ECLAIR: AN EFFICIENT CROSS LAYER ARCHITECTURE FOR WIRELESS PROTOCOL STACKS

Vijay T. Raisinghani*
Tata Infotech Ltd.(ATG) and
KR School of IT
IIT Bombay
rvijay@it.iitb.ac.in

Sridhar Iyer
KR School of IT
IIT Bombay
sri@it.iitb.ac.in

Abstract

Seamless mobility across heterogeneous mobile wireless technologies is now essential as mobile subscribers demand full and cost-effective wireless network coverage. Under such mobility conditions the *layered* protocol stack is inefficient. Significant research has been done for cross layer optimizations of the protocol stack. To enable rapid deployment of existing and new cross layer optimizations an architecture for cross layer optimizations is essential.

In this paper, we present our architecture *ECLAIR* which can serve as a blueprint for development of cross layer feedback systems. ECLAIR consists of two main components – *optimization subsystem* and *tuning layers*.The optimization subsystem contains *protocol optimizers* that effectuate cross layer optimizations. The protocol optimizers interact with the existing stack through the tuning layers.

In ECLAIR we exploit the fact that stack behavior is determined by the values stored in various protocol data structures. Our architecture facilitates easy manipulation of these values stored in the protocol data structures. To the best of our knowledge, ECLAIR is the first generic architecture for cross layer feedback.

1 Introduction

The new wireless networks – 3G and beyond [5, 12] – are expected to be all-IP and using standard protocol stacks, e.g. TCP/IP[11] (Transmission Control Protocol/Internet Protocol), to ensure interoperability.

The standard protocol stacks are *architected*[4] and *implemented* in a layered manner for the purpose of modularity. However, such *layered* stacks function inefficiently in mobile wireless environments[14]. This is due to the highly variable nature of wireless links and the resource-poor nature of mobile devices. To enhance the performance of these protocol stacks, significant research has been done for cross layer optimizations [2, 15, 6, 1, 9, 8]. See [7] for a survey on cross layer optimizations. Some examples of cross layer feedback are: (1) TCP packet loss information communicated to the application layer to enable application adaptation, (2) link/MAC layer tuning the transmit power of the physical layer based on the bit-error rate information from the physical layer.

As new wireless networks are deployed, to enhance the performance of the protocol stacks and to enable seamless mobility multiple cross layer feedback algorithms would be required. These algorithms would need to be easily integrated with the existing stack. An architecture for cross layer feedback would help standardize and ease the development, deployment and maintenance of these cross layer optimizations. An architecture for cross layer feedback is still an open research question.

We propose an architecture ECLAIR which provides a blueprint for designing and implementing cross layer feedback in an easy and efficient manner. Figure 1 shows a top-level view of our architecture. The main components are the *Optimizing SubSystem* (OSS) and the *Tuning Layers* (TL). OSS is the cross layer engine. It contains many *Protocol Optimizers* or POs. POs are the *intelligent* components of ECLAIR. The TLs provide the necessary APIs to the POs for interacting with various layers and manipulating the protocol data structures.

The POs take input from various layers and other device entities like the battery and decide the optimizing action to be taken. The optimizing action could be to reduce power consumption or reduce packet losses, etc. The optimizing actions are achieved by modifying existing protocol stack behavior. The POs manipulate the values stored in the protocol data structures so as to modify the protocol stack behaviors.

This paper is organized as follows: the design goals for a cross layer architecture, an overview of ECLAIR and its salient features are presented in section 2; details about ECLAIR are presented in section 3; in section 4 we discuss some example POs in detail; in section 5 we summarize our contributions.
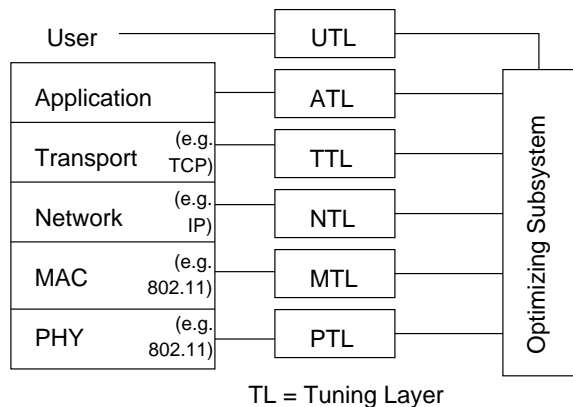

Figure 1: ECLAIR architecture

Figure 2: ECLAIR architecture details

2 ECLAIR Overview

2.1 Design Goals

From the software engineering perspective cross layer feedback is essentially a modification of the existing protocol stack. However, this would not be a one-time or ad hoc modification. As new wireless technologies are deployed, newer cross layer feedback ideas would be required. Multiple cross layer feedback algorithms would need to be easily integrated with the existing stack. In light of this, the following are the design goals for a cross layer architecture:

- **Rapid prototyping**: Enable easy development and deployment of new cross layer feedback algorithms, independent of existing stack.

