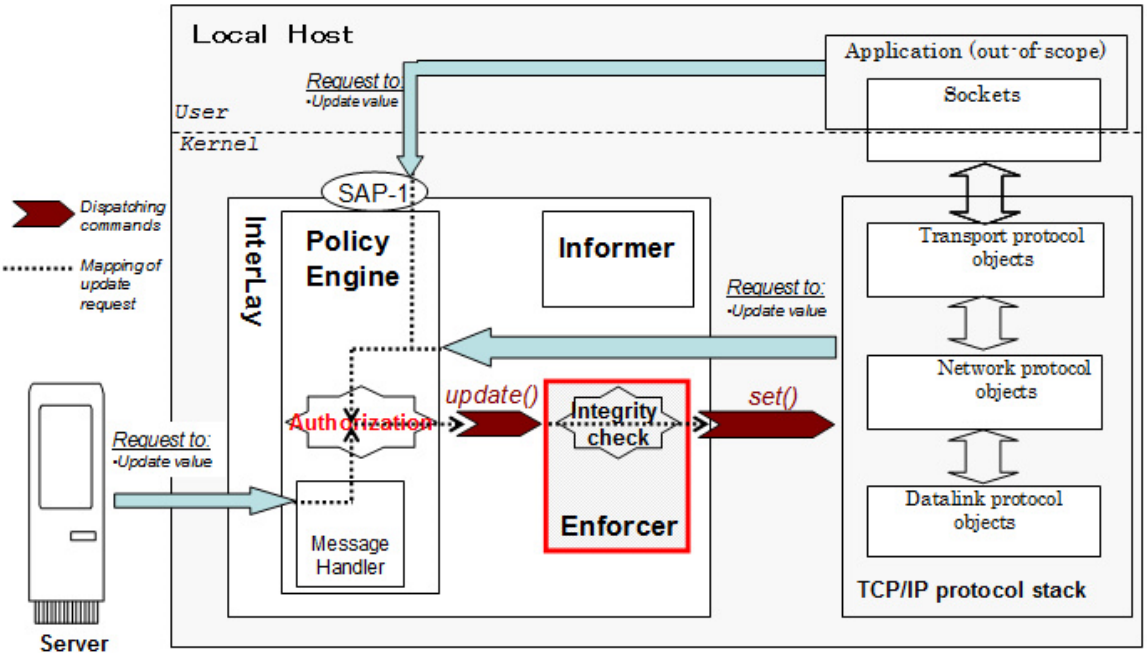

Figure 5. Enforcer and updating real-time parameters

The Enforcer also maintains a list of constrains on each parameter. When the update request is sent to the Enforcer, it may further validate the request, such as checking the

new value against the constrains of the parameter to see whether the new value is of valid type and range.

The Enforcer maintains a mapping between the ID of a real-time parameter and the *set()* method (or method to renegotiate/probe the value in the case the parameter obtains its value through negotiation/probing) of that parameter. The *update()* method will implement this mapping by using, for example a switch-case structure using the ID for selection. Any other actions (including security authorization) specific to the parameter that need to be performed, as well as action to locate the right object (in case several object instances of the protocol exist at the same time) can be performed in this switch-case (see table II).

3.7.3.2. Executing *action()* method

The Enforcer contains an *execute()* method that the PE calls to request the Enforcer to execute an *action()* method.

The Enforcer maintains a mapping between the ID of a *action()* method and the actual method. The *execute()* method will implement this mapping by using, for example a switch-case structure using the ID for selection. Any other actions (including security authorization) specific to the parameter that need to be performed, as well as action to locate the right object (in case several object instances of the protocol exist at the same time) can be performed in this switch-case, which is demonstrated in table II.

```
class Enforcer {
   public:
      void update(DWORD ID, DWORD *newVal) {
            switch (ID) {
             case 123456:
              //Any pre set()check or object location is done here
                 TCP.changeMSS(newVal);//the MSS should be updated
                                          //via renegotiation process, //not
                                          directly
              //Any post set(); // check is done here
                 break;
             case 111111:
              //Any pre set()check or object location is done here
              TCP.setDPort(newVal);
              //Any post set(); // check is done here
                 break;
             ...
             }
     };
      void execute(DWORD ID; void * val) {
            switch (ID) {
             case 123456:
                //Any pre check or object location is done here
                MIP.RO((string *) val);// in this case val is the address of
                                        // Corresponding Host
                //Any post set(); // check is done here
                                       break;
             ...
             }
     };

} Enforcer;
```

Table II. The switch-case of the update() method

Note: in the Table II the data type of newVal parameter is pointer to DWORD, however, the *update()* method can be overloaded to serve other data type, especially DWORD type for faster performance (most of the networking parameter will be of numeric type)

The example of the *update()* and *execute()* switch-case in C++ is explained in table II (suppose that the Enforcer is implemented as a separate object), while the operation of the Enforcer when executing *action()* method is depicted in Figure below.

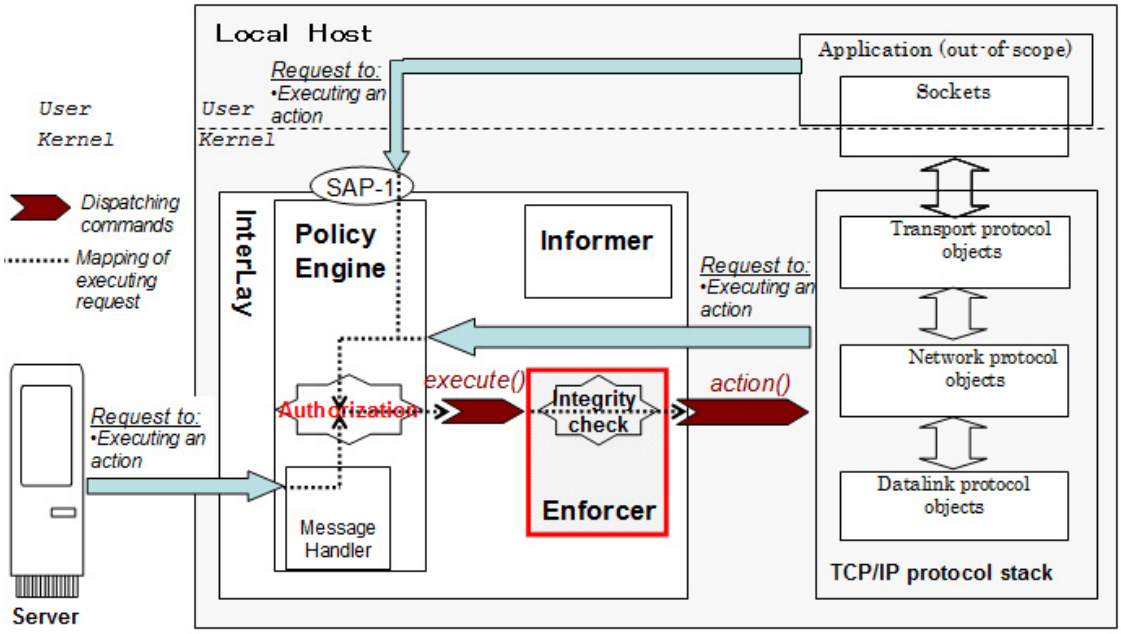

Figure 6. Enforcer and executing action() method

3.7.4. The Informer

As the name implies, the Informer is the entity that (i) returns the value of the real-time parameter, and (ii) notifies the requester or responses accordingly when a registered event occurs.

3.7.4.1. Returning the value for the real-time parameter

For real-time parameter, the Informer contains the *get_param()* method that returns the current value of real-time parameters at invoked time. Note that for fast changing parameter, the returned value might be outdated when reaching the requesting object.

The Informer maintains a mapping between the ID of a real-time parameter and the *get()* method of that parameter in the *get_param()* method. Similar to the *update()* method of the Enforcer, the *get_param()* method will implement this mapping by using, for example a switch-case structure using the ID for identifier. In general, there is no need for security check for a read operation, but it might be necessary to locate the right object in advance if there exist several protocol object instances of the same layer at the same time (for example several sockets are active – equaling to several TCP object exist). The example of the switch-case in C++ is described in Table III (suppose that the Informer is implemented as a separate object).

Because we expect that frequency of the request for *get()* method will be much higher than other type of request, to improve the performance of the networking subsystem, the Informer may choose to cache the value of the parameter that is judged to be static (for example the Home Address in Mobile IP protocol) so that it has to call the *get()* function for that parameter only once at the first time the parameter is requested.

How the requests to query value of real-time parameters are dispatched and served depends on where the requesters reside.

If the requesters are the protocols of the TCP/IP stack, they can call the *get_param()* function directly because they reside on the same kernel space with the InterLay. By contrast, end-user applications (in the user space) reside in different memory space from the InterLay object, and therefore they will interact with the Informer through the Service Access Point, namely the SAP-2 in figure 7. In the case the request for value of a real-time parameter comes from an external entity (i.e. an external server), the request will come through the PE.

When the Informer receives a request for value of a real-time parameter with the ID of that parameter through the *get_param()* method, the corresponding *get()* method of the parameter belonging to the target object will be invoked, and the Informer then returns the value to the requester, either directly if the requester resides in the kernel space, or through the SAP-2 interface if the requester is an end-user application. If the request comes from external server, the Informer returns the value to the PE, which then forwards the result to the external entity.

```
class Informer {
   public:
     DWORD * get_param(DWORD ID) {
      switch (ID) {
           case 222222:
              //object location is done here if required
              return TCP.getMSS() ;
              break;
           case 444444:
              //object location is done here if required
              return TCP.getDPort() ;
              break;
           case 666666:
              //object location is done here if required
              return MIP.getCCoA() ;
              break;
           ...
           }
     };
} Informer;
```

Table III. The switch-case of the *get_param()* method

Note: in the Table III the returned data type of newVal parameter is pointer to DWORD, however, the *get_param()* method can be overloaded to serve different data types, especially with DWORD type for faster performance (most of the networking parameter will be of numeric type)

The operations of the Informer regarding querying value of real-time parameters are depicted in Figure 7 below.

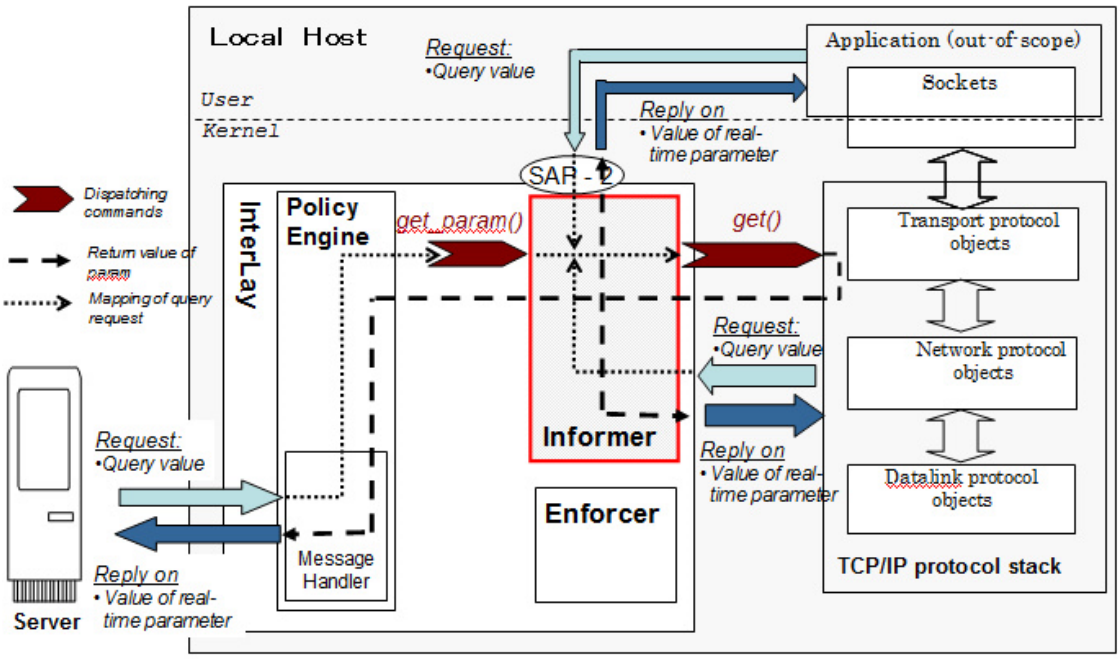

Figure 7. Informer and returning value of real-time parameters

3.7.4.2. Registering and notifying of the occurrence of an event.

Apart from real-time parameters, the Informer also handles event notification activities. It contains a *event_registration()* method for this purpose. The method is overloaded using OO technique to serve the different type of requesters with different types of input arguments.

For each event, the Informer will create and maintain a separate *notification chain* [27]. The Informer maintains a mapping between the event ID and the corresponding *notification chain.* The appropriate information (i.e. the *handler* which dictates the reaction when the event occurs) received from the PE through the call to the *event_registration()* method will be registered to the notification chain corresponding to the event ID. For the extra actions coming with the request to the PE, only the extra actions that are not registered yet will be registered.

For each notification request, the Informer keeps a timer, so that when the request expires, the requester will be purged from the list. This will reduce the overhead that is caused by notification that is no longer needed by the requester.

When the event occurs, the handlers registered in the notification chain will be executed as discussed previously in 3.5.2, the type of reactions indicated by the handler will be different depending whether the requesters reside in the kernel space and user space.

3.7.4.3. Registering and notifying requester in kernel space

The requesters in the kernel space include both the Message Handler of the PE (namely the MH to serve external systems) as well as protocol objects of the TCP/IP networking subsystem. As explained above, upon receiving the registration request from the PE, the Informer will register the handler to the appropriate notification chain, using the event ID. In this case the handler is the *send_event()* function of the PE or the callback function pointer from the protocol objects.

There are two ways to register the handler to the notification chains. First, the callback function pointer is registered directly into the notification chain. This requires no extra processing at the InterLay object, which is suitable for simple and sequential activities that simply require a linear chain of reactions.

However, as registering a function to a notification chain may require some extra efforts, the second option is to create a method at the InterLay that will takes care of invoking all of the callback function, and only this method is registered to the chain. The method will take care of removing expired registration. In this case only a single registration of that method to the notification chain is required. This option has another advantage that it is easier to manage/modify/maintain the notification chain because it is confined to a single method. Moreover, it can not only support linear processing of the callback function chains, but also support processing with condition and loop etc.

Upon the occurrence of the event, for the registration made by external system through the MH, the *send_event()* method is called with the ID of the event and the address of the server. This method will send the event ID to the external system, informing it of the occurrence of the event at the local system. For registration from local protocol objects (i.e. in the kernel), the corresponding call back function is executed.

The operations of notifying to requesters from inside the kernel space are illustrated in Figure 8 below.

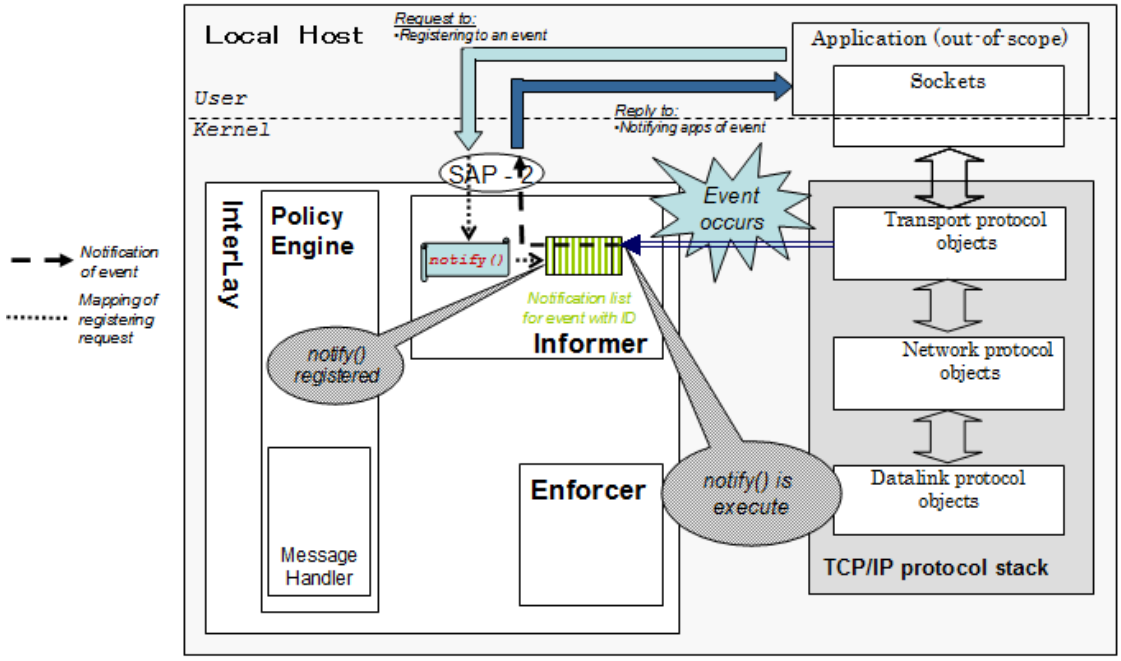

Figure 8. Registration and notification for events from inside the kernel

3.7.4.4. Registering and notifying requester in user space

For registration from the user application, the Informer carries out the following actions:

1. The Informer maintains a list of process ID for each event.
2. When an application registers for an event, the Informer adds the process ID to the list, after confirming that no such ID is already in the list.
3. The Informer maintains one separate *notify()* method for each event. When invoked, this function will go through the process ID list, and performing the notification to each and every application in the list.
4. The Informer then registers this method to the notification chain of the concerned event parameter. When the even occurs, the *notify()* method will be called, and it will notify all the applications in turn.

The registration for events from user applications is illustrated in Figure 9 below.

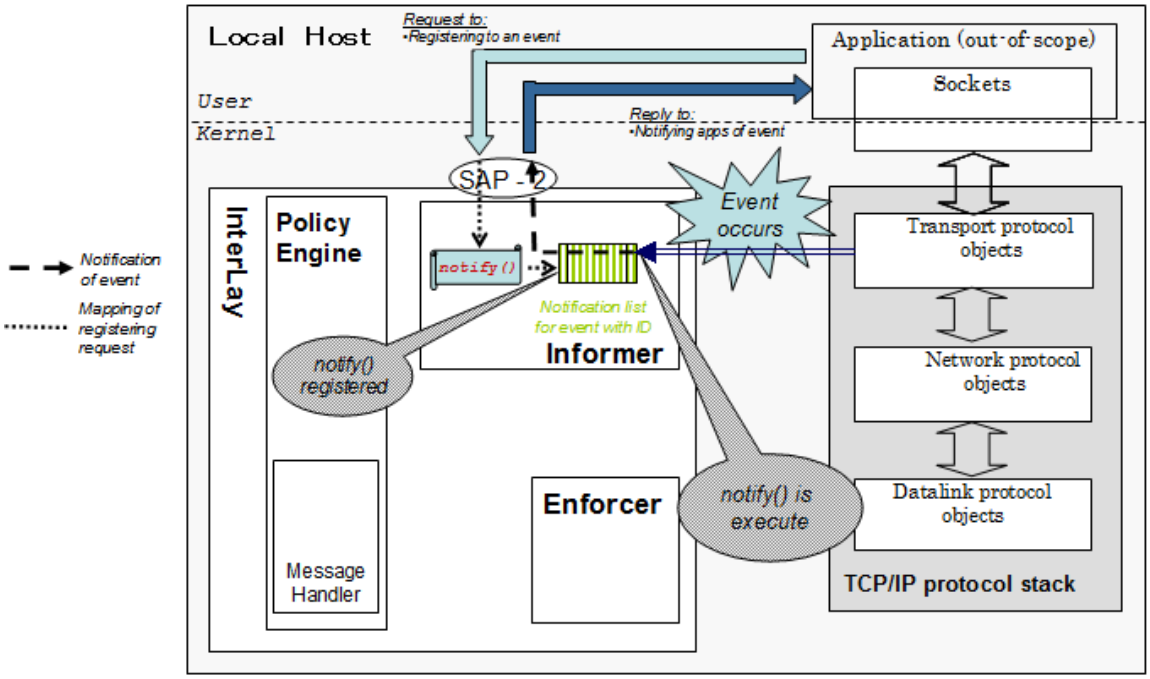

Figure 9. Registering for events from user applications

Notifying the user application from the kernel in a timely manner is difficult in InterLay because:

- The application needs several information to identify the correct event, namely the ID of the event and optionally the socket identifier (the address tuple) that the event belongs to.

- The only standard IPC (Inter Process Communications) mechanism that the kernel supports for notifying user application is through the kernel signal [34] [38]. There are two types of signals: normal and real-time, of which only real-time allows for an extra 32bits data, but the number of signal that is available is limited. In the case the application needs more information, signal will be used along with some other IPC mechanism such as mmap[47]. Another solution is to use NetLink socket.

We will discuss each user space – kernel space communication solution in Appendix I.

3.7.4.5. Other issues

Like the PE and the Enforcer, the Informer class also contains primitive methods, which are basic building blocks that perform supplementary actions to assist the invocation of the *get()* methods of the protocol objects, as well as to maintain the notification chain.

The Informer object also contains a list of predefined extra actions, each is given an ID. The entity that registers for an event can also specify in the registration request for the event some of these extra actions (apart from the even notification sent to the entity itself) that it wants the InterLay object (here namely the Informer) to perform when the event occurs.

And similar to the case of PE, the Informer can also be equipped with a rate-control mechanism to monitor and limit the request rate from a requester for a certain parameter (i.e. rate of calling the *get()/set()* for that parameter) as well as the overall number of requests sent to the Informer object from a certain requester, to a reasonable number in a certain period. This number can be decided based on the nature of the parameter, for example the upper limit for fast changing parameter can be set higher for slow changing parameter, or the limit is higher when the network activities are low than when they are high. Any request that arrives after the rate reaches limit will be discarded by the Informer. This will help to prevent the Informer object from being abused by the user applications and by the networking protocol objects themselves, in order to protect the performance of the new cross-layer enabled TCP/IP architecture.

3.8. On selection of the right protocol object

The TCP/IP implementation has the hour-glass shape, in which the IP layer is essentially the n-to-m multiplexer [71] between multiple Layer 4 and Layer 2 protocol instances. If the Interlay is requested to perform an action on all instances of Layer 4 and Layer 2, then it can walk through all instances and perform the requested action. However, in the case the requester of an action wants to target a specific Layer 4 or Layer 2 protocol instance, the InterLay needs to locate the right protocol object.

There may existed several instances of the same Layer 4 protocol at a time, and normally these instances are linked together in a link-list. For example, each UDP socket

will create an UDP instance at the networking subsystem, and these sockets are linked together in a link-list. There is one link-list per protocol, for example the link-list for UDP is separated from that of TCP and other Layer 4 protocols.

Instances of Layer 4 protocols are basically identifiable via the tuple (source + destination IP addresses and ports). Therefore when an entity (e.g. the user application or external system) wants to either manipulate a parameter or execute an action on a specific Layer 4 instance, the InterLay (namely the Informer or the Enforcer) will need the identifier of the instance (i.e. the tuple), and it will walk through the link-list of the concerned protocol in the networking subsystem until it finds the right instance, and execute the requested *get()* or *set()* or *action()* method.

On the other hand, Layer 2 protocol instances, which are basically controller object of network interfaces, can be identified by several means. The most popular means of identifying a network interface is to use the IP address associated with it. If network alias is used, then several IP addresses can be mapped to a network interface, but essentially a network interface will be unambiguously identified using an IP address. If the interface utilizes a technology from the IEEE 802 networking family, then the associated MAC address can be used by the InterLay to find the associated interface. Requesters from local system can also use the interface name to identify the interface they want to manipulate.

3.9. New system calls to enable SAP-1 and SAP-2

The division of the service access points into SAP-1 and SAP-2 is nominal, because to the user applications they are just a group of kernel system calls. Moreover, if the InterLay is implemented as one instead of three objects, then the division is purely logical. And as explained in section 3.7.4, SAP-2 can be easily combined with SAP-1 if we let the PE receives the request for looking-up value of real-time parameter from local system.

To realize these SAPs, several new system calls should be defined that carry out the interactions between the socket and the InterLay objects. As explained in section 3.5 and 3.6, each network parameters or *action()* method that is exposed by the InterLay will be given a unique predefined (DWORD) ID to be used with the system calls.

