CHARGING INFRASTRUCTURE

Prof. Apurv Kumar Yadav

Department of Electrical Engineering

Indian Institute of Technology Roorkee

Week-12

Lecture-59

Lec 59: Charging Stations-II

Hello everyone, welcome to lecture number 59 of this NPTEL lecture series on charging infrastructure. Today, we will continue our discussions on charging stations, which we started in the last class. So, we have seen what a charging station is. A charging station is nothing but a station that comprises several electric vehicle supply equipments. Now, those electric vehicle supply equipment could be an ACEVSE or a DCVSE.

In that EVSE, we have a charging gun which is used to charge the vehicle. Generally, the EVSEs are designed with a single charging gun or dual charging gun. A charging station also includes a central management system called CMS, sometimes also referred to as a charging station management system (CSMS). The central management system is nothing but a kind of server that actually maintains or manages the entire charging station. So, in the charging station, we have several EVSEs. It could be several—I am drawing two—but there could be several EVSEs which communicate with this central management system for various purposes: for data logging, metering, maintenance, understanding the status of EVSEs, understanding the status user behavior, and different experts can be involved.

Now, this communication between EVSE and the central management system happens using the OCPP protocol, which is the Open Charge Point Protocol. It is a very standardized protocol that ensures interoperability between different EVSEs and the central management system. When we look at our EV, between the EV and EVSE, we have one protocol which we have discussed for the AC charging system. For the DC charging system, some of the protocols we have discussed ensure what kind of messages, how the messages need to be sent, how the data should be scheduled, how it should be sent—all those things, including different stages of

charging we have seen. But whenever communication takes place between EVSE and CMS, it will be defined by the OCPP protocol, which is the Open Charge Protocol. Now, in this entire system, the OCPP is used to stabilize the communication or to facilitate data exchange between EVSE and CMS, and it ensures interoperability.

because EVSE could be manufactured by different charger manufacturer CMS central management system could be purchased from different developers or you can say sometimes the charging station operator itself have its own central management system sometimes the manufacturer which are manufacturing EVSE will provide its their own CMS or you can central management system so it actually helps the charging station operator to do the interpretability that means tomorrow if he wants to purchase EVSE from different manufacturer he can purchase and he can easily ensure that the EVSE which is purchasing has the OCPP protocol inbuilt into it so that it can easily do communication with the central management system so it provides the interpretability it also provides the scalability because you know in future new additional EVSCs are been added into a charging station one can easily keep on adding then it defines the data standardization so how the data need to be exchanged what is the format of data all those things are defined in OCPP protocol so that's when the data standardization is been done that's when it can be easily be implementable or manageable then it is simple Simple implementation or you can say the lead time to scale or lead time to develop the charging station will go down. So, they will also facilitate the simple implementation.

So, these are some of the reasons why the OCPP protocol got adopted across different countries. Although it started from Netherlands, but then slowly the distribution companies realized the charging station operator realizes that they need a standard which will define the communication between the EVSE and CMS, central management system. now this protocol which is there which actually has different components they are ocpp profiles now these profiles are the things which defines what kind of messages which are being exchanged for example there is core ocpp profile to configure the ocpp in the cms as well as dvsc Also there are some additional security profiles which are present which will use to enhance the security features of the communication. So different OCP profiles are there.

One need to understand what are those different OCP profiles are there. then in each OCB profile you have data types which defines the kind of data which are exchanged between the

EVSE and the center management system what are the data types which are included then you have messages what all different messages which are been there between the center management system and the EVSEs and then you have a configuration keys so configuration keys what you can say special kind of message which are used to configure the communication or you can say that which is used to configure some of the settings in the evs and the central management system then you have use cases where predefined or specific profiles are being designed so that they can easily use those use cases and build the communication over that for example start transaction and stop transactions are some of the use cases which are being defined in different stages of communication between the CMS and the EVSE. Then you have test cases.