- **Minimum intrusion**: Enable interfacing with existing stack without any significant changes in the existing stack.

- **Portability**: Enable easy porting to multiple systems.

- **Efficiency**: Enable efficient implementation of cross layer feedback.

In the next section we present an overview of ECLAIR and discuss how it achieves the stated design goals.
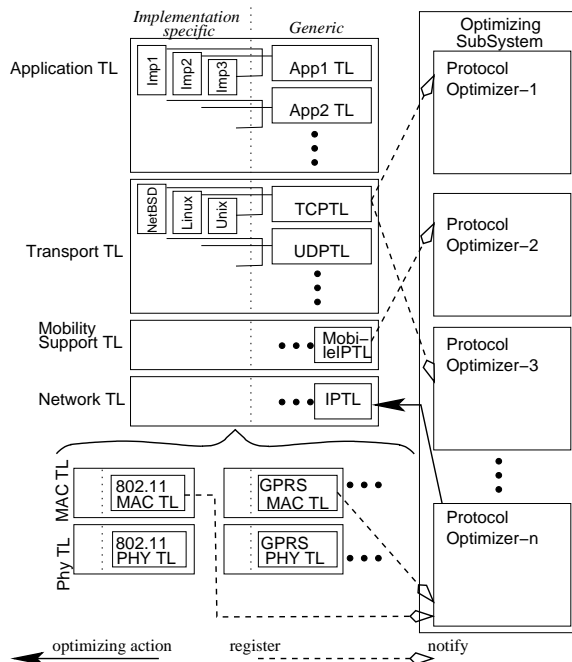
2.2 System Overview

The design goals discussed above indicate the need for separating the functionality of *cross layer feedback* into a separate system which would interact with the existing stack.

For enabling *rapid prototyping* of new cross layer feedback algorithms, ECLAIR splits the cross layer system into two subsystems – *Tuning Layers* and *Optimizing SubSystem*. Figure 1 shows the overview and 2 shows the details of ECLAIR.

- **Tuning Layers (TLs)**: The purpose of a tuning layer is to provide an interface to the internals of a protocol. E.g.: TCP tuning layer (TCPTL) is provided for TCP. Since the functionality for manipulating protocol data structures is built in to the TLs, no modification is required to the existing protocol stack. This facilitates incorporation of new cross layer feedback algorithms with *minimum intrusion*. Further, for the purpose of *portability* each TL is subdivided in to a generic and an implementation specific sublayer. For ease of reference we group the tuning layers according to their function. E.g.: Transport protocol tuning layers such as *TCPTL* for TCP, *UDPTL* for UDP, etc. are collectively referred to as *Transport Tuning Layer*.

- **Optimizing SubSystem (OSS)**: The optimizing sub-

system contains the algorithms and data structures for cross layer optimizations. It is the cross layer engine. The OSS contains many *Protocol Optimizers* (POs). A PO contains the algorithm for a given cross layer optimization. The OSS executes concurrently with the existing protocol stack and does not increase the stack processing overhead.

Besides meeting the design goals, ECLAIR provides additional benefits. In the next section we highlight the salient features of ECLAIR.

2.3   Salient Features

The design of ECLAIR enables easy deployment and control of cross layer feedback optimizations, enables interaction among multiple protocol stacks and facilitates *user feedback*. We present below the salient features of ECLAIR.

- *Event Notification*: The TLs provide the facility for POs to register for interesting events at a layer.

- *Switch on/off*: Since the cross layer system is separate, it can be easily/dynamically switched on or off. Also, individual POs may be switched on/off.

- *Seamless mobility*: ECLAIR can be used to enable seamless mobility on the mobile devices through POs that can monitor and control multiple protocol stacks.

- *User Tuning Layer*: Besides the layer specific TLs, ECLAIR also has a *User Tuning Layer* (UTL). UTL allows a device user or an external entity e.g.: a distributed algorithm or a base station, to tune the device behavior.

In the next section we present the details of ECLAIR.

3   ECLAIR Details

In the previous section we provided an overview of ECLAIR. In this section we discuss the internals of ECLAIR components.

