

# Cross-Layer Feedback Architecture for Mobile Device Protocol Stacks

Vijay T. Raisinghani and Sridhar Iyer, Indian Institute of Technology Bombay

## ABSTRACT

Applications using traditional protocol stacks (e.g., TCP/IP) from wired networks do not function efficiently in mobile wireless environments. This is primarily due to the layered architecture and implementation of protocol stacks. One mechanism to improve the efficiency of the stack is cross-layer feedback, that is, making information from within one layer available to another layer of the stack. For example, TCP retransmissions can be reduced by making it aware of network disconnections or handoff events. We highlight the need for a cross-layer feedback architecture and identify key design goals for an architecture. We present our ECLAIR architecture, which satisfies these design goals. We describe a prototype implementation that validates ECLAIR. We also discuss other cross-layer architectures and provide a cross-layer design guide.

## INTRODUCTION

To ensure interoperability with the existing Internet, standard protocol stacks (e.g., Transmission Control Protocol (TCP)/Internet Protocol (IP) [1]) are being deployed, even in mobile wireless setups, that is, on the mobile devices and intermediate nodes in the wireless network. However, these standard protocol stacks function inefficiently in mobile wireless environments [2]. This is primarily due to the layered *architecture* and *implementation* of protocol stacks. We highlight the inefficiencies of *layered* protocol stacks by using TCP as an example.

TCP is an end-to-end reliable transport protocol. TCP at the sender uses acknowledgments from the receiver as a signal to send additional packets. A missing acknowledgment is interpreted as an indication of packet loss due to congestion in the network. However, in mobile wireless environments packet losses are also caused by poor wireless channel conditions and disconnections. Since TCP is unaware of these channel conditions, it invokes its standard algorithm and reduces its throughput. It can be seen that TCP throughput can be improved by making it aware of the wireless channel conditions. For example, the retransmissions could be deferred until the channel conditions improve. There are various methods of improving TCP performance, which

entail modifications at the end station(s) or base station/router. We refer interested readers to [2] for an introduction to TCP algorithms, problems related to TCP in wireless environments, and the various solutions proposed in literature.

In wireless environments, the performance of other layers as well can be improved by enabling cross-layer feedback [3]. The feedback could be from layers above or below a layer (discussed further below).

As new wireless networks are deployed to enhance the performance of the existing protocol stacks, various cross-layer feedback optimizations would be required. These cross-layer optimizations would require easy integration with the existing stack. If the cross-layer optimizations were implemented in an *ad hoc* manner, it would lead to:

- Decreased execution efficiency of the stack
- Difficulty in ensuring protocol correctness of the protocols modified using cross-layer feedback
- Difficulty in maintaining the cross-layer optimizations

Thus, to help standardize and ease the development, deployment, and maintenance of the various cross-layer optimizations, appropriate architecture is essential. Existing approaches to cross-layer feedback do not satisfy all of these requirements.

Cross-layer feedback optimizations may need to be implemented at the intermediate nodes (base station or router) or mobile host (MH). We focus on cross-layer feedback in the MH since we believe that it would be easier to implement changes on the end-devices than in the network.

Our architecture ECLAIR [4] provides a guideline for designing and implementing cross-layer feedback in an easy and efficient manner on a mobile device. ECLAIR exploits the fact that protocol behavior is determined by the values stored in the protocol's data structures. In ECLAIR, a *tuning layer* (TL) for each layer, provides an interface to read and update these protocol data-structures. TLs are used by *protocol optimizers* (POs), which contain cross-layer feedback algorithms. The POs form the *optimizing subsystem* (OSS).

We briefly explain ECLAIR's prototype implementation and validation. We then provide guidelines for architecture selection and

An ad hoc approach could be used to implement cross-layer feedback, that is, blocks of code could be introduced in the existing layers to enable cross-layer feedback. For example, to enable TCP to receive hand-off information from the MAC layer, additional code would be introduced in the TCP and MAC layers.

ECLAIR deployment. Finally, we summarize and conclude the article.

## CROSS-LAYER FEEDBACK BACKGROUND

Cross-layer feedback means enabling interaction of a layer with any other layer in the protocol stack. A layer may interact with layers above or below it.

A survey of various cross-layer feedback proposals is presented in [3]. We list a few examples of cross-layer feedback for each layer as follows.

• **Physical:** Channel condition (e.g., bit error rate) status from the physical layer can be used by the link layer to adapt its error-control mechanisms. Also, the physical-layer transmit power can be tuned by the medium access control (MAC) layer to increase the range of transmission.

• **Link/MAC layer:** The number of retransmissions at the link layer can serve as a measure of channel condition. TCP may re-estimate its retransmission timers based on this data. The link layer may adapt its error correction mechanism based on the quality-of-service (QoS) requirements of the application layer, that is, acceptable delay, packet losses, and so forth.

• **Network:** Mobile-IP hand-off begin/end information can be used at TCP to manipulate its retransmission timer. The Mobile-IP layer can use link layer hand-off events to reduce Mobile-IP hand-off latency.