These are being used by the developer to basically ensure whether the OCPP is implemented properly or not. Before sending it to the compliance certification, one can easily test whether the CMS and the EVSEs are actually compliant to that OCPP or not so one can use those test cases directly they can import those test cases and they can test their implementation whether it is being properly implemented or not now in OCPP if you look very carefully we have also discussed this origination so it started in 2009 and till then it has gone up to OCPP 2.0.1 version now they have also come up with OCPP 2.1 version as well And in each version, the number of messages, the number of data types, the number of configuration keys, the use cases are being added. That means additional features were keep on added in each version of this OCPP.

So whenever if I say that my central management system or EVSE is compliant to OCPP 1.6, then one must ensure that they have the 60 use cases it has the 28 cross 2 messages around you know 56 messages whenever there is a communication between the central management system and the EVSE so it is a server and client kind of communication so one is the request message another one with the confirmation message so that is why it is written 28 cross 2 messages and there are 49 data types 43 configuration keys all those things will be defined in ocpp 1.6 and this ocpp is now managed by open charge alliance and it is an open source thing one can easily go to the oca website open charge alliance website and download the required files and then implement those ocpp 1.6 configurations or those messages and then one can say that their cms and evsc are ocpp compliant for whatever the version you are Referring to and implementing

that that you can say it is OCPP that particular version compliant. Now let us discuss the OCPP implementation.

There are two types of implementation for OCPP which is being defined in the OCPP document. They are OCPP SOAP implementation and OCPP JSON implementation. Now SOAP implementation is primarily used for OCPP 1.6 and earlier versions and it is basically the simple object access protocol which is nothing but a messaging protocol for data exchange which defines the data structure And how the messages are being shared? Basically it is the architecture using which messages are sent.

It utilizes XML to structure the messages. Here XML means extensible markup language. Now this XML format of data representation is a heavy duty primarily due to the addition of protocol headers which is due to the implementation of SOAP protocols. Now because of that the size of the data becomes large and those make the SOAP implementation less efficient and slower further it realizes on the more complex request response model that requires both the charge point and the cms to act as server to facilitate the two-way communication due to its two-way server requirement a evse using soap needed a unique ip address to reach to the

CMS. Now this affects the scalability of charging station when multiple EVSEs are needed to be installed. Now the implementation using SOAP is called as the OCPP-S. It is represented using this small form. Now the JSON implementation is basically the JSON is the JavaScript object notation, which is a lightweight data interchange format.

And that is happening over a persistent web socket connection. It is basically the data representation format alternative to XML. Now it is lightweight and flexible format of data representation. It is human readable. It is compact and the messages are sent in the form of strings.

So thus this has fast, reliable and easy readability. deployment as it is of the lightweight and compact data representations. Now it relies on a persistent web socket based connection for two way communication where web socket is nothing but a communication protocol which actually allows real time two way communication over a single always open connection. That means once the connection is being opened the communication channel is still be there and the

request and response message can be received. Further the security can be enhanced using security related protocols over the web sockets.

I mean one can implement the protocols related to security over the web socket and can further enhance the security of this particular connection. Now in this case since the communication is done with the CMS multiple chargers can be located behind the single router without requiring unique IP addresses for each EVSE which results in easy scalable implementation. now since the json is fast scalable reliable and easier implementable it has been adopted by most of the manufacturers pamri because they uses the json based data representation which is lightweight which is easily implementable which is understandable by the user easily and plus the reliability can be ensured by introducing different protocols over the WebSocket communication. Now one can also use the HTTP based connection instead of WebSocket and in case only one message has to be shared between the entities.

Now if the JSON implementation is being used it is called as the OCPP-J and let's say if one uses 1.6 version of OCPP then it is called as the OCPP 1.6 J based implementation if one uses 2.0.1 version of that So, then it will be OCPP 2.0.1 and it is been implemented using the JSON. So, it is called as a OCPP 2.0.1J based OCPP implementation. Now, this OCPP JSON implementation is actually been introduced from the OCPP 1.6 version and above. And in OCPP 2.1, it is been mandatory that the implementation has to be done using JSON implementation.

here the xml based soap implementation is actually the last size the data becomes larger in size that's when the speed also gets impacted however in case of json implementation because it is lightweight the fast communication can be obtained further the SOAP implementation is quite a strict implementation that means the messages have to be structured as per the different protocols however the JSON implementation is quite a less formal based implementation the messages are quite human readable it can be easily understandable by the users now if we see Different version differences, what are those different versions which are there? So, we start with OCP 1.6 because it is the widely accepted the OCP protocol. It includes several features such as status notification, remote start and stop of charging sessions.

