[](https://ida.interchain.io/)

[Interchain Developer Academy](https://ida.interchain.io/)/[Interchain Developer Academy](https://ida.interchain.io/academy/3-ibc/3-channels.html)



Search

[Interchain Developer Academy](https://ida.interchain.io/)[Interchain Developer Academy](https://ida.interchain.io/academy/3-ibc/3-channels.html)

Search



Filters

Interchain Developer Academy

[](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Week 0 - Getting Started](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Getting Started](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Blockchain 101](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Blockchain History](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Public and Managed Blockchains](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Consensus in Distributed Networks](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Cryptography](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Self-Assessment Quiz](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Go Introduction - First Steps](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Go Basics](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Go Interfaces](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Control Structures in Go](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Arrays and Slices in Go](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Standard Packages in Go](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Concurrency in Go](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Good-To-Know Dev Terms](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Docker Introduction](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Week 1 - Introduction to the Interchain](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Introduction to the Interchain](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Blockchain Technology and the Interchain](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[The Interchain Ecosystem](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Getting ATOM and Staking It](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[A Blockchain App Architecture](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Accounts](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Transactions](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Messages](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Modules](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Protobuf](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Multistore and Keepers](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[BaseApp](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Queries](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Events](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Context](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Testing](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Relaying with IBC](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Interchain Security](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Bridges](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Migrations](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Week 1 Quiz](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Week 2 - First Steps](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[First Steps](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Setup Your Work Environment](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Run a Node, API, and CLI](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Ignite CLI](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Exercise - Make a Checkers Blockchain](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Store Object](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Create Custom Messages](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Create and Save a Game Properly](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Add a Way to Make a Move](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Emit Game Information](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Record the Game Winner](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Week 2 Exercise](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Week 3 - Introduction to IBC and CosmJS](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Introduction to IBC and CosmJS](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[What is IBC?](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[IBC/TAO - Connections (OPTIONAL)](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[IBC/TAO - Channels (OPTIONAL)](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[IBC/TAO - Clients (OPTIONAL)](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[IBC Token Transfer](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Interchain Accounts (OPTIONAL)](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[IBC Middleware (OPTIONAL)](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Create a Custom IBC Middleware (OPTIONAL)](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Integrate IBC Middleware Into a Chain (OPTIONAL)](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[IBC Tooling](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[What is CosmJS?](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Your First CosmJS Actions](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Compose Complex Transactions](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Learn to Integrate Keplr](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Create Custom CosmJS Interfaces](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Week 4 - Ignite CLI and IBC Advanced](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Ignite CLI and IBC Advanced](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Keep an Up-To-Date Game Deadline](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Keep Track Of How Many Moves Have Been Played](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Put Your Games in Order](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Auto-Expiring Games](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Let Players Set a Wager](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Handle wager payments](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Integration tests](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Incentivize Players](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Help Find a Correct Move](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Play With Cross-Chain Tokens](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Understand IBC Denoms](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Go Relayer](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Hermes Relayer](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Week 5 - CosmJS Advanced](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[CosmJS Advanced](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Create Custom Objects](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Create Custom Messages](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Get an External GUI](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Integrate CosmJS and Keplr](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Backend Script for Game Indexing](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Week 6 - IBC Deep Dive](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[IBC Deep Dive](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[IBC Application Developer Introduction](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Make a Module IBC-Enabled](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Adding Packet and Acknowledgment Data](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Extend the Checkers Game With a Leaderboard](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Create a Leaderboard Chain](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Week 7 - From Code to MVP to Production and Migrations](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[From Code to MVP to Production and Migrations](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Run in Production](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Prepare the Software to Run](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Prepare a Validator and Keys](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Prepare Where the Node Starts](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Prepare and Connect to Other Nodes](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Configure, Run, and Set Up a Service](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Prepare and Do Migrations](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Simulate Production in Docker](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Tally Player Info After Production](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Add a Leaderboard as a Module](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Migrate the Leaderboard Module After Production](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Simulate a Migration in Docker](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Final Exam](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[What's Next?](https://ida.interchain.io/academy/3-ibc/3-channels.html)

