Chapter 2

CLAPDAWN: Cross Layer Architecture for Protocol Design in A Wireless Network

2.1 Introduction

In Chapter 1 we presented a survey of existing cross layer architectures and discussed their limitations. We have also identified the design objective for a cross layer architecture, namely: feedback loops free system, smaller development time, easy to recover from cross layer interactions, small footprint on referred architecture and support for multiple cross layer interactions. In this chapter we present our cross layer architecture CLAPDAWN, which is designed based on the above design goals.

2.2 CLAPDAWN Overview

As we saw in previous chapter that it is hard to manage multiple cross layer interactions in the system. Unmanageable cross layer interactions trigger the stability problem in the system. Literature covered in section 1.3.2 shows that available architectures are not addressing these problems at their depth. To overcome the limitation of existing
cross layer architectures and to provide rapid implementation of cross layer interactions, we have proposed the cross layer architecture CLAPDAWN: Cross Layer Architecture for Protocol Design in A Wireless Network. CLAPDAWN is designed to provide manageable and extendable cross layer interaction in the system.

In the section 1.3.1 we have discussed the three components of network protocol architecture namely primary communication part, primary control part and secondary control part. CLAPDAWN manages these three parts using two planes, Functional Plane (FP) and Control Plane (CP) as shown in figure 9.4. In CLAPDAWN, primary communication part and primary control part create the functional body of the cross layer architecture and we grouped them as Functional Plane (FP). Control Plane works over functional plane and governs the functionality of it to implement the cross layer interactions. In CLAPDAWN Control Plane (CP) manages all cross layer interactions taking place in the system.
2.3 Functional Plane (FP)

Functional Plane contains all the functionalities required to develop network protocol stack. As we discussed in our previous chapter that with cross layer interaction network protocol stack can be viewed as combination of three parts, primary control and communication parts and secondary control part. From these three parts, FP consists of two parts: primary communication part and primary control part. FP is network protocol stack without having any cross layer interactions in it.

FP communicates with the control plane through interfaces provided by control plane discussed in next subsection. Figure 9.4 shows the FP having four layers physical layer, data link layer, network layer and application layer. Normally only one active protocol resides at network, MAC and physical layer. But there are cases in which more than one protocols run actively at the given layer. For example, there may be more than one application existing at the application layer. To take care of it, FP contains the application interface that manages different applications running at application layer. Same can be designed for other layers having multiple protocols.

In literature, existing Cross Layer Architectures (CLAs) either only focus on layered architecture or completely forget the highly successful layered architecture. Revolutionary architecture completely ignores the layered architecture and evolutionary architecture tries to maintain the layered structure. In both the cases, the architecture looses valuable quality of single solution or layered structure. CLAPDAWN separates primary control and primary communication from secondary control. That provides additional flexibility and portability to the CLAPDAWN and expands the scope of it. Because of this separation any existing architecture can be used as reference architecture in CLAPDAWN.

To make the representation simple, we have shown the FP with four layers. In general, FP may represent TCP/IP protocol stack, or Open System Interconnect (OSI) seven layer stack or any other existing architecture. We call this architecture as reference or referred architecture. In general CLAPDAWN adopts existing architecture as FP and facilitates it for cross layer implementation.
2.4 Control Plane (CP)

Control plane supports the functional plane and governs the functionality of functional plane to implement Cross Layer Interactions (CLIs). Control plane manages all the cross layer interactions or secondary controls that take place in the system. Control plane divides complex functioning of cross layer interaction into following subcomponents: Control Network Interface (CNI), Layer Notification Interface (LNI), Cross Layer Engine (CLE), and Cross Layer Knowledge Base (CLKB). Figure 9.4 shows the organization of control plane in the CLAPDAWN. Cross layer engine and cross layer knowledge base implement the secondary control in control plane. Control plane gets its required information from functional plane using control network interface and provides the necessary updates to functional plane using layer notification interface. Control network interface and layer notification interface work as bridge between functional plane and control plane.

For simplicity here onwards we refer CLKB as knowledge base, CLE as execution engine, CNI as control interface and LNI as layer interface.