• **Transport:** Packet loss date at TCP can help the application layer adapt its sending rate. The link layer can adapt its error control mechanisms based on TCP retransmission timer information.

• **Application:** An application can use information about channel conditions from the physical layer to adapt its sending rate. Also, an application can indicate to the user the throughput it requires and the current throughput.

• **User:** A user may give the system an indication of impending disconnection. This information may be used by TCP to freeze its retransmission timer.

As can be seen from these examples, a large number of cross-layer interactions are possible in the stack. Hence, a systematic approach to cross-layer feedback is essential. Next, we discuss the implementation aspects of cross-layer feedback and motivate the need for a cross-layer feedback architecture.

### THE NEED FOR A CROSS-LAYER FEEDBACK ARCHITECTURE

From the software engineering perspective, cross-layer feedback is essentially a modification to the existing protocol stack. Since the stack forms an important part of the operating system, it is important that any modification to the stack:

- Imposes minimal overhead on the stack
- Does not introduce any errors in the stack
- Is easily extensible and reversible if required

**Cross-Layer Feedback: Ad Hoc Approach** — An ad hoc approach could be used to implement cross-layer feedback, that is, blocks of code

could be introduced in the existing layers to enable cross-layer feedback. For example, to enable TCP to receive hand-off information from the MAC layer, additional code would be introduced in the TCP and MAC layers. This additional code in TCP would query the MAC layer and determine the TCP adaptation, while the additional code in the MAC layer would provide an interface for querying the MAC layer's internal state.

We note that an ad hoc approach to cross-layer feedback has the following problems:

- Each additional cross-layer feedback code block would slow down the execution of a layer (e.g., TCP) and thus reduce the throughput of that layer. If a layer interacts with many other layers, this would lead to a large reduction in its throughput.
- The cross-layer feedback code will have to be rewritten for porting to other operating systems.
- Multiple cross-layer optimizations within a layer could lead to conflicts [5] and hence difficulty in ensuring correctness of the layer's algorithms.
- Once added to a layer, cross-layer feedback code would be difficult to update or remove, since the code would be intertwined with regular-layer code.
- Trial (fast prototyping) of new cross-layer feedback ideas would not be easy, since the layer code would need to be modified.

The above problems of an ad hoc approach highlight the need for an architecture for cross-layer feedback. From the above, we can derive the design goals of a cross-layer feedback architecture.

### Design Goals for a Cross-Layer Feedback Architecture

• **Rapid prototyping**, which would enable easy development and deployment of new cross-layer feedback optimizations, independent of existing stack.

• **Minimum intrusion**, which would enable interfacing with existing stack without *significant* changes in the existing stack. Here significant changes means too many or large code modifications to the layer(s). This would aid in maintainability, that is, easy extension or reversal of the cross-layer optimization and protecting the correctness of the stack, with minimal efforts.

• **Portability**, which would enable easy porting to different systems.

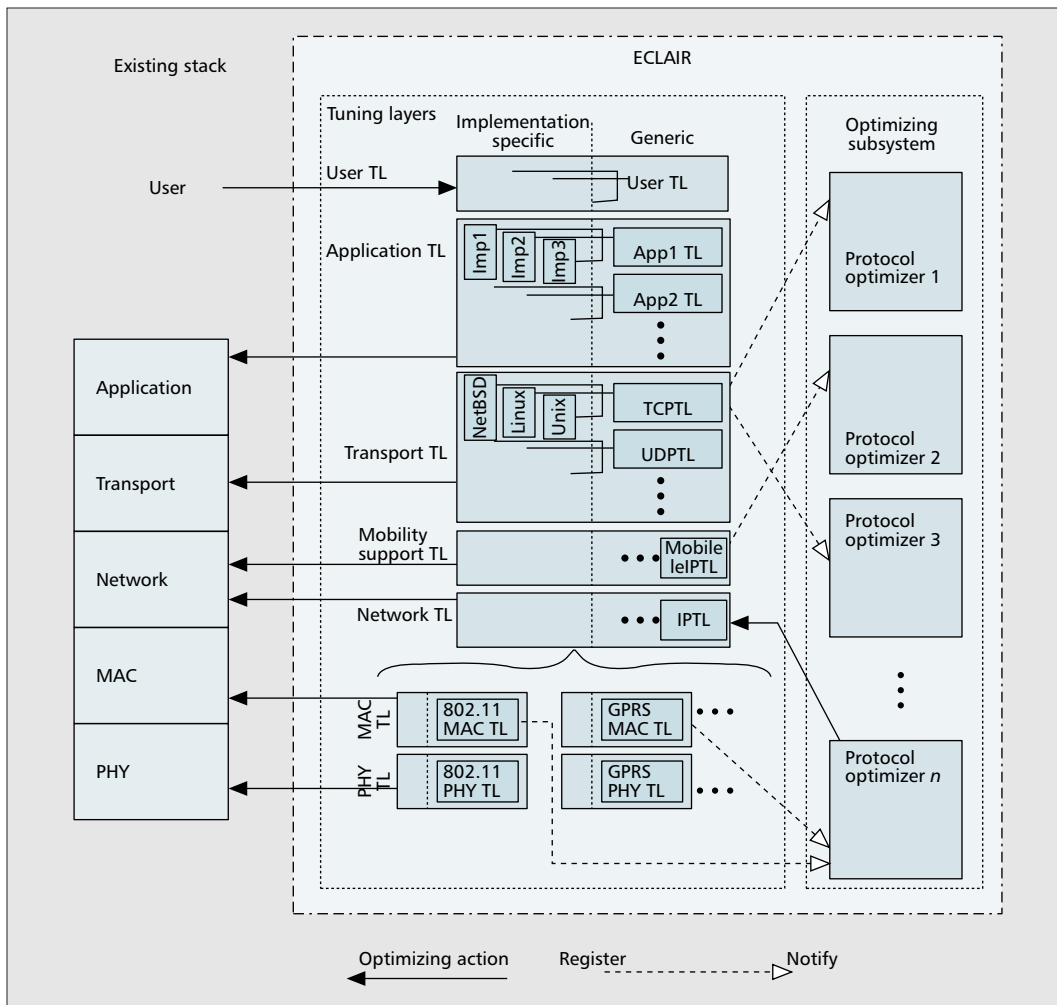
• **Efficiency**, which would enable *efficient* (minimum execution overhead) implementation of cross-layer feedback.