[Continue Your Interchain Journey](https://ida.interchain.io/academy/3-ibc/3-channels.html)

Docs Version Switcher

On this page

[Transport, Authentication, and Ordering Layer - Channels](https://ida.interchain.io/academy/3-ibc/3-channels.html#transport-authentication-and-ordering-layer-channels)

[Establishing a channel](https://ida.interchain.io/academy/3-ibc/3-channels.html#establishing-a-channel)

[Example code: ChannelOpenInit](https://ida.interchain.io/academy/3-ibc/3-channels.html#example-code-channelopeninit)

[Capabilities](https://ida.interchain.io/academy/3-ibc/3-channels.html#capabilities)

[Application packet flow](https://ida.interchain.io/academy/3-ibc/3-channels.html#application-packet-flow)

[Success case](https://ida.interchain.io/academy/3-ibc/3-channels.html#success-case)

[Synchronous and asynchronous acknowledgements](https://ida.interchain.io/academy/3-ibc/3-channels.html#synchronous-and-asynchronous-acknowledgements)

[Timeout case](https://ida.interchain.io/academy/3-ibc/3-channels.html#timeout-case)

[#Copy link](https://ida.interchain.io/academy/3-ibc/3-channels.html#transport-authentication-and-ordering-layer-channels) **Transport, Authentication, and Ordering Layer - Channels**



In this section, you will:

* Establish a channel.
* Learn about the application packet flow.

Connections and clients comprise the main components of the transport layer in IBC. However, application to application communication in IBC is conducted over **channels**, which route between an application module such as the module which handles Interchain Standard (ICS) 20 token transfers on one chain, and the corresponding application module on another one. These applications are namespaced by **port identifiers** such as 'transfer' for ICS-20 token transfers.



Note that it is possible to have either a *symmetric* case, where the application logic is independent of the direction of the packet being sent, or an *asymmetric* case, where packets should only be sent in one direction (or the logic can be different depending on the send direction). It makes sense (but is not imposed) to use the same port ID on both application modules in the symmetric case, and different port IDs on either chain in the asymmetric.   
  
For example, in the case of interchain accounts, there are two different port IDs for the host and controller submodules: in the interchain accounts module, icahost is the default port ID that the host submodule binds to, whereas icacontroller-... is the default port prefix that the controller submodule binds to.

Contrary to the core IBC transport layer logic, which handles only verification, ordering, and all around basic packet correctness, the application layer over channels handles only the application-specific logic which interprets the packets that have been sent over the transport layer. This concern separation between transport and application layer in IBC is similar to the concern separation between CometBFT's consensus layer (consensus, mempool, ordering of transactions) and ABCI layer (process of those transaction bytes).



A connection may have any number of associated channels. However, each channel is associated with only one connection ID, which indicates which light client it is secured by, and one port ID which indicates the application that it is connected to.

As mentioned above, channels are payload agnostic. The application modules sending and receiving IBC packets decide how to interpret and act upon the incoming packet data, and use their own application logic and handlers to determine which state transitions to apply according to the data contained in each received packet.

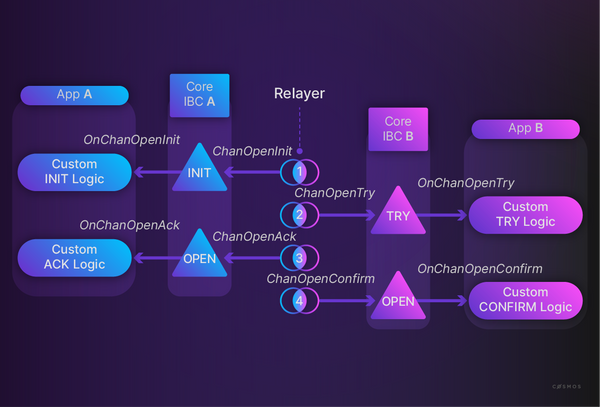


Remember the abbreviation *IBC/TAO* where the *O* represents *Ordering*. There are currently two different types of channels in terms of ordering:

* An **ordered channel** is *a channel where packets are delivered exactly in the order in which they were sent*.
* An **unordered channel** is *a channel where packets can be delivered in any order*, which may differ from the order in which they were sent.