2.4.1 Cross layer knowledge base

Knowledge base stores all the secondary controls that take place in the system. CLAPDAWN separates logic of any given secondary control from its execution and stores it in the form of Event Control Action (ECA) rules in knowledge base. Different cross layer interactions may have different design objectives, but they all are modeled using standard ECA rule format.

ECA rules are triggered by an event. An event might be change in routing table, packet loss or buffer overflow. For a given event, ECA rule contains the control part. This control part checks the set of conditions before performing defined actions. Once all the conditions are satisfied it executes the action part associated with the event. Knowledge base stores different secondary controls using common ECA format as shown in figure 2.2.

Figure 2.2 shows two cross layer interactions. First interaction shows an event of
2.4 Control Plane (CP)

Cross Layer Knowledge Base (CLKB)

<table>
<thead>
<tr>
<th>Cross Layer Interaction #1 Rule #1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event: Routing Table Update</td>
</tr>
<tr>
<td>Control: Neighbour &gt; MaxThreshold</td>
</tr>
<tr>
<td>Action: Decrease Tx Power</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cross Layer Interaction #2 Rule #1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event: LQI Update</td>
</tr>
<tr>
<td>Control: LQI &lt; Min Threshold</td>
</tr>
<tr>
<td>Action: Increase Tx Power</td>
</tr>
</tbody>
</table>

Figure 2.2: Cross Layer Knowledge Base (CLKB) with Cross Layer Interactions (CLI) stored in the form of Event Control Action (ECA) rules

network layer routing table updates. On routing table update event, it checks the control statement that whether the number of neighbors is greater than the defined maximum threshold. If number of neighbors is greater than the defined threshold then it performs the action part and decreases the physical layer transmission power. In second ECA rule, on an event of Link Quality Indicator (LQI) update, it checks whether LQI is below the allowable threshold. If LQI falls below the allowable threshold then it increases the transmission power to improve LQI.

With uniform representation of different cross layer interactions it becomes an easy task to check conflicting condition and feedback loops occurring between multiple cross layer interactions. Each event has input parameter and each action shows the affected parameters. Using group of ECA rules, CLAPDAWN generates the dependency graph. Dependency graph models the relation between different parameters and cross layer interactions.

A graph node in the dependency graph represents network parameter. The directed edge between two graph nodes represent the cross layer interaction. For the directed edge, starting graph node is an input parameter and destination node is the parameter affected by that cross layer interaction. Dependency graph makes it easy for CLAPDAWN to identify conflicting conditions and feedback loops in the system.

Multiple incoming edges from different nodes show multiple cross layer interactions affecting the same output parameter. That identifies the possible conflicts among CLIs.
Further, any loop in the graph highlights the feedback loop in the system. CLAPDAWN equipped with ECA rules in knowledge base provides a way to identify conflicting scenarios and feedback loops in the system in advance.

Figure 2.3 shows an example of a dependency graph of multiple cross layer interactions. In the given example we have nine parameters and five cross layer interactions. Each cross layer interaction may have more than one rule to execute. Here CLI 1 has three rules associated with it CLI1-R1, CLI1-R2 and CLI1-R3. CLI1-R1 is triggered by update in parameter 1 and it affects or modifies the value of parameter 2. Here a rule may update more than one parameters as shown in case of CLI3-R1. This rule updates more than one parameters, e.g. parameter 3 and parameter 7 in this example.

As mention above, in a dependency graph conflicting scenario takes place when there are multiple input edges for the given parameter. Same way feedback loops can be identified by finding loops in the dependency graph. In given example CLI 2, 3 and 4 create the feedback loop in the system and CLI 1 and 3 generate the conflict at parameter 3.

Populating ECA rules is easy compared to implementing cross layer interaction using some programming language. We have developed an editor that helps in writing ECA rules. The editor converts the rules into intermediate representation. This intermediate
representation is knowledge base programming language representation. Knowledge base programming language is easy to work with and complex interaction can directly be implemented using simple programming. Further details of ECA editor is provided in appendix A.

Here, cross layer interaction designer selects the event parameters, condition parameters and action associated with that event from the available list. The editor helps in defining the condition statements and action statements. Currently it is a menu driven editor. Editor is in its primary stage and able to craft simple cross layer interaction. Future objective is to provide graphical drag and drop editor to form complex ECA rules.