The kernel defines several new system calls to be used by the applications to invoke the functionalities of the InterLay object as follows.

3.9.1. The new *net_set_param()* socket API

A new *net_set_param()* socket API function is used at the SAP-1 interface, which allows the application to assign new values to the real-time attributes of lower layer objects (normally the Transport) through PE. *net_set_param()* requires a set of 2 input arguments: {the predefined ID of the real-time parameter; the new value for that parameter}.

net_set_param() will invoke the *set_param()* method of the PE functional group with two more additional input parameters to correctly identify L3 and L4 protocol objects: the caller's Process ID and the socket identifier (i.e. the socket type and socket's address tuple.)

The *set_param()* method will use the Process ID to find the priority of the request (either the default value for application or a explicit value set by the user) and perform priority test or any other necessary authorization test and if everything is OK it will ask the Enforcer to invoke the *set()* method for the parameter that corresponds to the requested parameter ID and the socket's identifier (i.e. the L4 session), and the result of *set()* method (namely SUCCESS or FAILURE) is returned to the caller of the *net_set_param()*.

3.9.2. The new *net_get_param()* socket API

A new *net_get_param()* socket API function is used at the SAP-2 interface, which allows the socket object to query the value of a real-time parameter of lower layer objects through the Informer. The *net_get_param()* requires the predefined ID of the real-time parameter as input parameter.

net_get_param() will invoke the *get_param()* method of the Informer functional group with two more additional input parameters to correctly identify L3 and L4 protocol object: the caller's Process ID and the socket identifier (i.e. the socket type and the socket's address tuple).

get_param() method will invoke the *get()* method for the protocol object's attribute that corresponds to the requested real-time parameter's ID and socket's identifier (i.e. the L4 session) and the returned value is subsequently returned to the requesting application.

3.9.3. The new *net_invoke_action()* socket API

A new *net_invoke_action()* socket API function is used at the SAP-1 interface to allow the application to invoke an *action()* method of a protocol object, with one function parameter which is {the predefined ID for the concerned *action()* method}.

The *net_invoke_action()* system call will be mapped to the *ivk_action()* of the PE with two more additional input parameters to correctly identify L3 and L4 protocol object: the caller's process ID and the socket identifier (i.e. the socket type and the socket's address tuple).

The *net_invoke_action()* system call will be called by the application by indicating a predefined ID. The *ivk_action()* method will use the caller's Process ID to find the priority of the request (either the default value for application or an explicit value set by the user) and perform priority test or any other necessary authorization test and if everything is OK the PE will ask the Enforcer to invoke the *action()* method of the appropriate protocol object.

Optionally the application can call a sequence of *action()* method by indicating the corresponding sequence of ID.

3.9.4. The new *net_reg_event()* socket API

A new *net_reg_event()* system call is used at the SAP-1 interface to allow the application to register for a specific event. The *net_reg_event()* function will take as input arguments the predefined ID of the concerned event and optionally the socket identifier (i.e. the socket's address tuple) if the event belongs to Transport protocol object.

The *net_reg_event()* system call will be mapped to the *request_register()* of the PE with two more additional input parameters to correctly identify L3 and L4 protocol object: the caller's Process ID and the socket identifier (i.e. the socket type and the socket's address tuple).

After the request is authorized, *request_register()* of the PE will invoke the *event_registration()* method of the Informer. Subsequently the Informer will register the process ID to the notification chain and will inform the application when the event occurs.

As the name implies, the job of the *net_reg_event()* is only to notify the InterLay object of the request to learn about the event by the application. The actual notification process will be carried out by the Informer as described in Appendix I.

3.9.5. Implementation of the system calls

The four system calls *net_set_param(), net_get_param(), net_reg_event()* and *net_invoke_action()* can be implemented either as socket API functions or generic system calls. The advantage of implementing system calls as socket API calls is that there is no need for socket identifier to be included as input arguments. In the case the system calls are implemented as generic system calls, there is no need for new socket, so deployment might be faster, but socket identifier (the address tuple) might be needed.

3.10. Illustrations of InterLay operations

In this section we will provide some illustrations of how the InterLay works to provide services to various requesters.

In the following sections, we will consider the following interactions between:

- InterLay and lower layers
- InterLay and user applications
- InterLay and external systems

For each interaction, we will consider the following scenarios:

- Querying value of real-time parameter
- Setting value of real-time parameter
- Invoking *action()* method
- Event registration

3.10.1. InterLay and lower layers

Lower layers mean transport layer and those below. Interlay and lower layers belong to the same kernel module, therefore it can call directly *set()* and *get()* method for real-time attributes.

A. Querying value of real-time parameters

Figure 10 explains how a lower layer returns the data for real-time parameters to another lower layer.

Step 1: Lower layers call the *get_param()* method with the parameter ID
Step 2: Informer calls the *get()* method corresponding to the parameter ID
Step 3: Informer returns value from the *get()* method to the caller of *get_param()*

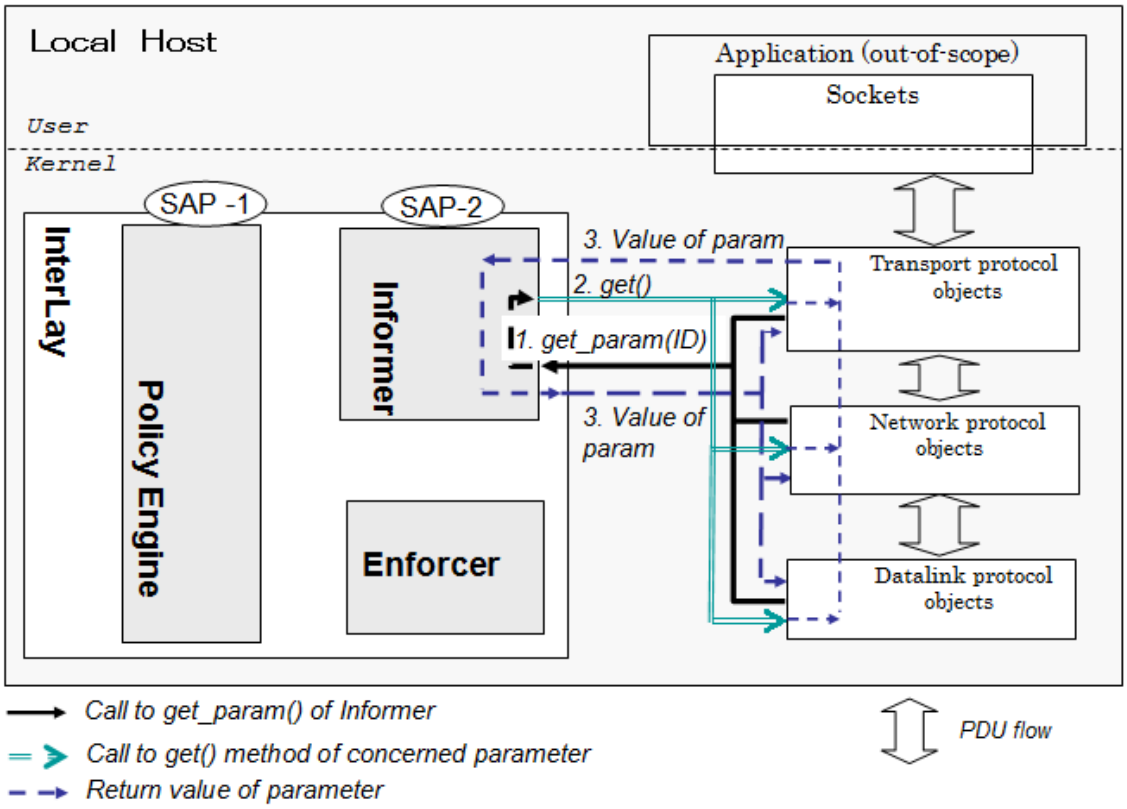

Figure 10. Querying value by lower layers

B. Updating value of real-time parameter

Figure 11 explains how a lower layer updates the data for real-time parameters to another lower layer.

Step 1: Lower layers call the *set_param()* method with the parameter ID
Step 2: Policy Engine authorizes the request and calls the *update()* method
Step 3: Enforcer calls *set()* method to update value of the parameter ID

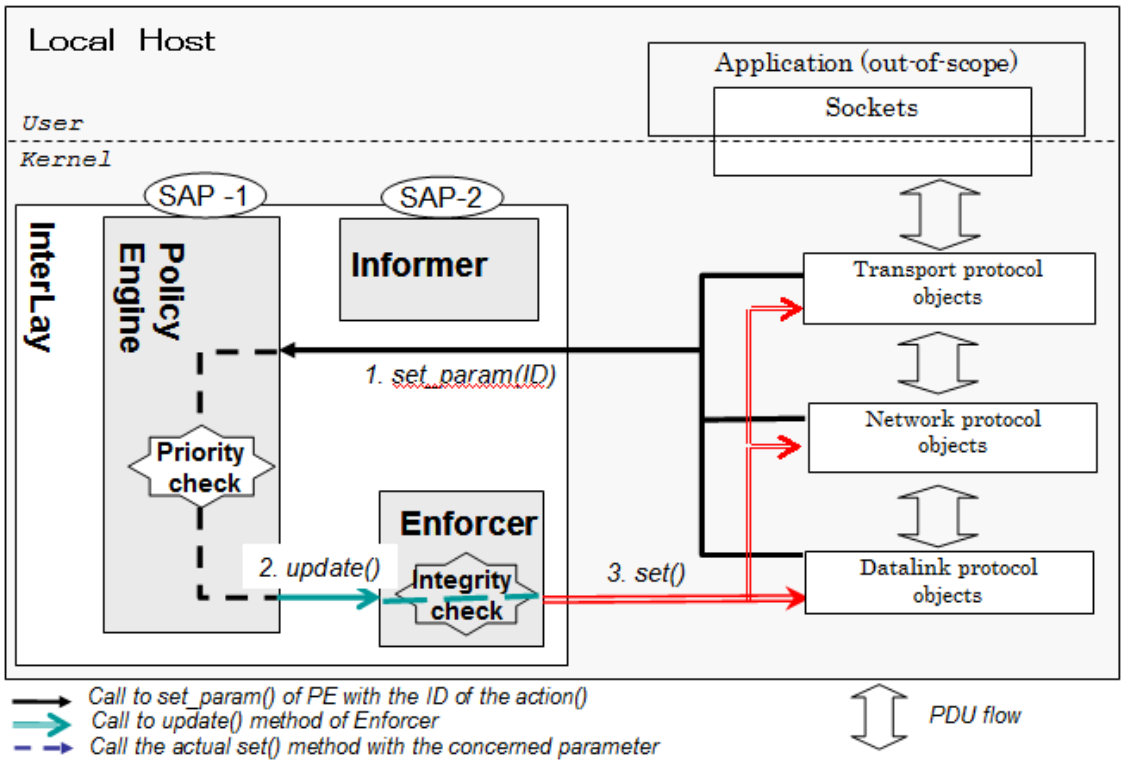

Figure 11. Updating value by lower layers

C. Invoking *action()* methods

Figure 12 explains how lower layers invoke an *action()* method.

Step 1: Lower layers call the *ivk_action ()* method with the action ID
Step 2: Policy Engine authorizes the request and calls the *execute()* method
Step 3: Enforcer calls *action()* method corresponding to the action ID

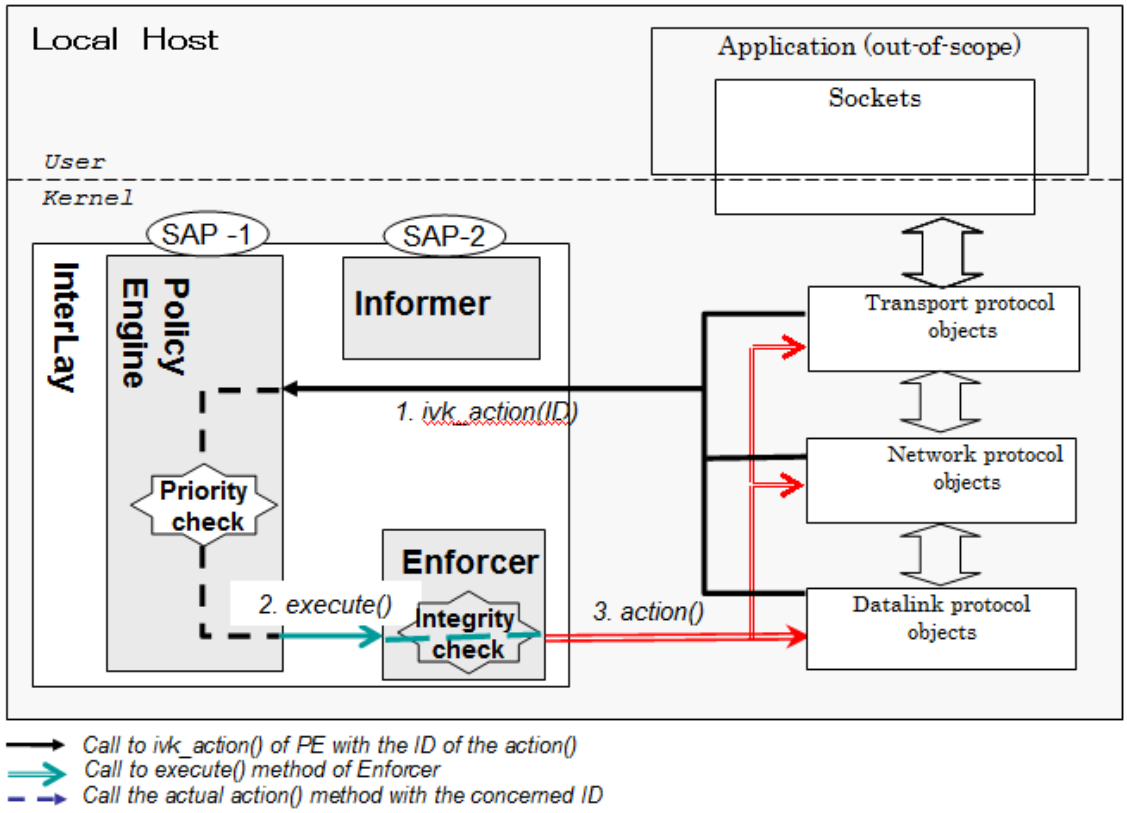

Figure 12. Invoking action() methods

D. Event registration and notification

Figure 13 explains how lower layer objects register and be notified for an event

Step 1: Lower layers call the *request_register()* method with the event ID and callback function
Step 2: Policy Engine calls the *event_registration()* method
Step 3: Informer puts the call back function to the notification chain.
Step 4: The event occurs.
Step 5: The notification chain invokes the callback function for the lower layer object.

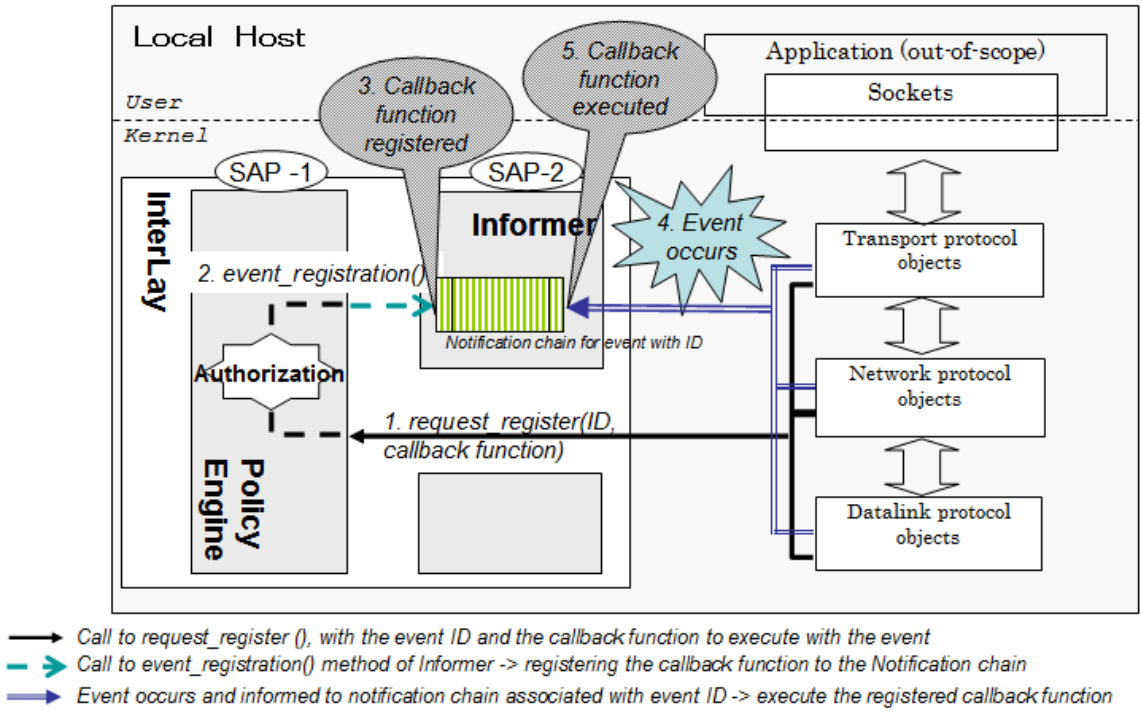

Figure 13. Registration for events from lower layers

3.10.2. InterLay and user applications

The InterLay will interact with higher layers through the system calls of SAP-1 and SAP-2.

A. Querying value of real-time parameters

For real-time parameters, the application can query directly the Informer through the standardized interfaces as depicted in Figure 14

Step 1: Applications call the *net_get_param()* socket call with the parameter ID
Step 2: Informer calls the *get()* method corresponding to the parameter ID
Step 3: Informer returns value from the *get()* method to the caller of *net_get_param()*

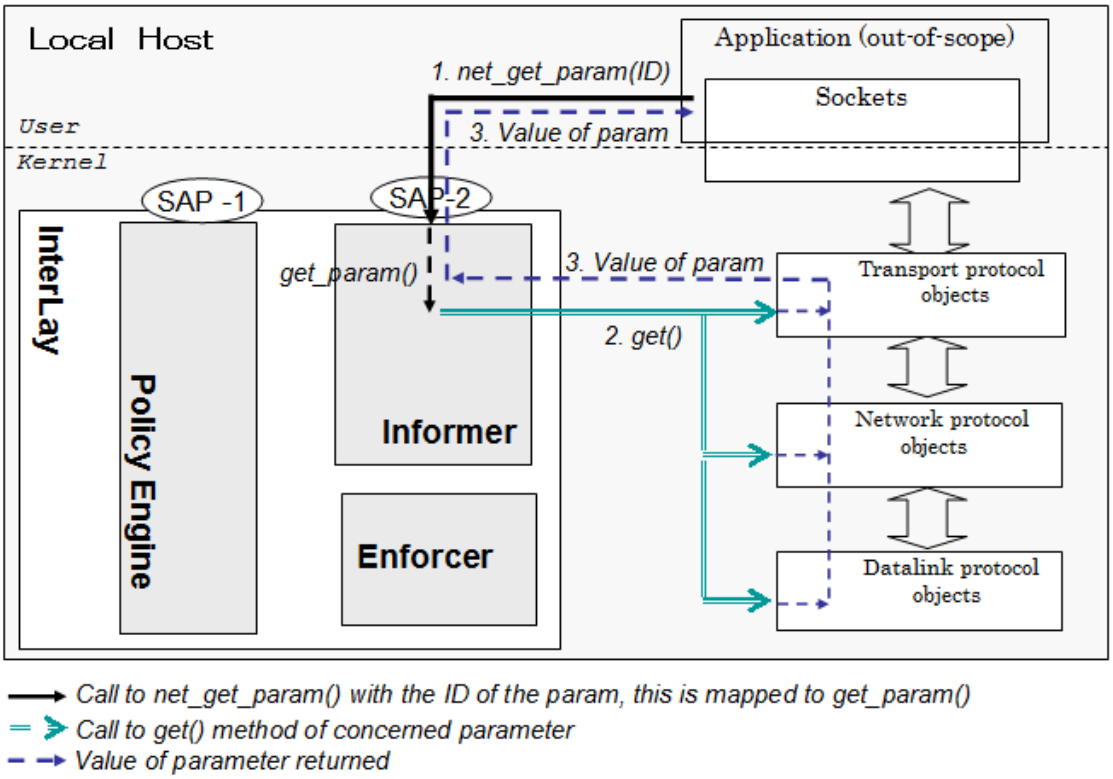

Figure 14. Querying value by user applications

B. Updating value of real-time parameters

Figure 15 explains how the applications update the value for real-time parameters by user applications.

Step 1: Applications call the *net_set_param()* socket call with the parameter ID
Step 2: Policy Engine calls the *update()* method
Step 3: Enforcer performs necessary integrity check and calls *set()* method to update value of the parameter ID

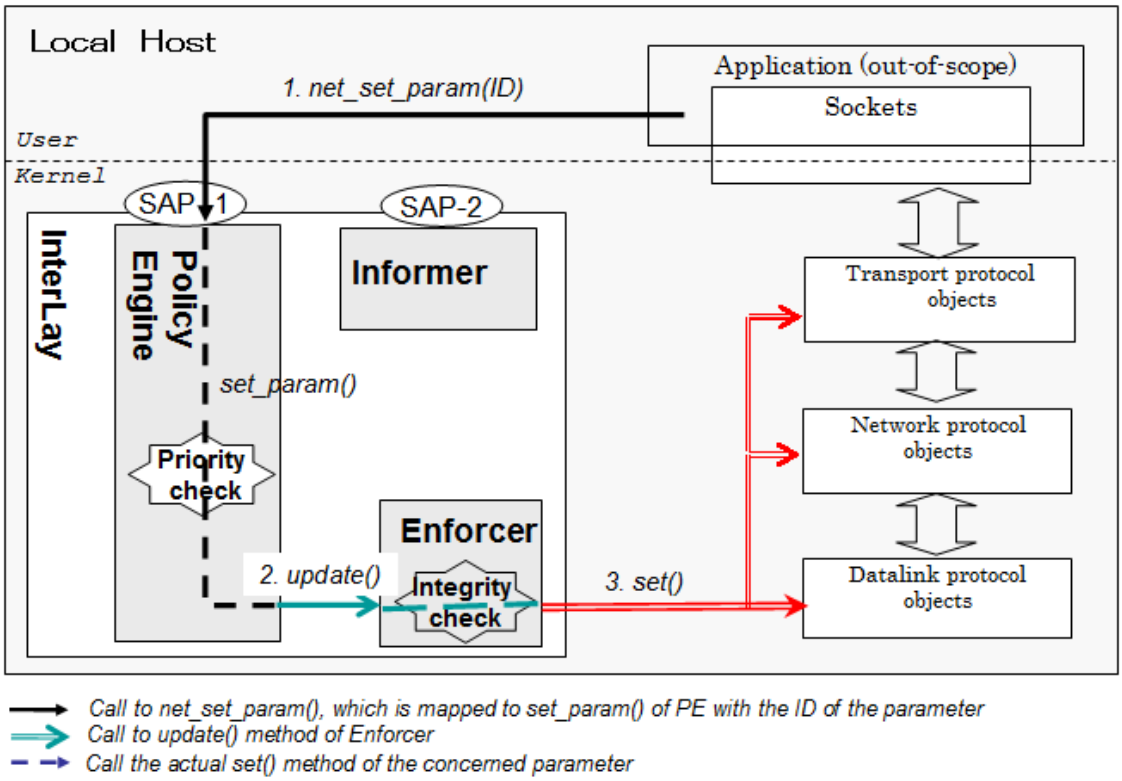

Figure 15. Updating value by user applications

C. Invoking *action()* methods

Figure 16 explains how the applications invoke an *action()* method.