The above goals motivated our study of the need for a cross-layer feedback architecture. In the next section, we discuss existing approaches to cross-layer feedback.

## RELATED WORK

We discuss various architectures for cross-layer feedback within the mobile device.

One of the early proposals is the Physical Media Independence (PMI) [6] architecture. In PMI, cross-layer feedback is achieved through *guard modules* and *adaptation modules*. PMI is



MobileMan recommends replacing the standard protocol layer with a redesigned network-status-oriented protocol, so that the protocol can interact with Network Status. MobileMan has been deployed on experimental testbeds for ad hoc networks.

■ Figure 1. ECLAIR: cross-layer feedback architecture.

aimed at monitoring the network interface *availability*. *Guard modules* monitor interface characteristics such as *connected*, *powered*, and so forth. *Adaptation modules* attached to each layer of the network stack receive *policy-related* information from higher-layer modules, and event indications from lower-layer modules. The adaptation modules adapt the respective layers using the operating system utilities. The information about interface events propagates layer by layer. For example, if the MAC layer receives an event for adaptation, it would adapt its behavior first and then propagate the information to the next higher layer.

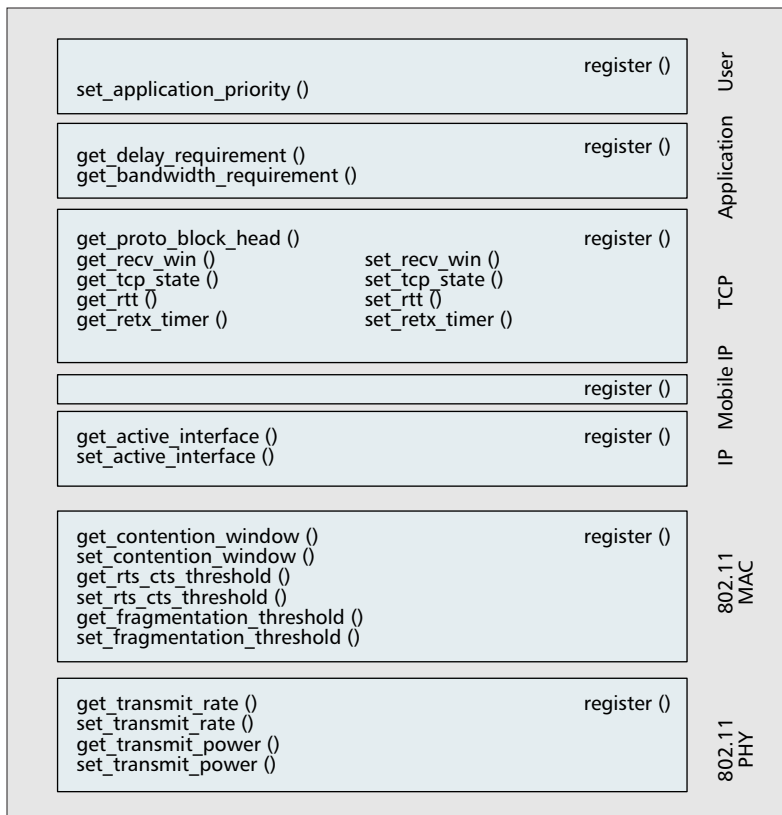
An architecture that is focused on the network environment was proposed in [7]. In this architecture, cross-layer feedback is achieved through Internet Control Message Protocol (ICMP) messages. The physical/MAC layers, network layer, and application layer/user monitor the network for events such as bandwidth change, hand-off, and so on. When an event occurs, the information is propagated to the upper layers through ICMP messages (we refer to this as the *ICMP-arch*). These ICMP messages are generated by some module running on the system and contain all the event-related information. A special handler at the socket layer traps these messages, adapts protocols, and

also propagates the information to the applications. The applications register for events using the Application Programming Interface (API) provided. The protocol adaptations are defined by the application developer using the API provided.