For given functional plane CLAPDAWN manages the token repository. Currently the token repository is generated manually considering the referenced FP. Based on this token repository rule, the editor provides the selection options to the CLI designer. If any change takes place in referred architecture in FP then knowledge base has to make some configuration chances for new FP. Knowledge base requires to update its repository for new FP. New cross layer interaction can be designed using new token repository.

Old cross layer interaction can work with the new FP. Here, it is the responsibility of knowledge base to maintain smooth functioning of secondary control. To use existing rules with changing FP, knowledge base requires to migrate current parameter set to new parameter set. That can be done my providing the XML mapping files. Knowledge base imports this mapping file and updates the parameter configuration accordingly.

Because of the standard ECA representation these updates are easy to perform. With ECA representation it is now easy to introduce new secondary control in the system. Uniform representation of different cross layer interactions makes execution simple. On occurrence of an event it is the responsibility of execution engine to execute the relevant ECA rules from knowledge base.
2.4 Control Plane (CP)

2.4.2 Cross layer engine

CLAPDAWN separates cross layer interaction logic from its execution. It maintains the logic of cross layer interaction in knowledge base in the form of ECA rules. Based on event received from control interface, its the responsibility of execution engine to execute the relevant rules from knowledge base. Cross Layer Engine executes the required rules from the knowledge base and produces the updates from the FP which it conveys to layer interface.

Execution engine is the central entity in CP. It communicates with control interface by receiving events from control interface, with knowledge base for rules and with layer interface by sending messages. Execution engine performs the critical job of finding the rules, mapping them with parameters and executing them. Figure 2.4 shows the organization of execution engine.

Execution engine consists of the following components: event receiver, parameter mapper, rule fetcher, rule cache, policy configuration and parameter dispatcher. Event receiver receives events from control interface and passes the input parameter to the parameter mapper. Parameter mapper gets the relevant rules from rule cache. Rule fetcher gets required rules from knowledge base. By itself rules do not have any value associated with them. It is the responsibility of parameter mapper to attach appropriate

---

The diagram shows the components of the Cross Layer Engine (CLE) in CLAPDAWN with Policy configurations.

- **Event Receiver**
- **Parameter Mapping**
- **Rule Cache**
- **Rule Fetcher**
- **Execution of Rules**
- **Policy Configuration**
- **Output Parameter dispatcher**
- **Policing Output Parameter**

Figure 2.4: Cross Layer Engine (CLE) in CLAPDAWN with Policy configurations
2.4 Control Plane (CP)

value to them. It stores the rule with mapped parameter values in the rule cache. Section
2.5.1 covers the detail example of it.

Execution engine executes the rules mapped with parameter and produces the results. These results are further processed by Output parameter checker using policy configuration. Following paragraph explains the policy configuration. Finally Output parameter dispatcher forwards the generated output parameters to layer interface.

Fetching rules from knowledge base and mapping it with the parameter requires configuration time. Repeating these steps for every event creates an overhead on the system. To reduce this overhead execution engine uses rule cache in which it stores the rules with the parameter mapping.

To fetch the rules from knowledge base, execution engine uses hashing on input parameters. Once execution engine gets its required rules. It checks the control statement. If it satisfies the condition then execution engine executes the action part of ECA rule. Execution of action part generates the output parameter which goes through policing.

Knowledge base identifies the possible conflicts in the system. It identifies the parameters and rules which generates conflicts and feedback loops in the system. To handle them CLAPDAWN provides policy mechanism. At the time of populating the knowledge base with ECA rules, knowledge base highlights possible conflicts and loops with CLIs with affected parameters. Further to resolve them, CP offers the option of policy configuration. Using which one can restrict the execution of the rules or change the way the final output parameter is updated. Policy configuration stores these policy options for affected parameters.

To mitigate the effect of conflict it provides the option of a) running average of output parameters, b) averaging output of all the conflicting rules, c) prioritize the conflicting rules or d) lock some of the conflicting rules. To remove the feedback loops in the system it provides option of locking. By locking one of the CLI in the loop it removes that CLI from the system which breaks the loop. Policy configuration reduces the effect of conflict and removes the feedback loops from the system. Uniform representation of rules and separating execution from logic provides conflict resistant and loop free features to
2.4 Control Plane (CP)

CLAPDAWN.

