Object-Oriented Approach to a New Cross-Layer Information Manipulation Model for TCP/IP Architecture

Vu Truong THANH and Yoshiyori URANO

Abstract

The TCP/IP networking model was originally designed with a focus on connectivity adhering to layering principle. While this principle has facilitated the speedy expansion of the Internet, it also prevents application layer services from carrying out service optimization and customization due to the lack of current status from lower layers. The purpose of this research is to propose a new TCP/IP networking architecture that can facilitate the sharing of information across layers' borders. Object oriented design will be used to analyze and develop the new architecture. Some examples on the application and coverage for future development of the new architecture will also be discussed.

Keywords: *Layering principle, cross-layer, service optimization, service customization*

1 Introduction

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.

With the original focus on connectivity, conventional IP networking implementation is more suitable for routers than end-hosts and it facilitates very well the phenomenal development of the Internet. At the end-hosts, 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. This 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 mechanisms [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 them are nomadic-related issues, become widely available. The TCP/IP architecture is built on the assumption that the terminal's access point to the network was static, with stable electrical signal, therefore IP addresses are used for both routing and identification, and loss is due mostly to congestion.

As mobile access introduces more volatile properties to communication sessions such as change of subnet, unstable transmission quality, together with the proliferation of new and more demanding ubiquitous services, it is time that the conventional model of self-contained, status-hiding layering approach of the Internet be revised to provide applications with more information and control of the underlying operations to better adapt to the changes of the Internet realm as shown in Table 1.

Table 1 The change of the Internet paradigm

	Conventional Internet	Advanced Internet
Feature	Fixed and static	Mobile and dynamic
End devices	Desktops	Smart phones, tablets
Connection type	Wired, few types, slow	Wireless, many types, very fast
Data and Traffic	Homogenous traffic, low volume	Heterogeneous traffic, exponentially growth
Service	Some few major text based services	Various real-time interactive multimedia services

In the next part of the paper, we will discuss further about the need for a new approach to the layering architecture of the Internet. Based on this discussion, we will design the new approach (called the Inter-Lay scheme) using object-oriented design in chapter 3 and chapter 4 discusses some related works and compares this research with them, as well as provides information on some possible application and how the proposed scheme can provide for possible changes of the Internet. Chapter 5 concludes the research with final conclusions and future works. In addition, the test questions to find the suitable network parameters to be exposed will be discussed in the appendix section.

2 The issues with the original layering approach

The ARPAnet began with just one core protocol, the Transmission Control Program (TCP), which was formally described in RFC 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].

TCP/IP stack follows strict layering principles, which in general requires that the operations and internal states of each layer are known only to the layer itself, and a layer communicates only to adjacent layers. The main advantage of this layered 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.

This localization of changes and incremental development were very important in the early development stage when there was almost no information on how the architecture would behave in different settings and environment. For example, TCP congestion control algorithm was refined many times as the network experienced various new types of communications services and networking technologies.

The disadvantage of layer enclosure is that except the Protocol Data Unit (PDU), higher layers have virtually no status information from lower layers therefore it has to accept general assumption that lowers 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. In other words, it might be beneficial for higher layers to learn more information from lower layers to optimize as well as to provide seamless operations to end-user communication services.

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]. Or if there is a way for the application to be informed that the TCP layer is constantly experience congestion, it can cooperate by reducing its transmission rate for example by using a slower codec.

On the other hand, lower layers such as the Data Link layer can make a better decision when performing a handover if it receives the preferences of the above applications.

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). So 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 to allow a layer to expose more internal data/states to other layers in order to give user applications more flexibilities and customizations.

The following chapter 3 will describe such a new TCP/IP networking stack architecture that can facilitate cross-layer boundary communications. The new architecture is proposed to be implemented with Object Oriented (OO) Technology to facilitate the development process.

3 Cross-layer data provision networking model

In this part, a new layering architecture is proposed, in which lower layers will reveal selected internal information to higher layers, either adjacent or several layers away. This new and more complex architecture would be deployed in end devices to support service development, while routers continue to use the traditional version of the TCP/IP architecture, because the new model adds extra capabilities and complexity indented for user applications that do not exist in routers.

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

As a practical guideline for real-world development of the networking stack, we will base our discussion 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 Operating Systems (OSs) are similar in capabilities so it can be extended easily to other platforms.