So, one can remotely control from the CMS itself the EVSE start and stop sessions. It offers firmware management. It offers data transfer capabilities over WebSocket connections using JSON. implementation it also supports SOAP as well however the 1.6 implementation it has certain limitations for example related to security feature for advanced security features are not been there it does not support ISO 15118 protocol and we know how important the ISO 15118 protocol whenever you have the CCS2 charging system there and also it supports only the basic smart charging capabilities it does not support the advanced ISO 15118 protocol however in 2.0.1 it has enhanced security features it introduces advanced smart charging capabilities it is compatible with ISO 15118 standard can easily scale it up different EVS is compliant to that ISO 15118 standard can be implemented and do it it can be used to implement complex

complex communication architecture so all those things can be done then it also supports the direct vehicle communication so all those things are been introduced in ocpp 2.0.1 one can easily go through its documentation and understand those things Now let us discuss some of the message format which are being used in OCPPJ implementation. Why we are discussing OCPPJ because it's a very compact lightweight implementation. So that's when many people uses this OCPPJ implementation. So let us see how this OCPPJ implementation can take place.

So again it has the messages which are there. There are three kinds of messages or three types of messages which are there. One is a call message. which is a kind of you can say it is a request kind of thing a request message and you have the call result which is kind of respond to that call which is being requested then you have call error which is generally initiated from the server to client in case there is some error while doing the communication taking place Now the messages between the central management system and EVAC is basically this set of or you can say is the strings of calls and call result that means you know several set of request and respond messages we have between the CMS central management system and the EVAC.

Now each message type which we discussed three call, call result, call error has a specific message type number because that is where server as well as the client or you can say the CMS as well as the EVIC realizes what kind of request and what kind of response is been expected so this is nothing but the message type number 2 is for call 3 is for call result 4 is for call error by seeing this message type number one can able to understand whether it is a call message or is it

a response message or is it a error message And this will actually help the CMS as well as VVLC to segregate between the respond and request messages. Now, let us see one by one what are those call messages formats. So, in the call message, it basically comprises of first is message type ID.

If you have understood what is message type ID for call, it is 2. And it is a variable. data type is integer and it is the message type number which is used to identify the type of message so we have seen here in the previous slide for call the message type number is 2 then comes the message id so which is nothing but a string of 36 characters and it is nothing but it is a unique identifier which is used to match the request and the result so this is the unique identifier now this message id used to identify the request the message id for call message is different so every message which is being shared between the cms and evse is different so that's why the message id has to be different for call message is different from all the previously used message ids obviously it's over the same web socket communication so every message id must be having a unique number and that must be different from the previously shared message id however the message id for call and the call error messages which is kind of a response to that request must be having the same id as that of the call message id

so every message has the message id and message id for call is unique different than previous messages however the respond to that will have the same message id so that the entity which is receiving that response must understand that it is response to that particular call which is being initiated by the entity so that is why the call result and call error message will have the same message id as that of the call message id since it is a respond to that particular call so the respond to that particular call must have the same message id and it is nothing but a strings of a 36 character and then it comprises of another field which is the action action field it is the name itself say action field and this is nothing but string type field And it is basically the name of the remote procedure or action which is supposed to happen. And this field will contain a case sensitive string. So, for example, if there is a boot notification request, if there is a boot notification request, that's the action which has to be done.