3.1   Tuning Layers (TLs)

ECLAIR provides one TL for each protocol. For the ease of reference the *tuning layers* having similar function are grouped together. E.g.: TCP tuning layer, UDP tuning layer, etc. are collectively referred to as *Transport Tuning Layer*. For the purpose of portability, a TL is subdivided into a *generic tuning sublayer* and an *implementation dependent access sublayer*. The generic tuning sublayer provides an

implementation independent interface to a specific protocol. The implementation dependent sublayer provides implementation specific interfaces for a protocol. E.g.: Since the TCP implementations in Unix, NetBSD and Linux are different, there are separate components for each of the TCP implementations. The implementation specific layer has knowledge about the protocol implementation in a particular operating system and is used to manipulate or monitor the values in that protocol's data structures.

The TLs provide an interface to the POs for registering for events. For e.g. a PO can register for *handoff* events with the MobileIP TL. Multiple POs can register for the same event with a TL. The TL monitors the protocol for the events for which the POs have registered. When an event occurs, the TL *notifies* the registered POs.

For effecting a protocol optimization a PO would invoke the API provided by the generic sublayer of a protocol tuning layer (e.g. TCPTL). The generic sublayer would in-turn invoke the API of the implementation specific access sublayer (e.g. TCP on Linux). We present some detailed examples of POs in section 4.

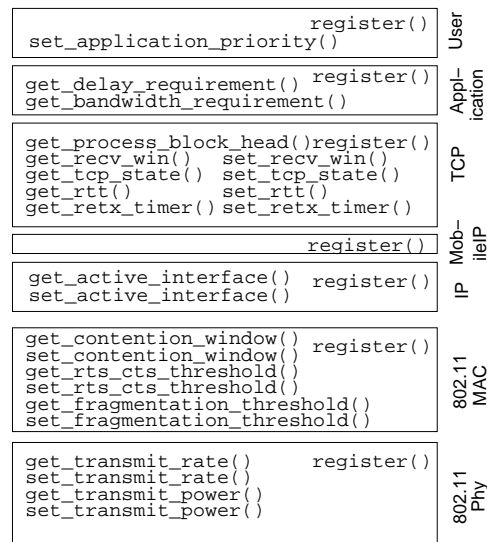So far we have explained the internal details about the TL. Next, we present some APIs of two TLs.

Figure 3: Tuning layer APIs: Examples

3.1.1   Tuning Layer APIs

MAC Tuning Layer: The MAC tuning layer or MTL provides interfaces for the various MACs. For e.g.: it provides an interface for 802.11, GPRS or CDMA MAC.

Due to paucity of space we only present some example generic APIs for 802.11 [3] MAC (see figure 3). *register* API is used for registering for an event at the 802.11 MAC. Other APIs are provided for reading and changing the *contention window*, *RTS/CTS threshold* and *fragmentation threshold*.

Transport Tuning Layer: The Transport tuning layer OR TTL provides interfaces for various transport protocols. We discuss TCPTL as an example. In a typical operating system, on system startup, TCP creates the *head* of its protocol data structure. See [13] for details on TCP/IP implementation in NetBSD. This information is accessed by TCPTL for subsequent manipulation of the data structure values. The API `get_protocol_block_head()` (see figure 3) is used to get the *head* of the TCP data structure. Other APIs are used for manipulating the *receiver window*, *retransmission timer*, etc.

Figure 3 shows some more sample *generic* APIs for some of the TLs.

3.2 Optimizing SubSystem (OSS)

The *protocol optimizers* (POs) in the OSS contain the cross layer feedback algorithms. The PO decides the *optimizing action* to be taken based on the *events* occurring at the various layers and the current *state* of the protocol layer which is to be modified (e.g.: TCP may be in *congestion avoidance* or *slow start* phase). The optimizing action modifies the target protocol's behavior.

As can be seen the POs interact with various layers for *events*, *state information* and *optimizing actions*. This interaction is done through the *tuning layers* (TLs). The POs *register* with the respective layer's TL for getting information about *events* at that layer.