Currently the TCP/IP networking stack is implemented in structured programming fashion, where the operations are carried out in sequential procedures. However, keeping the existing programming model, and adding codes to expose a layer's internal parameters will be risky as variables are accessed directly, there is no way to guarantee the integrity of network parameters even in a read-only procedure in the case of structured programming and 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 allows for the exposure of internal data of a layer without the danger of (mistakenly) altering the data natively. Another aspect is that with conventional procedural programming, whenever the parameter is accessed or

updated, the codes for basic protection mechanisms (for authorization, integrity check etc.) will have to be repeated, while in OO programming, all of these basic protection is carried out only once in the *set()* and *get()* method for the parameter, and whenever the parameter is accessed or updated, the 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 more advantages in applying OO paradigm to the new cross-layer communication model as follows:

- The layering approach (and protocols) and OO technology have the same principles, namely self-contained internal attributes, interactions using pre-defined interfaces, the modification of one entity does not affect other existing entities. As the data/ operation of each layer are extensively analyzed and documented, converting TCP/IP layer to object should be straightforward.
- The new cross-layer communication architecture concentrates on data to be provided cross-layer. This data-centric purpose obviously means suitability with OO Programming. Moreover these data should be accessed and modified with utmost discretion which can be easily done by OO Programming natively.
- Unlike the existing networking model, 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 method to a protocol class in OO Programming than to find the right place and right mechanism in procedural programming.
- As new protocols are being introduced into TCP/IP to accommodate new communication requirements, the ability of OO to reuse common codes with inheritance and polymorphism will make 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 use as a template to develop TCP and the newer SCTP (*Stream Control Transmission Protocol*) of which the original methods will be overridden with the correct input parameter using polymorphism.
- As new protocols and features are constantly introduced, the networking subsystem will have to be actively and continuously maintained for a very long period of time. The advantage of OO Programming documentation will make the succession process among programmers more smoothly.
- Because the query and update activities are carried out independently among classes, we can assign the object of each class in Figure 1 below in a separate thread, and as multi-processor CPU 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 to develop the cross-layer enabled networking architecture. In this model the protocols of each layer is conceptualized into classes. The overall system is depicted in Figure 1. The model also needs some supplement classes such as buffers for PDU which are not depicted in the figure for simplification.

In this model, each layer will be represented by a generic class with all the basic functionalities and parameters of that layer, and a specific protocol class will inherit the generic layer class and add the attributes and functionalities specific to itself. The network parameters of the protocol become the corresponding attributes of the class. Note that while the class contains all parameters of the protocol, in this discussion we only concern the parameters that should be revealed across layers. For Data Link layer, the object might be designed as a wrapper object of the device driver.

Moreover, a protocol can be mapped individually to a class, or if there is a group of protocols that share common major properties, they can be mapped into one class. For example, the IEEE 802.11 a, g family maybe mapped into one single object if their differences in operating frequency and modulation techniques are out of interest of applications. 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.

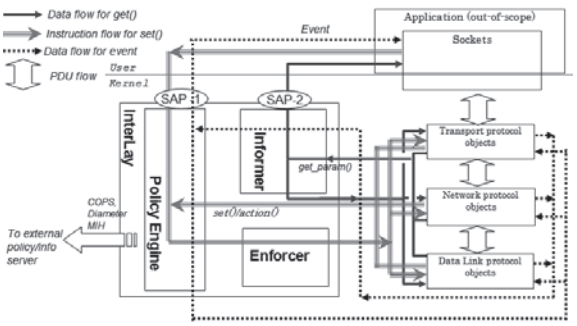

Figure 1. The new TCP/IP architecture

There are two types of attributes for a protocol class. One is real-time attribute, where the current value of the layer's parameter is of importance, and the other is event attribute, where the importance is not the instant value but whether that value has crossed above or below a certain value (which generates an event) or an external event is received, such as the Explicit Congestion Notification (ECN). For example, the current sequence number of TCP object is real-time attribute, but the round-trip time (RTT) of a packet is only important when it is greater than that of the round-trip time timeout (RTO). As an event can happen anytime during a certain period, other protocol objects (including the applications) cannot read the event arbitrarily but will register/de-register to be notified of the event for a certain period. The event can be associated with additional information when be informed to other protocol objects.

Each real-time attribute will be associated with a *get()* method as a mean for the upper layer to learn the value of the associated parameter. The implementation of the *get()* method will decide how to calculate the value of the associated parameter, and special attention is required so that the calculation process does not accidentally alter or damage the system.