So, it will be represented as, you know, boot notification. the request will not be there in the boot notification this is one example similarly if there is a start transaction request so the action will be the start transaction so that will be the field description so in the call message first is

obviously message type id then comes the message id so in the entire this thing you have message type id which is integer you have message id which is the string after that you have the action field which is again a string and then finally you have the payload field which is nothing but the json kind of data type it is json object containing the arguments relevant to the action so payloads are the data part of the messages that carries specific information which is been exchanged between EVHC and CMS it has particular fields against which the data are being sent inside the messages which is being used to perform the required actions now these payload fields can be required or optional fields which is given in the OCPP documents and if there are no payload in that particular JSON then a notation must be null or and empty object you can say define so if there is no payload then you have to either define null or empty object or in indicated by this way you can you know implement if there are no fields to that particular call message for example one simple message is the heartbeat request message

so heartbeat request message are being sent to keep the connections live it does not have any field in the payloads because there are no action which one has to perform so the message field comprises of message type id message id action payload so action defines the name of the particular procedure which has to be done payload defines the data content of the message which is being used to perform required action or you can say actually data part of the message then comes the another message format which is the call result now in call result as we have discussed just like in the case of call we have a message id which is integer and in case of call result which is nothing but the response kind of message it has the message type id nothing but three so if you look here is nothing but a message type number which is three so that's the call result message type id now again for every call message there is a message id so for every response to that call We will also have a message ID and that message ID is same as that of the particular call which is been initiated by the required entity. Now it is again a 36 string comprises of 36 characters and this has the same ID which is there in the call request.

So that the recipient which is receiving that particular call result will understand that this is the response to that particular call which they have initiated. so it is easier for the recipient to match the request and result so that is why this particular message id will be the part of the response message and then again just like in the case of call here also we have the payload which comprises of the arguments relevant to that particular action which is been defined in the call

request now comes another kind of message which is called as the call error message now in the call error message again it is the field is message type id which is again integer same as that of call and a call result again the message type id which is defined as the 4 if we see here we have defined the message type is nothing but the 4 so the moment the receiver receives that 4 they indicate that it is the call error message which has come error message which has come now message id is again the string of 36 characters and if you look very carefully it has the same message id which is there in the call request now for example let's say the charger demand some data from the CMS then they will attach that id towards that call and if by sending the request if they find there is some error then the CMS will respond to that call with the call error message and that's why they have the same message id

Then in this call error message we have message type ID that's the kind of message one has to send. Message ID has to be there. Then you have error code. Again it is a string. This has to be string from the error tables.

Again this is given in the OCPP document. One can see what error code has to be sent whenever the error has been taken place. So that's when while doing the implementation one needs to define that error code. Again it is a string type of field. Then you have an error description.

Again it is a it is what you can say See this error code is a fixed error code which is given in the OCPP. So they will define what kind of error is just the name of that error. While the error description will define some description about the error. Most of the time if possible it has to be filled in.

But if it is nothing is there then one need to provide it as a empty string. Then comes the error details. Again, it is the JSON object which describes the error details in an undefined way in a certain manner. And if no error details, then it has to be kept empty object. So, it is the error details one need to keep.

Again, what should be there, what kind of errors and what error codes, all those things will be defined in that OCPP document. So here we have seen three different messages, call message, which is a kind of request message. We have a call result. We have call error. So these messages which are sent has to be sent with a certain format and that format we have discussed.

Now one can have OCPP 1.6 compliant by implementing 1.6 version of OCPP. Similarly one can have OCPP 2.0.1 compliant by implementing 2.0.1 version of OCPP. Now in OCPP 1.6 there are several messages mostly there are 56 request and confirm message as we have indicated it is 28 cross 2 messages. so there are 28 requests and corresponding to those 28 requests there are 28 confirmed messages so there are total of 56 messages some of them i have indicated as authorized boot notification cancel reservation change availability change configuration clear cache clear charging profile data transfer diagnostic status notification firmware status notification get configuration get diagnostic heartbeat remote start transaction remote stop transaction meter values different messages are being incorporated in that ocpp 1.6 protocol so one can implement various functionality using this set of messages and if one says the system is ocpp 1.6 compliance then they

must use these sets of messages. Further addition to this other messages can also be defined but these are minimum set of messages which have to be implemented. So, in this one, one can go through the OCPP 1.6 specification document, easily downloadable. It is an open access document one can download and one can see those messages. Now, each message, as we mentioned, has message payloads which defines the data content in the message.