Cross-layer information can also be exchanged through an Interlayer Signaling Pipe (ISP) [8], that is, through packet headers. This is suitable for cases where some adaptation may be required at lower layers for each packet from higher layers. However, this requires that lower layers be able to read higher-layer headers. This necessitates modification to the layer code where adaptation is required. For cross-layer feedback from lower to higher layers, the lower layers would need to change the packet header, which could lead to packet errors.

Cross-Layer Signaling Shortcuts (CLASS) was proposed in [9]. CLASS allows direct interaction between the layers. For example, the application layer can directly interact with the link layer. However, CLASS has drawbacks similar to that of an ad hoc approach.

MobileMan [10] adds another stack component called *Network Status*. This component is a repository provided for network information sharing among the layers. The access to Network Status is standardized. MobileMan recommends



■ Figure 2. Generic tuning sublayer: example APIs.

replacing the *standard* protocol layer with a redesigned *network-status-oriented* protocol, so that the protocol can interact with Network Status. MobileMan has been deployed on experimental testbeds for ad hoc networks.

The framework in [11] proposes a *cross-layer manager*. The protocol layers expose *events* and *state variables* to the cross-layer manager. *Management algorithms* are woken up by the events. The cross-layer manager uses the state variables to query/set the protocol internal state. Four interlayer coordination *planes* are identified, namely, security, quality of service, mobility, and wireless link adaptation. Internal details of this framework are not available.

The above examples of cross-layer feedback focus on improvements *within* the protocol stack. The Global Resource Adaptation through Cooperation (GRACE) framework [12] is aimed at cross-layer adaptation across the hardware, software (OS), and application layers. However, GRACE does not address adaptation of any the protocol stack layers.

The cross-layer architectures proposed in the literature that focus on cross-layer interaction within the stack on the mobile device [6–11] do not address the all the design goals identified above. These architectures do not fully address the goals of rapid prototyping, maintainability, portability, and efficiency. In MobileMan [10], it is recommended that the protocol layer be replaced by a redesigned protocol. This would lead to increased implementation and maintenance efforts. Further, the layers may need to be changed in case the Network Status component is enhanced. Efficiency would be lower in archi-

tectures such as ICMP-arch [7] since the information is wrapped in ICMP messages, which increases the event communication overheads. In PMI [6] as well the event information propagates layer by layer, which would decrease the cross-layer execution speed. In ISP [8] the overhead of scanning each packet and adaptation would slow down the execution of the lower layers and thus reduce throughput. Further, there is no provision for any-to-any layer-event communication in either PMI [6], ICMP-arch [7], or ISP [8].

We refer interested readers to [5] for useful caveats and principles related to cross-layer feedback design and to [3] for a survey of cross-layer feedback optimizations.

Thus far we have presented the existing approaches to cross-layer feedback. As discussed, these approaches do not fully address the design goals identified in the previous section. In the next section we present our proposal for a cross-layer feedback architecture — ECLAIR — which is based on the design goals presented above.

## ECLAIR DESIGN

For enabling *rapid prototyping* of new cross-layer feedback optimizations, ECLAIR is split into two subsystems: TLs and OSS. Figure 1 shows the details of ECLAIR.

**Tuning Layers** — The purpose of a TL is to provide an interface to protocol data structures that determine the protocol’s behavior. For example, a TCP tuning layer (TCPTL) is provided for TCP.

For ease of reference, we group the TLs according to their function. For example, Transport TL refers to the collection of transport protocol TLs such as TCPTL for TCP, UDPTL for UDP, and so forth.

A TL can read and update the protocol data-structures. A protocol implementation typically has data-structures for *control* and *data*. A protocol’s behavior is determined by its control data-structures. For example, in Linux, TCP control information is stored in a data structure `struct tcp_opt` embedded within the socket data structure `struct sock`. The interested reader can refer to standard texts on Linux TCP/IP internals for details.

For the purpose of portability, a TL is subdivided into a *generic tuning sublayer* and an *implementation-specific sublayer* [4] (Fig. 1).

**Optimizing Subsystem** — The OSS contains the algorithms and data-structures for cross-layer optimizations. The OSS contains many *protocol optimizers* (POs). A PO contains the algorithm for a particular cross-layer optimization. For an *optimizing action* (Fig. 1; solid line, solid arrowhead), a PO invokes a function in the TL, using the TL’s API. The PO (or POs) *registers* for *events* with TLs, using the register API (Fig. 1; dashed line, hollow arrowhead). The TLs *notify* the registered POs whenever an event occurs. The PO also uses TL APIs for querying the current *state* of the protocol layer which is to be modified (e.g., the TCP’s state could be *congestion avoidance* or *slow start* phase).



The OSS executes concurrently with the existing protocol stack and does not increase the stack-processing overhead.

Some example APIs of the generic tuning sublayer are presented in Fig. 2, where MAC and Physical TL APIs for the 802.11 Wireless LAN standard are shown.