While most of the attributes will be read-only (i.e., associated with only a *get()* method), a few real-time attributes will be also assigned with a *set()* method to adjust that attribute to a value suitable with the judgment of the higher layer on the current state of the connection, and the *set()* method is more appropriate to real-time attributes. The event attributes are not suitable for *set()* method and in the case the system needs to initiate some processing related to an event, it should be implemented as a separate method. Because of the potential damage that could be caused if the *set()* method is not carried out correctly, discussion on the *set()* method will be given in full separately in section 3.3 below.

The actions or procedures that a protocol performs in its execution are implemented as methods of the corresponding class. For example the Mobile IP object can contain an *RO()* method that once called, it will perform route optimization procedure to a given destination. These methods will be called "*action()* method", and they can be invoked by other layers as well. Apart from those intrinsic *action()* methods of the protocol, a protocol class will include the methods to manipulate the PDU, especially the protocol headers, as well as method to prepend/append certain information to the PDU.

And because the *action()* method also has the potential to affect the operation of the system, they will be discussed more in section 3.3 together with the *set()* method.

3.2. The InterLay object

In the proposed model, the protocol object will perform all functionalities of the relevant protocol. The only extra processing are *set()* and *get()* method. On the other hand, 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 layer (i.e. not limited to the Application layer) can query and update certain parameters of protocol objects from lower layers.

As shown in figure 1, the InterLay is divided into 3 functional groups: the Policy Engine (PE), the Enforcer and the Informer.

The Policy Engine

The PE communicates with external entities for supplement policies and information using standard protocols (Diameter, COPS, MIH …). It also performs all necessary permission and integrity checks when updating a network parameter, normally to guarantee that the update does not negatively affect the networking subsystem or other processes. The Policy Engine can also be equipped with a rate-control mechanism to monitor and limit the access rate to a certain parameter (i.e. rate of calling the *get()/set()* for that parameter), as well as the overall request to the InterLay object of a certain application to a reasonable number. This number can be decided based on the nature of the parameter, for example the limit for fast changing parameter can bet set higher for slow changing parameter.

For event-based attribute, the PE also accepts registrations/de-registration from other protocol objects for events coming from the Informer, and complements the handler for the event with any extra processing that is required by the registration/de-registration activities.

For real-time attributes, each is associated with a certain priority, and a requesting protocol object will also be assigned with a certain priority. Excluding the application layer, in general higher layer protocol has a higher priority than a lower one because it has more comprehensive view of the current condition. The priority of an application object is highest for parameter that is dedicated to itself (i.e. parameters from layer 4 (L4) protocol object created by the application) and an application object has a default lowest priority for parameters of layer 2 and layer 3. On the other hand, the priority of requests coming from external policies can be set on a case-by-case basis.

An update request for the value of an attribute is served only if its priority (equal to that of the requesting object) is higher or equal to the current priority of the parameter, and requests 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 by setting its priority

to the highest exclusive priority with infinite lifetime.

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

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

The Enforcer

The Enforcer performs actual changes of real-time parameters as requested by the PE, with necessary corresponding processing of other parameters/procedures. It also contains *action()* methods to react to a event which is also triggered by the PE.

The Informer

The Informer provides objects of other layers with current value of real-time parameters at request time. Note that for fast changing parameter, the value might be outdated when reaching the requesting object. It also informs Policy Engine of event and adds the handler from the PE to the notification changes for that event.

Also, the Informer may cache the value of the parameter that is judged to be static (for example the Home Address in Mobile IP protocol) so that the InterLay has to call the *get()* function for that parameter only once at the first time the parameter is requested. This will improve the performance of the networking subsystem.

The InterLay class also contains *primitive methods*, which are basic building blocks that perform supplementary actions to assist the invocation of the *get()/set()* and *action()* methods of the protocol objects. These primitive methods may perform, for example, integrity or authentication checks.

The Interlay resides in the kernel space therefore it can interact directly with lower layer protocol objects. However, for higher layer (namely the application layer) standard interface (i.e. service access point-SAP) with standardized APIs should be defined to allow the access to InterLay's functionalities.

InterLay object and lower layers

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

For event attributes, InterLay can get events from lower layers by taking advantage of kernel's *notification chains* [27]. There are two ways to use *notification chains.* One is to register relevant action blocks from the Enforcer directly to the notification chains of the event, and the other is to introduce a method at the PE that will take care of the necessary processing. The advantage of the first approach is its simplicity in implementation, while the latter one can provide not only linear chain of reaction but also reaction with condition and loop.

