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

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



Search

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

Search



Filters

Interchain Developer Academy

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

[Week 0 - Getting Started](https://ida.interchain.io/academy/3-ibc/4-clients.html)

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

[Blockchain 101](https://ida.interchain.io/academy/3-ibc/4-clients.html)

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

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

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

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

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

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

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

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

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

[Arrays and Slices in Go](https://ida.interchain.io/academy/3-ibc/4-clients.html)

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

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

[Good-To-Know Dev Terms](https://ida.interchain.io/academy/3-ibc/4-clients.html)

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

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

[Week 1 - Introduction to the Interchain](https://ida.interchain.io/academy/3-ibc/4-clients.html)

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

[Blockchain Technology and the Interchain](https://ida.interchain.io/academy/3-ibc/4-clients.html)

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

[Getting ATOM and Staking It](https://ida.interchain.io/academy/3-ibc/4-clients.html)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[Week 1 Quiz](https://ida.interchain.io/academy/3-ibc/4-clients.html)

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

[Week 2 - First Steps](https://ida.interchain.io/academy/3-ibc/4-clients.html)

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

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

[Run a Node, API, and CLI](https://ida.interchain.io/academy/3-ibc/4-clients.html)

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

[Exercise - Make a Checkers Blockchain](https://ida.interchain.io/academy/3-ibc/4-clients.html)

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

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

[Create and Save a Game Properly](https://ida.interchain.io/academy/3-ibc/4-clients.html)

[Add a Way to Make a Move](https://ida.interchain.io/academy/3-ibc/4-clients.html)

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

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

[Week 2 Exercise](https://ida.interchain.io/academy/3-ibc/4-clients.html)

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

[Week 3 - Introduction to IBC and CosmJS](https://ida.interchain.io/academy/3-ibc/4-clients.html)

[Introduction to IBC and CosmJS](https://ida.interchain.io/academy/3-ibc/4-clients.html)

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

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

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

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

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

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

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

[Create a Custom IBC Middleware (OPTIONAL)](https://ida.interchain.io/academy/3-ibc/4-clients.html)

[Integrate IBC Middleware Into a Chain (OPTIONAL)](https://ida.interchain.io/academy/3-ibc/4-clients.html)

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

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

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

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

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

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

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

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

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

[Keep an Up-To-Date Game Deadline](https://ida.interchain.io/academy/3-ibc/4-clients.html)

[Keep Track Of How Many Moves Have Been Played](https://ida.interchain.io/academy/3-ibc/4-clients.html)

[Put Your Games in Order](https://ida.interchain.io/academy/3-ibc/4-clients.html)

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

[Let Players Set a Wager](https://ida.interchain.io/academy/3-ibc/4-clients.html)

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

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

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

[Help Find a Correct Move](https://ida.interchain.io/academy/3-ibc/4-clients.html)

[Play With Cross-Chain Tokens](https://ida.interchain.io/academy/3-ibc/4-clients.html)

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

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

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

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

[Week 5 - CosmJS Advanced](https://ida.interchain.io/academy/3-ibc/4-clients.html)

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

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

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

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

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

[Backend Script for Game Indexing](https://ida.interchain.io/academy/3-ibc/4-clients.html)

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

[Week 6 - IBC Deep Dive](https://ida.interchain.io/academy/3-ibc/4-clients.html)

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

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

[Make a Module IBC-Enabled](https://ida.interchain.io/academy/3-ibc/4-clients.html)

[Adding Packet and Acknowledgment Data](https://ida.interchain.io/academy/3-ibc/4-clients.html)

[Extend the Checkers Game With a Leaderboard](https://ida.interchain.io/academy/3-ibc/4-clients.html)

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

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

[Week 7 - From Code to MVP to Production and Migrations](https://ida.interchain.io/academy/3-ibc/4-clients.html)

[From Code to MVP to Production and Migrations](https://ida.interchain.io/academy/3-ibc/4-clients.html)

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

[Prepare the Software to Run](https://ida.interchain.io/academy/3-ibc/4-clients.html)

[Prepare a Validator and Keys](https://ida.interchain.io/academy/3-ibc/4-clients.html)

[Prepare Where the Node Starts](https://ida.interchain.io/academy/3-ibc/4-clients.html)

[Prepare and Connect to Other Nodes](https://ida.interchain.io/academy/3-ibc/4-clients.html)

[Configure, Run, and Set Up a Service](https://ida.interchain.io/academy/3-ibc/4-clients.html)