Besides meeting the design goals highlighted earlier, ECLAIR provides additional benefits. Since the cross-layer system is separate, it can be easily/dynamically enabled or disabled. Also, individual POs may be enabled or disabled. Besides the layer-specific TLs, ECLAIR also has a User TL (UTL). UTL allows a device user or an external entity (e.g., a distributed algorithm or a base station) to tune the device behavior. Lastly, ECLAIR allows any-to-any layer communication through the POs.

In the next section, we present a prototype implementation of ECLAIR. The prototype is based on user feedback to TCP [13, 14]. User feedback has also been proposed by other researchers in different contexts. However, we restrict the focus of this article to the architectural aspects of cross-layer feedback.

## ECLAIR IMPLEMENTATION: USER FEEDBACK

Users can provide useful feedback for improving the performance of the stack or the *user experience* [13, 14]. One example is when a user may want to control the throughput of applications running on the device. For example, a user may want one file download to get more bandwidth than another.

One method of controlling the application's bandwidth share is through manipulation of the *receiver window* of its TCP connection [13, 14]. TCP uses congestion- and flow-control mechanisms to avoid swamping the network or the receiver [1]. The receiver reflects its receive buffer status by the *advertised window* field in the acknowledgments to the sender. When the network losses are *low*, the send rate of a TCP sender is determined by this advertised window. This property can be exploited to intentionally restrict the throughput of some applications on the mobile device. This would lead to increased throughput for the rest of the applications.

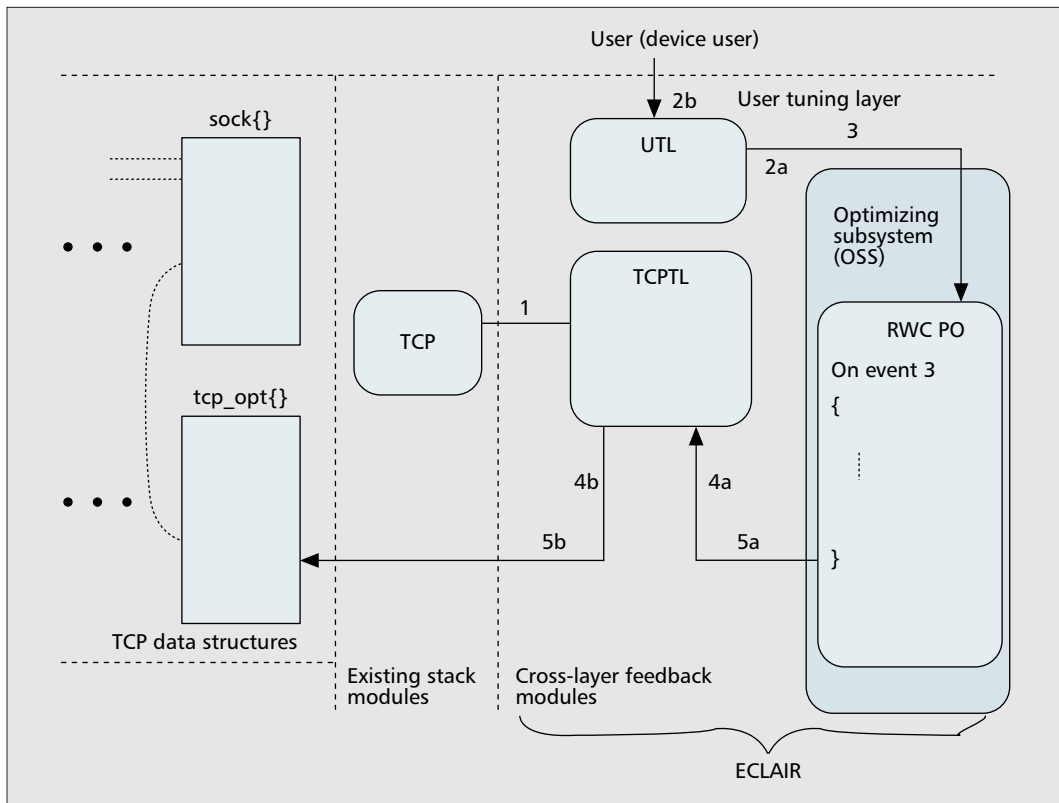
**Algorithm:** The user assigns some priority number to each application. An application's priority number is used to calculate its receiver window.

**Implementation:** The use of ECLAIR for the above PO, the *receiver window control* PO (RWC PO), is shown in Fig. 3.

The explanation of the sequence shown in Fig. 3 is as follows: (1) TCPTL reads data-structure location information at system start; (2a), (2b) the PO registers for user events and the 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 re-calculate the new receiver window values for the various applications, and it is assumed that the application can be identified by the sockets); (5a), (5b) the receiver window values are set for each application.

In Fig. 3, the dotted lines from *sock* repre-

Users can provide useful feedback for improving the performance of the stack or the user experience. One example is when a user may want to control the throughput of applications running on the device. For example, a user may want one file download to get more bandwidth than another.