InterLay object and application layer

Socket resides at the user space therefore it cannot interact directly with the InterLay object. Also, the relationship is asymmetric: InterLay provides information on and receives instruction of what to do with lower layers.

For real-time attribute, the socket can query directly the Informer through the standardized interface, while for events the socket should register for the event with the PE when it needs to be informed about that event. The registration will have a certain lifetime, or the socket can actively deregister (to improve overall performance)

The Policy Engine also provides the application with an interface to exchange application-specific policies (to simplify the operation of PE, it does not exchange policy directly with the application server of a specific application).

Separating cross-layer related processing to a separate InterLay object instead of exposing directly the networking parameters via protocol objects has several advantages. The protocol objects can concentrates on doing its main job therefore we can reuse existing algorithms and (possibly) code which allows even faster development. And because InterLay does not differentiate between layers, a standard procedure can be established to add/modify/remove access to the parameters of all layers. Moreover, no complex mechanism to prevent deadlocks among competitive requestors is necessary because all accesses to *set()* and *action()* methods are carried out via Interlay. Finally, if the InterLay object is implemented as a separate kernel module and interacts with the networking subsystem through standardized kernel mechanisms (such as kernel symbol table in Linux) then any failure in the InterLay object will only affect the additional cross-layer functionality but the conventional networking functionality remains intact, thus increase the reliability of the communications system close to that of the conventional networking model.

New system calls for InterLay scheme

Several new system calls should be defined to realize the service access points (SAP) that carry out the interactions between the socket and the InterLay objects. Each network parameters or *action()* method that is exposed by the InterLay will be given a unique predefined (DWORD) code to be used with the system calls, and there are 3 separate naming domains: one is for real-time parameter, one is for event parameter and one for *action()* method.

A new *net_set_param()* socket API function is used at the SAP1 interface, which allows the socket object to assign new value to the real-time attributes of lower layer objects (normally the Transport) through PE. *net_set_param()*

requires a set of 2 input parameters: {the predefined code of the network parameter, the new value}. It invokes the *InterLay.set_param()* method of the PE functional group with two more additional function parameters: the caller's process ID and the socket identifier (i.e. the socket's address tuple). The *InterLay.set_param()* will use the caller 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 it will ask the Enforcer to invoke the *set()* method for the protocol object's attribute that corresponds to the requested network parameter's code and socket's identifier.

A new *net_get_param()* socket API function is used at the SAP2 interface, which allows the socket object to query the value of a real-time attribute of lower layer objects through the Informer. The *net_get_param()* requires the predefined code of the network parameter as input parameter and invokes the *InterLay.get_param()* method of the Informer functional group with two more additional function parameters: the caller's process ID and the socket identifier (i.e. the socket's address tuple). *InterLay.get_param()* will invoke the *get()* method for the protocol object's attribute that corresponds to the requested network parameter's code (and socket's identifier if Transport Layer information is requested) and returned the value to the requesting application.

A new *net_invoke_action()* socket API function is used at the SAP1 interface to allow the application to invoke an *action()* method of a protocol object, with one function parameter which is {the predefined code for the concerned *action()* method}. The *ivk_net_action()* system call will be mapped to the *InterLay.ivk_action()* with two more additional function parameters: the caller's process ID and the socket identifier (i.e. the socket's address tuple) will be called by the application by indicating a predefined flag, and the application can call a sequence of *action()* by indicating the corresponding sequence of flags. The *InterLay.ivk_action()* method will use the caller 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 *action()* method of the appropriate protocol object.

A new *net_reg_event()* system call is used at the SAP1 interface to allow the application to registered for a specific event. The *net_reg_event()* function will take the predefined code for the concerned event and (optionally the socket identifier -i.e. the socket's address tuple- if the event belongs to Transport protocol object) as input parameters. The PE functional group of the InterLay will add the ID of the calling process to the handler of the concerned event. When the concerned event happens, the InterLay can inform the application using any appropriate Inter Process Communication mechanism.

Note that *net_set_param(), net_get_param(), and net_invoke_action()* are implemented as socket API functions while *net_reg_event()* is implemented as general system call due to the fact that an application might create several connections at the same time and a L2/L3 event is common to all of them.

3.3. On the 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 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 other 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 requestor. 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 only.