Step 1: Applications call the *net_invoke_action ()* with the action ID
Step 2: Policy Engine calls the *execute()* method
Step 3: Enforcer call *action()* corresponding to the action ID

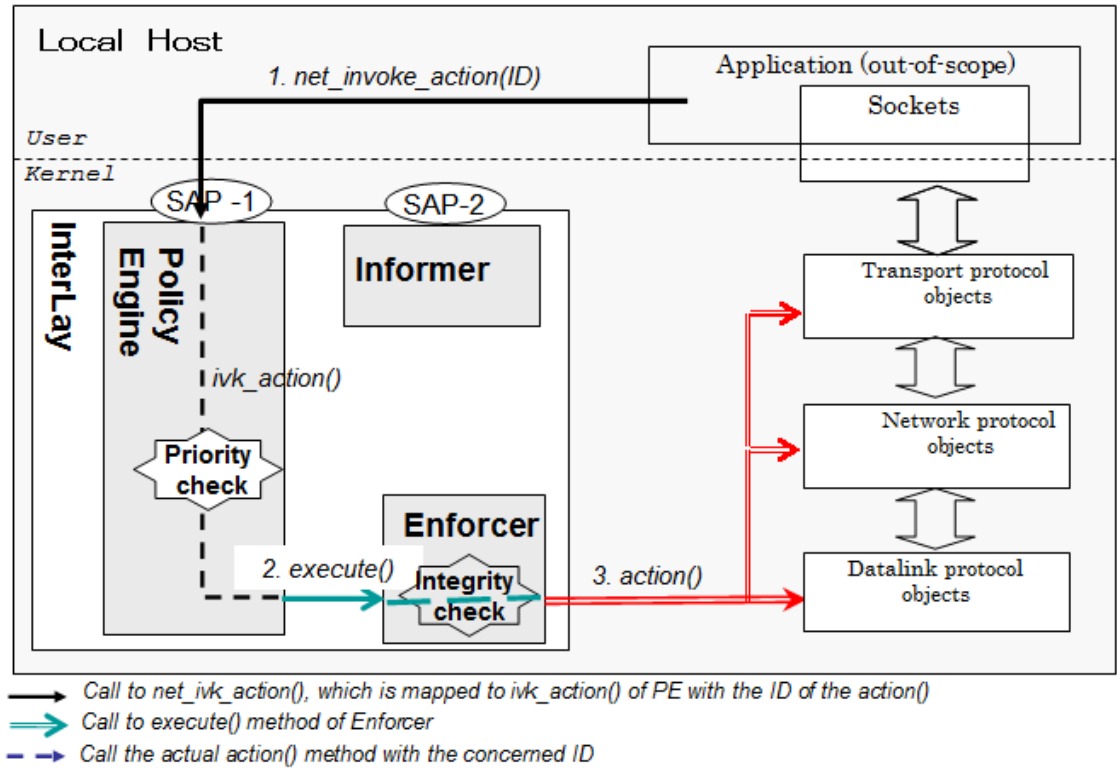

Figure 16. Invoking *action()* methods

D. Event registration and notfication using NetLink socket

Figure 17 explains how user applications register and get notification for an event using NetLink socket.

Step 1: The kernel registers the protocol family for the event, and InterLay and applications create the NetLink socket with the right protocol family.
Step 2: The event occurs.
Step 3: Notification chain asks NetLink to send notification to the NetLink socket in the kernel.
Step 4: The notification message reaches the NetLink socket in the application.
Step 5: The callback function that is registered with the NetLink socket in the application is invoked to handle the event.

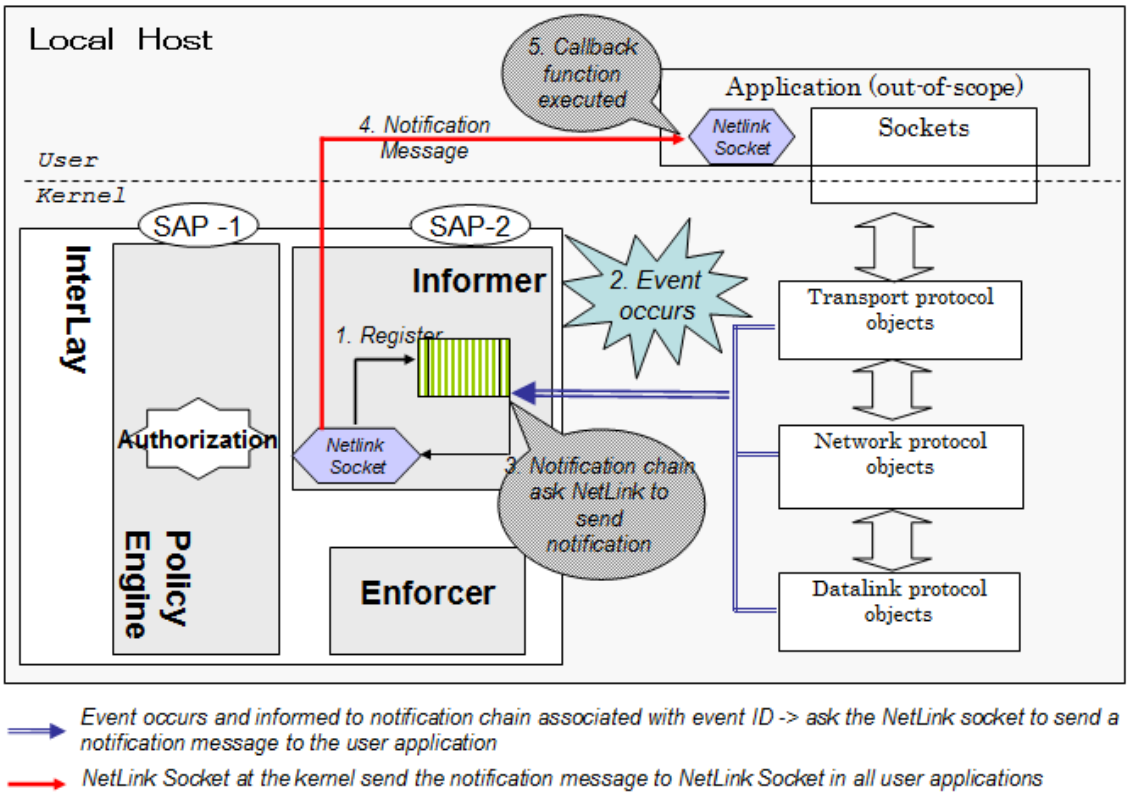

Figure 17. Event registration and notification (using NetLink socket)

E. Event registration and notfication using signals

Figure 18 explains how user applications register and get notification for an event using signals.

Step 1: Applications call the *net_reg_event()* system call with the event ID.
Step 2: Policy Engine authorizes the request and calls the *event_registration()* method with the process ID.
Step 3: Informer puts the process ID in the *notify()* method and register the *notify()* method to the notification chain.
Step 5: The event occurs.
Step 4: The notification chain invokes the *notify()* method and sends the signals to all process IDs that are contained in the *notify()* method.

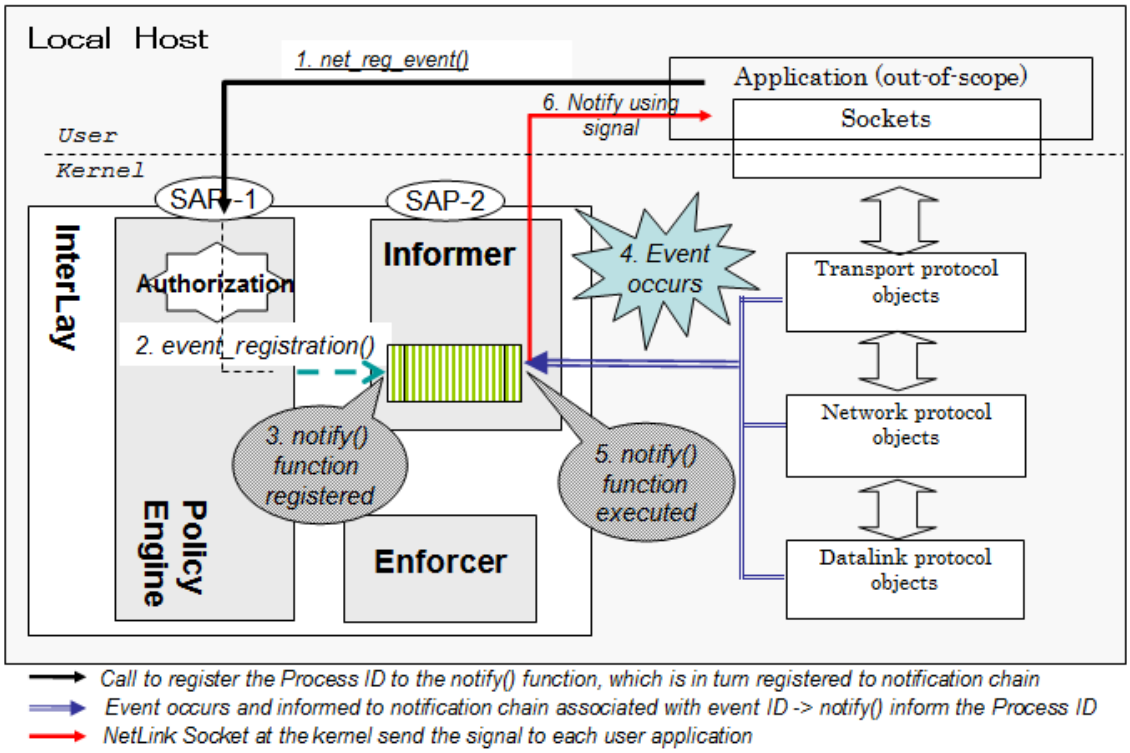

Figure 18. Event registration and notification (using signals)

3.10.3. InterLay and external systems

The InterLay will interact with external systems through the Message Handler of the PE.

A. Querying value of real-time parameters

The external server queries real-time parameters as depicted in Figure 19.

Step 1: External system sends a message requesting the local system to query the value of parameter with certain ID.
Step 2: The message is mapped to the *request_receive()* method .
Step 3: The *get_param()* of the Informer is called
Step 3: Informer returns value from the *get()* method to the MH which in turn composes a message and sends the value to the external system

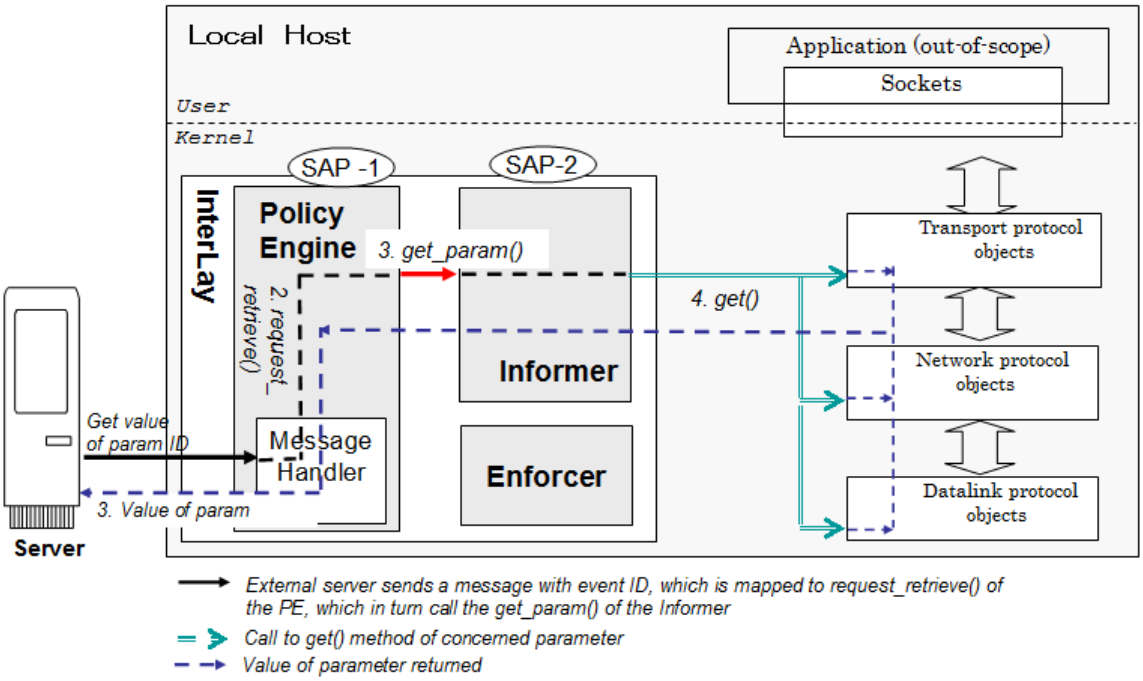

Figure 19. Querying value by external servers

B. Updating value of real-time parameters

The external system updates real-time parameters as depicted in Figure 20.

Step 1: External system sends a message requesting the local system to update the value of parameter with certain ID.
Step 2: The message is authorized and mapped to the *set_param()* method.
Step 3: The *update()* method of the Enforcer is called.
Step 3: Enforcer checks the integrity of the update request and calls the associated *set()* method of the concerned parameter.

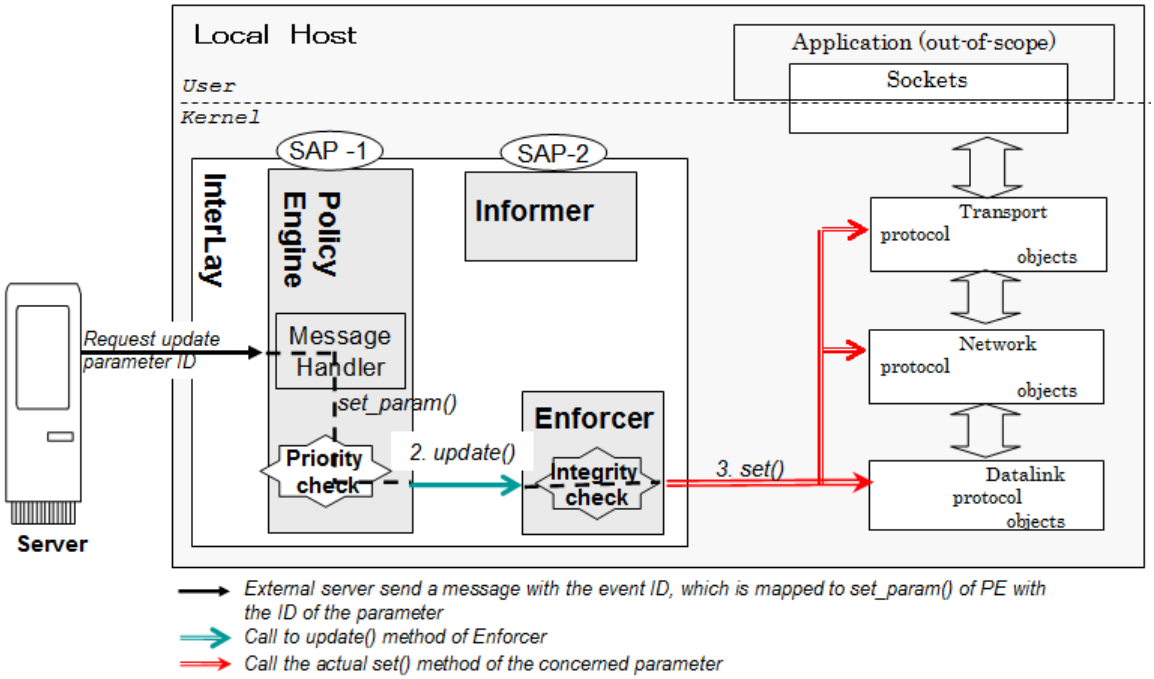

Figure 20. Updating value by external servers

C. Invoking *action()* methods

Figure 21 explains how the external system invoke an *action()* method.

Step 1: External system sends a message requesting the local system to execute an *action()* method with a certain ID.
Step 2: The message is authorized and mapped to the *ivk_action()* method.
Step 3: The *execute()* method of the Enforcer is called.
Step 3: Enforcer checks the integrity of the update request and calls the associated *action()* method of the concerned ID.

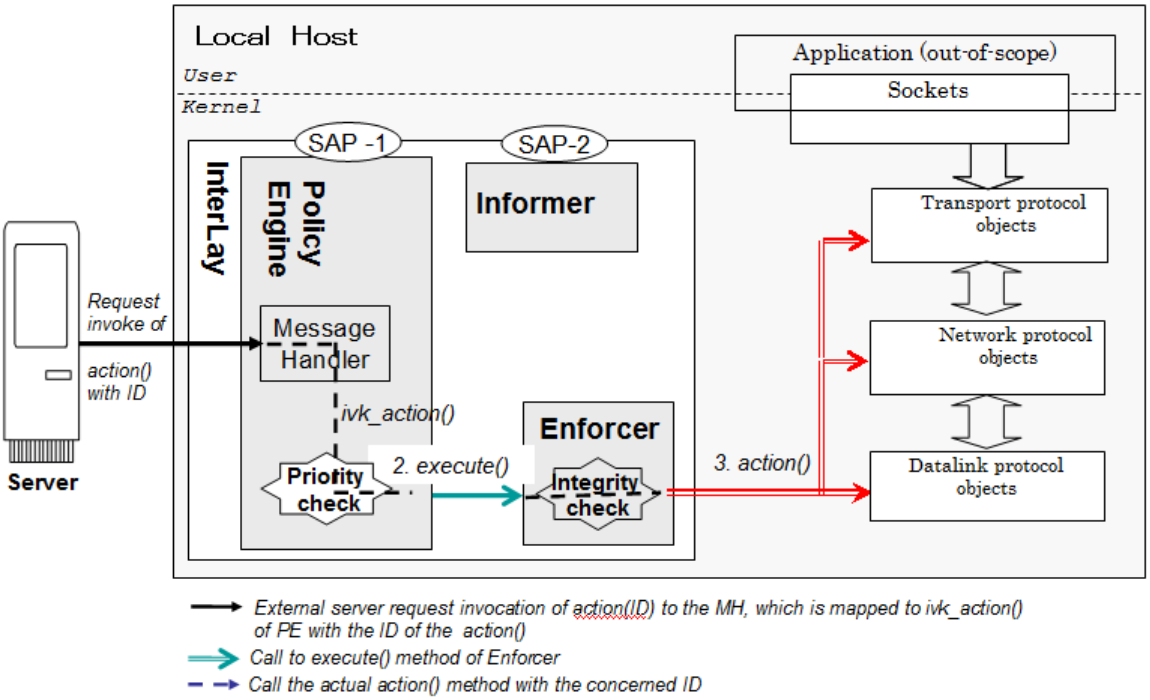

Figure 21. Invoking action() methods

D. Event registration and notification

Figure 22 explains how external system register and be notified for an event

Step 1: External system sends a message requesting the local system to register for the event with a certain ID and the message is authorized and mapped to the *request_register()* method.
Step 2: The PE calls the *event_registration()* of the Informer with the function pointer to *send_event()* method of MH.
Step 3: The *send_event()* method of MH is registered to the notification chain
Step 4: The event occurs.
Step 5: The notification chain invokes the *send_event()* method of MH, which in turn forms a notification message with the ID of the event that has just taken place and sends to the external system.

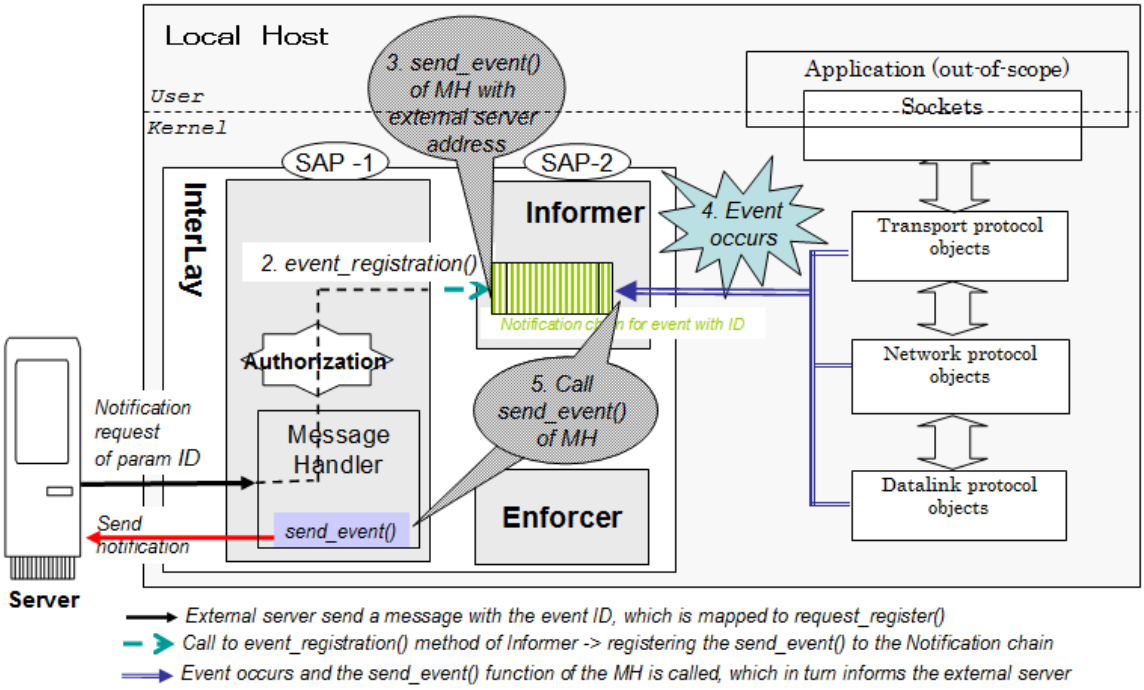

Figure 22. Registration for events from external servers

3.11. Service logic in the InterLay model

Service logic means how to organize the functionalities provided by the networking subsystem to provide a certain service feature. For example, in section 5.1.2.1, the service logic to maintain TCP session over IP address changes includes the following actions:

- Learning of the handoff event to a new Access Point (IP address).
- Freezing the TCP connection to prepare for mobility procedures.
- Requesting the Corresponding Node to update the destination address to the new IP address.
- Unfreezing the TCP connection and restart exchanging data between both ends.

The components of the service logic are the *set()* and *get()* methods of real-time parameters and notification of events, as well as the *action()* method.

In the InterLay model, the service logic will be carried out by the application, because the application has the most knowledge about the need of itself. The TCP/IP stack would merely provide the building blocks for the service logics. The advantage of this approach

is that it allows for the unlimited service scenarios; therefore it provides the utmost flexibility for communication service customization.

However, for certain service scenarios that are common or popular among many applications or services, forcing the service developers to re-create the same service logic each time would be inconvenience and causing unnecessary burden. In this case, when a certain service logic is deemed as useful for many services then it can be generalized and implemented as either an *action()* method of the TCP/IP stack or as a system call.

3.12. On the security of set() and action() methods

In most cases, real-time attributes are exposed by only a *get()* method. This read-only access protects the networking subsystem from potential error caused by incorrect implementation or being mistakenly set to a wrong value.

However, there are cases where it would be beneficial if the upper layers can change a certain attribute with a *set()* method, because when upper layers know the exact condition of the networking subsystem, there might be a need for them to change the state of the lower layers to a specific value for optimization or seamless operation. Because kernel modules are developed with more stringent quality management and testing, there would be no adversary effects when parameters of lower layer protocols are updated by either Transport or Network protocol objects (because these objects belongs to the kernel). However, there is no such guarantee for the development process of user applications, therefore socket objects (i.e. user applications) should be limited to manipulate only attributes that affect the specific session created by itself. This ensures that any improper use of the *set()* method will only affect sessions belonging to calling applications but not those belonging to other applications. In other words, this means that the socket object should normally not be able to manipulate *set()* method of attributes belonging to layer 3 (L3) object and below or otherwise it will affect all ongoing sessions.

Also, it is clear that the *set()* method should not be applied directly to protocol's attribute that has value obtained through negotiation with the corresponding peer without re-negotiation with that specific peer. For example, the TCP protocol should not expose *set()* method for the MSS (Maximum Segment Size) attribute, but a

MSS_renegotiate(new size) action method that carries out the re-negotiation of MSS will be provided instead.

For the *action()* method, it should be exposed by the InterLay only if it does not affect other ongoing connections except that belong to the requester. For example, IP protocol of Layer 3 can expose the *action()* method for route optimization procedure, but the method should affects only the binding cache for the destination associated with the requesting socket/application only.

CHAPTER 4. TEST QUESTIONS FOR SELECTION OF FINE-TUNABLE PARAMETER LIST

When the InterLay is deployed and implemented, initially the application developers might first experience some difficulty because they do not know what kind of information is available and how to use them. They need to know the behaviors and possible usages of parameters of all lower layer protocols. To reduce the burden of the developers in learning the behavior and parameters of all lower layer protocols, we need some mechanism to find all the suitable parameters for the InterLay model and document them properly.

Moreover, the InterLay model must also build a list of parameters from lower layers that should be opened for control from protocols of higher layers, either adjacent or several layers away. (we call this the fine-tunable parameter list).