so each ocpp j message will have its corresponding payloads and these payloads are also being defined in the ocpp document if one wants to implement ocpp 1.6 version then ocpp 1.6 j document to be looked for if one wants to implement ocpp 2.0.1 version then corresponding document need to be looked for so all these messages which i am mentioning each message will have its own payload The response will have one kind of payload, the request will have some other kind of payload. So, for example, the boot notification request will have one kind of payload and boot notification response will have another kind of payload. So, each message will have its own payload or you can say arguments which has been defined and which you can say in simpler manner the data content of the messages. Now, each message payload has some field which can be required or optional field against which the data is being sent.

So, for example, the payload for remote start transaction request message are shown over here. It has the field which are connector ID, it is of integer type and it is the optional field. It is basically the connector number on which the transaction is starting and the connector ID has to be greater than zero. Now the second field in the payload of remote start transaction request

message is ID tag which is of ID token type and it is the required field which has to be sent in the remote start transaction request message. So it is the identifier

which is used by the EVSE to start the transaction then comes the charging profile which uses charging profile type and it is used by the EVSE or charge point for the requested transaction and it is an optional field which is been defined over here similarly if you see the heartbeat confirmation message which has the field name current time which has to be there with that particular message and this contains the current time of the central management system it is the required field for this payload of heartbeat confirmation message now heartbeat is there to ensure that the connection is live and it has to be sent in between to ensure the connection is live similarly here we are showing you the boot notification request message payloads now it is The message which is sent from the EVSE to CMS immediately after EVSE starts up, confirming its availability and providing initial details. Now the payload of this request message include the fields which are charge box serial number, charge point model, charge point serial number, charge point vendor, firmware version, ICCID, IMSI, meter serial number and meter type.

Now the different field types have been defined. now here in this some fields are optional some fields are required for example charge point model is the required field and it contains the value that identifies the model of the charge point or you can say or EVAC to which the central management system is communicating similarly the charge point serial number is optional the charge point vendor is the required field in this payload now this contains a value that identifies the vendor of the charge point or you can say or evac similarly there are some fields which are optional some fields which are necessary to be put in the payload of boot notification request message similarly on the other side if you see we are showing the payload of boot notification confirmation message now it has the field which is current time interval and status current time is of date time type and it is a required field this contains the central systems current time or you can say CMS current time it has the field interval which is the nothing but the integer it is also a required field now it tells the minimum wait time before sending the next notification request message then you have the status field which is of type registration status and again it is a required field and it contains whether the charge point has been registered within the central management system or not within the central system so here the charge point

analogous to the EVSE while the CSMS is analogous to CMS or you can say charging station management system or central management system or also sometimes called as the central system so if you see in the payload some of the fields are required some are optional so the developer must decide whether they want to keep that optional field or not during the time of implementation similarly one can go through the ocppj specification document to find out payloads of other messages as well and all the messages in the OCPP will have the defined payloads which is been documented in detail in OCPP specification document and let's say if you are implementing 1.6 version of OCPP you can go and see OCPP 1.6j specification document if you are implementing OCPP 2.0.1 you can go and see OCPP 2.0.1 specification document and can able to understand what are the payloads of the messages which has to be included inside the messages when it is being sent so let us understand the different steps of development of OCPP implementation in this first the designer has to select the OCPP version they have to understand the different aspect of that version and they have to select the language which will be used to implement that particular ocpp now the language could be python java java scripts now they can use the existing ocpp libraries for example if they are using the python language then python libraries corresponding to ocpp can be imported or can be used and after that they have to