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

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

[Tally Player Info After Production](https://ida.interchain.io/academy/3-ibc/4-clients.html)

[Add a Leaderboard as a Module](https://ida.interchain.io/academy/3-ibc/4-clients.html)

[Migrate the Leaderboard Module After Production](https://ida.interchain.io/academy/3-ibc/4-clients.html)

[Simulate a Migration in Docker](https://ida.interchain.io/academy/3-ibc/4-clients.html)

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

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

[What's Next?](https://ida.interchain.io/academy/3-ibc/4-clients.html)

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

Docs Version Switcher

On this page

[Creating a client](https://ida.interchain.io/academy/3-ibc/4-clients.html#creating-a-client)

[Updating a client](https://ida.interchain.io/academy/3-ibc/4-clients.html#updating-a-client)

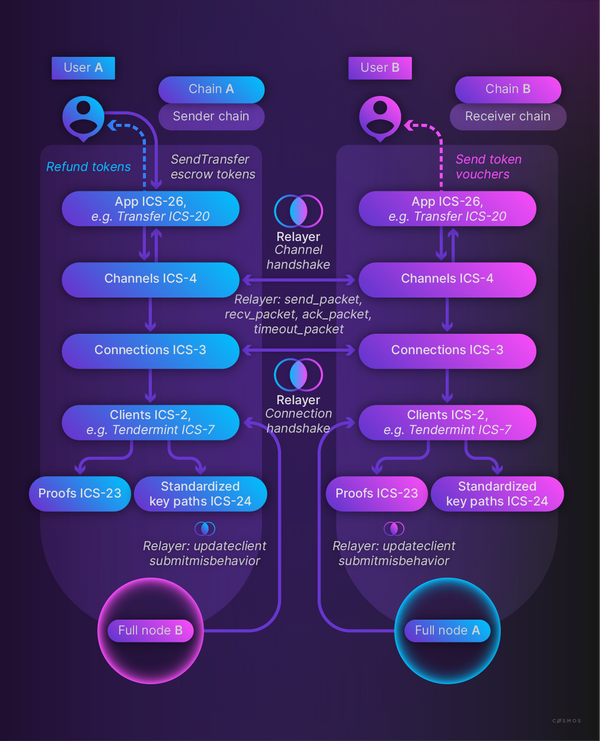
[Verifying packet commitments](https://ida.interchain.io/academy/3-ibc/4-clients.html#verifying-packet-commitments)

[#Copy link](https://ida.interchain.io/academy/3-ibc/4-clients.html#transport-authentication-and-ordering-layer-clients) **Transport, Authentication, and Ordering Layer - Clients**



In this section, you will learn:

* How a client is created.
* How client state and consensus can be verified.
* How packets are verified.



As previously shown, IBC is structured as several layers of abstraction. At the top, applications such as [Interchain Standard (ICS) 20 token transfers (opens new window)↗](https://github.com/cosmos/ibc/tree/master/spec/app/ics-020-fungible-token-transfer) implement the [ICS-26 IBC standard (opens new window)↗](https://github.com/cosmos/ibc/blob/master/spec/core/ics-026-routing-module/README.md), which describe the routing and callback functionality used to connect the application layer to the transport layer. Underneath the application are channels, which are unique for each application (for example, a channel that allows a transfer application on chain A to speak to a transfer application on chain B). [Connections](https://ida.interchain.io/academy/3-ibc/2-connections.html), which may have many channels, are used to connect two clients (for example, to allow the entire IBC stack of chain A to connect to the IBC stack of chain B). These clients, which may have many connections, comprise the foundational layer of IBC.



IBC application developers will primarily interact with [IBC channels](https://ida.interchain.io/academy/3-ibc/3-channels.html). This layer is comprised of the handshakes and packet callbacks.

In the IBC setup, each chain will have a **client** of the other chain in its own IBC stack. IBC clients track the consensus states of other blockchains, and the proof specs of those blockchains that are required to properly verify proofs against the client's consensus state. The packets, acknowledgements, and timeouts that off-chain relayers send back and forth can be verified by proving that the packet commitments exist inside of these clients on each chain.