Naturally, to take advantages of InterLay model, it is needed to cover as many suitable parameters to be exposed as possible. It means that the more comprehensive the list is the better. However adding a new network parameter to the list would require recompilation of the networking subsystems that means more development work, more complexity, higher possibility of failure etc.

Therefore, it is necessary to have a method to identify and categorize only parameters that are appropriate and exclude ones that would not be used across layers. In this chapter, the test questions are proposed as a method to identify all of relevant parameters, and because each test question in general defines a category, based on the test question we can roughly identify the usage of the parameter that agrees with a specific test question.

4.1. The test questions

The activities associated with the data exchange procedure between end-hosts can be divided into two categories of functions as in [3]: (i) data manipulation functions, which actually manipulate the data packet (such as read/write from network line, packet error detection, buffering for retransmission, encryption etc.) and (ii) transfer control functions, which regulate the transfer of the data (such as flow control, detecting network problems, acknowledgement etc). In addition, in certain types of protocols (such as reliable transport

protocols) connection establishment and termination must be carried out prior to and after the exchange of data.

Of the above two functional categories, intuitively the parameters that should be revealed to other layers should be those belonging to the transfer control functions, including information that reports changes of the network conditions/states that can affect either the transfer of data or the capacity of the networking subsystems received from external entities on the network. The test questions are then defined to determine whether the parameter should be exposed or not.

As depicted in Figure 23, each protocol will be screened to find its parameters, and these parameters will be examined with the test questions. The test questions are designed to find parameters that can improve performance of the connection or help the connection to overcome current problems of TCP/IP model such as mobility or fault-tolerance. The parameters that satisfy the test question (i.e. the answer is yes) will be added to the fine-tunable list and will be exposed to other protocols of other layers.

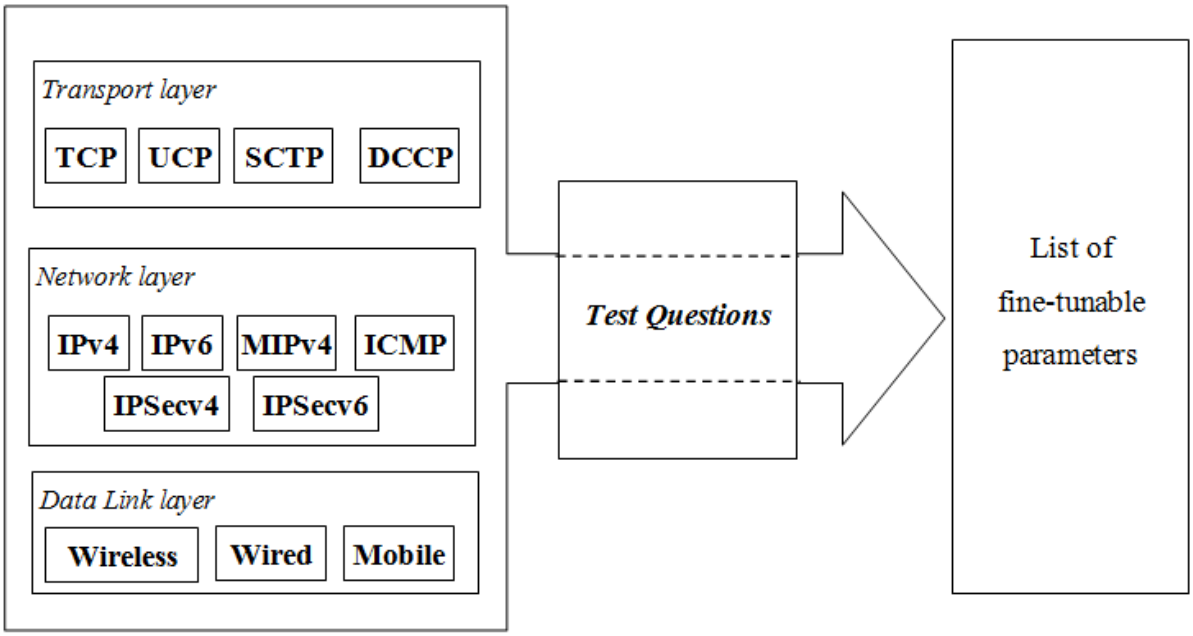

Figure 23. Screening of protocol for fine-tunable parameters

As a rule of thumb, real-time parameter that is associated with an event (i.e. the change in value of the parameter might trigger the event) should not be modified by outside entities. Also, a direct change to parameter obtained through probation or negotiation with

other peer is not preferable. If these types of parameters need to be updated then it is preferable to change it through the associated probation/negotiation procedures.

From current and foreseeable network capabilities vs. service demands, the following test questions have been created to examine such parameters based on their type, their role in improving performance or flexibility of the connection and their suitability for higher layer's condition and preference. Parameters that pass the test will be added to the list. The usage of the parameter that agrees with a specific test question will also be roughly identified.

4.1.1. For event parameters

We propose test question #1 as follows:

Question #1: Does the event parameter (both internally generated and externally received) signal a critical change of condition or status of the network?

The relevance of this question is that if the information about a critical change of condition of the network is available, it will allow higher layers (including user applications) to tune its activities to the current or near future condition of the network.

The internal generated event can be calculated from local parameters. For example, the time-out event happens when the TCP layer calculates that the value of RTT of a packet is larger than that of the RTO. Another example is when the Data Link layer (namely the network driver) finds that the radio signal of the Wireless interface is lower than a predefined value. On the other hand, the event can also be caused by the operation of the protocol. For example, the successful change of IP address can trigger some appropriate actions from the higher layers, such as updating transport layer connection's parameters or renegotiation of QoS for the existing flow(s), etc.

The external received event can be either directly or through some policy protocol. An example of directly received event is when TCP layer receives an ECN (Explicit Congestion Notification) from its peer. Upon receiving the indication that the network load is at critical load via the time-out or ECN event, the application can cooperate by switching to a slower codec to reduce the sending data rate. Or in the

case when the Wireless signal reaches a threshold so that an L2 handoff to another base station is imminent, the Mobile IP instance can carry out preparation for a re-registration to the Home Agent so that the actual re-registration process can be shorten.

4.1.2. For real-time parameters

We propose the following 4 test questions (the number is continued from the last section for convenient reference):

Question #2: Does the parameter indicate extra capability of individual communication session or of the system as a whole?

By asking protocols belonging to other layers to fine-tune its performance parameter, a protocol can synchronize its performance with other protocols to match with network status so that optimized performance can be achieved. For example, the type of access technologies being used can implicitly inform the application of the maximum data rate it can send or receive. If the application is informed that a 3G connection is being used, then a network video application should provide standard definition version of a movie instead of high definition one. However, this question will be more relevant in the future where the use of controlled network such as NGN is ubiquitous since the networking subsystem will be provided with more detailed information regarding the performance of the network.

Question #3: Does the parameter indicate the constraint of a configuration of the communication session?

A protocol may have many parameters that indicate upper or lower bounds for certain activities. Knowing these boundaries will help optimize or synchronize with parameters of other protocols. For example, knowing the value for Maximum Segment Size parameter of TCP protocol will help the implementer of Application Level Framing scheme [5] to set the Application Data Unit (ADU) size to multiple of MSS size, so that there will be no partial-filled packet at the end of each ADU.

Question #4: Does the parameter indicate the current progress of the communication session?

Checkpoint and acknowledged sequence number are good examples of markers for progress of a communication session. Knowing these values will help to recover a session to the exact state before failure for example. Together with parameters identified in question 5, it can also help in the process of handover of a connection from one machine to another for fault-tolerant/failsafe/failover purposes since it provides the state of the connection in the original machine before the handover.

Question #5: Does the parameter help to indicate a configuration of or identify the connection?

This question is relevant with more advanced or future capabilities of the networking subsystem, such as fault-tolerance/failsafe/failover of connection, or session mobility, security, etc. Currently, a session is still identified by IP addresses. However, a separate ID would make it easier for issues such as mobility [6] [7].

One example of the application of this question is that it is very complicated to provide reliable service across failure for TCP sessions within the current model of TCP/IP. However, if the application keeps track of the real-time static parameters (those from question #3) as well as the current progress (such as last checkpoint, sequence number which relate to question #4) then it will be easier for the application to restore the relevant session after a reboot.

The 5 test questions above come from current and foreseeable network's capabilities and services' demands, and it is needed to be constantly examined to find out new test questions and parameters to serve in different types of system or service scenarios. However, the principles of the test questions are universal enough so that when new service demands/network capabilities arise, new test questions can be introduced to find out new suitable parameters for the fine-tunable list.

4.2. Finding the appropriate network parameters

Figure 24 shows the TCP/IP protocol umbrella that will be examined with the test questions, with some new protocols that were developed and added to TCP/IP to meet

new demands. Note that only those protocols that are related to data communications connection management are shown.

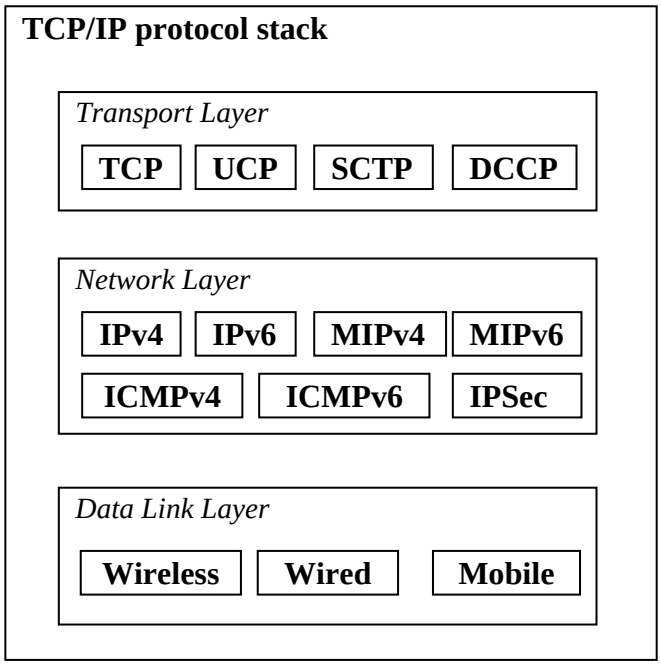

Figure 24. The TCP/IP protocol umbrella

By applying the above questions to each layer, we can summarize the list of some major parameters from common networking protocols in each layer that should be revealed to the applications and other higher layers in the tables of following sections (if the first column is checked then the parameter is an event parameter, otherwise it is a real-time parameter).

4.2.1. Lists of parameters for Layer 2 (Data Link Layer)

Table IV. For all type of Data Link protocols

R/E	*Protocol Family*	*Parameter name*	*Selected with test question #*	*Usage/Purpose*
	All	Type of access technology	Question #2	The general type of access network: WIRED/WIRELESS/MOBILE, or more detailed such as

				ETHERNET10M, WI-FI_G, 3G etc. From the type of the quest the application can infer in general the rate, QoS, Security, coverage area etc.
	All	MAC	#5	Because in general MAC is unique globally, therefore MAC can be used to support the creation of ID for higher layer address, security nonce etc.

Table V. For Wireless access Data Link Protocol

(Note: these parameters are recompiled from MIH [106])

R/E	*Protocol Family*	*Parameter Name*	*Selected with test question #*	*Usage/Purpose*
*	Wireless	Link Up	#1	The link is active and Layer 3 protocols can start exchange data through this link
*	Wireless	Link Going Up	#1	Upon receiving this event, the NS knows that the time to connect to an Access Point (AP) is longer than expected then the NS may consider connecting to another AP
*	Wireless	Link Down	#1	This event notifies that a link is not active and cannot be used for data transmission
*	Wireless	Link Going Down	#1	This event notifies that a link down event may be fired soon therefore the

				NS should, for example, prepare for a Layer 2 handover
*	Wireless	Link quality cross threshold	#1	This event notifies that the quality of the active link is under a pre-configured threshold for a sufficient time, which means that the NS should start preparing for a imminent handover
*	Wireless	Better quality AP available	#1	An Access Point with a better signal is available. The NS may consider to handover to this AP because it may, for example, provide higher and more stable data rate
*	Wireless	Link Handover Complete	#1	Notification of a fresh handoff. The application may need to readjust its data rate or transport layer may handover its transport connection to the new IP address (if Layer 3 handover also happens)

Table VI. For wired line access Data Link Protocol

R/E	*Protocol Family*	*Parameter name*	*Selected with test question #*	*Usage/Purpose*
*	All Wired tech.	Electrical Signal Instability	#1	The event of instability of electrical signal in an interface may suggest that the interface is going to be fail. Upon receiving this event the NS may choose to shutdown the interface to

				prevent route damping in routers or in end devices with more than one wired line interfaces
*	Ethernet	BCN	#1	Receiving the Backward Congestion Notification [11] can signal the congestion/non-congestion condition in the Ethernet environment

4.2.2. Lists of parameters for layer 3 (Network Layer)

Table VII. For all type of Network Layer protocols

R/E	*Protocol Family*	*Parameter Name*	*Selected with test question #*	*Usage/Purpose*
	IPv4, IPv6,	Source IP address	#5	Useful to restore connection over networking subsystem reboot or ad-hoc handover of transport layer connection
	IPv4, IPv6	Dest. IP address	#5	Useful to restore connection over networking subsystem reboot or ad-hoc handover of transport layer connection
	IPv4, , IPv6	IP packet options	#2	In the future options may be added to report network-wide Layer 3 condition, and this may be of interest to higher layers. Higher layers can also use options as a command code to instruct other ends to perform certain action on Layer 3 protocols

Table VIII. For Mobile IP protocols

R/E	*Protocol Family*	*Parameter Name*	*Selected with test question #*	*Usage/Purpose*
	MIPv4	Foreign Agent CoA	#5	This parameter shows the current attachment point that can be used for any ad hoc route optimization
	MIPv4 MIPv6	Colocated CoA	#5	This parameter shows the current attachment point that can be used for any ad hoc route optimization (Note: MIPv6 is not a separate protocol but a part of IPv6 protocol)

Table IX. For IPSec protocols

R/E	*Protocol Family*	*Parameter Name*	*Selected with test question #*	*Usage/Purpose*
	IPSec	Security association	#5	Useful to restore security connection over NS restart

Table X. For ICMP protocols

R/E	*Protocol Family*	*Parameter Name*	*Selected with test question #*	*Usage/Purpose*
*****	ICMP-v4 ICMP-v6	Destination Unreachable Event	#1	This is an external event received from the network. Upon receiving this report the application may choose to connect to another backup destination

				address or process. Also depending on the exact code the application may carry out other reconfiguration activities such as allowing fragment or changing TOS field
*	ICMPv4 ICMPv6	Time Exceeded	#1	Depending on the code the application will learn about the problem of either the network or its peer host
*	ICMPv4 ICMPv6	Source Quench	#1	Transport layer and application should carry out procedure to reduce transmission rate

4.2.3. Lists of parameters for layer 4 (Transport Layer)

Table XI. For all Layer 4 protocols

R/E	*Protocol Family*	*Parameter Name*	*Selected with test question #*	*Usage/Purpose*
	All protocols	MSS	#5	Knowing this value, the application can synchronize the ADU size of Application Level Framing protocol to that of multiple of MSS to optimize the transmission
	All protocols	Source Port	#5	Used for ad hoc mobility or fault-tolerance support for transport layer connections
	All protocols	Destination Port	#5	Used for ad hoc mobility or fault-tolerance support for transport layer connections

Table XII. For congestion enabled protocols

R/E	*Protocol Family*	*Parameter Name*	*Selected with test question #*	*Usage/Purpose*
	TCP, SCTP, DCCP	MSS	#5	Knowing this value, the application can synchronize the ADU size of Application Level Framing protocol to that of multiple of MSS to optimize the transmission
*	TCP, SCTP, DCCP	ECN received	#1	The event implies congestion in the Tx direction. The application can respond by tuning its speed by changing to lower codec etc.
*	TCP, SCTP, DCCP	RTO event	#1	The application knows that a RTO just has taken place so it can adjust its sending rate
	TCP, SCTP, DCCP (with CCID 2)	Window size	#5	The application can lower the windows size to response to network congestion report, for example, via ECN or ICMP etc.
	TCP, SCTP, DCCP (with CCID 2)	Sequence Number	#4	Used for fault-tolerance support for transport layer connections

4.2.4. Lists of parameter for layer 5

In the TCP/IP layering architecture, there is no official definition and implementation of layer 5 (Session). Instead, the features of layer 5 protocol are implemented individually by application protocols which require such functionalities. Because in the conventional TCP/IP architecture this information resides in the application layer, the parameters are basically out-of-scope of the InterLay model. However, finding these parameters using the test questions and documenting them is also beneficial to application developers, as they have more information on what kind of parameters are available for service optimization and customization.

Table XIII. For parameter of Layer 5

R/E	*Protocol Family*	*Parameter Name*	*Selected with test question #*	*Usage/Purpose*
	N/A	Check-pointing value	#4	This item can be used to mark the current status of the connection to use with fault-tolerance activities etc…
	N/A	Session ID	#4	This item can be used to mark the current status of the connection to use with fault-tolerance activities etc…

4.3. Characteristics of the test questions' approach

The test questions approach to identify and categorize parameters in this paper has the following characteristics:

- It provides an appropriate methodology that can identify suitable parameters that should be added to the fine-tunable list, while excluding those ones that are not suitable. This reduces the amount of parameters that should be opened to other layers, resulting in less development time and effort, as well as less fault for InterLay scheme.

- The test questions can be used as a means to categorize parameters based on functionality, therefore it can make the development of specialized systems faster with less work, because parameters belonging to the category that are not needed will not be opened altogether. For example, embedded system for real-time Internet-radio streaming devices can remove the support for all parameters related to question #4, because in real-time streaming services there is no need for session check-pointing.
- The methodology of using parameter types and test questions are general enough to be applied not only to InterLay scheme, but also to any other network architectures that support cross-layer communication among protocols.
- And because of this well documentation of network parameters, the developer does not need to learn about all protocols, he just needs to concentrate on the parameters that seem most relevant to the application protocol he wants to design.

CHAPTER 5. DISCUSSION AND ANALYSES

5.1. Coverage of InterLay scheme

In this section we will examine how the InterLay covers existing modifications to the TCP/IP networking stack. Due to the development of the Internet that is explained in Table 1, mobility, fault-tolerance and security are the new add-ons to the original TCP/IP networking model.

5.1.1. InterLay scheme and TCP fault-tolerance across local networking subsystem restart

The manipulation of the Internet Protocol Control Block (Internet PCB) and TCP Control Block (TCB) can be used to save the TCP session over the restart of the subsystem.

To control and manipulate TCP session, the following *action()* methods should be provided by the TCP/IP stack. These methods belong to Group B of extra functionalities by the TCP/IP stack as explained in section 3.4:

- *act_freeze()* and *act_unfreeze():* These two *action()* methods will act upon on a `FREEZE_FLG` flag. The TCP object will be required to make sure that the `FREEZE_FLG` to be off before sending or receiving a datagram to/from the IP object. The *act_freeze()* method will turn on the `FREEZE_FLG` which will freeze the sending/receiving activities at the TCP and IP protocol object. However, the socket will continue to read data that is already in the receiving buffer until it is empty because the data have been ACKed. On the other hand, the *act_unfreeze()* method will turn off the `FREEZE_FLG` which will restart the sending/receiving activities at the TCP and IP protocol object. The application can invoke this *action()* method using the *net_invoke_action()* system call with the *`FREEZE_ID`*

and *UNFREEZE_ID* of the *act_freeze()* and *act_unfreeze()* action methods, respectively.

Let's suppose that the networking subsystem (NS) is about to restart due to some errors, and that it can communicate with active sockets before restarting, then the preservation of the TCP session is carried out as follows (see Figure 25):

Step 1: The application that wish to have its connections to be fault-tolerant registers for the NET-RESTART-EVENT in advance with the InterLay using the socket's *net_reg_event()*

Step 2: The NS informs the registering applications with the imminent PRE-NET-RESTART-EVENT through the InterLay.

Step 3: The application requests the NS to freeze the sending/receiving activities for the socket, by calling the *net_invoke_action()* system call with the pre-assigned FREEZE code as input argument.

Step 4: After confirming that the calling socket has the right to the concerned TCP object (session), the PE requests the Enforcer to invoke the *TCP_object.act_freeze().* This will freeze the sending/receiving activities. However, the socket will continue to read data that are already in the buffer until it is empty because the data have been ACKed.

Step 5: The application calls the *net_get_param()* system call, with the TCP-TCB parameter code to get the TCB of the socket.

Step 6: The Informer will query the *TCP_object.get_TCB()* and returns the result to the application (the returned result contains all the status information of the TCB, as well as any unsent buffer)

Step 7: The application calls the socket's *net_get_param()* with the IP-PCB parameter code to get the PCB of the socket

Step 8: The Informer will query the *IP_object.get_PCB()* for the concerned socket, and returns the result to the application (the returned result contains all the status information of the PCB)

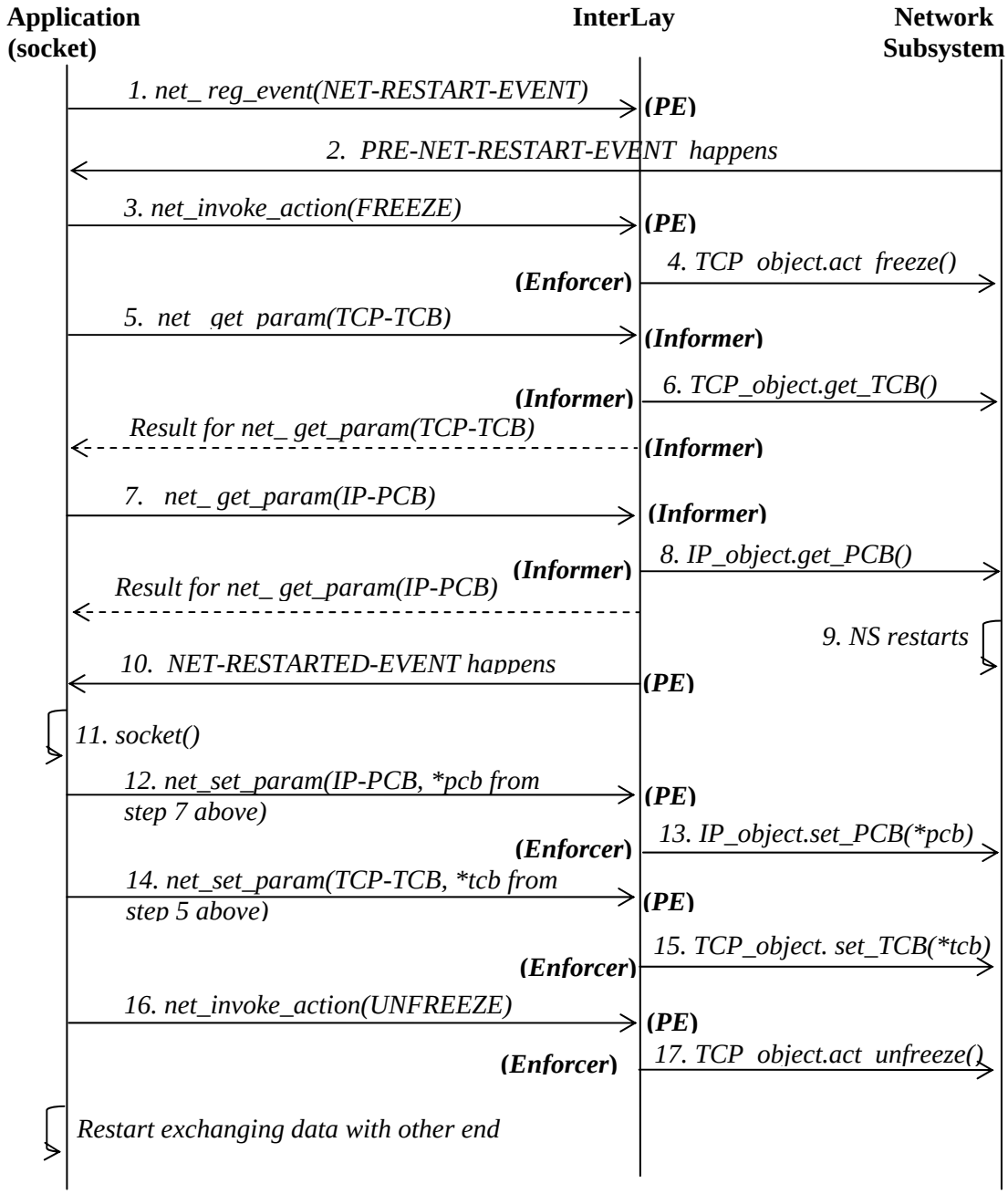

Figure 25. The interaction diagram for TCP's fault-tolerance procedure
(Note: the dotted lines denote returning value for net_get_param())

Step 9: Now the NS will restart itself and the application will delete the socket instance.