4 Discussion and analysis

4.1. Related works

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 takes into consideration some component 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, and 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 transmissions 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 organized 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.

Media Independent Handover architecture [31] bears resemblance to the InterLay scheme, however MIH confines to handover related activities only while InterLay can support new and extra functionalities for all kind operations at all layers, and in practice InterLay can replace the functionalities of MIH.

In summary, the existing researches on cross layer application examine the benefit of cross-layer design from the performance approach by asking different layers to adapt itself according to the current status of the network on a case-by-case basis while the coverage of InterLay scheme is much more broader to better adapt to the changes of the Internet realm as described in chapter I.

4.2. Coverage of InterLay scheme

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

InterLay scheme and TCP mobility

Mobility can be introduced in IP layer [28], or the TCP layer. In [18] we have proposed the mobile TCP socket that supports mobility for TCP session. Basically, this mobile socket provide an interface to change (i.e. the *set()* method) the PCB (Protocol Control Blocks) parameters to maintain TCP session across address changes. We have proved that the inter-layer exchange of information provide some advantages to 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 (both traffic and processing) and latency.

InterLay scheme and TCP fault-tolerant across local networking subsystem restart

Similar to the mobility above, the manipulation of the PCB can be used to save the TCP session over the restart of the subsystem.

Suppose that the networking subsystem (NS) is about to restart due to some errors, and it can communicate with active sockets before restarting. The preservation of the TCP session is carried out as follows (see figure 2):

Step 1: The application that wishes 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 registered 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 parameter.

Step 4: 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 that turns on a FREEZE_FLG. *TCP_object* is the TCP object that is created by the concerned socket. Because the TCP object is required to check that the FREEZE_FLG to be off before sending or receiving a datagram to/from the IP object, this will freeze the sending/receiving activities. However, the socket will continue to read data that is already in the buffer until it is empty because those data have been ACKed.

Step 5: The application calls the socket's *net_get_param(),* with the TCP-TCB parameter code to get the TCB

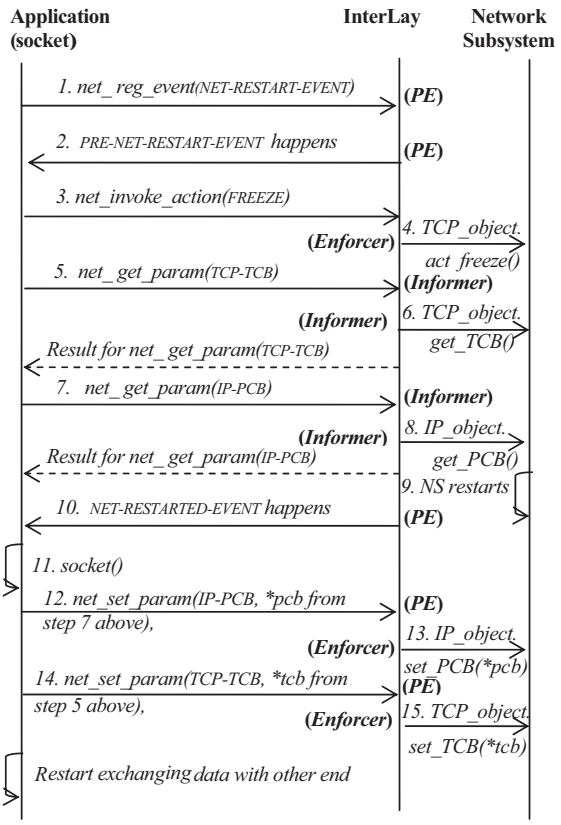

Figure 2. The interaction diagram for TCP's fault-tolerance procedure (dotted lines are returning value for *net_get_param()*)

of the socket.

Step 6: The Informer will query the *TCP_object.get_TCB()* and returns the result to the applications. (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 applications. (The returned result contains all the status information of the PCB.)

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 registered for the *NET-RESTART-EVENT*. (The list of those applications is stored 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 parameters, 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()* 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.update_TCB()* action method.

The application now can issue *send()* and *receive()* requests to the socket. Note that the application might need to send three datagrams with the same ACKed number so that the other end retransmits any lost data.

Note that this scheme is extendable to the case of UDP (with 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.

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 [30].

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.