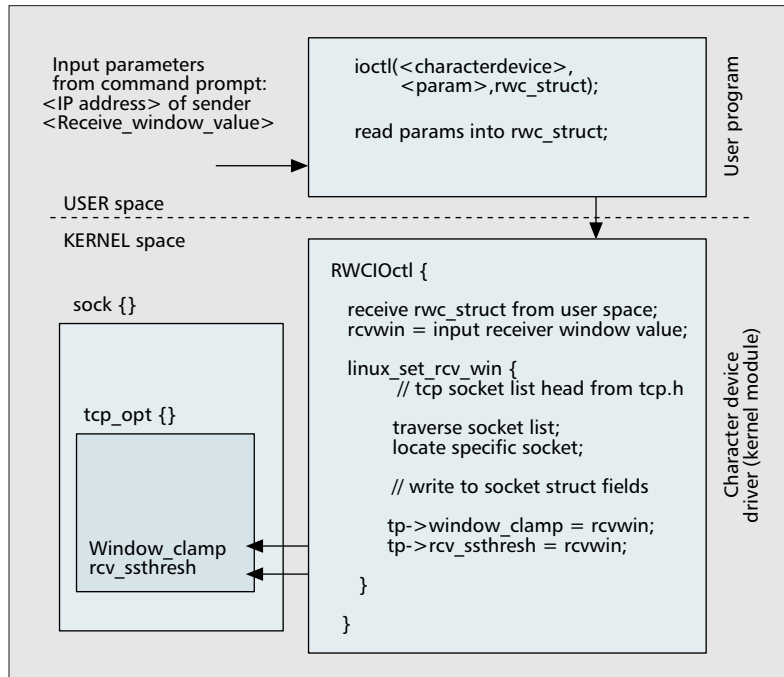
■ Figure 3. ECLAIR architecture: receiver window control.

sent the memory references from `sock` to other data structures.

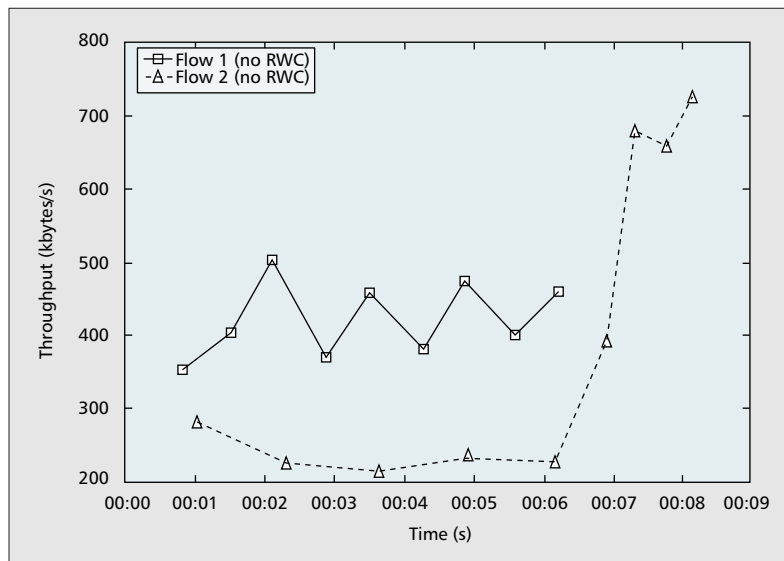
We refer the interested reader to [4] for the design of Mobile-IP and TCP interaction, using ECLAIR.

Next, we present the implementation details of user feedback (RWC) based on the architecture presented above.

**User Feedback: Implementation Details** — We chose Linux for the implementation since its source code is freely available and modifiable. The relevant TCP data-structures are in the header file `sock.h`. `tcp_opt` is TCP's control data-structure. `sock` is the *socket* data-structure. `window_clamp` and `rcv_ssthresh` are used



■ Figure 4. Call flow: RWC using ECLAIR.



■ Figure 5. Receiver window control experiments over WLAN (802.11): no RWC.

for controlling the advertised window in TCP. Figure 4 shows the call flow of the RWC prototype implementation using ECLAIR. Our current implementation largely has TL functionality only. In this prototype the RWC *calculation* is performed by the user. The parameters are passed to the TL to change the control parameters (receiver window) in the socket. The *IP address* parameter is used to identify the application's TCP socket within which the receiver window value is to be changed.

The RWC PO and TL are coded in a single kernel-loadable module. No modification was required to the existing TCP layer code. Interested readers can refer to standard texts about Linux device drivers for details about writing Linux kernel modules.

Using the above implementation, on a Linux desktop, we conducted experiments over our department's wireless LAN. The desktop was connected to our department LAN using 802.11 wireless LAN equipment. We started two *http* file-transfer sessions from the desktop to two Web servers on our department LAN. The desktop was the receiver. Figure 5 shows the result when RWC was not invoked. The flow that starts first (flow 1) gets most of the bandwidth. In another set of experiments, during the transfer, RWC was invoked on the desktop in order to reduce the receiver window of flow 1. The resulting graph (Fig. 6) shows the decrease in throughput of the session (flow 1) controlled by RWC and an increase in throughput of the other session (flow 2). These experiments validate our ECLAIR implementation.

In this section, we discussed ECLAIR design, its prototype implementation, and validation. In the next section we present a cross-layer feedback design guide.