Step 10: After the NS is restarted, the `NET-RESTARTED-EVENT` is sent to all applications that register for the `NET-RESTART-EVENT` *(t*he list of those applications is saved by the NS before restarting)

Step 11: The application responses by creating a socket with the *socket()* function.

Step 12: The application then requests the InterLay to update the Internet PCB for the created socket with the *net_set_param*() function and 2 input arguments, one is the `IP-PCB` parameter code for the PCB and the other is the `*pcb` value that is returned in step 7 above.

Step 13: The PE will inform the Enforcer to invoke the *IP_object.update_PCB()* action method.

Step 14: The application then requests the InterLay to update the TCP's TCB for the created socket with the *net_set_param()* system call function and 2 input parameters, one is the `TCP-TCB` parameter code for the TCB and the other is the `*tcb` value that is returned in step 5 above.

Step 15: The PE will inform the Enforcer to invoke the *TCP_object.set_TCB()* action method.

Step 16: Upon receiving the confirmation of successful update of the new socket, the application requests the network subsystem to unfreeze the sending/receiving activities for the socket, by calling the *net_invoke_action()* system call with the pre-assigned `UNFREEZE` code as input argument.

Step 17: After confirming that the calling socket has the right to the concerned TCP session, the PE requests the Enforcer to invoke the *TCP_object.act_unfreeze()* action method that turns off the `FREEZE_FLG` to restart the sending/receiving activities for the socket.

When the two sides can resume actual sending and receiving data:

After step 17, the newly created TCP socket is the exact duplicate of the original one, and it is ready to send any data that is available at the buffer, and start receiving the data from the application residing at the corresponding node.

However, at that time the TCP session at the corresponding node side might already time out (this is due to the restart of the Networking Subsystem at the local node, NOT due to congestion), so even if the TCB is updated, no data exchange is possible until the time out is over, which can be very long. The corresponding node can be stimulated to send data sooner by letting the local node to send three consecutive ACKs for the last received data so that the corresponding node can enter the fast retransmit state.

When the networking subsystem is in the process of restarting, if the TCP data packets from the corresponding node arrive, then the CN will be returned with an ICMP's Unreachable, or the maximum retransmission timeout for the TCP session has been reached (e.g. because restarting of the networking subsystem at the local node takes too long), which might resets the TCP at the corresponding node. If this is the case, then after step 2 the application at the local node can either ask the InterLay directly or through the peer application at the corresponding node to not terminate the concerned TCP session even if an ICMP's Unreachable is received or the maximum retransmission timeout for the TCP session has been reached. Of course, this protection should only be activated for a certain short period so that if the local node does not go back online then the TCP session will eventually be terminated.

Note that this scheme is extendable to the case of UDP (no need for step 5 and 14, and any data sent by the other end during the period will be lost) as well as to the case the OS reboots provided that the application is given enough time to perform the above procedures and the restoration process is fast enough so that the connection is not aborted first by the other end.

5.1.2. InterLay scheme and maintaining TCP session over IP address change

Mobility in the Internet can be introduced in IP layer [28], or the TCP layer. In this section, we will examine how InterLay can maintain a TCP session without underlying IP layer's supports when the mobile node performs a handover to a new IP address, or from one interface to another of the same device. There are three scenarios: maintaining TCP session over handoff at one end only in 5.1.2.1, between interfaces in 5.1.2.2, and over handoff at both ends in 5.1.2.3.

Because there are also existing researches that provide the same protection of TCP session, the comparison among them is described in Appendix II.

To provide TCP mobility, the following extra *action()* methods should be provided by the TCP/IP stack. These methods belong to Group B of extra functionalities by the TCP/IP stack as explained in section 3.4:

- *act_mobi()*: The TCP session is identified by the IP addresses of both ends, therefore conventionally when the IP address changes (due to handoff) the TCP session will be terminated by the TCP/IP stack. However, InterLay model provides the application with an *act_mobi()* action method in the TCP/IP stack that basically turns on a flag that will protect the TCP from being terminated when the IP address changes (i.e. maintaining TCP state information, including Internet PCB, TCB, and all the necessary buffers). The application can invoke this action method using the *net_invoke_action()* system call with the `ACT_MOBI_ID` of the *act_mobi()* action method.

5.1.2.1. Maintaining TCP session over handoff at one end only

Existing solutions require the modification of TCP protocol or the TCP/IP architecture as a whole, while the InterLay approach can support TCP mobility through manipulation of the Internet PCB and TCB, as well as events from lower layers.

In [18] we have proposed the mobile TCP socket that supports mobility for TCP session. Basically, this mobile socket is a special version of the InterLay model which specializes in supporting only TCP mobility, by providing an interface to change (i.e. the *set()* method) the PCB and TCB parameters to maintain TCP session across address change. We have analyzed in [18] that the inter-layer exchange of information provides some advantages over other approaches, namely (i) the maintenance of TCP session across handoff is carried out only if the application finds it beneficial, and (ii) the maintenance process can make use of existing security association, which reduce overhead (in terms of both traffic and processing) and latency.

The operation of TCP mobility will be carried out as follows (Figure 26):

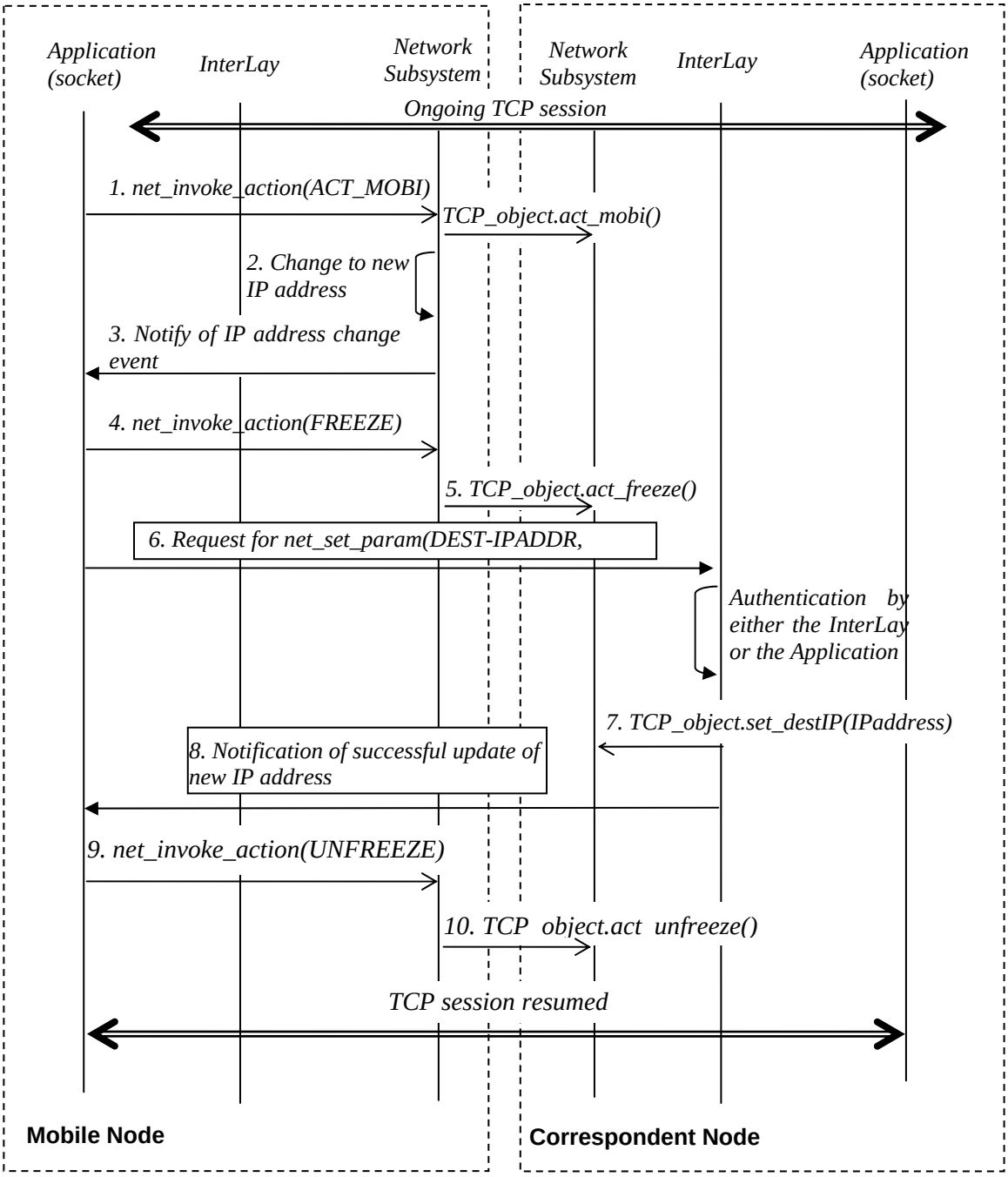

Figure 26. The interaction diagram for maintaining TCP session over IP address change

Step 1: An application at Mobile Node (MN) is sharing an active TCP connection with another application in the Correspondent Node (CN). The mobile host can query Layer 2 via the InterLay on the possibility of a near future handoff. The possibility can be

decided via for example signal strength or whether the interface has just been handed-off more than twice recently, which is a signal that the mobile host is moving fast and hence a handoff is imminent. If the mobile host is deemed to experience a handoff in the near future, and the application knows that the TCP session would be a long one, then the application will call the *act_mobi()* method to maintain the socket even in the case the IP address is changed due to a handoff. It will also register for the Link-Up event from the networking interface.

Step 2: The interface that is used by the TCP connection at the MN is being handed-off to another access point, and as a result the IP address changes to a new one. Because the *act_mobi()* method has been invoked by the application, the TCP session that is associated with the socket is maintained.

Step 3: The application also registers in advance (namely step 1) to the notification of address change for the concerned interface. Therefore, the application is notified of the address change, and the notification will include the new IP address.

Step 4: The application requests the network subsystem to freeze the sending/receiving activities for the socket, by calling the *net_invoke_action()* system call with the pre-assigned FREEZE code as input argument.

Step 5: After confirming that the calling socket has the right to the concerned TCP object (session), the PE requests the Enforcer to invoke the *TCP_object.act_freeze()* method that turns on a *FREEZE_FLG*. *TCP_object* is the TCP object created by the concerned socket at the MN. This will freeze the sending/receiving activities at the TCP and IP object, but the application can still process any data that is already available at the socket's receiving buffer.

Step 6: This new address should be informed to the CN. In this service scenario, the application at the MN will send a request to the InterLay at the CN to perform the *net_set_param()* action to update the destination IP address (the address of the MN is the destination address at the CN side). The request can be relay by any message-exchange protocol.

The Message Handler (MH) of the InterLay at the CH is the destination of the request to update destination address. As explained in section 3.7.2.1, there are two ways to authenticate this request:

A) Authentication by the Interlay: When the application at the MN establishes the TCP connection with the peer at the CN, it also exchanges a share secret with the CN's peer and the latter informs the MH of this share secret. The InterLay will use this secret to authenticate the request from the MN. In Figure 26 we depict case A.

B) Authentication by the application at CN: Upon receiving the request from the MN, the MH at the CN will send the request to the application (in this case the MH does not need to understand the request; the application will have to parse the request and learn that the application at the MN wants it to update the MN's address).

Step 7: In this step, the InterLay will ask the Enforcer to update the destination address from the old to the new address of the MN. Depending on how the request is authenticated in step 6, the request to the Enforcer will come directly from the CN's application using the *net_set_param()* system call in case B of step 6, or it comes from the application at the MN via the Policy Engine in case A. In Figure 26 we depict the case A.

Step 8: If the update of destination address in step 7 succeeds, then either the MH (in case A of step 6) or the application (in case B of step 6) will inform the application at the MN about this success. In Figure 26 we depict the case A.

Step 9: Upon receiving the confirmation of successful update of the new IP address from CN, the application requests the network subsystem to unfreeze the sending/receiving activities for the socket, by calling the *net_invoke_action()* system call with the pre-assigned `UNFREEZE` code as input argument.

Step 10: After confirming that the calling socket has the right to the concerned TCP session, the PE requests the Enforcer to invoke the *TCP_object.act_freeze()* method with the `UNFREEZE` code that turns off the *FREEZE_FLG*.

When the two sides can resume actual sending and receiving data:

The MN will be ready to receive data after sending the *update_pcb()* request to the CN (in step 7), and it can start sending data (i) after receiving the confirmation of the successful update to the new IP address from the CN and unfreezing the sending activity (in step 10) or (ii) after receiving the data from the CN (in the case the data from CN comes before the confirmation).

On the other hand, the application at the CN side will be ready to send and receive data together with the confirmation of the successful update to the new IP address to the MN.

5.1.2.2. Transferring TCP session between network interfaces

Supporting handover between multiple interfaces will be an important feature, since more and more mobile devices will be equipped with both wireless WAN and wireless LAN interfaces. Because InterLay supports the update of both IP address and Port for a TCP session, it can support the handover between interfaces of the same device.

There are two cases where the application might consider a local handoff of TCP session between two local interfaces: (case 1) if the signal quality in the currently used interface is going below a threshold but another access point is not available and (case 2) if an interface more suitable to the application than the currently used one is going up (for example a faster nominal speed, or cheaper cost or wider coverage etc).

In general, the process for the application to switch the TCP session from one to another interface is that first it creates a new socket on the target interface, and then uses the *set_PCB()* and *set_TCB()* action method to copy the information from the old to the new socket, in a similar manner to what happens in section 5.1.1. Next, the application sends out the notification to the CN to update the new IP address and Port, just like in the case explained in 5.1.2.1.

The detailed procedure for TCP mobility between two local interfaces (interface A and B) is explained as follows (see Figure 27).

Step 1: An application at Mobile Node (MN) is sharing an active TCP connection with another application in the Correspondent Node (CN).

In case 1 discussed above, the application at the MN needs to register for the Link-Going-Down event or the Link-Down event from the wireless interface that is used by the TCP socket, while in case 2 the application will register for all Link-Up and Link-Down events (these events are described in table V in chapter 4).

Step 2: In case 1 above, when the application receives the Link-Down event, if after a certain amount of time it does not receive a Link-Up event it will start the procedure to handoff the TCP session to another active interface that is available, otherwise the session might be closed. The wait-time for a Link-Up event can be decided case-by-case depending on the nature of the application, but for example it can be started after the application receives a time-out event from the TCP object.

In case 2, the application constantly monitors the link-up and link-down event and maps the available Internet connection at the MN with its characteristics. The mapping can be made using information provided in advance by the user, or can be retrieved through some kind of policy exchange with the access network, or both. When it receives the information of a Link-up event for a network interface that is judged to be better fit for the communication session than the existing one for the ongoing session, it will start the procedure to handoff the TCP session to the new interface.

The application might consider the following criteria when making such judgment in case 2:

- The expected remaining lifetime of the session.

- The expected stability of the current and new network interface connection (by using the information mentioned in the above paragraph).

- The more suitability of the characteristics of the new network interface connection compared with the current one.

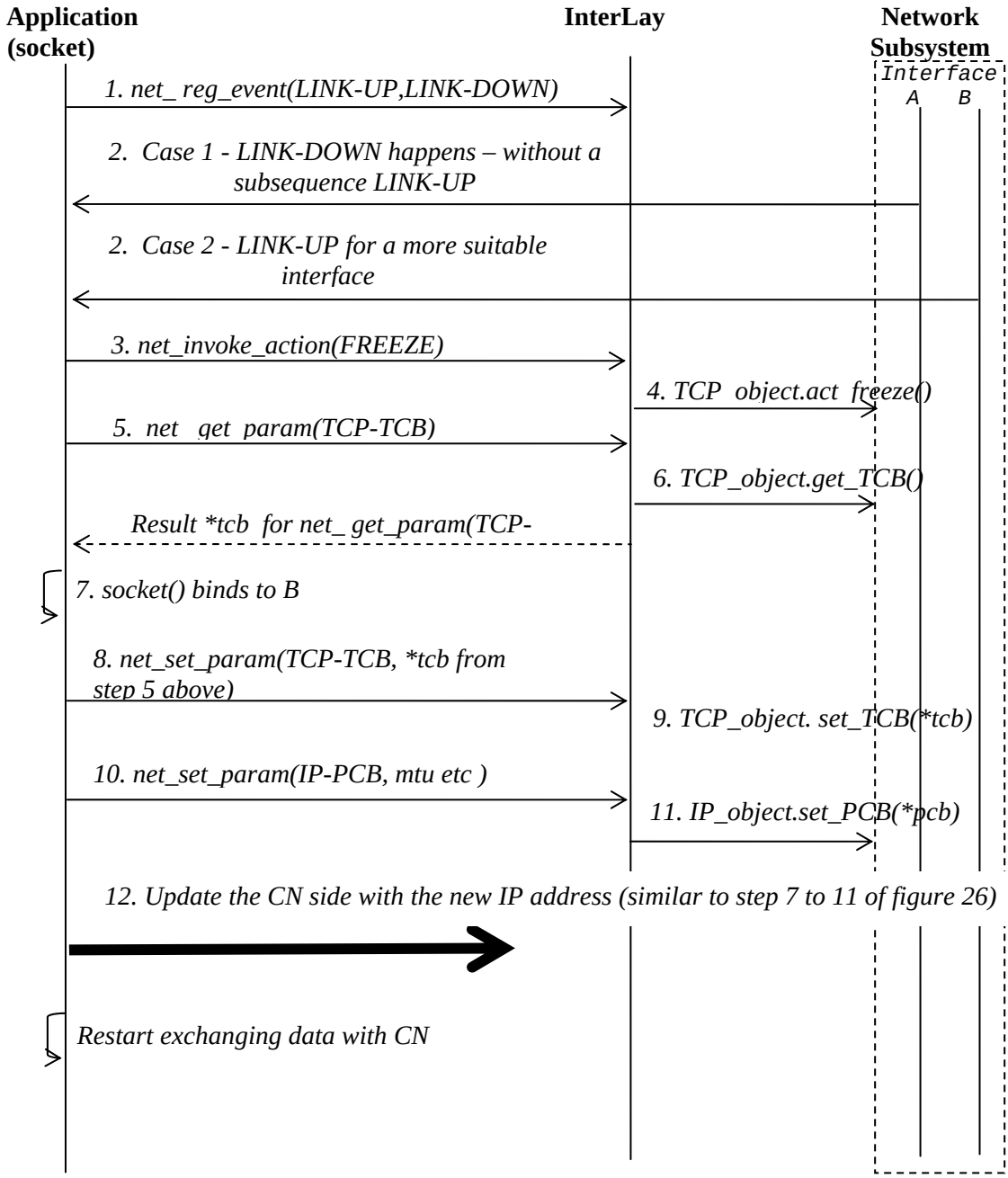

Figure 27. The interaction diagram for maintaining TCP session between interfaces

Step 3: After making the decision to transfer the TCP session from one interface to another, the application starts the procedure by first requesting the network subsystem to freeze the sending/receiving activities for the socket by invoking the *net_invoke_action()* system call with the pre-assigned FREEZE code as input argument.

Step 4: After confirming that the calling socket has the right to the concerned TCP object, the PE requests the Enforcer to invoke the *TCP_object.act_freeze()* method. This will freeze the sending/receiving activities at the TCP and IP object, but the application can still process any data that is already available at the socket's receiving buffer.

Step 5: The application calls the *net_get_param()* with socket A, with the `TCP-TCB` parameter code to get the TCB of the socket.

Step 6: The Informer will query the *TCP_object.get_TCB()* and returns the received `*tcb` structure to the application (the returned result contains all the status information of the TCB for A, as well as any unsent buffer)

Step 7: The application creates a TCP socket and binds it to the target network interface B.

Step 8: The application then requests the InterLay to update the TCP's TCB for the newly created socket with the *net_set_param()* function and 2 input parameters, one is the `TCP-TCB` parameter code for the TCB and the other is the `*tcb` value that is returned in step 5 above.

Step 9: The PE will inform the Enforcer to invoke the *TCP_object.set_TCB()* action method with the `*tcb` received in step 5 above.

Step 10: The application requests to update IP protocol specific parameters, such as MTU etc,

Step 11: The PE will inform the Enforcer to invoke the *IP_object.set_PCB()* action method for socket B.

After this step we have a ready and identical socket binding to interface B at the MN. The only difference is that the data packets will now be routed to/from another IP address associated with another interface.

Step 12: The next step is to update the CN side with the new IP address. Because to the application at the CN, the change of the interface is just the change of IP address, therefore the step can be fulfilled by repeating step 7 to 11 of 5.1.2.1.

After this step the TCP session is ready for exchanging of data, and the two sides can resume actual sending and receiving data in the same manner as in the case of section 5.1.2.1.

5.1.2.3. Maintaining TCP session in the case both ends handoff simultaneously

When both ends handoff simultaneously, the procedure in 5.1.2.1 cannot be applied, because the MN and CN do not know of the new IP address of each other so that they can not inform each other of the change of address. In this case, a rendezvous point is needed that forwards the request to update the new target IP address from the MN to the CN and vice verse.

The rendezvous point is a fixed node that

- keeps track of the address of both parties
- has some security association with both parties (for authentication of requests)

Because both ends handoff simultaneously, it means that both ends are mobile devices, which in normal case both are user devices. Therefore two parties are participating not in a client-server communication model, but rather in a peer-to-peer communication service. Such a service can be for example, voice call or video phone call between two customers. For this type of service, normally a server for the service will act as the intermediary agent that helps both users to find each other and to connect to each other. For example in SIP (Session Initiation Protocol) service, the SIP server always knows about the address of its SIP users through the registration process, and always establishes security associations with its SIP clients.

The major hurdle for maintaining TCP session in the case of simultaneous handoff is to find an extra rendezvous point. However, as explained in the above paragraph, in the case of simultaneous handoff, both sides are normally involving in a peer to peer communication service, and it is intuitive to see that the server for the service satisfies the above two requirements for a rendezvous point, so fortunately a rendezvous point is always available for this kind of service scenario.

The procedure to maintain TCP session in simultaneous handoff scenario is described as follows:

Stage 1: Handoff at end devices

In this stage both ends invoke the *act_mobi()* method and register for the LINK-UP event. When the interface at each ends finishes a handoff, the SIP application at each end will be notified and in turn it will send the request to update the IP address for the TCP session to the SIP server using a SIP message (after it re-registers with the SIP server at the new IP address).

The actions related to TCP mobility in this stage are similar to step 1 to 6 in Figure 26, with an exception that the request to update the IP address is sent to the SIP server instead of directly to the other peer.

Stage 2: Dispatching of IP address update requests to corresponding peer

After the SIP server receives the request to update the IP address from a SIP application, it will forward the request to the other SIP application. There are two possibilities regarding the status of the other SIP application:

- It is reachable at the current registered address: In this case, the IP address that the target end node registers with the SIP server is still valid, either because it has not experienced a handoff, or just completed a handoff and re-registration with the SIP server. In this case the request is relayed using a SIP message as normal.
- It is in the process of being handing-off: The other SIP application is not reachable at the moment, and the SIP server will wait for the re-REGISTER message from the SIP application to find its new address and then relay the request again to the new registered address.

Stage 3: Updating of the destination IP address at the corresponding peer

In this stage, after the SIP application at the corresponding peer authenticates the SIP message, it will ask the InterLay to update the destination IP address of the corresponding TCP session to that of the request. The procedure is similar to step 7 and 8 (for case B) of Figure 26. After the success of step 8, the SIP application at the corresponding peer will send a success notification to the SIP server, and start sending data to the new destination IP address.

Stage 4: Informing of the update success to the host peer

Upon receiving the success notification from the corresponding peer, the SIP server will notify the host peer that sends out the update request of this result by using a SIP message. The SIP application will then activate the TCP session using procedures in steps 9 and 10 of Figure 26. After this stage the TCP session is ready to be used for exchanging data.

We can see that the procedure for simultaneous handoff composes of essentially two procedures for single handoff (described in section 5.1.2.1), with the exception that the notification is not sent directly to the corresponding peer but is relayed through the SIP server, which knows the address of both ends even in the case of simultaneously handoff. It is also intuitive that the 4-stage procedure above is applicable in the case only one end performs a handoff.