Example PO: We present an example PO to illustrate the working of a PO. We describe a *Receiver Window Control PO* that apportions download link bandwidth among applications based on their *priority* numbers. The PO *registers* with the User TL for the *event* of user changing the application priorities. Whenever this event occurs, the PO is *notified*. The PO's algorithm maps the new priority value to a new receiver window value for the application's TCP connection. This in turn changes the application's bandwidth share on the download link. The PO invokes the following TCPTL APIs (see figure 3) *get_recv_win()* and *set_recv_win()*. This example and two others, about network layer feedback to TCP and seamless mobility, are discussed in detail the next section.
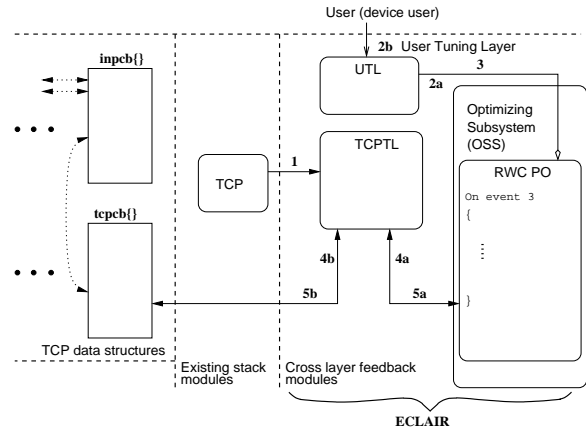

Figure 4: ECLAIR architecture: User feedback

4 Optimizing SubSystem Operation

In this section we show three examples which use ECLAIR. The first two examples are about cross layer feedback to TCP from the *user* and network layer (MobileIP). The last example describes how ECLAIR supports seamless mobility.

4.1 User feedback

Users can provide useful feedback to improve the performance of the stack or the *user experience*[9, 8]. One example is when a user may want to control the throughput of the running applications. E.g.: a user may want one file download to get more bandwidth than another.

Algorithm: One method of controlling the application's bandwidth share is through manipulation of the *receiver window* of its TCP connection [9]. The user assigns some priority number to each application. The sum of the receiver windows of all applications R is assumed to be set according to the available bandwidth. An application's priority number p is used to calculate its receiver window, $r = R \times p / \sum p$ [9].

Implementation: The use of ECLAIR for the above PO (*Receiver Window Control PO* or *RWC PO*) is shown in figure 4.

The explanation of the sequence shown in figure 4 is as follows: **(1)** TCPTL gets *protocol block head* information at system start. **(2a),(2b)**PO registers for *user* events. User changes priorities for running applications. **(3)** Application and respective priority information is passed to the RWC PO. **(4a),(4b)** Current receiver window/buffer information is collected via TCPTL. This information is used to recalculate

the new receiver window values for the various applications. It is assumed that the application can be identified by the sockets. **(5a),(5b)** The receiver window values are set for each application.
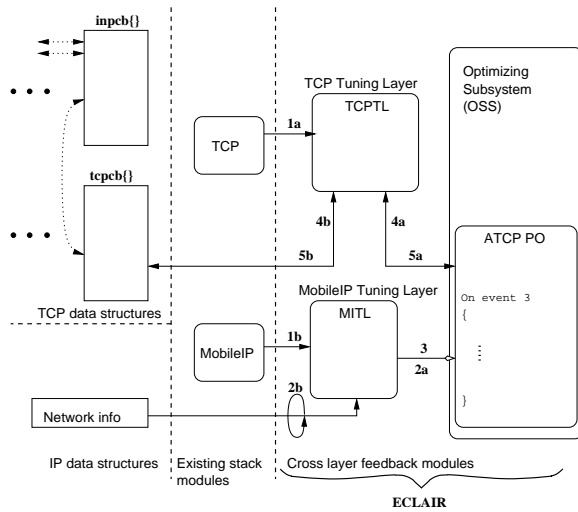

Figure 5: ECLAIR architecture: Adapted TCP

4.2   Adapted TCP

TCP is known to perform poorly when there are disconnections since it misinterprets disconnections as network congestion and decreases its sending rate [14]. TCP performance can be improved through feedback from other layers.

Algorithm: Feedback from the network layer about disconnections can be used to adapt TCP behavior. A number of such improvements are proposed in [10]. One of the improvements suggested is: if disconnection occurs and if the *congestion window*(*cwnd*) is open, then cancel TCP retransmission timer and do not reduce the *cwnd*. On reconnection, use the earlier *cwnd* and set a new retransmission timer. This action results in improved TCP throughput since unnecessary reduction in *cwnd* is avoided.

Implementation: The use of ECLAIR for the above PO (*Adapted TCP PO* or *ATCP PO*) is shown in figure 5.

The explanation of the sequence shown in figure 5 is as follows: **(1a),(1b)** The data structure location information is given to the respective TLs at system startup. **(2a),(2b)**PO registers for disconnection(reconnection) event. MITL monitors the network status for disconnection(reconnection). **(3)** Disconnection(reconnection) occurs and ATCP PO is notified. **(4a),(4b)** Current state of TCP is queried via TCPTL and the action is determined (e.g. changing the value of the

TCP retransmission timer). **(5a),(5b)** The TCP retransmission timer is set to the new value determined in step 4.