implement the communication for example they have to implement the web socket communication in this EVSE opens the web socket connection to CMS while CMS always listens for incoming web socket connections from the charge point or EVSE and that is when one can easily scale the number of EVSE in the charging station and then after that one has to implement the OCPP messages to perform different functionalities they have to structure the messages like which message to go first then which message will come then what messages will be received all those things can be defined and after that one has to design the architecture on the server side for example obviously on the server side they have to implement the web socket server which listens to the connections sent by the EVSEs and then they have to also understand with what logic they will be implementing for example how the messages to be processed what data is to be stored how to handle the communication all those things will be defined and then they have to also implement how the different functionalities will be managed for example how they manages the different states sessions and transactions and finally once connections are been implemented once the messages are been implemented once the architecture on the server has

been implemented like how they manages different functionalities finally they have to do the testing with the either simulator or the real hardware so this is how one can implement this they can use the messages which is defined in the ocpp document and they can develop the communication between the cms and the evse and this communication is primarily done using the web socket based communication because it opens the persistent or real time two-way communication further one can also do it using the http communication but

There one set of messages are being sent the communication gets completed or gets ended and again it has to be initiated once more. So in order to implement the faster communication between the CMS and the EVLC they use the web socket based communication. so once the communication is been opened the messages are being sent between the EAVSE and the CMS and the required functionality will be performed so if we take some example so for example if we see the boot notification so this is you know the message ID and then these are the different field of boot notification messages so first field is message type ID since it is the call message so it is a 2 the message ID type is 2 Then you have the unique ID, message ID, which is nearly 36 characters. It is a 36 character string.

And then you have an action field, which includes what kind of action you are going to do. So it is a boot notification. It is a request, but we will not write request. We will just write boot notification. It is a case sensitive field one need to understand.

then comes the payload payload field in the payload field as we have understood in the payload field what are required thing charge point model and charge point vendor so if you see the charging station will write model so if you see charge point model so we have defined model that model you can define be any number to that and you have to define the vendor which is again required thing so the vendor is been defined which is a vendor name and they have been defined one vendor name then what is the reason one can also include this reason to that you know another thing which is nothing but the power up case we can say then comes the response to that boot notification since it is a response the message id is three the unique id if you see the call at this is a call id which i said So the message ID for call message and the response message is the same. And that's why the recipient will understand that the message is correspond to the boot notification request which they have sent.

And in the payload ABC payload there are three current time interval status. So they have sent the current time what is the time. What is the interval? 10 interval. So, interval means whenever the registration status is accepted, this contains the heartbeat interval in second.

So, 10 seconds of heartbeat interval. If the central system returns something other than accepted, the value will be the minimum wait time. So, within 10 seconds, the heartbeat has to be sent, and then comes the status, which is nothing but 'accepted'. Similarly, one can also see the heartbeat message. In the heartbeat request, the message ID is the call message.

Since it is 2, we understand it is a call message. They define the unique ID, message ID, which is nothing but a 36-character string. And they have defined the action, which is 'heartbeat'. We don't have to write 'request'; just 'heartbeat'. Payload: since there is no payload for heartbeat, they define the payload to be empty.

That is empty—nothing. No argument is being sent. Just the heartbeat request is being sent to respond to that request. The response is being sent. So that's why the message type ID is 3.

The unique message ID is the same what was there here. And if you see the boot notification message ID and heartbeat message ID is different. So both should be the independent of each other. Then they define the payload. If you see the call result message format, there is no action.

Only the response is there. So there is no action directly the payload. If you look the payload in the heartbeat confirm message, the payload field is nothing but current time of the central management system. So they have defined the current time, which is this. So, this is how the messages are being sent, this is how the OCPP can be implemented in the central management system and in the EVSE and that is when they both communicate with the standardized message formats and standardized intervals.

this is how the communication between the evsa and cms will be taking place although if you want to go more into detail of this ocpp implementation you can easily read ocpp 1.6 or 2.0.1 now they have also come up with ocpp 2.1 detailed document you can just open up those document and you can keep on read those document with that you will come to know a lot of information. This is an example we have shown over here using the understanding developed here one can easily go through those document and read through those document and can able

to understand how to implement those OCPP protocol in the CMS system as well as the EVSE. So, thank you very much for patience listening to this lecture. We will meet you in the next lecture and we will continue our discussion.

Thank you.