This service scenario can also be applied to the case both ends transfer the TCP session from one interface to another, with stage 1 now using steps 1 to 11 of Figure 27.

Because there are two scenarios for handoff (namely directly between end devices as explained in section 5.1.2.1 or via an intermediate server as in this section), with the differences in the destination for the request to update address, the criteria to choose a suitable scenario are:

- In client-server communication model, one end (namely the server) normally has a fixed IP address; therefore for application that employs this communication model the handoff scenario described in section 5.1.2.1 is more suitable.
- In the peer-to-peer service model, both ends are potentially mobile, and normally an intermediate service server is available that can be used as rendezvous point as described in this section , therefore the handoff scenario described in this section is more suitable.

5.1.3. InterLay scheme and SHIM layer

Because the InterLay scheme allows for prepending extra information to the PDU, it can easily handle SHIM-layer type of modification, such as that of Host Identity Protocol

[29]. Moreover, using this prepending ability, we can create tunnels without the need to introduce new protocols such as [49].

5.1.4. InterLay scheme and Route optimization

By exposing the internal activities of a protocol to outside layers, we can provide even more flexibility to the end user application. For example, in Mobile IPv6, when route optimization (RO) is used then even if the communication session is short and small, RO signaling is still being carried out, which creates processing and signaling overhead. This is especially can be troublesome if, for example, the Correspondent Host is a popular http server with many small html pages, such as microblogging service, or if the Mobile Host is moving fast from one access point to another.

If InterLay scheme is used, then the application will be provided with a interface to a *action()* method at the MIP protocol object, which accepts the destination (CH's) IP address as parameter, that performs the RO procedure to that destination. The advantage of this approach is that the application is the one that knows about its communication need the most, therefore it can make the best decision. For example RO is activated only if the application decides that its communication session is traffic-heavy or long-lived. In addition, the Data Link (i.e. network driver) object can provide a *get()* method to inform the application about the handoff frequency, and the application can further decide that if the frequency of handoff is high then RO should not be activated.

5.2. Advantages and benefits of InterLay model

5.2.1. Advantages of the InterLay model

The new InterLay model helps overcome some limitations of the traditional TCP/IP architecture and shows its superiority over other related schemes which are of narrower approach based on a case-by-case basis. This is due to its advantages described as follows:

5.2.1.1. Advantages that come from the approach of separating the cross-layer activities from the main networking activities

These advantages include:

- Any changes in the cross-layer communication mechanism will not affect the TCP/IP stack and vice versa.
- The core TCP/IP stack is similar to conventional implementation, performing all activities related to sending/receiving data. Therefore, all expertise/code of existing implementation can be reused, reducing development time as well as possible errors.
- Because the InterLay entity can be modified, enhanced, and upgraded independently in term of cross-layer related functionality/capability without requirement to change the core TCP/IP stack, it lessens the chance of the TCP/IP stack inadvertently being wrongly modified.
- The centralization of the management for cross-layer activities in one entity allows the uniformed access to a network parameter from all other layers. This simplifies the analysis and design process as well as reduces the confusion for developers when accessing the parameter from various layers.
- Moreover, nowadays network policies are widely employed in various forms for various purposes, the InterLay model will also allow the exchange with (and possible control from) external policy servers to synchronize the operations of the local networking subsystem with the conditions of the overall external network.

5.2.1.2. Advantages that come from the wide coverage of the Interlay model

The InterLay model is a single implementation but it can support many developments of the TCP/IP architecture.

It allows for networking subsystem and user applications not only to be able to adapt the performance according to lower layers status, but also to support for:

- more choices beside performance (such as arbitration decision for route optimization) and
- coverage for future changes and requirements (for example, it can support new extensions to original TCP/IP model such as TCP mobility, SHIM-layer activities, IP tunneling … without difficulty). For example the InterLay scheme can support new requirements without requiring development of new protocols at the

This supports more timely development and deployment of future services, because the application designers now do not have to wait first for a new protocol to be established, standardized and implemented in the OS kernel to enable the newly emerged changes and requirements.

However, there are cases when eventually a new protocol that standardizes the new requirements will be rolled out. In this case the application designers can just update their application with the new protocol, and the abilities of the InterLay object will continuously be used to serve other newly surfacing problems and requirements.

5.2.1.3. Advantages that come from the InterLay model's ability to coexist with other different cross-layer design proposals

The Interlay model provides not only attributes (and associated *set()* and *get()* methods) but also *action()* methods, it can serve as a general platform for other proposals to implement their algorithms upon. For example, the algorithms proposed in [19] can be inserted as *action()* methods in relevant protocols in this new TCP/IP architecture, together with exposing needed information among layers.

5.2.2. Benefits of the InterLay model

With its advantages mentioned above, the InterLay model brings about main benefits that can be summarized as follows:

Firstly, the InterLay model provides a general, comprehensive, secure approach to cross-layer manipulation of networking subsystems, which better serves optimization, customization of communication infrastructure and services.

More specifically, the general aspect includes:

- The information is accessed in a uniform way

- The information is accessible by all types of entities, including
  - ✧ End-user applications (in the user space)
  - ✧ Lower layers protocol objects (in the kernel space)
  - ✧ External policy servers (outside the local system)

The comprehensive aspect includes:

- The model allows the manipulation of all kind of information: *real-time parameter, event parameter,* and *action()* method
- The model allows both read and write operations
- The model allows manipulation of information from both local and external entities

And the secure aspect includes:

- Protection using priority
- Security of *set()* and *action()* method (when implementing in OO programming)
- Protection given by *get()* and *set()*(when implementing in OO programming)
- For write (update) operations as well as *action()* method, there are several check points: in the PE, in the Enforcer and in the *set()* method

Secondly, the InterLay model supports faster service development because there is:

- no need for update to the system every time a new requirement appears
- no need for the standardization of new protocols but making use of programming skill

Thirdly, by providing fine-tuning capability to other communication applications, the new model allows the “intelligent use” of underlying network status and functions by the end-user applications.

Fourthly, due to its broad coverage, the Interlay is able to replace previous works that also use cross-layer communication of network parameter approach but on a case by case basis. For example:

- To maintain a TCP session when a handover to a new IP address takes place, existing solutions require the modification of TCP protocol, while InterLay can support TCP mobility through manipulation of the PCB (Protocol control blocks) and TCB (Transmission Control Block), as well as events from lower layers. From section 5.1.2, we can see that the InterLay model can support ***more flexibility than any other*** existing works, enabling not only TCP mobility over handoff at one end, but also support transfer of TCP session across local interfaces as well as in the case both ends handoff simultaneously. The latter two cases are not possible in any existing works.
- InterLay model allows for prepending/appending extra information to the PDU, therefore it can handle SHIM-layer type of modification and replace Host Identity Protocol (HIP) at end devices (the InterLay provides the feature similar to Virtual IP [13] which can replace the functionality of HIP. It can also replace IP in IP tunneling protocol
- By executing *action()* methods from external entities, the InterLay model can replace the functionality of the IEEE 802.21 Media Independent Handover (MIH) protocol.
    - ✧ MIH's functions and commands to gather network characteristics and seamless handovers can be implemented as *action()* method of L2 object.
    - ✧ Station initiated and network initiated handovers will be implemented by calling the appropriated action() method.

And finally, even though focusing on TCP/IP protocol suite, the principles and methodologies of InterLay model are general enough to be extended to other networking models that use layering approach.

5.3. Comparison of InterLay and related systems

5.3.1. InterLay and Simple Network Management Protocol (SNMP) system

5.3.1.1. Overview of SNMP

SNMP is defined by IETF to be used as a network management protocol for the TCP/IP network. It defines the mechanism to manage devices on IP networks, including servers, workstations, routers, switches, modem racks and other networking devices. SNMP management architecture not only manages the performance and the configuration of the hardware but also the software running over it, such as the OS itself, web server software, etc…

The information model of SNMP-based network management includes the Simple Network Management Protocol (SNMP) itself, the Structure of Management Information (SMI) and the Management Information Base (MIB).

The SNMP Protocol defines the format of messages exchanged by the management systems and agents. It also includes the commands (i.e. operations) to gather and enforce data between the manager and the managed using the *GET* (and other *GET* -related commands), *SET* and *TRAP* commands.

The SMI provides the specifications on the format used to define managed objects on the network accessed by the SNMP protocol. In essence, it is a subset of Abstract Syntax Notation One (ASN.1) language adapted for use with MIB.

The MIB is a conceptual repository of management information recorded in a text file. It contains the management information (so-called the managed objects) provided by the managed device.

The architecture of network management based on SNMP includes the following three key components:

- *Managed device* is a device that needs to be monitored and managed. This includes both or either the hardware and any applications running on it.
- *Agent* is a software application running on managed devices. It gathers and/or modifies the information described in the MIB text file and responds to SNMP

commands from the NMS (described below) as well as sends notifications to the NMS.

- *Manager* is a device that runs the network management system (NMS) software which generates commands and receives information and notifications from the agents on managed devices for management purposes.

The SNMP protocol contains the following commands (i.e. operations) that are used between the NMS and agents to exchange information:

- The commands to request the managed objects of the MIB sent to the agents by the NMS, including the *GET, GETNEXT* and *GETBULK* command.
- The command to change/update the value of a managed object of the MIB sent to the agents by the NMS, which is called the *SET* command.
- The unsolicited notification sent by the agent to the NMS reporting of an (typically unexpected) event in the managed device. The related command is called the *TRAP* command.

The *GET* (and related commands) and the *SET* commands operate in the "client pull" fashion, in which the NMS actively "pulls" data from the agent, while the *TRAP* command operates in a "server push" fashion, in which the agent "pushes" out a *TRAP* message to the NMS.

In the above commands, the concerned managed object is identified by its Object Identifier (OID). The OIDs are organized in a tree-like hierarchy, and the OID of a specific object is represented by a series of integers separated by a dot, with each integer represents the node that must be traversed in order to reach the object, starting from the root node.

SNMP network management architecture is designed to manage configuration of devices (and its software), monitor performance as well as react to events from managed devices.

5.3.1.2. Comparison between SNMP and InterLay

From the discussion in section 5.3.1.1., we can see that the agent in the SNMP architecture runs at the local device that can be used to get various data from the device,

to change or update configuration parameters as well as to notify of events in the local system. This is similar to what the InterLay is designed for.

However, the purpose of SNMP is for network management, so its target is the information on the performance of OS, application and hardware, while the target of InterLay is the networking subsystem specifically. Because of the difference in targets, the SNMP and InterLay architecture have the following differences:

- The SNMP focuses on retrieving statistic data focusing on performance while the InterLay focuses on getting real-time value of parameters.
- The data that are changed/updated by SNMP are typically static configuration parameters, while InterLay can change/update both static and dynamic parameters of the networking subsystem.
- Because SNMP uses IP address to address the source and destination of SNMP commands, the SNMP agent cannot serves local applications.
- Moreover, events are notified directly from the InterLay in the kernel to the external device, which are more efficient in terms of processing load than that of SNMP, because in the SNMP the event must go through an intermediate SNMP agent running in the user space.
- Because of the way SNMP identifies managed information (i.e. the OID), the information must be specified in advance between the NMS and the SNMP agent. Therefore, SNMP is not suitable for manipulating the networking subsystem where certain objects (such as TCP objects) are created and destroyed dynamically.
- SNMP is not designed with *action()* methods in mind like InterLay.

Because of the above differences, SNMP cannot fulfill all the tasks that InterLay must do. It is also obvious that because the differences in purposes, *InterLay and SNMP do not exclude each other, instead they complement each other,* in which InterLay provides more flexibility in the manipulation of communication sessions, while SNMP better suits with network management purposes.

5.3.2. InterLay and Media Independent Handover (MIH) system

5.3.2.1. Media Independent Handover

Media Independent Handover architecture is being developed by the IEEE 802.21 Working Group to enable the smooth interaction and media independent handover between 802 technologies and other access technologies.

Central to the architecture is a Media Independent Handover Function (MIHF), which is an entity that interacts crossed-layer with multiple layers (IP, transport and application) to provide support for mobility when IP sessions are being handed over from one access technology (i.e. a network interface) to another.

The MIFH facilitates the handover process by relaying messages among the Internet access technologies and upper layers (namely IP and above). The messages can be categorized into three types of services as follows:

- *Event Service* which passes notification of events from both local interface as well as remote to higher layers. Some examples of event are Link_Up, Link_Down, etc.
- *Command Service* which provides a set of commands for the MIH users to control the handover process. The MIH commands are similar to *action()* methods of the InterLay model. Some examples of command are MIH_Configure_Link, MIH_Connect_3G, MIH_Link_Switch, etc. Similar to the InterLay model, to receive an event the requester must subscribe to that event in advance.
- *Information Service* provides the mobile device with the ability to query supporting information to facilitate it in selecting the correct target interface. Some examples of supporting information include list of available networks, link layer information, neighbor information, neighbor maps, etc.

The MIHF only provides the above services to different layers of the network subsystem. A mobility management agent is needed at the mobile device, which uses the services of the MIHF to calculate and decide which access network the mobile node should handover into. IEEE 802.21 does not specify the requirements or operations of the mobility management agent.

5.3.2.2. Comparison between MIH and InterLay

MIHF and InterLay object are similar in that they both allow higher layers to learn of the events of lower layers and to issue commands to lower layers in reacting to such events.

However, MIH and InterLay have some differences, of which the most prominent difference is the scope. MIH concentrates on handover, so its main target is Layer 2 interface, and it is used only to support handover and related activities. For user application, the only benefit that MIH provides is that the requirements for the access link (such as speed, QoS etc.) from the application could be taken into account when a new access technology is being selected. Meanwhile, InterLay is more comprehensive, in the sense that it allows the application not only to have a role in the handover process, but also to control and manage the behaviors of the IP and transport layer.

Another weakness of MIH is that it does not support the update/modification of real-time parameter due to the lack of means to identify a specific parameter of a specific protocol instance, as well as the necessary authorization mechanism to perform a “write” operation on the parameter. However, we can see from section 5.1 that this write operation is the main enabler for service customization and flexibility.

Therefore the MIH does not really provide any support in terms of customization and flexibility for the communication applications/services, as possible with the InterLay model.

Using the services of the Policy Engine, the Informer and the Enforce (as described in section 3.7) the InterLay can also provide the services of the MIH for Layer 2 handover. Therefore the InterLay can be said to supersede the MIH.

5.3.3. InterLay model and Control Information Exchange between Arbitrary Layers (CEAL) system

5.3.3.1. Overview of CEAL

Control Information Exchange between Arbitrary Layers (CEAL) is designed to provide similar cross-layer functionalities to that of Media Independent Handover, but not confining to layer 2 but extending to other layers as well [107] [112] [113].

As the name implies, CEAL is designed to exchange information, not to enforce changes to the networking subsystem.

CEAL introduces the Abstract Entity (AE) as the interface to communicate and cooperate with the protocol entity (PE). The PE is a representation of a specific protocol as a whole. The AE abstracts the protocol-specific information of the PE to some common and protocol independent information. Each PE will require a specific AE for itself.

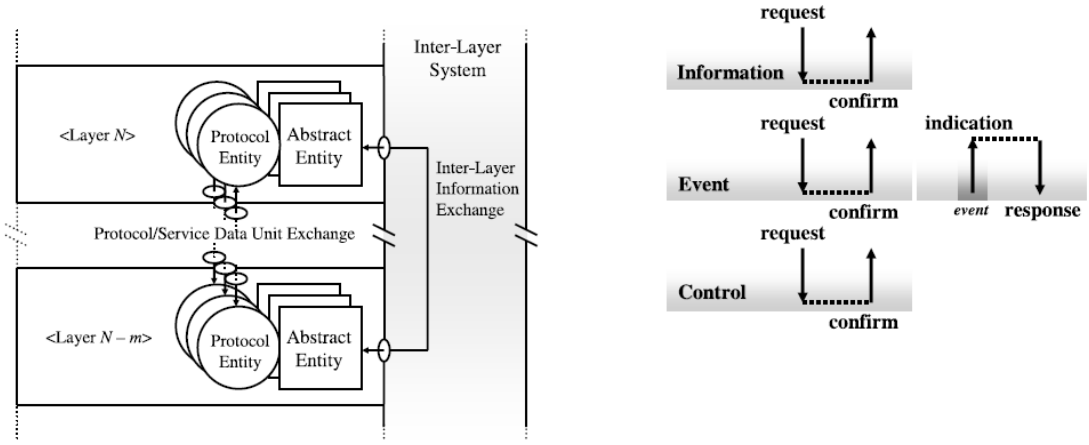

Figure 28. CEAL architecture and Primitive Interaction model [107]

AEs provide services to each other using four primitives: *Request* (primitive is used to request a certain service from another layer), *Confirm* (acknowledges the *Request* primitive), *Indication* (notification of the information that is requested by the *Request* primitive), and *Response* (acknowledges the *Indication* primitive).

The four primitives will be used to exchange 3 types of interaction: exchange of information of a protocol (similar to the *get()* method for real-time parameter of the InterLay model), notification of events (similar to the event parameter of the InterLay model) and exertion of control action (similar to the *action()* method of the InterLay model).

CEAL contains the Inter-Layer System (ILS) as a dispatcher of message between AEs to enable the AEs to exchange cross-layer information. AEs use a data structure called “Abstract Entity Parameter Packet” (AEPP) to exchange information with each other

through the ILS. The AEPP includes the source layer ID and protocol ID of the source and destination AEs, the type of primitive and interaction and other supporting data.

By using the four primitives and the three interaction types, the AE of a certain protocol can learn about the status of another protocol through its respective AE, and response correspondently.

5.3.3.2. Comparison between CEAL and InterLay

In this part we will compare CEAL and InterLay in 6 different areas.

1. Centralized vs. Distributed management of cross-layer activities

In CEAL the cross-layer manipulation tasks are carried out by the AEs, while in the InterLay model they are fulfilled by the InterLay object. So in essence CEAL is a collection of distributed cross-layer manipulation objects, one for each protocol, while in InterLay model the manipulation tasks are centralized in the InterLay object.

The centralized approach of InterLay model provides the advantage of removing the need to duplicate common functionalities such as authentication/authorization of requests at each AE, which reduces the size of the cross-layer system. Moreover, when these common functionalities must be upgraded or modified, then in InterLay model this action must be performed once, while in CEAL it must be repeated for each AE.

And as experienced in the development of the Intelligent Network [56] of the Telecom network or in the Next Generation Network, it is beneficial for a service/application provisioning system to employ common service independent building blocks (SIBs) [110], and a service can be built up by assembling these SIBs together in a specific order (of course there are cases when new SIBs are developed if required by a new service). The centralized management of the InterLay makes it possible to create and maintain such SIBs for the PE, the Informer and Enforcer. On the other hand the AEs are distributed and independent from each other so CEAL does not suit well with the SIBs approach.

One drawback of centralized approach is that it makes the size of the switch statements (illustrated in table II and III) of the Informer and Enforcer, which are used to find the right protocol instances and *set()/get()/action* methods, to become larger because

they have to deal with all parameters and *action()* methods. We can remedy this problem by assigning the parameters and *action()* method of each protocol with a consecutive range of ID, therefore we can easily identify the owner protocol of a parameter or *action()* method from its ID, and the switch-case now has to deal with finding the right parameters/*action()* method only, which is similar to what AEs has to do in CEAL (that uses distributed approach).

2. Handling newly introduced feature

In InterLay model parameters are identified individually for each protocol, while in CEAL each parameter is abstracted by a common representative for all the protocols of the same layer. While this abstraction in CEAL is expected to allow existing protocol entities to be able to interact with new ones with ideally no modifications, but in order to actually make use of a newly introduced protocol, its new parameters need to be abstracted again which ultimately requires the update of the system.

More concretely, when changes are introduced (either by a new protocol or new capabilities such as new events or parameters or actions to existing protocols), for CEAL it needs to update the AEs to handle the new aspects for the protocol entity, and for InterLay the ID for the new parameter/event/action should be identified, and the new ID should be mapped to the right parameter of the right protocol. So the distributed architecture in terms of cross-layer manipulation tasks of CEAL does not provide any advantages than the centralized architecture of InterLay.

3. Requirements of knowledge from developers

In the case of CEAL, the service/system developers need to know about the AE of the concerned protocol, as well as what kind of control is available for that protocol, which is not readily available.

In the case of InterLay model, the developers only need to know the ID of the target (i.e. the real-time or event parameter, or *action()* method) and the rest will be taken care of by the InterLay object. The information on the list of the target is readily available by using the test questions as described in Chapter 4 above.

4. Working at the instance level vs. the protocol-as-a-whole level

In CEAL, the AE is used as the destination to access the concerned protocol as a whole. Therefore a requester can access to the protocol as a whole, not to individual instances of the protocol. As a result, in CEAL any change will be applied to all instances of a protocol, for example a notification of a congestion event might result in the setting of `cwnd` of all TCP sessions to 1, while in reality only the TCP sessions that use the congested link should be affected.

On the contrary, in InterLay model the InterLay object allows for the protocol instances to act as the originator and executor of the request, therefore the InterLay model can provide more customization to individual services or network protocol. For example as explained in subsection 5.1.2, an application can decide whether a single TCP session should be maintained over address change or not without affecting other ongoing TCP sessions.

Moreover, working at the instance level also allows the InterLay model to execute an internal action of a protocol for a specific instance only. This is beneficial such as in the case of route optimization described in subsection 5.1.4 where the route optimization is required for a single IP session only. This is not possible with CEAL, because the route optimization action will be applied to all IP sessions to a same care of address.

Therefore the InterLay model provides finer-grained service customizations than that of CEAL. An example of the finer-grained of InterLay is the optional Route Optimization described in subsection 5.1.4. Because InterLay supports individual instance of IP session, it can perform RO for an individual Correspondent Host, while calling the Route Optimization method in CEAL will cause all mobile IP sessions to be terminated at the same destination Correspondent Host.

5. The ability to perform a "write" operation on the real-time parameter

As explained in section 1.2 of chapter 1 the ability to perform a *"write"* operation is the main enabler for (obj-b) of a cross-layer system.

Because in CEAL, the actual parameter as well as its owner protocol instance are abstracted by the AE, there is no way to access to the parameter to change its value. Therefore "write" operation is not supported in CEAL.

However, as discussed in section 5.1, “write” operation is the key factor for the arbitrary manipulation of protocol instances, which equals to inventing a new protocol via programming skills. This is an exclusive strength of InterLay model.

6. Working with external systems

Using the Message Handler in the Policy Engine, the InterLay model can easily interact directly with external systems.

CEAL cannot interact directly with external systems, and a separate user application will be needed to relay the commands with external system, which will impose more overhead than that of Message Handler in the InterLay model.

5.4. Overhead issues in OO Programming

In terms of operation speed, comparing with conventional procedural programming, the TCP/IP stack experiences the following extra overhead in OO programming:

(i) The extra overhead to create and destroy the objects belonging to Layer 2, 3, and 4.

(ii) The extra overhead to look-up the implementation of the virtual functions (in the so called v-table) that carry out the sending/receiving data in each respective object.

(iii) The extra overhead to look-up the implementation of any other virtual function that performs any other functions for that protocol.

Because objects of Layer 2 and Layer 3 are created when TCP/IP Networking Subsystem (NS) in the kernel is initiated and destroyed when the networking subsystem is shutting down, the overhead of (i) for objects of Layer 2 and Layer 3 does not affect the performance of the NS. Layer 4 object is created or destroyed whenever a transport session is established or torn down, but because at user terminals new session is created sporadically, the effect on the performance is negligible.

Because the major activities of the TCP/IP stacks are to move the data (PDU) up and down the protocol stack, the overhead in (iii) should be negligible, and the main extra overhead will come with (ii).

However, the look-up of the v-table takes only several CPU cycles [50] [51], so the impact of this extra load on modern CPU should not be noticeable. Moreover,

optimization techniques such as loading the v-table into the CPU cache (which is common for Just-In-Time compilers) will reduce the look-up time to one or two cycle, further reducing the impact of this overhead.