4.3. Discussion and analyses

From the above discussion and analysis, we can see that the research in this paper allows for the networking subsystem and user application not only to be able to adapt the performance according to lower layers status, but it also allows (i) more choices beside performance (such as arbitration decision for route optimization) and (ii) 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 networking subsystems for mobility and fault tolerance. This can support the 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, 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 arisen problems.

Moreover, because the InterLay object is designed as a separate object, the main advantage of layering principle, namely the modification/revision of a protocol would not affect other protocols of the protocol stack, is maintained. For example, if a parameter of a protocol is obsolete (or introduced), and the TCP/IP stack is revised to remove/add the parameter, then only the called to the *set()/get()* of the parameter is affected. In this case only the application that uses the removed (or added) parameter has to be updated, which is also a practice applied to conventional TCP/IP implementation. As such, the InterLay imposes no additional limitations on the flexibility development of the TCP/IP networking family.

Moreover, as the research in [24] has pointed out, one important question with cross-layer design idea is "How do the different cross-layer design proposals coexist with one another?" Because this research proposes to provide not only attributes (and associated *set()* and *get()* method) but also *action()* methods, it can serve as a general platform for other proposals to implement their algorithm 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, and then we have a ready-to-deploy implementation of the networking subsystem that supports the ideas in [19].

In terms of operation speed between OO programming and conventional procedural programming, the TCP/IP stack will experience 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 NS 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 tear 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 neglect-able, and the main extra overhead will come with (ii).

However, the look-up of the v-table takes only several CPU cycles [32] [33], 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.

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

5 Conclusion

5.1. Major contributions

The Internet has the characteristic of "dump 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. This research aims at resolving this problem. 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. As a result, a new architecture called InterLay is introduced to provide cross-layer manipulation in TCP/IP networking stack. Several examples are provided to signify the usefulness and advantages of the new architecture in chapter 4.

The research proposes the mapping of the cross-layer model to object-oriented design. This streamlines the development of the new architecture abstraction into real implementation, by making use of existing OO conversion tools and framework for protocol development. Moreover, the OO design turns the proposed new architecture into an implementation platform for other ideas on cross-layer design, as explained in the end of chapter 4. And by the introduction of the InterLay object, we can separate the *set()/get()* method of a protocol object from the procedure of interacting with protocol objects, reducing the possibility of bugs.

And although the discussion focuses on TCP/IP protocol suite, it is general enough to be extended to other networking model that uses layering approach as well. In this sense, it is also useful to be used as a reference model for the design of new network architecture in the future. Moreover, it can also be used as a guideline for embedded system where customization and optimization are very crucial.

5.2. Future works

Future works include 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.

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", RFC 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", RFC 4881, 2007

[7] J. Crowcroft et. al, "Is Layering Hamful", 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, August 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", RFC 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", RFC 4340, 2006

[26] R. Stewart, Ed., "Stream Control Transmission Protocol", RFC 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", RFC 3344, 2002

[29] R. Moskowitz et. al, "Host Identity Protocol (HIP) Architecture", RFC 4423, 2006

[30] W. Simpson, "IP in IP tunnelling", RFC 1853, 1995

[31] IEEE 802.21 WG, IEEE Draft Standard for Local and Metropolitan Area. Networks: "Media Independent Handover Services", IEEE P802.21/D10.0, 2008

[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

APPENDIX

Exposing new network parameters to the InterLay object would require recompilation of the networking subsystems and that would be inconvenience. Therefore the more comprehensive the list of parameters to be exposed the better. However, not all parameters would be used across layers, and exclude them would reduce the size of InterLay object meaning less development work, lower possibility of failure due to smaller size. As a result, we need to find a method to find as many as possible of the suitable parameters to be exposed.

In order to find the right parameters, we propose to define the test questions to determine whether a parameter should be exposed. The test questions are defined to find the parameters that improve performance or flexibility of the connection. From current and foreseeable network's capabilities vs. services' demands, we propose the following three test questions.

● *Question #1: Does the information signal a critical change of condition 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 allows higher layers (including user applications) to tune its activities to the current or near future condition of the network.

The information can be either calculated from a direct parameter/attribute of the layer or the information it received. For example, the application can switch to a slower codec to reduce the sending data rate if congestion is judged as imminent when the TCP layer either reports that the calculated RTT shows a time-out event has occurred, or if the TCP layer receives an ECN (Explicit Congestion Notification) from its peer.