[#Copy link](https://ida.interchain.io/academy/3-ibc/3-channels.html#establishing-a-channel) Establishing a channel

Similarly to how connections are established, **channels are established through a four-way handshake**, in which each step is initiated by a relayer:



1. ChanOpenInit: will set the chain A into INIT state. This will call OnChanOpenInit so application A can apply the custom callback that it has set on INIT, e.g. check if the port has been set correctly, the channel is indeed unordered/ordered as expected, etc. An application version is also proposed in this step.
2. ChanOpenTry: will set chain B into TRY state. It will call OnChanOpenTry so application B can apply its custom TRY callback. Application version negotiation also happens during this step.
3. ChanOpenAck: will set the chain A into OPEN state. This will call OnChanOpenAck which will be implemented by the application. Application version negotiation is finalised during this step.
4. ChanOpenConfirm: will set chain B into OPEN state so application B can apply its CONFIRM logic.



"Crossing hellos" refers to a situation when both chains attempt the same handshake step at the same time.   
  
Crossing hellos have been removed from ibc-go v4 onward, as referenced in [this PR (opens new window)↗](https://github.com/cosmos/ibc-go/pull/1317). The PreviousChannelId in MsgChannelOpenTry has been deprecated.

Similarly, there are also [callbacks for the closing of channels (opens new window)↗](https://github.com/cosmos/ibc/tree/main/spec/core/ics-004-channel-and-packet-semantics#closing-handshake). However, it is up to the application developers to decide if it makes sense for users to be able to trigger the closing of a channel. For token transfers for example, this is not desirable due to the risk of locking liquidity on a closed channel, hence that is [why this feature has been disallowed in the canonical transfer module (opens new window)↗](https://github.com/cosmos/ibc-go/blob/v7.0.0/modules/apps/transfer/ibc_module.go#L151).

[#Copy link](https://ida.interchain.io/academy/3-ibc/3-channels.html#example-code-channelopeninit) Example code: ChannelOpenInit

You can find the implementation of ChannelOpenInit in core IBC's [msg\_server.go (opens new window)↗](https://github.com/cosmos/ibc-go/blob/v7.0.0/modules/core/keeper/msg_server.go#L166)

The important part to note in this code snippet is that an application module has capabilities for the requested port. Therefore, an application module can only use a channel and port if the application owns the capability for that port and the module which attempting to open a channel is the module we have granted capabilities to in app.go:



Copy

// ChannelOpenInit defines a rpc handler method for MsgChannelOpenInit.

// ChannelOpenInit will perform 04-channel checks, route to the application

// callback, and write an OpenInit channel into state upon successful execution.

func (k Keeper) ChannelOpenInit(goCtx context.Context, msg \*channeltypes.MsgChannelOpenInit) (\*channeltypes.MsgChannelOpenInitResponse, error) {

ctx := sdk.UnwrapSDKContext(goCtx)

// Lookup module by port capability

module, portCap, err := k.PortKeeper.LookupModuleByPort(ctx, msg.PortId)

if err != nil {

return nil, sdkerrors.Wrap(err, "could not retrieve module from port-id")

}

...

// Perform 04-channel verification

channelID, cap, err := k.ChannelKeeper.ChanOpenInit(

ctx, msg.Channel.Ordering, msg.Channel.ConnectionHops, msg.PortId,

portCap, msg.Channel.Counterparty, msg.Channel.Version,

)

...