To roughly calculate the overhead, suppose that when invoked by a *send()* system call, the network stack receives the amount of data that equals to *m* segments. We can calculate the processing time for the TCP/IP stack as follows

- For procedural programming, it is:

$$m \times t_{tcpPrc} + m \times t_{ipPrc} \qquad (1)$$

- For OO programming, it is:

$$(t_{tcpLkp} + m \times t_{tcpPrc}) + (m \times t_{ipLkp} + m \times t_{ipPrc}) \qquad (2)$$

[Note: t_{Lkp} is the average time to look-up and access the object, while t_{Prc} is the average processing time for a segment or packet]

The calculation for processing time in (2) is based on the assumption that the TCP object invokes the IP object for each segment that is ready. If there is a mechanism for the TCP object to call the IP object to process a batch of *n* segments, then (2) would become:

$$(t_{tcpLkp} + m \times t_{tcpPrc}) + ((m/n) \times t_{ipLkp} + m \times t_{ipPrc}) \qquad (3)$$

[Note: it is obviously that $n \leq m$ and for simplicity n is chosen to be a factor of m]

The mechanism to enable (3) can be for example the TCP object to hold call to IP object until *n* segments are ready in the buffer. Therefore *n* is bounded by for example a upper delay time for processing the segment in the IP layer, and this delay is controlled by the application as a network parameter (for example non real-time application can select larger *n* than real-time application).

By comparing with (1), the extra processing time (i.e. overhead) for a packet in the case of (2) is 1/m $t_{tcpLkp} + t_{ipLkp}$, and in case of (3) the extra processing time is $[t_{tcpLkp}/m + t_{ipLkp}/n]$. As such the extra processing (overhead) per packet would be reduced proportionally to *m* and *n*, so for applications with high communication traffic (which require more processing power) the overhead is neglected, therefore for end devices with

high computing capacity, the affect of OO programming on the performance of the TCP/IP networking stack is negligible.

Moreover, in the case multiprocessor is used at the end devices, then each protocol object can be programmed as a single thread and executed by a separated processor. Assume that TCP object and IP object execute simultaneously, then (2) becomes

$$(t_{tcpLkp} + t_{tcpPrc}) + (m\text{-}1) \times \max\{t_{tcpPrc};(t_{ipLkp} + t_{ipPrc})\} + (t_{ipLkp} + t_{ipPrc}) \quad (4)$$

The performance measurement in [32] shows that t_{tcpPrc} is more than 3 times that of t_{ipPrc}, so $t_{ipLkp} + t_{ipPrc}$ is probably less than t_{tcpPrc}, then comparing (1) and (4) is equal to comparing $[(m\text{-}1) \times t_{ipPrc}]$ with $[t_{tcpLkp} + t_{ipLkp}]$, and in case *m* is large (i.e. high traffic applications) (4) can even be smaller than (1).

The same calculation can be done reversely for the receiving activities.

As for performance of the InterLay object, because this object will be called sporadically, and normally via system call, which is already processing intensive, the extra overhead incurred by OO programming is negligible.

OO programming will also cause overhead in creating and destroying TCP objects when a TCP session is established and terminated, but this is one time expense for the whole duration of the TCP session and not contributes to the processing of actual data.

However, it should be noted that the InterLay architecture can be implemented in both OO and procedural programming, and in the case performance is strictly needed then InterLay architecture can always be implemented in procedure programming, with extra care for actions that alter network parameters because the protection intrinsic with the *set()* method of OO is not available anymore.

5.5. Deployment strategies

This section discusses how the InterLay scheme can be rolled out in real world deployment. We propose that the deployment can be divided into two modes: disruptive and non-disruptive deployment.

When disruptive mode deployment is used, essentially the usage of existing service/application will be suspended until all the parties concerned finished the upgrade

to the new architecture, in both the OS and end-user applications. Disruptive deployment is further divided into two sub-categories: Complete disruptive and Partial disruptive.

In complete disruptive mode, all parties have to upgrade either or both the application and OS in order to continue using existing service. This also includes those who will not communicate with new features.

In partial disruptive mode, all partners that communicate with new features need to upgrade regardless of the features being used together with the OS, but they can still communicate with un-upgrade parties using the un-upgrade features.

In non-disruptive mode, only the parties that want to use new features provided by the InterLay have to upgrade and they can use the new features even with un-upgraded parties.

Let us consider one example of non-disruptive service. Suppose that a voice application can originally negotiate to change codec on the fly, and the application at end device A is upgraded to utilize cross-layer communication feature. When the application at A receives information from the lower layer regarding the possible change of link speed (for example Layer 2 can inform of the change from 802.11b to 802.11g, or Layer 4 inform of ECN or RTO event) then the application can actively negotiate to use a higher/lower codec to provide a smoother experience. Therefore, party A can use the new feature of InterLay without the requirements of the

For the example of partial disruptive service, suppose that a TCP application will work better if the TCP connection can be preserved across AP handoff. The TCP socket at both end device A and B can be upgrade to communicate with lower layers to preserve the TCP connection. But because that socket is still a TCP socket, the application can work with device with the current version without any trouble.

The new InterLay networking model proposed in this research can be deployed in several steps, depending on the nature of the applications that will be used. This will make the acceptance of cross-layer communication model faster and smoother. Also, the deployment of InterLay model can start with mobile devices, where the requirement for flexible services is more pressing, while the deployment for desktop can be rolled out later.

Moreover, the deployment of OS and end-user applications can be separated. Regarding the OS deployment, the new networking model can be rolled out together with OS upgrade. Until every host supports the new networking model, only non-disruptive and partial disruptive deployment of services are possible.

Regarding the application deployment, at first non-disruptive deployment can be carried out right away, and then partial disruptive and finally new version of the service can be rolled out in full-feature deployment (the two latter cases can be carried out only after the InterLay-enabled OS is available at the end device).

Regarding the IPC mechanisms between the InterLay object and other protocol objects, at first we can use existing and proven mechanism such as mmap, so that faster development can be achieved (as developers are used to these mechanism). However, new APIs (especially for the communication between InterLay ⇔ user-space socket) can be introduced later for each specific communication to improve efficiency. These new APIs can be developed with reference to existing implementation of other APIs, thus reducing the developing time.

Moreover, this new and more complex architecture could be deployed in end devices to support service development, while routers continue to use the traditional version of the TCP/IP architecture. The reason for this distinction is because the new model added extra capabilities and complexity as well as processing overhead that are indented for user applications that do not exist in routers.

5.6. Other performance and security issues

User applications can abuse the architecture by constantly issuing requests to InterLay to retrieve/alter network parameters, and this may affect the performance of the whole system.

There are two cases such a problem can happen. The first case is that the application is a malicious program intended to negatively affect the user's device, and the second is that the application is poorly written. However, these two cases are common issues of the IT world, not specific to the Interlay architecture itself. The solution to the first case is to raise the awareness of not using untrusted software, and to eliminate the second case, the

user application should be developed by adhering established quality control practice in the software industry.

And the PE and the Enforcer entities of the InterLay object are equipped with rate control to minimize the effects of bad applications from abusing the InterLay functionalities.

CHAPTER 6. CONCLUSIONS AND FUTURE WORKS

6.1. Major Contributions

The Internet has the characteristic of “dumb network, intelligent end devices”. However, the current layering model does not allow the “intelligent use” of underlying network status and functions by the end-user applications. Meanwhile, as explained in Chapter 1, communication applications now have the need and the ability to process information from lower layers for more customization and optimization. This research aims at addressing this problem.

From the initial idea, through the process of design and implementation, *a new TCP/IP model – the Interlay – has been envisaged, formed and introduced.* In Chapter 3, 4 and 5, the design, working mechanism of the new model have been described and illustrated. Its wide coverage, multifaceted advantages and noteworthy benefits have also been discussed and analyzed. The deployment strategy of new model into real world has also been outlined with different modes and steps.

The new TCP/IP model with InterLay entity as one important part can overcome limitations of conventional TCP/IP model and that of existing related works on cross-layer communication manipulation. It allows the provision of information from lower layers so that applications not only be able to adapt their performance according to lower layers status, but they can also (i) make more choices beside performance (such as arbitrary decision for route optimization) and (ii) adapt better with future changes. It can also provides *new service scenarios that are not possible with any other proposals*, such as supporting TCP mobility in the case of handoff simultaneously at both ends or between local interfaces.

The results of the research are discussed in more details as follows:

Firstly, the research explores and provides the case for the need of a new architecture of TCP/IP which allows protocol's internal activities and states to be access across layers, especially by the user application. We have provided several examples to signify the usefulness and advantages of the new architecture in chapter 5.

The research examines numerous existing works on cross-layer information manipulation, which seek benefits of cross-layer design from the performance aspect by asking different layers to adapt themselves according to the current status of the network. These works have brought about benefits such as optimizing the utility function of end to end throughput in ad-hoc network, optimizing the exchange of information and conserving energy in sensor networks. For example, Coordination between Layer 2 and Layer 3 during a handoff has reduced the interruption time for Mobile IP and SIP applications.

The review of existing works indicates the major limitation of these works: they are based on a case by case approach. Every time a new requirement appears, the system has to be modified, which makes it slow to develop and introduce new features.

Secondly, the research provides a *new broader approach* that better supports the exchange and manipulation of cross-layer information *without* requiring individual modification of the TCP/IP stack for each new development. The resulting InterLay model has been proposed and is the main topic of this research, which includes:

- the core conventional TCP/IP stack
- a separate InterLay entity that handles all activities related to cross-layer communications.

Thirdly, as the InterLay model allows the ability for the application to invoke internal operation of the networking protocol stack, as well as the ability to query and update the internal state of the networking subsystem, the *new architecture provides the application with the ability close to implementing new protocols*!

The InterLay model is the only solution for cross-layer manipulation that supports the "write" operation of protocols' parameters. As illustrated in section 5.1, by using the "write" operation, the InterLay model allows for the implementation of new features by

just using the programming skill instead of requiring a new protocol to be developed first. And as recommended in Request For Comment 1958 [103], “Nothing gets standardised until there are multiple instances of running code”, the InterLay can be used in this sense to implement and monitor various aspect of a new feature, and the information obtained from this process can be used to speed up the development of the correspondent protocol. So the InterLay model can be used as a testbed to develop protocols for the TCP/IP architecture!

Moreover, apart from using the priority mechanism to protect the TCP/IP protocol stack from disastrous modification of internal state, the new model provides the ability to enforce *protection mechanisms* for any alternation in the following 3 places: (i) the PE (upon receipt of the request), (ii) the Enforcer (before the *set()* is called) and (iii) inside the *set()* method itself. The implementers are free to put any perceivable security and integrity mechanism in these checkpoints. This provides better safeguard against mistakes that are caused incidentally or by negligence. Also, by making use of the Message Handler, the InterLay object can receive the policies from external entities to be performed at these checkpoints.

And by the introduction of the InterLay object, a call made to the *set()/get()* method of a protocol object does not have to interact directly with the protocol stack, which *reduces the possibility of bugs.*

Fourthly, the model also allows for external servers to monitor and control the behaviors as well as learning about critical events of the local networking subsystem. This allows for *greater service control and optimization*. In addition, as explained in section 3.7.2.1, the InterLay object allows the possibility of executing scripts provided by external server. This is done by the external server to send information to the networking subsystem, together with executing the appropriate *action()* method and updating the appropriate real-time parameters or executing selected action whenever an event takes place.

Fifthly, to select the right parameters to be exposed for cross-layer manipulation, *five test questions are defined and applied* to various protocols from layer 2 to layer 5 and suitable parameters are identified in Chapter 4, together with the possible usage of these

parameters. This methodology can effectively *help save resources* (manpower, time and other costs) in developing the InterLay scheme. It also helps application developers to get ideas on what are available and how they can be used to support communication applications.

And lastly, the OO design turns the proposed new architecture into an *implementation platform for other ideas on cross-layer design*, as explained in Chapter 5. Even though the research focuses on TCP/IP protocol suite, its principle and methodology are general enough to be extended to other networking model that uses layering approach as well. In this sense, the InterLay model is also useful as a *reference model* for the design of new network architectures in the future. Moreover, it is also suitable as a guideline for embedded system where customization and optimization are of importance.

6.2. Future works

Future work includes more detailed specifications to realize the OO design of the new architecture using OO conversion tools and other existing OO framework for protocol design and development, as well as creation of more applications based on the InterLay scheme.

Moreover, the test questions and parameter list in Chapter 4 should be constantly monitored, revised, updated and appended as infrastructure and service demands evolve, they need to be constantly examined to find out new test questions and parameters to serve in different types of system or service scenarios. The approach of the test questions proposed in this research is universal enough to support future expansion with ease.

As explained in subsection 3.7.2, the mechanisms for the networking subsystem to contact and receive information from external servers or devices is a very important issue, as it enables the networking subsystem to execute scripts from external entities, which enables better service coordination. Therefore further studies on this matter as well as service coordination scenarios are highly recommended.

Finally, there is a need on finding a more generic Inter Process Communication between Kernel and User space. One possibility to do this is to provide another kind of kernel signal that can carry more data.

APPENDIX I. USER SPACE – KERNEL SPACE COMMUNICATION SOLUTIONS FOR EVENT NOTIFICATION

In section 3.7.4.4, two inter-process communication solutions that can be used by the kernel to notify user applications about the occurrence of event are mentioned. They are kernel signal and NetLink socket. This appendix explains in details how these solutions work.

I. Solution 1: Using the signal

In this case, one real-time signal is reserved for the InterLay. When the event occurs, the corresponding *notify()* method will send the signal to the list of registering applications, with the event ID as the data. At the application side, one callback function that handles all events the application is interested in must be registered as the handler for the signal belonging to InterLay. When the signal is sent to the application, the callback function will get the event ID from the associated data, and react to this event.

The advantage of this solution is that it is straight forward and easier to implement at both the InterLay and the application. The drawback is that implementing a signal is a complicated job, and one dedicated signal is required. And if more data is needed (such as identifier of the socket), then another kernel space– user space IPC mechanism has to be used, such as mmap [47]

II. Solution 2: Notifying using the NetLink socket.

If NetLink is used for event notification in the InterLay model, first a new INTERLAY protocol family is declared in the AF_NETLINK address family.

The InterLay opens one NetLink socket (in the kernel) of INTERLAY protocol to communicate with the application of all events (i.e. one socket for all events' ID).

The user application creates one function (for example with the name *event_handler()*) which contains the code to parse the notification from the InterLay, and after identifying the specific event the function will handles it correspondingly.

The user application then creates a NetLink socket of the same INTERLAY protocol. It then registers the *event_handler()* as the callback function for the socket. If the application registers for several events, then either all of events are handled in one *event_handler()* function, or the application will have to create the same amount of NetLink socket, each is registered with a function that handles one of the event.

When the event occurs in the kernel, the corresponding *notify()*method is called. In this case the *notify()*method will send one message containing the event ID to the NetLink socket. Due to the characteristic of the NetLink socket, all applications that have an open NetLink socket of the same INTERLAY protocol will receive the message, and the *event_handler()* function will be called. The function will read the event ID, and if it is waiting for the event, it will process it accordingly, else the notification is ignored.

The advantage of this approach is that it is very simple. The problem with this approach is that the application has to process all events, regardless of whether it is interest in the events or not. In the case event is fired constantly, this will greatly reduce the performance of the user application.

One solution for this problem is for the InterLay to declare a protocol type for each of the event. In this case it will have to create one kernel socket for each event, using the corresponding protocol type. When the *notify()*method is invoked as the result of the occurrence of the event, the kernel socket will send a message. For the application, it must create one NetLink socket for each event it is interested to, and register a handler function to the socket. When the kernel sends the message, the handler function in the application will be invoked, and the event is processed. The advantage of this solution is that the application is notified only when the appropriate event happens. The drawback is that more sockets must be used at both the kernel and the application.

Note that because of the characteristic of the NetLink socket, the application does not have to register in advance with the InterLay object. This will save some extra overheads.

However, one problem of the Netlink approach is that applications will be notified arbitrarily about the event, without the authorization of the InterLay object. Nonetheless as the event notification is of the type of read-only operation, and local applications can be trusted for such read-only operations so normally authorization is just used for reducing the load, and the load of NetLink does not depend on the number of registered processes.

Another drawback is that if the application needs to know about the socket associated with the event, then the overhead is rather high if many applications are interested in the event, because the kernel socket will have to send all socket identifiers to every applications (and in this case the application has to make a registration in advance).

APPENDIX II. COMPARISON OF PROPOSALS TO MAINTAIN TCP SESSION OVER IP ADDRESS CHANGE

In this part we will summarize some existing researches on maintaining TCP session across IP address changes, and comparing them with the solution provided by InterLay that is described in subsection 5.1.2.

I. Related works

In mobile networking, there is a distinction between identifier and locator. An application is reached at a specified location and distinguished by a specific identity. The location of the application may change but the data can always find the right application using the identifier.

The traditional TCP/IP model does not support mobility (namely, change of PoA (Point-of-Attachment)) due to the fact that the model uses address (i.e. location) as the identifier to distinguish between sessions. Therefore as a handover (change of PoA) changes the location, it also invalidates the identity (and as a consequence, it closes the session as well).

Adding mobile capability can be realized at several layers. In the **Session Layer** (of the OSI model) approach, such as the Session Layer Mobility (SLM) framework [80], or the Session Layer migrate approach [10]. These proposals insert a session layer management entity between the Applications and the Transport layer, and this entity will carry out the job of re-establish the data session to the MN's new attachment point. The drawbacks of this Session Layer approach are (i) because Session Layer is incorporated into the application layer in the TCP/IP model, it requires changes to existing applications as well as (ii) it is not backward compatible (*) with existing applications.

(*) In this research, backward compatibility means that the application/system that implements the proposal could work/interact with legacy application/system in case the host device does not change its

PoA. In other words, the application that employs the new solution can work with existing server, client or peer application at the other end.

There are also proposals to provide mobility at the **Network Layer**, the most famous example is MIP (Mobile IP) [28]. In MIP, locator is the current address of the Mobile Node at the foreign network (the Care-of Address), while the identifier is the MN's Home Address. The advantage of this approach is that no changes are required at the transport and above layer. The main drawbacks are the overhead due to triangle routing and tunneling, as well as the requirement of extra infrastructure (Home-Agent, Foreign-Agent).

Other researches propose to add a **Shim layer** between the Network and Transport layer for mobility (in fact, the Session Layer approach above can be said to be a special Shim layer between the Application and the Transport layer). This Shim layer will hide away the IP layer (including IP address) from the above layers. In Host Identity Protocol [29], a Host Identifier (HI) acts as the identifier and hides the change of the locator (i.e. IP address) from the Transport layer. A kind of DNS service will map between HI and the current IP address, and at the end host the HIP layer will correctly map between the HI and the TCP port. Virtual Internet Protocol [13] utilizes a similar approach, and the VIP layer will do the same job of the HIP layer, using the VIP address as the identifier to hide the changes of IP address from the Transport layer. Both approaches introduce a Shim Header between the IP and the TCP/UDP header.

The drawback of this Shim approach is that it causes overhead because of the Shim Header. Also, it requires changes to applications, because sockets are now bound to identifiers at the Shim layer, not the actual IP address. Furthermore, HIP requires IPSec (ESP) for all data exchange, as well as a kind of DNS service to map between HI and IP address. These activities induce certain processing overhead, which reduces battery life at the mobile device. Moreover, HIP hosts will not be able to talk with non-HIP host, which means that backward compatibility is a big issue. In the VIP case, the interaction between VIP host and non-VIP host requires a conversion gateway, but applicable only for a limited case of mobility.

The third approach is to solve mobility issue at the **Transport Layer**. Several proposals have been proposed, such as **(i)** creating new transport layer protocol, and an example is the SCTP protocol [78], **(ii)** split-connection approach [86, 87] as well as **(iii)** adding new states and options to existing TCP protocol to support change of attachment points [14, 15, 16, 79]. The common advantages of transport layer approach are that we can have dedicated congestion control scheme for handoff, no new infrastructure is required and no state (except packet routing function) is stored in the network like that of MIP.

More specifically, the advantage of [78] is that it does not only support mobility but also multihoming etc. The main problem with [78] is that it does not compatible with existing applications using TCP, therefore it could not be used as a solution for existing applications. Security is also a problem, because by knowing the *vtags* (authentication code), a third party can not only just inject data into a connection like what happens to sequence number of TCP, but also actually steal the whole SCTP session due to the mobile feature of SCTP.

The problems with **(ii)** are that it requires extra infrastructure, breaking the end-to-end syntax of TCP session, and single point of failure. But it allows two different congestion control scheme for fix and wireless portion of the link.

Approach **(iii)** has the problem that both sides have to perform additional works to exchange a share-secret in advance, regardless of whether the Mobile Node will actually performs the handover to a new Access Point (AP) or not, or whether the TCP session lives long enough to experience a handover. The proposal in [16] relieves this matter somewhat by initiating the preparation process only if the TCP connection exists longer than a threshold, but if the mobile device performs a handover before that then all unprepared TCP connections are lost. On the other hand, in all proposals of **(iii)** if the Mobile Node does not perform a handover, then all of the preparations for the long-live connections are wasted.

II. Comparison of InterLay and related works

II.1. Specific Pros and Cons of InterLay

The general pros and cons of solving IP mobility at different layers are specified in section I above. TCP mobility support by InterLay can be regarded as the solution at the Transport layer, and more specifically it belongs to approach **(iii)** therefore it has all the pros and cons of approach **(iii)** at the Transport Layers.

Some specific pros and cons of InterLay TCP mobility scheme compared to other solutions of approach **(iii)** are:

- Con-1: To fully take advantage of InterLay scheme the applications have to be rewritten.
- Pro-1: The application is able to perform any necessary extra processing that might be needed for the connection to the new IP address.
- Pro-2: If the applications at both ends establish some security association, then this association can be used to authenticate the request to update IP address, which can reduce the network and processing load.
- Pro-3: For specific applications like SIP, both end hosts can simultaneously handover to new IP address. In this case, the end applications will be written to use SIP server as Rendezvous Point.

II.2. Summary of different solutions of maintaining TCP session across IP address changes

In this section, different solutions of TCP mobility will be compared using the following criteria:

- **Backward Compatibility**

  Backward Compatibility with existing TCP-based applications is perhaps the most important issue for a solution to be really useful and deployable.

  Solution of InterLay is fully compatible with existing applications in the sense that it can perform like existing TCP socket (i.e. it can connect to existing TCP socket) when connects with traditional TCP socket.

- **Requirement of new infrastructure or changes to existing infrastructure entities**

The slow adoption of Mobile IP shows that if new infrastructure is requires for a solution to be in effect, then the adoption maybe slow.

The InterLay model requires no new infrastructure in order to provide mobility to TCP session.

- **Requirements of changes to applications**

  If a solution requires changes to existing applications then that will also slowdown the adoption process considerably.

  In order to fully take advantage of the InterLay model, it is required that applications be rewritten to use the InterLay. But this problem can be alleviated because the InterLay model is compatible with traditional TCP socket, therefore in the case new features are not needed right way the application can use existing features without any changes.

- **Support handover between multiple interfaces**

  Support handover between multiple interfaces is an important feature, since more and more mobile device will be equipped with both wireless WAN and wireless LAN interfaces.

  Because InterLay supports the changes of both IP address and Port, it can support the handover between interfaces of the same device.

- **Allow for extra processing**

  There are cases when it is preferable for the application to be able to perform certain pre-/post-update activities regarding changing of IP address. Such actions might be critical for premium/critical services.

  InterLay model allows such activities to take place in the case the application is rewritten to support InterLay explicitly.

- **Complexity (in both the working of the proposal and implementation)**

  The working of proposed solution should be simple in terms of limited exchange of control data and algorithm to process such data, and few or no protocol states in the

network. Also the amount of code changes to application (if any) should be minimized.

InterLay model does not requires any state at the network, and the changes of codes to application is minimized (only a few function calls to update the IP address and send the request to update the IP address).

- **Overhead (in both processing and network load)**

Overhead has close correlation with complexity. Here the overhead is the amount of data sent to the network to exchange control messages between end hosts. Also, normally Mobile Node runs on battery while on the move, therefore little processing at the network stack/applications to reduce power consumption is very important.