Although relayers do not perform any verification of the packets, and therefore do not need to be trusted, relayers have a particularly important role in IBC setup in addition to IBC network liveness through submission of packets. They are responsible for submitting the initial messages to create a new client, as well as keeping the client states updated on each chain, so that proof verification on a submitted packet is successful. Relayers are also responsible for sending the connection and channel handshakes to establish connections and channels between chains. Furthermore, relayers can submit evidence of misbehaviour if a chain on the other end of a connection tries to fork or attempts other types of malicious behaviour.

[#Copy link](https://ida.interchain.io/academy/3-ibc/4-clients.html#creating-a-client) Creating a client

Start with [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#L25), which is where the messages come in. This is the first appearance of the CreateClient function, which will be submitted by a relayer through the relaying software to create an IBC client on the chain that the message is submitted to:



Copy

// CreateClient defines a rpc handler method for MsgCreateClient.

func (k Keeper) CreateClient(goCtx context.Context, msg \*clienttypes.MsgCreateClient) (\*clienttypes.MsgCreateClientResponse, error) {

ctx := sdk.UnwrapSDKContext(goCtx)

clientState, err := clienttypes.UnpackClientState(msg.ClientState)

...

consensusState, err := clienttypes.UnpackConsensusState(msg.ConsensusState)

...

... = k.ClientKeeper.CreateClient(ctx, clientState, consensusState);

...

}

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 "L25" \t "_blank)

It creates a client by calling [ClientKeeper.CreateClient (opens new window)↗](https://github.com/cosmos/ibc-go/blob/v7.0.0/modules/core/02-client/keeper/client.go#L16):



Copy

// CreateClient creates a new client state and populates it with a given consensus

// state as defined in https://github.com/cosmos/ibc/tree/master/spec/core/ics-002-client-semantics#create

func (k Keeper) CreateClient(

ctx sdk.Context, clientState exported.ClientState, consensusState exported.ConsensusState,

) (string, error) {

...

clientID := k.GenerateClientIdentifier(ctx, clientState.ClientType())

...

k.SetClientState(ctx, clientID, clientState)

...

// verifies initial consensus state against client state and initializes client store with any client-specific metadata

// e.g. set ProcessedTime in CometBFT clients

... := clientState.Initialize(ctx, k.cdc, k.ClientStore(ctx, clientID), consensusState);

...

EmitCreateClientEvent(ctx, clientID, clientState)

return clientID, nil

}

modules /

core /

... /

keeper /

client.go

[View source→](https://github.com/cosmos/ibc-go/blob/v7.0.0/modules/core/02-client/keeper/client.go" \l "L16" \t "_blank)

A local, unique identifier clientID is generated for each client on the chain. This is not related to the chainID, as IBC does not actually use the chainID as an identifier.



The IBC security model is based on clients and not specific chains. This means that the IBC protocol does not need to know who the chains are on either side of a connection, provided that the IBC clients are kept in sync with valid updates, and these updates or other types of messages (i.e. ICS-20 token transfers) can be verified as a Merkle proof against an initial consensus state (root of trust). This is analogous to IP addresses and DNS, where IP addresses would be the corollary to IBC clientIDs, and DNS the chainIDs.   
  
Because of this separation of concerns, IBC clients can be created for any number of machine types, from fully-fledged blockchains to keypair-based solo machines, and upgrades to chains which increment the chainID do not break the underlying IBC client and connections.

In addition, you can see that the function expects a ClientState. This ClientState will look different depending on which type of client is to be created for IBC. In the case of Cosmos-SDK chains and the corresponding implementation of ibc-go, the [CometBFT (Tendermint) client (opens new window)↗](https://github.com/cosmos/ibc/tree/main/spec/client/ics-007-tendermint-client#client-state) is offered out of the box:



Copy

interface ClientState {

chainID: string

trustLevel: Rational

trustingPeriod: uint64

unbondingPeriod: uint64

latestHeight: Height

frozenHeight: Maybe<uint64>

upgradePath: []string

maxClockDrift: uint64

proofSpecs: []ProofSpec

}

The CometBFT ClientState contains all the information needed to verify a header. This includes properties which are applicable for all CometBFT clients, such as the corresponding chainID, the unbonding period of the chain, the latest height of the client, etc.

TrustingPeriod determines the duration of the period since the latest timestamp during which the submitted headers are valid for upgrade. If a client is not updated within the TrustingPeriod, the client will expire. This does not mean the client is irrecoverable. However, recovery of an expired CometBFT client will require a [governance proposal (opens new window)↗](https://ibc.cosmos.network/main/ibc/proposals.html#preconditions) for each client which has expired. If both clients on either side of a connection have expired, then a governance proposal will be required on each chain in order to revive each client.