// Perform application logic callback

if err = cbs.OnChanOpenInit(ctx, msg.Channel.Ordering, msg.Channel.ConnectionHops, msg.PortId, channelID, cap, msg.Channel.Counterparty, msg.Channel.Version); err != nil {

return nil, sdkerrors.Wrap(err, "channel open init callback failed")

}

// Write channel into state

k.ChannelKeeper.WriteOpenInitChannel(ctx, msg.PortId, channelID, msg.Channel.Ordering, msg.Channel.ConnectionHops, msg.Channel.Counterparty, msg.Channel.Version)

return &channeltypes.MsgChannelOpenInitResponse{

ChannelId: channelID,

}, nil

}

modules /

core /

keeper /

msg\_server.go

[View source→](https://github.com/cosmos/ibc-go/blob/v7.0.0/modules/core/keeper/msg_server.go" \l "L166" \t "_blank)



[#Copy link](https://ida.interchain.io/academy/3-ibc/3-channels.html#capabilities) Capabilities

IBC is intended to work in execution environments where modules do not necessarily trust each other. This security is accomplished using a [dynamic capability store (opens new window)↗](https://github.com/cosmos/cosmos-sdk/blob/master/docs/architecture/adr-003-dynamic-capability-store.md). This binding strategy prevents other modules from using the particular port or channel since those modules do not own the appropriate capability.   
  
While this background information is useful, IBC application developers should not need to modify this lower level abstraction, other than setting the capabilities appropriately in app.go.

[#Copy link](https://ida.interchain.io/academy/3-ibc/3-channels.html#application-packet-flow) Application packet flow

As stated previously, application modules communicate with each other by sending packets over IBC channels. However, IBC modules do not directly pass these messages to each other over the network. Rather, the module will commit some state reflecting the transaction execution to a precisely defined path reserved for a specific message type and a specific counterparty. For example, as part of an ICS-20 token transfer, the bank module would escrow the portion of tokens to be transferred and store the proof of this escrow.

A relayer will monitor channels for events emitted when updates have been submitted to these paths, and (after first submitting a MsgUpdateClient to update the sending chain light client on the destination chain) relay the message containing the packet data along with a proof that the state transition contained in the message has been commited to the state of the sending chain. The destination chain then verifies this packet and packet commitmentment proof against the state contained in the light client.

Take a look at the [packet definition (opens new window)↗](https://github.com/cosmos/ibc/tree/main/spec/core/ics-004-channel-and-packet-semantics#definitions) to see the packet structure:



Copy

interface Packet {

sequence: uint64

timeoutHeight: Height

timeoutTimestamp: uint64

sourcePort: Identifier

sourceChannel: Identifier

destPort: Identifier

destChannel: Identifier

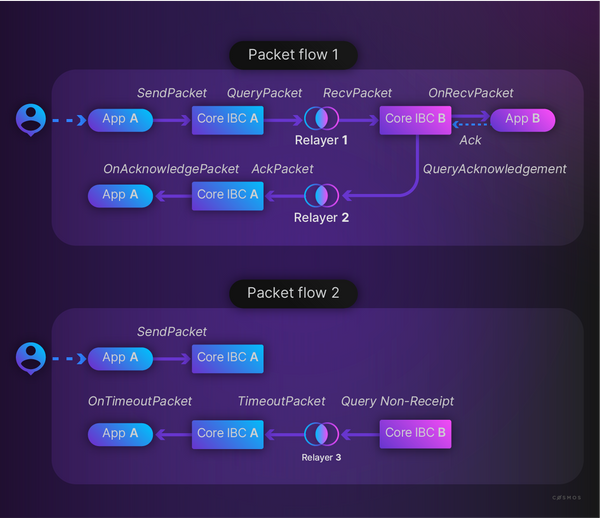
data: bytes

}

Sequence denotes the sequence number of the packet in the channel.