4.3   Seamless Mobility PO

Seamless mobility means the continuation of a session on a mobile device even as it roams across networks provided by heterogeneous wireless technologies. The key requirement is that the session should continue uninterrupted. We discuss an example PO for seamless mobility between GPRS and WLAN networks.

Algorithm: For achieving seamless mobility the MACs of the GPRS and 802.11 interfaces are monitored for *vertical* handoffs. When the network changes the corresponding wireless interface is made active in the IP layer.

Implementation: We call this the *Seamless PO* or *SPO*. The PO may register with the MAC TLs of GPRS and 802.11. The SPO thus gets information about *vertical* handoffs. When a handoff occurs , the SPO changes the *active* interface at the IP layer (e.g.: if the mobile moves from a 802.11 WLAN to a GPRS network, the GPRS interface will be made active). For this the SPO accesses the data structures in IP using the IP tuning layer. A PO of this type is shown in figure 2.

Other SPOs for handoff across other types of networks can be along the above lines. Details are beyond the scope of this paper.

5   Summary

In this paper, we presented ECLAIR an architecture for cross layer feedback. ECLAIR separates the cross layer system from the existing protocol stack. ECLAIR further subdivides the cross layer system into components. The optimization algorithms form the *optimizing subsystem*. The functionality for manipulating and accessing the existing protocol stack is built into the *tuning layers*. The tuning layers are further subdivided into a generic and implementation specific part to support portability. Their is no processing overhead on the existing stack since the optimizing subsystem executes in parallel to the protocol stack. ECLAIR provides a structured approach to cross layer feedback and enables *rapid deployment* of new cross layer feedback algorithms. Eventually, as these cross layer algorithms are widely accepted they can be incorporated into the next version of the stack.

Our future research shall focus on validating and further refining ECLAIR.

References

[1] R. Cáceres and L. Iftode. Improving the Performance of Reliable Transport Protocols in Mobile Computing Environments. *IEEE JSAC*, 13(5):850–857, June 1995.

[2] J.-P. Ebert and A. Wolisz. Combined Tuning of RF power and Medium Access Control for WLANs. *Mobile Networks and Applications*, 6(5):417–426, 2001.

[3] IEEE Std 802.11-1997. Wireless LAN Medium Access Control (MAC) And Physical Layer (PHY) Specifications, 18 Nov. 1997.

[4] ITU. Information technology - OSI - Basic Reference Model, July 1994. X.200.

[5] A. Jamalipour and S. Tekinay, editors. *Fourth Generation Wireless Networks and Interconnecting Standards*, volume 8 of *IEEE Personal Communications*. Oct. 2001.

[6] M. Methfessel, K. F. Dombrowski, P. Langendörfer, H. Frankenfeldt, I. Babanskaja, I. Matthaei, and R. Kraemer. Vertical Optimization of Data Transmission for Mobile Wireless Terminals. *IEEE Wireless Communications*, 9(6):36–43, 2002.

[7] V. T. Raisinghani and S. Iyer. Cross-layer Design Optimizations in Wireless Protocol Stacks. *Computer Communications (Elsevier)*, 2003.

[8] V. T. Raisinghani and S. Iyer. User Managed Wireless Protocol Stacks. 23rd *ICDCS*, Providence, RI, USA, May 2003. Poster.

[9] V. T. Raisinghani, A. K. Singh, and S. Iyer. Improving TCP Performance over Mobile Wireless Environments using Cross Layer Feedback. In *IEEE ICPWC*, New Delhi, India, Dec. 2002.

[10] A. K. Singh and S. Iyer. ATCP: Improving TCP Performance over Mobile Wireless Environments. In *IEEE MWCN*, Stockholm, Sweden, Sept. 2002.

[11] W. R. Stevens. *TCP/IP Illustrated, Volume I, The Protocols*. AWL, 1994.

[12] UMTS Forum. Glossary. http://www.umts-forum.org/glossary.asp, 2003.

[13] G. R. Wright and W. R. Stevens. *TCP/IP Illustrated, Volume II, The Imlpementation*. AWL, 1995.

[14] G. Xylomenos and G. C. Polyzos. Internet Protocol Performance over Networks with Wireless Links. *IEEE Network*, 13(4):55 – 63, July/Aug. 1999.

[15] G. Xylomenos and G. C. Polyzos. Quality of Service Support over Multi-Service Wireless Internet Links. *Computer Networks*, 37(5):601–615, 2001.