TrustLevel determines the portion of the validator set you want to have signing a header for it to be considered as valid. CometBFT defines this as 2/3, and the IBC CometBFT client inherits this property from CometBFT.



Properties such as TrustLevel and TrustingPeriod can be customised, such that different clients on the same chain can have different security guarantees with different tradeoffs for efficiency of processing updates.   
  
As stated before, TrustLevel is inherited from CometBFT and will be 2/3 for all CometBFT clients. However, this could change for other client types.   
  
It is recommended that TrustingPeriod should be set as 2/3 of the UnbondingPeriod.   
  
It is also recommended that MaxClockDrift should be set to at least 5sec and up to 15sec, depending on expected block size differences between the chains in the connection. The Hermes (Rust) relayer will compute this value for you if you do not manually set it.



It is important to highlight that certain parameters of an IBC client cannot be updated after the client has been created, in order to preserve the security guarantees of each client and prevent a situation where a relayer unilaterally updates those security guarantees. These parameters are: MaxClockDrift and TrustLevel.

CreateClient additionally expects a [ConsensusState (opens new window)↗](https://github.com/cosmos/ibc/tree/main/spec/client/ics-007-tendermint-client#consensus-state). In the case of a CometBFT client, the initial root of trust (or consensus state) looks like this:



Copy

interface ConsensusState {

timestamp: uint64

nextValidatorsHash: []byte

commitmentRoot: []byte

}

The CometBFT client ConsensusState tracks the timestamp of the block being created, the hash of the validator set for the next block of the counterparty blockchain, and the root of the counterparty blockchain. The initial ConsensusState does not need to start with the genesis block of a counterparty chain.



The next validator set is used for verifying subsequent submitted headers or updates to the counterparty ConsensusState. See the following part on [Updating clients](https://ida.interchain.io/academy/3-ibc/4-clients.html#updating-a-client) for more information about what happens when a validator set changes between blocks.

The root is the **AppHash**, or the hash of the application state of the counterparty blockchain that this client is representing. This root hash is particularly important because it is the root hash used on a receiving chain when verifying [Merkle (opens new window)↗](https://en.wikipedia.org/wiki/Merkle_tree) proofs associated with a packet coming over IBC, to determine whether or not the relevant transaction has been actually been executed on the sending chain. If the Merkle proof associated with a packet commitment delivered by a relayer successfully hashes up to this ConsensusState root hash, it is certain that the transaction was actually executed on the sending chain and included in the state of the sending blockchain.

The following is an example of how the CometBFT client handles this Merkle [proof verification (opens new window)↗](https://github.com/cosmos/ibc-go/blob/v7.0.0/modules/core/23-commitment/types/merkle.go#L133). The [ICS-23 spec (opens new window)↗](https://github.com/cosmos/ibc/tree/master/spec/core/ics-023-vector-commitments) addresses how to construct membership proofs, and the [ICS-23 implementation (opens new window)↗](https://github.com/confio/ics23) currently supports CometBFT IAVL and simple Merkle proofs out of the box.



Non-CometBFT client types may choose to handle proof verification differently.



Copy

// VerifyMembership verifies the membership of a merkle proof against the given root, path, and value.

func (proof MerkleProof) VerifyMembership(specs []\*ics23.ProofSpec, root exported.Root, path exported.Path, value []byte) error {

if err := proof.validateVerificationArgs(specs, root); err != nil {

return err

}

// VerifyMembership specific argument validation

mpath, ok := path.(MerklePath)

if !ok {

return sdkerrors.Wrapf(ErrInvalidProof, "path %v is not of type MerklePath", path)

}

if len(mpath.KeyPath) != len(specs) {

return sdkerrors.Wrapf(ErrInvalidProof, "path length %d not same as proof %d",

len(mpath.KeyPath), len(specs))

}

if len(value) == 0 {

return sdkerrors.Wrap(ErrInvalidProof, "empty value in membership proof")

}

// Since every proof in chain is a membership proof we can use verifyChainedMembershipProof from index 0

// to validate entire proof

if err := verifyChainedMembershipProof(root.GetHash(), specs, proof.Proofs, mpath, value, 0); err != nil {

return err

}

return nil

}



IBC on-chain clients can also be referred to as **light clients**. In contrast to the full nodes, which track the entire state of blockchain and contain every single tx/block, these on-chain IBC "light clients" track only the few pieces of information about counterparty chains previously mentioned (timestamp, root hash, next validator set hash). This saves space and increases the efficiency of processing consensus state updates.   
  