TimeoutTimestamp and TimeoutHeight (pick one of these) dictate the time before which the receiving module must process a packet.

The diagram below shows the application packet flow for the success case (top) and unsuccessful or timeout case (bottom). Both cases will be discussed in more detail in the next paragraphs.



[#Copy link](https://ida.interchain.io/academy/3-ibc/3-channels.html#success-case) Success case

In the first step of a successful packet flow, application A will send a packet (call sendPacket) to application B. SendPacket can be triggered by a user, but applications can also trigger this as the result of some other application logic.

Core IBC A will commit the packet to its own state and the relayer can query this packet and send a RecvPacket message to core IBC B. Core IBC handles a number of verifications, including verifying that the packet was indeed sent by chain A, that the packet came in the correct order if it was sent over an ordered channel, that the state commitment proof is valid, etc. If this verification step is successful, core IBC will then route the packet to application B.

Note that core IBC is unopinionated about the actual content of the packet data, as this data is at this point just bytes. It is the responsibility of the applications on either end to marshal and unmarshal the data from and to the expected data structures on either side. This is also why application version negotiation as discussed previously in the channel handshakes is important, as different versions of an application may result in different expected data structures on either end of the channel and application.

After receiving the packet data from core IBC, application B will then marshal the data blob into the expected structure and apply the relevant application logic. In the case of an ICS-20 token transfer, for example, this would entail the minting of the received tokens on chain B to the specified receiver user account. Application B will then send an Acknowledgement message to core IBC B, which will again commit it to its own state so it can be queried and sent by a relayer to core IBC A.



[#Copy link](https://ida.interchain.io/academy/3-ibc/3-channels.html#synchronous-and-asynchronous-acknowledgements) Synchronous and asynchronous acknowledgements

Acknowledgements can either take place synchronously or asynchronously. What this means is that the OnRecvPacket callback has a return value Acknowledgement which is optional.   
  
In the case of a synchronous Acknowledgement, the callback will return an Acknowledgement at the end of the process and a relayer can query this Acknowledgement packet and relay immediately after the process has finished. This is useful in cases in which application A is expecting an AckPacket in order to initiate some application logic OnAcknowledgePacket. For example, the sending chain of an ICS-20 token transfer will do nothing in the case of a successful AckPacket, but in the case where an error is returned, the sending chain will unescrow the previously locked tokens.   
  
In the case of applications like Interchain Security, there is an asynchronous Acknowledgement flow. This means that the Acknowledgement is not sent as part of the return value of OnRecvPacket, but it is sent at some later point. IBC is designed to handle this case by allowing for Acknowledgements to be committed or queried asynchronously.   
  
In either case, even if there is no application specific logic to be initiated as a direct result of a received Acknowledgement, OnAcknowledgePacket will at the very least remove the commitment proof from the store to avoid cluttering the store with old data.

[#Copy link](https://ida.interchain.io/academy/3-ibc/3-channels.html#timeout-case) Timeout case

In the case that a packet is time-sensitive and the timeout block height or timeout timestamp specified in the packet parameters **based on chain B's time** has elapsed, whatever state transitions have occured as a result of the sent packet should be reversed.

In these cases, the initial flow is the same, with core IBC A first committing the packet to its own state. However, instead of querying for the packet, a relayer will submit a QueryNonReceipt to receive a proof that the packet was not received by core IBC B. It can then send the MsgTimeout to core IBC A, which will then trigger the relevant OnTimeoutPacket application logic. For example, the ICS-20 token transfer application will unescrow the locked up tokens and send these back to the original sender OnTimeoutPacket.

synopsis

To summarize, this section has explored:

* How application to application communication in IBC is conducted over channels, which route data between corresponding modules on different chains, and how a single connection between applications can have any number of associated channels.
* How channels are payload agnostic and simply deliver data packets over the transport layer which application modules use their own logic and handlers to interpret and act upon.
* How channels can be **ordered** (where data packets are delivered exactly in the order they were sent) or **unordered** (where packets can be delivered in any order, which may differ from that in which they were sent).
* How channels are established through a four-way handshake, which allows only the two end modules to make use of the channel, securing them against malicious entities.
* How packet flow between modules is not direct - instead, the sending module commits some particular state, thus emitting an event which is detected by a relayer, which delivers to the destination module both the packet data and a proof of the sending module's state commit, which is then verified on the destination chain.
* How it is possible for time-sensitive packets to trigger a reversal of the sending module's state change in the event that a timeout block height or timestamp has elapsed.

previous

[](https://ida.interchain.io/academy/3-ibc/2-connections.html)

**[IBC/TAO - Connections (OPTIONAL)](https://ida.interchain.io/academy/3-ibc/2-connections.html)**

up next

**[IBC/TAO - Clients (OPTIONAL)](https://ida.interchain.io/academy/3-ibc/4-clients.html)**

[[](https://ida.interchain.io/academy/3-ibc/4-clients.html)](https://ida.interchain.io/academy/3-ibc/4-clients.html)

Rate this Page

icon smile

icon meh

icon frown

Would you like to add a message?

Submit

Thank you for your Feedback!

[](https://ida.interchain.io/ida-course/discord-info.html)

On this page

[Transport, Authentication, and Ordering Layer - Channels](https://ida.interchain.io/academy/3-ibc/3-channels.html#transport-authentication-and-ordering-layer-channels)

[Establishing a channel](https://ida.interchain.io/academy/3-ibc/3-channels.html#establishing-a-channel)

[Example code: ChannelOpenInit](https://ida.interchain.io/academy/3-ibc/3-channels.html#example-code-channelopeninit)

[Capabilities](https://ida.interchain.io/academy/3-ibc/3-channels.html#capabilities)

[Application packet flow](https://ida.interchain.io/academy/3-ibc/3-channels.html#application-packet-flow)

[Success case](https://ida.interchain.io/academy/3-ibc/3-channels.html#success-case)

[Synchronous and asynchronous acknowledgements](https://ida.interchain.io/academy/3-ibc/3-channels.html#synchronous-and-asynchronous-acknowledgements)

[Timeout case](https://ida.interchain.io/academy/3-ibc/3-channels.html#timeout-case)

#### **Get Cosmos updates**

Unsubscribe at any time. [Privacy Policy↗](https://v1.cosmos.network/privacy)

     Next

Documentation

[Cosmos SDK](https://docs.cosmos.network/)[Cosmos Hub](https://hub.cosmos.network/)[CometBFT](https://docs.cometbft.com/)[IBC Protocol](https://ibc.cosmos.network/)

Community

[Interchain blog](https://blog.cosmos.network/)[Forum](https://forum.cosmos.network/)[Discord](https://discord.gg/cosmosnetwork)

Contributing

[Source code on GitHub](https://github.com/cosmos/sdk-tutorials)

[](https://ida.interchain.io/)

[Interchain Developer Academy](https://ida.interchain.io/)

**[](https://blog.cosmos.network/)[](https://twitter.com/cosmos)[](https://discord.gg/cosmosnetwork)[](https://www.linkedin.com/company/interchain-foundation/about/)[](https://reddit.com/r/cosmosnetwork)[](https://t.me/cosmosproject)[](https://www.youtube.com/c/CosmosProject)**



Dark mode

† This website is maintained by the Interchain Foundation (ICF). The contents and opinions of this website are those of the ICF. The ICF provides links to cryptocurrency exchanges as a service to the public. The ICF does not warrant that the information provided by these websites is correct, complete, and up-to-date. The ICF is not responsible for their content and expressly rejects any liability for damages of any kind resulting from the use, reference to, or reliance on any information contained within these websites.

Cosmos is a registered trademark of the [Interchain Foundation.](https://interchain.io/)[Privacy](https://v1.cosmos.network/privacy)