## CROSS-LAYER FEEDBACK DESIGN GUIDE

### ARCHITECTURE SELECTION

To ensure correctness and efficiency, one of the primary criteria for selecting a cross-layer feedback architecture is the *type* of adaptation. The adaptation can be *synchronous* or *asynchronous*. In synchronous adaptation, whenever a layer receives some cross-layer feedback, it proceeds with its *regular* execution only after executing the adaptation required. For example, assume that a *network disconnection* event is detected and TCP adaptation is required. In the synchronous case, TCP's regular execution would proceed only after the required adaptation is completed. In the asynchronous case, the control data-structures of TCP would be updated so as to effect a change in TCP behavior while TCP execution is in progress.

ECLAIR is suited for asynchronous adaptation, since it is separate from the existing stack.

Cross-layer feedback correctness would be affected if an architecture suitable for asynchronous adaptation were used for synchronous adaptation. For example, cross-layer feedback adaptation, which is to be triggered by information contained in each packet, would fail if an asynchronous architecture like ECLAIR were

used. Furthermore, the efficiency would be reduced if a synchronous architecture were used for an adaptation, which can be done asynchronously.

To highlight the impact on efficiency, we consider RWC, as explained above. In this case, the primary requirement is to apportion application bandwidth, which can be done through asynchronous adaptation. It may not be essential to tune application bandwidth *synchronously*. In the implementation proposed in [14], each `read()` of the application invokes RWC algorithm, that is, the adaptation is synchronous with application execution. This would reduce the application execution speed hence throughput. If an asynchronous architecture (e.g., ECLAIR) is used, the application speed would not be reduced.

Subsequent to the architecture choice based on the type of cross-layer feedback, it is essential to minimize the overheads of the cross-layer feedback implementation. In the following subsections we discuss the design guide for ECLAIR implementations for single and multiple PO cases.

### ECLAIR USAGE

**Single Cross-Layer Optimization** — Separating POs and TLs into a separate cross-layer system, outside the stack, introduces the overhead of additional function calls. Hence, in case only a single cross-layer optimization is planned and the cross-layer system is not to be ported/deployed on multiple operating systems, then it would be better to modify the layer code. This would reduce the overhead of multiple function calls between the PO and the TL and hence increase the efficiency of the implementation.

**Multiple Cross-Layer Optimizations** — In case of multiple cross-layer optimizations, POs and TLs should be implemented as specified in the ECLAIR architecture.

If multiple cross-layer optimizations or POs directly access the layers, then there is high dependency of the POs on the layer's code. Any change to the layer code will lead to a change in all the POs interacting with that layer. This would lead to the maintainability and portability issues highlighted above. Reducing such *dependence* is useful for easy maintainability of the cross-layer system. Introduction of a TL leads to reduction in the dependency between the layer code and POs. While a TL is *dependent* on the layer code, the impact of a change to the stack is reduced and localized to the TL's implementation specific sublayer. Multiple protocol optimization modules that use the TL need not be changed, since the generic tuning sublayer interface remains unchanged. Similarly, when the cross-layer system needs to be ported to another operating system, only the implementation-specific sublayer needs to be changed.

In summary, ECLAIR should be used if the cross-layer type is asynchronous. Furthermore, POs and TLs should be implemented, as proposed in ECLAIR, if multiple cross-layer optimizations are to be implemented or if the cross-layer system is to be ported to multiple operating systems.

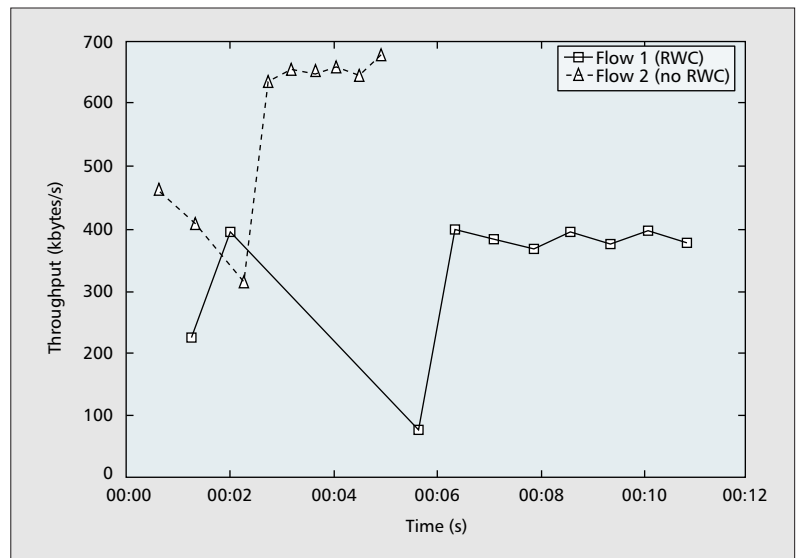


Figure 6. Receiver window control experiments over WLAN (802.11): RWC.

## CONCLUSION

Layered protocol stacks are inefficient when deployed in wireless networks. Hence, cross-layer feedback is essential. However, ad hoc cross-layer feedback implementations lead to problems related to easy development/deployment, maintainability, portability, and execution efficiency. Thus, an appropriate architecture for cross-layer feedback is essential.