2.4.3 Interfaces

In its modular design CLAPDAWN separates the secondary control logic from primary communication and control part. CP manages the secondary control part and FP manages the primary control and communication part. Interfaces are part of CP and they provide the bridge between FP and CP. CP provides two interfaces control interface and layer interface using which it communicates with the FP.

Control interface and layer interface both have their own importance. Through control interface, FP propagates the events in the form of messages to the CP. Control interface receives these input messages and converts them into events and propagates them to the execution engine. Each component or layer sends its input message to their defined points in CP. Using layer interface, FP receives response of its input parameter from CP. Layer interface receives the result of ECA rule execution and updates the FP using output messages. Here, control interface sends information to the CP and layer interface sends information from the CP.

Referenced architecture in FP modifies itself to make it adjustable to control interface and layer interface. These modifications are minute compared to changes required to implement cross layer interaction within the layer. CLAPDAWN embeds the layer or components of FP to expose information to the CP and creates the catch in FP that accepts information from CP.

Mainly these interfaces are designed considering following two design aspects. First, these interfaces hide the details of cross layer implementation from the FP. Second, it also hides the changing functional plane from the CP. Both are necessary, for stable protocol stack and changing CLI requires the first abstraction. In case of evolving protocol stack like Wireless Sensor Network we require the second type of abstraction.
2.5 Working of CLAPDAWN

In CLAPDAWN FP contains the referred architecture and CP provides controlled communication between modules or layers or components of FP. CP and FP communicates using asynchronous message passing through control interface and layer interface. On update of interested parameters, FP forwards them to CP using messages through standard interface provided in control interface. Control interface gets these messages and converts them into events. With changing FP the message content may change but control interface converts these messages into events which maintains the uniformity in CP.

On event, execution engine fetches rule from knowledge base. Event in execution engine provides input parameter, based on which execution engine gets affected rules from knowledge base. For each rule, execution engine checks the control statement and if it satisfies the condition then execution engine executes the action part and generates the output parameters. These output parameters are further processed by policy configuration. Finally the output parameters are forwarded to layer interface. Layer interface updates the FP by means of output messages.

CLAPDAWN maintains the two unidirectional information flows between CP and FP. One is inflow from FP to CP using control interface and second is outflow from CP to FP using layer interface. The well organized information flow controls the problem of continuous loops in the system. In other words, currently generated information will not trigger next sequence of event until they reached the FP. Once they update the FP then only they are allowed to re-enter the CP.

In CLAPDAWN, FP sends updates to CP using messages. Control interface receives these messages and converts them into events. These events are then propagated to execution engine. This information flow from FP to CP is either synchronous or asynchronous. In asynchronous information flow, FP sends message to CP and continues its task. With synchronous message FP sends the message and waits for the reply from CP in the from of layer interface updates. In section 1.2.2 we discussed different types of information flows in the cross layer interaction. Some are information sharing typed
while others are more profound and modify layer parameters. CLAPDAWN manages them using synchronous and asynchronous information flows.

CLAPDAWN manages information updates using asynchronous messages. FP sends updates to CP using asynchronous information flow which takes care of all the information sharing type cross layer interactions. Mainly the synchronous information flow is used for packets. Each packet which requires processing from cross layer interaction will use synchronous information flow. The patch provided to referred architecture handles this. CP has rule base, using which it sends configuration request to FP. FP has standard interface which receives them and provides the required asynchronous and synchronous updates. Section 2.5.2 provides example of asynchronous and synchronous updates.

2.5.1 Cross layer interaction example

To explain design and working of CLAPDAWN with an example we have implemented a cross layer interaction involving application layer and data link layer. Application layer runs the video streaming application and data link layer (MAC) uses the IEEE 802.11e. Objective of CLI is to increase the quality of video streaming by dynamically managing the prioritized queues at MAC layer.

Performance of IEEE 802.11e can be fine tuned by various mechanisms like Contention Window Size, TXOPLimit or data transmission rate. Different configurations of these parameters vary the performance of IEEE 802.11e. Required performance for the applications running at application layer can be achieved by configuring these parameters. In video streaming, different parts of encoded video data have different significance. Required performance for video streaming application can be achieved by configuring the IEEE 802.11e parameter based on significance of encoded video data.