The characteristic of different protocols are summarized in the following table:

	Compatibility	New or changes to infrastructure	Changes to applications	Handover between multiple interfaces	Extra processing	Complexity	Overhead	Other Pros & Cons
Session Layer								
SLM [4]	No	No	Yes	N/A	Yes	Much	Fair	
SL Migrate [10]	No	No	Yes	N/A	Yes	Much	Fair	
Network Layer								
MIP [1]	Yes	Yes	No	No	No	Fair	Much	Support UDP at the same time
Shim layer between the Network and Transport layer for mobility								
HIP [2]	No	Yes	Yes	No	No	Much	Much	
VIP [3]	No	Yes	Yes	No	No	Much	Much	
Transport Layer								
SCTP [78]	No	No	Yes	Yes	No	Fair	Fair	Multi-homing etc…

	Compatibility	New or changes to infrastructure	Changes to applications	Handover between multiple interfaces	Extra processing	Complexity	Overhead	Other Pros & Cons
MSOCK [11]	Yes	Yes	No	N/A	No	Much	Fair	
I-TCP [12]	Yes	Yes	No	N/A	No	Much	Much	
TCP-R [7]	Yes	No	No	No	No	Fair	Fair	
End-to-End [8]	Yes	No	No	No	No	Fair	Fair	
Preserving [9]	Yes	No	No	No	No	Fair	Fair	
Mobility Support [10]	Yes	No	No	No	No	Fair	Fair	
InterLay Scheme	*Yes*	*No*	*Yes*	*Yes*	*Yes*	*Fair*	*Fair*	*Simultaneous handoff of MH and CH for certain services such as SIP is possible*

(In column with grey background, a negative value is better, while in column with white background a positive value is better.)

From the above table, we can see that InterLay scheme is almost similar to other solution at the Transport Layer in terms of the said criteria, but with the additional features of handover between multiple interfaces, allowing for pre- and post-update processing of IP address changes and simultaneous handoff of Mobile Node and Correspondent Node for certain services such as SIP.

REFERENCES

[1] D.D.Clark, “The Design Philosophy of the DARPA Internet Protocols”, Proc SIGCOMM 88, ACM CCR Vol 18, Number 4, pp. 106-114 1988(reprinted in ACM CCR Vol 25, Number 1, 1995, pp. 102-111).

[2] R. Bush et. al, “Some Internet Architectural Guidelines and Philosophy”, Request for Comments 3439, 2002

[3] D. D. Clark, “Architectural Considerations for a New Generation of Protocols”, Proc. *SIGCOMM'90, pp. 200-208, 1990*

[4] International Standards, “Open Systems Interconnection - Basic Reference Model Organization - Information Processing”, International Standard 7498-1

[5] Jon Postel, “Comments on Internet Protocol and TCP”, Internet Engineering Note number 2 (IEN 2),

[6] K. E. Malki et al, “Low-Latency Handoffs in Mobile Ipv4”, Request for Comments 4881, 2007

[7] J. Crowcroft et. al, “Is Layering Harmful”, IEEE Network Magazine, pp. 20 - 24, 1992

[8] Seunghun Oh, et al., “Seamless Fast Handoff in Mobile IPv4 Using Layer-2 Triggers”, Second International Conference on Systems and Networks Communications, 2007

[9] Stefan Böcking, “Object-oriented Network Protocols”, Proc. INFOCOM '97, pp. 1245 - 1252 vol.3, 1997,

[10] Snoeren, A.C. et al., “Reconsidering Internet mobility”, Proc. 8th Workshop on Hot Topics in Operating Systems, pp. 41 - 46, 2001.

[11] Landfeldt, B. et al., “SLM, a framework for session layer mobility management”, 8th International Conference on Computer Communications and Networks, pp. 452 – 456, 1999.

[12], R. Moskowitz et. al”Host Identity Protocol (HIP) Architecture”, Request for Comments 4423, 2006

[13] Teraoka,F. et al., “Host migration transparency in IP networks: the VIP approach”, ACM SIGCOMM Computer Communication Review, Volume 23 , Pp.: 45 - 65 , 1993

[14] FUNATO D et al., “TCP Redirection for Adaptive Mobility Support in Stateful Applications”, IEICE transactions on information and systems, pp. 831 – 837, 1999

[15] A. C. Snoeren et al., “An end-to-end approach to host mobility”, Proceedings 6th ACM International Conference on Mobile Computing and Networking, pp. 155 - 166, 2000

[16] Vassilis Prevelakis and Sotiris Ioannidis, “Preserving TCP Connections Across Host Address Changes”, Lecture Notes in Computer Science, Springer Berlin / Heidelberg, pp. 299-310, 2006

[17] D. Tennenhouse, "Layered multiplexing considered harmful", Proceedings of the IFIP Workshop on Protocols for High-Speed Networks, Rudin ed., North Holland Publishers, pp. 143–148 , 1989

[18] Vu Truong Thanh, Yoshiyori Urano, “Mobile TCP socket for secure applications ", The 12th International Conference on Advanced Communication Technology (ICACT), 2010

[19] Warrier, Ajit et al., “Cross-layer optimization made practical”, Fourth International Conference on Broadband Communications, Networks and Systems, pp. 733 -742, 2007

[20] Lijun Chen, Stevenh. Low, Mung Chiang, John C. Doyle, “Optimal cross-layer congestion control, routing and scheduling design in ad hoc Wireless networks”, In Proc. IEEE INFOCOM, pp. 1 - 13, 2006

[21] Weilian Su and Tat L. Lim, “Cross-layer design and optimisation for Wireless sensor networks”, International Journal of Sensor Networks , Volume 6, Number 1, pp. 3-12, 2009

[22] Christophe J. Merlin, “Adaptability in Wireless Sensor Networks Through Cross-Layer Protocols and Architectures”, PhD Dissertation, University of Rochester, New York, 2009

[23] M. V. Schaar and N. S. Shankar, ”Cross-layer Wireless multimedia transmission: Challenges, principles, and new paradigms” IEEE Wireless Communications, Vol.12, Issue 4, pp. 50-58, 2005.

[24] Vineet Srivastava and Mehul Motani, “Cross-Layer Design: A Survey and the Road Ahead”, IEEE Communications Magazine, Vol. 43 Issue 12, pp. 112-119, 2005

[25] E. Kohler et. al, ”Datagram Congestion Control Protocol”, Request for Comments 4340, 2006

[26] R. Stewart, Ed., “Stream Control Transmission Protocol”, Request for Comments 4960, 2007

[27] Christian Benvenuti, “Understanding Linux Network Internals”, ISBN: 978-0-596-00255-8, O'Reilly Media, 2005,

[28] C. Perkins et. al, “IP Mobility Support for IPv4”, Request for Comments 3344, 2002

[29] R. Moskowitz et. al, “Host Identity Protocol (HIP) Architecture”, Request for Comments 4423, 2006

[30] Sami Iren , Paul D. Amer , Phillip T. Conrad, The transport layer: tutorial and survey, ACM Computing Surveys (CSUR), v.31 n.4, p.360-404, Dec. 1999

[31] J. Rosenberg, H. Schulzrinne,”An Extension to the Session Initiation Protocol (SIP) for Symmetric Response Routing”, Request for Comments 3581, August 2003

[32] Xavier Leroy, “Compilation techniques for functional and object oriented languages”, PLDI tutorial, 1998

[33] Marcin Chady, How C++ Works, Radical Entertainment INC. available online at: http://pages.cpsc.ucalgary.ca/~bdstephe/585_W11/d403_C++.pdf

[34] Linux Manpage, “SIGNAL(7)”, Linux Programmer’s Manual

[35] Kernel Korner, “Why and How to Use NetLink Socket”, Linux Journal, retrieved online in October, 2010

[36] M. Degermark et. al, “IP Header Compression“, Request for Comments 2507, 1999

[37] V. Jacobson, “Compressing TCP/IP Headers for Low-Speed Serial Links”, Request for Comments 1144, February 1990

[38] Moshe Bar, “The Linux Signals Handling Model”, Linux Journal, retrieved online Oct, 2010

[39] Linux Manpage, “netlink(3)”, Linux Manual Page

[40] Yavatkar, R., et. al, "A Framework for Policy-based Admission Control", Request for Comments 2753, 2000.

[41] Deepti Chafekar et. al, “Cross-Layer Latency Minimization in Wireless Networks with SINR Constraints”, MOBIHOC’07, Canada, 2007

[42] Torsten Braun, Christophe Diot, “Protocol implementation using integrated layer processing”, ACM SIGCOMM Computer Communication Review, v.25 n.4, p.151-161, 1995

[43] Ernesto Exposito, et. Al, “Introducing a cross-layer interpreter for multimedia streams, Computer Networks: The International Journal of Computer and Telecommunications Networking”, Vol.52 No. 6, p.1125-1141, 2008

[44] Carl Wijting, Ramjee Prasad, “A Generic Framework for Cross-Layer Optimisation in Wireless Personal Area Networks”, Wireless Personal Communications: An International Journal, Vol. 29 No.1-2, p.135-149, 2004

[45] Robbert van Renesse, “Masking the overhead of protocol layering”, ACM SIGCOMM Computer Communication Review, Vol.26 No.4, p.96-104, 1996

[46] Mark B. Abbott, Larry L. Peterson, “Increasing network throughput by integrating protocol layers”, IEEE/ACM Transactions on Networking (TON), Vol.1 No.5, p.600-610, 1993

[47] Linux Manpage, “mmap(2)”, Linux Programmer’s Manual

[48] José Orlando Pereira et. al, “Open Implementation of Reliable Communication Protocols” , 1997. Retrieved on June, 2010, from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.50.7030

[49] W. Simpson, “IP in IP tunneling”, Request for Comments 1853, 1995

[50] Brad Calder , Dirk Grunwald , Benjamin Zorn “Quantifying behavioral differences between C and C++ programs” journal of programming languages, Vol. 2 No. 4, 1994

[51] R. Radhakrishnan ,et. al, “Performance Impact of Object Oriented Programming” Retrieved from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.50.7030

[52] Vu Truong Thanh, Yoshiyori Urano, “Link-based service customization for NGN”, The 10th International Conference on Advanced Communication Technology, Korea 2008

[53] Alessandro Rubini, Jonathan Corbet, “Linux Device Drivers“, 2nd edition, Oreilly & Associates Inc, 2001

[54] D. Durham, Ed. et. al, “The COPS (Common Open Policy Service) Protocol”, Request for Comments 2748, 2000

[55] K. Chan et. al, “COPS Usage for Policy Provisioning (COPS-PR), Request for Comments 3084, 2001

[56] Willner, R.; Lee, L.W.; “IN service creation elements: variations on the meaning of a SIB”, Intelligent Network Workshop, 1997, USA

[57] P. Calhoun et. al, “Diameter Base Protocol”, Request for Comments 3588, 2003

[58] Hakima Chaouch, et. al, "A trial towards unifying control protocols: COPS versus Radius/DIAMETER and Mobile IP” 4th International Workshop on Mobile and Wireless Communications Network, pp. 537- 541, 2002

[59] Hakima Chaouch, “A new policy-aware terminal for QoS, AAA and mobility management”, International Journal of Network Management”, Vol.14 Is. 2, 2004

[60] Hanish, A.A., "Operating systems and communication protocols” Fourth International Workshop on Object-Orientation in Operating Systems, pp.166- 170 , 1995

[61] Cena, G., et. al; , "Object oriented models and communication protocols in the factory “ 21st International Conference on Industrial Electronics, Control, and Instrumentation, 1995 vol.2, pp.1573-1579, 1995

[62] Hanish, A.A., "Operating systems and communication protocols” Object-Orientation in Operating Systems, 1995., Fourth International Workshop on , vol., no., pp.166-170, 1995

[63] Information Sciences Institute, University of Southern California, “TRANSMISSION CONTROL PROTOCOL”, Request for Comments 793, 1981

[64] Information Sciences Institute, University of Southern California, “INTERNET PROTOCOL”, Request for Comments 791, 1981

[65] S. Deering and R. Hinden, “Internet Protocol, Version 6 (IPv6) Specification”, Request for Comments 1883, 1995

[66] Rad, M.S.; Shafiee, M.; “A mobility-aware cross-layer congestion control, routing and scheduling design in ad-hoc wireless networks”, OPTIM 2008: 11th International Conference on Optimization of Electrical and Electronic Equipment, 2008.

[67] S. Kent, “Security Architecture for the Internet Protocol”, Request for Comments 4301, 2005

[68] Hanish, A.A.; Dillon, T.S., "Object-oriented modelling of communication protocols for re-use” Fourth International Conference on Computer Communications and Networks, pp.18-26, 1995

[69] Andrew A. Hanish and Tharam S. Dillon. “Communication protocol design to facilitate re-use based on the object-oriented paradigm. Journal of Mobile Networks and Applications, Vol. 2 Iss. 3, pp. 285-301. 1997

[70] Gregory R. Lavender et. al, “Technical Report Implementing Communication Protocols Using Object-Oriented Techniques”, Technical Report, Virginia Polytechnic Institute & State University Blacksburg, 1992

[71] T. Socolofsky and C. Kale, “TCP/IP tutorial”, Request for Comments 1180, 1995

[72] R. Braden, Editor, “Requirements for Internet Hosts -- Communication Layers”, Request for Comments 1122, October 1989

[73] Hidetoshi Yokota et. al, “Link layer assisted mobile IP fast handoff method over wireless LAN networks”, Proceeding of MOBICOM ‘02, pp. 131-139, 2002

[74] Mohamed Alnas et. al, “A Cross-Layer Decision for Mobile IP Handover”, Proceedings of 24th IEEE International Conference on Advanced Information Networking and Applications, 2010

[75] Tanja Lang and Daniel Floreani, “Performance evaluation of different TCP error detection and congestion control strategies over a wireless link”, ACM SIGMETRICS Performance Evaluation Review, Volume 28 Issue 3, pp. 30-38, 2000

[76] C. Caini and R. Firrincieli, “TCP Hybla: a TCP Enhancement for Heterogeneous Networks”, in International Journal of Satellite Communications and Networking, John Wiley & Sons, Vol. 22 , No. 5, pp 547 – 566, 2004

[77] Liu Feng et. al, “TCP-ATCA: Improved Transmission Control Algorithm in Satellite Network”, Chinese Journal of Aeronautics, Vol. 21, Iss. 2, pp. 155-161, 2008

[78] R. R. Stewart et al., “SCTP Extensions for Dynamic Reconfiguration of IP Addresses”, Internet draft, 2002

[79] W. Eddy, “Mobility Support For TCP”, Internet-draft, http://tools.ietf.org/html/draft-eddy-tcp-mobility-00

[80] Landfeldt, B. et al., “SLM, a framework for session layer mobility management”, 8th International Conference on Computer Communications and Networks, 1999

[81] D. Yon et. ai, "TCP-Based Media Transport in the Session Description Protocol (SDP)",Request for Comments: 4145, IETF, September 2005

[82] Huei-Wen Ferng et. ai, "A SIP-Based Mobility Management Architecture Supporting TCP with Handover Optimization", Proc. Of Vehicular Technology Conference, pp. 1224-1228, Apr. 2007

[83] Elin Wedlund ;Henning Schulzrinne, “Mobility Support using SIP”, Second ACM/IEEE International Conference on Wireless and Mobile Multimedia (WoWMoM’99), Aug. 1999.

[84] H. Schulzrinne and E. Wedlund, “Application layer mobility using SIP,” ACM SIGMOBILE Mobile Computing and Commun. Rev., vol. 4, no. 3, pp. 47–57, 2000.

[85] Bill N. Schilit et al, "Context-Aware Communication," IEEE Wireless Com. Mag., pp. 46-54, Oct. 2002.

[86] David A. Maltz et al., “MSOCKS: An Architecture for Transport Layer Mobility”, pp. 1037 – 1045, Vol.3, Proc. of INFOCOM , 1998

[87] Bakre, A. et al., “I-TCP: indirect TCP for mobile hosts”, 5th International Conference on Distributed Computing Systems, 1995

[88] Peer Azmat Shah et al., “End-to-End Mobility Management Solutions for TCP: An Analysis”, 11th International Conference on Computer and Information Technology, 2008

[89] Hala Elaarag, “Improving TCP performance over mobile networks”, ACM Computing Surveys, Vol. 34 , Iss. 3, pp.: 357–374, September 2002

[90] Mika Ratola, “Which Layer for Mobility ? - Comparing Mobile IPv6, HIP and SCTP”, Helsinki university of Technology, 2004

[91] Eddy, W.M., “At what layer does mobility belong?”, Communications Magazine, IEEE, pp. 155- 159, 2004

[92] C. Ahlund and A. Zaslavsky, "Multihoming with Mobile IP," 6th IEEE International Conference on High Speed Networks and Multimedia Communications, 2003

[93] W. Richard Stevens, “TCP/IP Illustrated, Volume 1: The protocols”, Addison-Wesley Publishing, 1994

[94] W. Richard Stevens, “TCP/IP Illustrated, Volume 2: The Implementation”, Addison-Wesley Publishing, 1995

[95] Amirthalingam, K.; Moorhead, R.J., "SNMP-an overview of its merits and demerits," System Theory, 1995., Proceedings of the Twenty-Seventh Southeastern Symposium on , vol., no., pp.180-183, Mar. 1995

[96] Michael G. WILLIAMS, “Directions in Media Independent Handover” IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences, Vol.E88-A, No.7, pp.1772-1776, Jul. 2005

[97] Antonio de la Oliva et. al, “A, IEEE 802.21 enabled mobile terminals for optimized WLAN/3G handovers: a case study”, ACM SIGMOBILE Mobile Computing and Communications Review, Vol. 11, Iss. 2, Apr. 2007

[98] Hayoung Oh et. al, “A Robust Seamless Handover Scheme for the Support of Multimedia Services in Heterogeneous Emerging Wireless Networks”, Wireless Personal Communications: An International Journal, Vol. .52 No.3, p.593-613, Feb. 2010

[99] J. Case et. al, “Introduction to version 2 of the Internet-standard Network Management Framework”, Request for Comments 1441, Apr. 1993

[100] J. Case et. al, “Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2)”, Request for Comments 1905, Apr. 1999

[101] J. Case et. al, “Management Information Base for version 2 of the Simple Network Management Protocol (SNMPv2)”, Request for Comments 1450, Apr. 1999

[102] J. Case et. al, “Introduction to version 3 of the Internet-standard Network Management Framework”, Request for Comments 2570, Apr. 1999

[103] B. Carpenter, Editor, “Architectural Principles of the Internet”, Request for Comments 1958, June 1996

[104] Michael G. WILLIAMS, “Directions in Media Independent Handover” IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences, Vol.E88-A, No.7, pp.1772-1776, Jul. 2005

[105] Antonio de la Oliva et. al, “A, IEEE 802.21 enabled mobile terminals for optimized WLAN/3G handovers: a case study”, ACM SIGMOBILE Mobile Computing and Communications Review, Vol. 11, Iss. 2, Apr. 2007

[106] IEEE 802.21 WG, IEEE Draft Standard for Local and Metropolitan Area. Networks: “Media Independent Handover Services”, IEEE P802.21/D10.0, 2008

[107] 後郷 和孝, 寺岡 文男, “動 的なネットワーク環境に適応するためのクロスレイヤシステムの設計と実装”, 電子情報通信学会論文誌Vol.J91-D, No.3, pp.733-743, March 2008

[108] Vu Truong Thanh, Yoshiyori Urano “Agent based handover scheme for SIP communication – The case of TCP sessions” The 24th International Technical Conference on Circuits/Systems, Computers and Communications, Korea, 2009

[109] Teemu Koponen, et. Al, ”A data-oriented (and beyond) network architecture”. Proceedings of the ACM SIGCOMM 2007, pp. 181-192, Kyoto, Japan. August 2007.

[110] Zhang, L.; et. Al, “Named Data Networking (NDN) project”, PARC technical report TR-2010-02; October, 2010

[111] Jacobson, V., et. al, “VoCCN: voice over content-centric networks”, Proceedings of ReArch 2009, Rome, Italy, December, 2009,

[112] Teraoka, F, “Redesigning Layered Network Architecture for Next Generation Networks”, IEEE GLOBECOM Workshops, December, 2009

[113] AKARI Architecture Design Project “AKARI Architecture Conceptual Design [translated version 2.0] (Preliminary) released”, National Institute of Information and Communications Technology (NICT), May, 2010

[114] Yunsop Han, Fumio Teraoka, “SCTPmx: An SCTP Fast Handover Mechanism Using a Single Interface Based on a Cross-Layer Architecture”, IEICE Transactions on Communications, Vol.E92-B, No.9, pp.2864-2873, September 2009

[115] Hayoung Oh et. al, “A Robust Seamless Handover Scheme for the Support of Multimedia Services in Heterogeneous Emerging Wireless Networks”, Wireless Personal Communications: An International Journal, Vol.52 No.3, p.593-613, Feb. 2010

[116] Jennifer Rexford and Constantine Dovrolis, “Future Internet architecture: clean-slate versus evolutionary research”, Communications of the ACM , Vol. 53 No. 9, pp. 36-40 , September, 2010.

LIST OF ACADEMIC ACHIEVEMENTS

Category (Subheadings)	Paper Title	Relation with the disseration
Articles in refereed journals	○ Vu Truong Thanh, Hidetoshi Yokota and Yoshiyori Urano, "Defining Cross-Layer Fine-Tunable Parameter List in TCP/IP Networking Architecture for Communication Service Optimization and Customization", Journal of Computer Science and Cybernetics, Vol. 27, No. 1, pp. 36-50, June 2011	Incorporated in chapter 3 and 5 of the dissertation
	○ Vu Truong Thanh and Yoshiyori Urano, "Object-Oriented Approach to a New Cross-Layer Information Manipulation Model for TCP/IP Architecture", Forthcoming in GITS/GITI Research Bulletin 2011	Incorporated in chapter 4 of the dissertation
Presentations at International conferences	○ Vu Truong Thanh and Yoshiyori Urano, "Mobile TCP socket for secure applications", The 12th International Conference on Advanced Communication Technology, pp. 971-974, Gangwon, Korea, Feb. 2010	Incorporated in chapter 2 and 5 of the dissertation
	○ Vu Truong Thanh and Yoshiyori Urano, "Agent based handover scheme for SIP communication – The case of TCP sessions", The 24th International Technical Conference on Circuits/Systems, Computers and Communications, Paper Number: C-10-0599, session S16-D3, Jeju, Korea, Jul. 2009	Incorporated in chapter 2 of the dissertation
	○ Vu Truong Thanh and Yoshiyori Urano, "Link-based service customization for NGN", The 10th International Conference on Advanced Communication Technology, pp.57-60, Gangwon, Korea, Feb. 2008	Incorporated in chapter 2 of the dissertation

	Vu Truong Thanh and Yoshiyori Urano, “Agent based low loss low delay handover scheme for SIP communication – The case for UDP traffic”, The 11th International Conference on Advanced Communication Technology, pp. 2099-2102, Gangwon, Korea, Feb. 2009	
Presentations at International conferences (No refereed)	Vu Truong Thanh and Yoshiyori Urano “Development of “Next Generation Network Basics” e-Learning Courseware based on cooperation among AIC members”, Asia Info-Communication Council 35th Conference, Document Number 215, Saitama, Japan, Mar. 2007 Vu Truong Thanh and Yoshiyori Urano, “Distant E-Learning and Information sharing by Geographic Information System on Community Network in rural area, Asia Info-Communication Council 35th Conference, Document Number 216, Japan, Mar. 2007 Vu Truong Thanh and, Yoshiyori Urano “Experiment of Session Initiation Protocol’s instant messaging”, Asia Info-Communication Council 32nd Conference, Document Number 143, Halong, Vietnam, May 2005	

Others	KDDI, Waseda University, NIICS, “Joint Research on an Advanced Learning & Training System based on Social Networking Site service, Hanoi, Vietnam”, Final Report, Asia Pacific Telecommunity Project, Nov. 2009, Hanoi, Vietnam KDDI, Waseda University, Phutho P&T Dept., “Distant E-Learning and Information sharing by Geographic Information System, Phutho, Vietnam”, Final Report, Asia Pacific Telecommunity Project, May 2007, Phutho, Vietnam Waseda University (Japan), Posts and Telecommunications Institute of Technology (Vietnam), “NGN Advanced”, E-learning Material, Feb. 2008, Tokyo, Japan Waseda University (Japan), Posts and Telecommunications Institute of Technology (Vietnam), “NGN Basics”, Feb. 2007, Tokyo, Japan Waseda University (Japan), Universiti Malaysia Sarawak (Malaysia), “IP-Mobile Advance”, Mar. 2006, Tokyo, Japan Waseda University (Japan), Posts and Telecommunications Institute of Technology (Vietnam), “IP-Mobile Basics”, Mar. 2005**,** Tokyo, Japan	