On the other hand, the information can be the result of an operation of the protocol. For example, the successful change of IP address can trigger some appropriate action from the higher layers, such as updating flows parameters or renegotiation of QoS for the existing flow(s), etc. Another example is when the wireless signal reaches a threshold so that an L2 handoff to another base station is imminent, the L2 can either inform this threshold event immediately (which gives more time for upper layers to prepare), or wait until the network interface has successfully connected and been authenticated to the new base station.

● *Question #2: Does the information a static parameter that regulates the performance of the transfer?*

A layer can have many state variables; some are threshold type, such as RTT, while others are real-time and static, such as MSS, buffer size. Real-time static information can be used to synchronize between parameter of different layers etc.

Threshold parameters are not useful by its direct reading, but when it is associated with a threshold value, then it can either be regarded as a static parameter (and belongs to this question), or it can be categorized as belonging to question #1.

For example, the RTT by its straight value is not of much use, but if it passes the threshold that generates an RTO event, the event can be regarded as a signal of the change in network condition (e.g. becoming congested) which then belongs to category of question #1 above.

● *Question #3: Is the information a parameter that helps to identify the connection or its current progress?*

This question is relevant with more advance or future capabilities of the networking subsystem, such as fault-tolerant/failsafe/failover of connection, or session mobility, QoS, security, etc. Currently, a session is still identified by IP addresses. However, a separate ID would make it easier for issues such as mobility [12] [13]. Also, checkpoint and acknowledged sequence number are good example of markers for progress of a communication session.

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 #2) as well as the current progress (such as last checkpoint, sequence number) then it will be easier for the application to restore the relevant session after a reboot.

The previous 3 test questions come from current and foreseeable network's capabilities and services' demands, and they need to be constantly examined to find out new test questions and attributes/*action()* method 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.

By applying the above questions to each layer, we can summarize the list of some major parameters from famous networks protocols in each layer that should be revealed to the applications and other higher layers in the following tables (if the first column is checked then the parameter is an event parameter).

For layer 2 (Data Link Layer) (Note: the first 7 parameters come from MIH [31]

R/E	Proto. Family	Param.Name	Usage/Purpose
*	Wireless	Link Up	The link is active and ready
*	Wireless	Link going up	Support the preference of a network
*	Wireless	Link Down	Link cannot be used for data transmission
*	Wireless	Link Going Down	A link down event might be fired soon → should start prepare for HO
*	Wireless	Link quality reaches threshold	Link quality is under a pre-config. thres. for a long time → should start prepare for HO
*	Wireless	Better quality AP avail.	An Access Point with a better signal is available. Might provide higher data rate
*	Wireless	Link Handover Complete	Notification of a fresh handoff. The app. might need to readjust its data rate
	All	Type of access technology	(WIRED/WIRELESS/3G) → the application can infer in general the rate, QoS, Security…
	Wired tech.	Electrical Signal Stability	Preventing route damping in routers
	All	MAC	Support ID creation/ manipulation activities

For layer 3 (IP Layer)

R/E	Pro.Fa	Param.Name	Usage/Purpose
	All IP version	Source IP address	Useful to restore connection over network subsystem reboot or adhoc handover
	All IP version	Dest. IP address	Useful to restore connection over network subsystem reboot or adhoc handover
	All MIP version	CoA	Know the current attachment point for any adhoc route optimization
	IPSec	Security association	Useful to restore security connection over network subsystem reboot …

For layer 4 (Transport Layer)

R/E	Pro.Fa	Param.Name	Usage/Purpose
	All	MSS	To support Application Level Framing
	All	Source Port	Adhoc Mobility support
	All	Destination Port	Adhoc Mobility support
	TCP	Sequence Number	Adhoc Fault-Tolerant activities etc…
*	TCP	ECN received	Application tuning such as changing codec etc.
*	TCP	RTO event	The app. knows that a RTO just has taken place so it can adjust its sending rate
	TCP	Window size	Adhoc adjustment of transmit rate by application

For layer 5 (i.e. equal to Session layer in OSI reference model):

R/E	Pro.Fa	Param.Name	Usage/Purpose
	N/A	Checkpointing value	Fault-Tolerant activities etc…
	N/A	Session ID	Fault-Tolerant, handoff activities etc

In addition, some useful action() methods for IP protocol are described as follows

Pro.Fa	Method Name	Usage/Purpose
IP	MTU_change()	Used to renegotiate the MTU after adhoc handover of IP connection
IP	Route_Optimized()	Used to perform RO to a selected destination for a selected IP connection