Video streaming which uses MPEG-4 standard divides the video data into 3 frames I-Frame, P-Frame and B-Frame. Each frame has different significance. I-Frames are most critical frames for video decoding than P-Frames and B-Frames. B-Frame has lowest significance among the three frame types. IEEE 802.11e has different access queues (AC0-AC3) having separate buffer. These queues work independently from each other.
2.5 Working of CLAPDAWN

Different access queues have different priority. AC0 is highest priority queue while AC3 is least priority queue among AC0-AC3. In Cross layer approach [10], MPEG-4 video packets are dynamically mapped to appropriate AC based on both the significance of the video data and the network traffic load.

2.5.2 CLAPDAWN implementation

CLAPDAWN implementation of above CLI is shown in figure 2.5. Two layers are involved in this cross layer interaction, application layer and data link layer (MAC). Figure 2.5 shows them with data flow and secondary control flow. Solid line shows the dataflow in the system. Secondary flows are of two types synchronous and asynchronous. Secondary data flow that deals with data packet are synchronous while one which updates parameter in CP are asynchronous.

Using asynchronous update events, application layer provides the significance of each frame (Probtype) and MAC layer provides queue length to the CP, shown as step 1, 2 and 3 in figure 2.5. CP gets these events and updates its parameter, shown as step 4. If execution engine finds rules related to these parameters then it executes them. These asynchronous messages take care of the communication occurring across the layers.
2.5 Working of CLAPDAWN

Secondary control which handles per packet communication works synchronously. On arrival of packet it generates the event in CP and waits for the decision, shown as step 5 and 6. Event carries the frame type information with the packet identity. Execution engine obtains related rules from the knowledge base and executes them by applying policy configuration, step 7 and 8 in figure 2.5. Execution of these rules decide the queue number for the current frame and sends that queue number to FP using output messages. FP receives that message containing packet identity and queue number and puts the packet in the selected queue, as shown in step 9. Following are the cross layer interaction steps:

# CLI-1 Rule-1
\[ P_{r_{new}} = \frac{P_{type} \times (QL0 - \text{Threshold}_{low})}{(\text{Threshold}_{high} - \text{Threshold}_{new})} \]
\[ rnd = \text{random number between } (0, 1) \]

# CLI-1 Rule-2
\[ \text{If}(QL0 < \text{Threshold}_{low}) \{ \]
\[ Queu_{number} = 0 \]
\[ \} \]

# CLI-1 Rule-3,4
\[ \text{else if}(QL0 < \text{Threshold}_{high}) \{ \]
\[ \text{if } (rnd > P_{r_{new}}) Queu_{number} = 0 \]
\[ \text{else } Queu_{number} = 1 \]
\[ \} \]

# CLI-1 Rule-5,6
\[ \text{elseif}(QL0 > \text{Threshold}_{high}) \{ \]
\[ \text{if}(rnd > P_{r_{new}}) Queu_{number} = 1 \]
\[ \text{else } Queu_{number} = 2 \]
\[ \} \]
Above shown set of rules are executed on occurrence of events. These rules are encoded into ECA rules and stored in knowledge base. The following code snippet shows one of the ECA rule present in knowledge base.

CLI 1 - RULE 2 PARA LL frametype 2 1
CLI 1 - RULE 2 CONDITION 1 LL frametype CO 1 e
CLI 1 - RULE 2 CONDITION 2 LL QL0 LL threshold low l
CLI 1 - RULE 2 ACTION 1 LL Queue number CO 0 CO 0 d E

The above is ECA representation of rule 2 of cross layer interaction 1. It shows that the rule has two conditions to check and one action to perform. It checks for the frame type (Link Layer) and queue length (QL0) for queue zero. If frame type and queue length of queue zero satisfies the condition then it assigns the priority queue zero to the given frame. ECA rules are generated with ECA rule editor.

2.6 Summary

We have discussed our proposed architecture in this chapter and shown structure and working of it. For better handling of cross layer interaction, architecture is divided into two parts a) functional plane and b) control plane. Control plane manages all the cross layer interactions of the system through its subcomponents. Structure and working of our architecture highlights various feature of our architecture. We have compared them with the other architectures in the next chapter.