The key design goals for a cross-layer feedback architecture are *rapid prototyping*, *minimum intrusion*, *portability*, and *efficiency*. Our architecture ECLAIR satisfies these design goals by splitting the cross-layer system into TLs and an OSS. Our prototype implementation of user feedback (RWC) validated ECLAIR.

Cross-layer feedback can be asynchronous or synchronous. For ensuring the correctness and efficiency of cross-layer feedback, the right architecture needs to be selected. ECLAIR is suitable for asynchronous cross-layer feedback. ECLAIR POs and TLs would introduce some overheads; however, POs and TLs are useful for cross-layer feedback design, implementation, and evolution.

ECLAIR and various other architectures proposed in the literature do not solve *all* the issues related to cross-layer feedback. For example, one of the important issues is cross-layer feedback conflict [5]. ECLAIR provides components that can be used for implementing conflict-resolution mechanisms.

## REFERENCES

- [1] W. R. Stevens, *TCP/IP Illustrated, Volume I, The Protocols*, AWL, 1994.
- [2] Y. Tian, K. Xu, and N. Ansari, "TCP in Wireless Environments: Problems and Solutions," *IEEE Commun. Mag.*, vol. 43, no. 3, 2005, pp. S27–S32.
- [3] V. T. Raisinghani and S. Iyer, "Cross-Layer Design Optimizations in Wireless Protocol Stacks," *Comp. Commun.*, vol. 27, no. 8, 2004, pp. 720–24.
- [4] V. T. Raisinghani and S. Iyer, "ECLAIR: An Efficient Cross-Layer Architecture for Wireless Protocol Stacks," *World Wireless Cong.*, San Francisco, CA, May 2004.
- [5] V. Kawadia and P. R. Kumar, "A Cautionary Perspective on Cross-Layer Design," *IEEE Wireless Commun.*, vol. 12, no. 1, 2005, pp. 3–11.

- 
- [6] J. Inouye, J. Binkley, and J. Walpole, "Dynamic Network Reconfiguration Support for Mobile Computers," *ACM/IEEE Int'l. Conf. Mobile Comp. and Net.*, Budapest, Hungary, Sept. 1997.
- [7] P. Sudame and B. R. Badrinath, "On Providing Support for Protocol Adaptation in Mobile Networks," *Mobile Nets. and Apps.*, vol. 6, no. 1, 2001, pp. 43–55.
- [8] G. Wu *et al.*, "Interactions between TCP and RLP in Wireless Internet," *IEEE GLOBECOM*, Rio de Janeiro, Brazil, Dec. 1999, vol. 1B, pp. 661–66.
- [9] Q. Wang and M. A. Abu-Rgheff, "Cross-Layer Signaling for Next-Generation Wireless Systems," *WCNC*, vol. 2, Mar. 2003, pp. 1084–89.
- [10] M. Conti *et al.*, "Cross-Layering in Mobile Ad Hoc Network Design," *IEEE Comp.*, vol. 37, no. 2, 2004, pp. 48–51.
- [11] G. Carneiro, J. Ruela, and M. Ricardo, "Cross-Layer Design in 4G Wireless Terminals," *IEEE Wireless Commun.*, vol. 11, no. 2, 2004, pp. 7–13.
- [12] W. Yuan *et al.*, "Design and Evaluation of a Cross-Layer Adaptation Framework for Mobile Multimedia Systems," *SPIE/ACM Multimedia Comp. and Net. Conf.*, Santa Clara, CA, 2003.
- [13] V. T. Raisinghani, A. K. Singh, and S. Iyer, "Improving TCP Performance over Mobile Wireless Environments using Cross-Layer Feedback," *IEEE ICPWC*, New Delhi, India, Dec. 2002.
- [14] P. Mehra, A. Zakhor, and C. Vleeschouwer, "Receiver-Driven Bandwidth Sharing for TCP," *IEEE INFOCOM*, San Francisco, CA, Apr. 2003.

## BIOGRAPHIES

VIJAY T. RAISINGHANI (rvijay@it.iitb.ac.in) is a Ph.D. student at the School of Information Technology at Indian Institute of Technology (IIT) Bombay. His Ph.D. is sponsored by TATA Infotech Ltd, Mumbai, where he is working as an associate consultant. His research interests include cross-layer feedback and seamless mobility. He received his M.Tech. from the School of Information Technology at IIT Bombay. Additional information is available at <http://www.it.iitb.ac.in/~rvijay>

SRIDHAR IYER (sri@it.iitb.ac.in) is presently an associate professor in the School of Information Technology at IIT Bombay. Prior to this, he was a faculty member in the Department of Computer Science and Engineering at IIT Guwahati. His research interests include networking protocols and multimedia tools for distance education, wireless networking, mobile computing frameworks, and some areas of program/protocol verification. He received his B.Tech., M.Tech., and Ph.D. degrees from the Department of Computer Science and Engineering at IIT Bombay. Additional information can be found at <http://www.it.iitb.ac.in/~sri>