The objective is to avoid a situation where it is necessary to have a copy of chain B on chain A in order to create a trustless IBC connection. However, full nodes which track the entire state of a blockchain are useful for IBC relayer operators as an endpoint to query for the proofs needed to verify IBC packet commitments. This entire process maintains the trustless, permissionless, and highly secure design of IBC. As proof verification still happens in the IBC client itself, no trust in the relayer operator is needed and anyone can permissionlessly spin up a relaying operation, provided that they have access to a full node endpoint.

[#Copy link](https://ida.interchain.io/academy/3-ibc/4-clients.html#updating-a-client) Updating a client

Assume that the initial ConsensusState was created at block 50, but you want to submit a proof of a transaction which happened in block 100. In this case, you need to first update the ConsensusState to reflect all the changes that have happened between block 50 and block 100.

To update the ConsensusState of the counterparty on the client, a MsgUpdateClient containing a Header of the chain to be updated must be submitted by a relayer. For all IBC client types, CometBFT or otherwise, this Header contains the information necessary to update the ConsensusState. However, IBC does not dictate what the Header must contain beyond the basic methods for returning ClientType and GetClientID. The specifics of what each client expects as important information to perform a ConsensusState update will be found in each client implementation.

For example, the CometBFT client Header looks like [this (opens new window)↗](https://github.com/cosmos/ibc/tree/main/spec/client/ics-007-tendermint-client#headers):



Copy

interface Header extends TendermintSignedHeader {

identifier: string

validatorSet: List<Pair<Address, uint64>>

trustedHeight: Height

trustedValidatorSet: List<Pair<Address, uint64>>

}

The CometBFT SignedHeader is a header and commit that the counterparty chain has created. In the MsgUpdateClient example, this would be the header of block 100 which will contain the timestamp of the block, the hash of the next validator set, and the root hash needed to update the ConensusState on record for the counterparty chain. The commit will be a signature of at least 2/3 of the validator set over that header, which is guaranteed as part of CometBFT's consensus model.

ValidatorSet will be the actual validator set, as opposed to the hash of the next validator set stored on the ConsensusState. This is important for the CometBFT UpdateClient method because, in order to preserve the CometBFT security model, it is necessary to be able to prove that at least 2/3 of the validators who signed the initial header at block 50 have signed the header to update the ConsensusState to block 100. This ValidatorSet will be submitted by the relayer as part of the MsgUpdateClient, as the relayer has access to full nodes from which this information can be extracted.

TrustedValidators are the validators associated with that height. Note that TrustedValidators must hash to the ConsensusState NextValidatorsHash since that is the last trusted validator set at the TrustedHeight.

The TrustedHeight is the height of a stored ConsensusState on the client that will be used to verify the new untrusted header. You can see the code that takes the ConsensusState at the TrustedHeight and uses it to verify the new header [here (opens new window)↗](https://github.com/cosmos/ibc-go/blob/v7.0.0/modules/light-clients/07-tendermint/update.go#L42). This code proves that the submitted header is valid and creates a verified ConsensusState for the submitted header, as well as updating the client state to reflect the new latest height of the submitted header. This verified ConsensusState will be added to the client as part of the set of ClientConsensusStates, and can subsequently be used as a trusted state at its corresponding height.



If you want to see where ConsensusState is stored, see the [Interchain Standard (ICS) 24 (opens new window)↗](https://github.com/cosmos/ibc/tree/master/spec/core/ics-024-host-requirements), which also describes the paths for other keys to be stored and used by IBC.

[#Copy link](https://ida.interchain.io/academy/3-ibc/4-clients.html#verifying-packet-commitments) Verifying packet commitments

As shown in the deep dive on [channels](https://ida.interchain.io/academy/3-ibc/3-channels.html), a relayer will first submit a MsgUpdateClient to update the sending chain client on the destination chain, before relaying packets containing other message types, such as ICS-20 token transfers. The destination chain can be sure that the packet will be contained in its ConsensusState root hash, and successfully verify this packet and packet commitment proof against the state contained in its (updated) IBC light client.

The pseudo-code snippet below from [03-connection (opens new window)↗](https://github.com/cosmos/ibc/tree/main/spec/core/ics-003-connection-semantics#helper-functions), which illustrates how a client verifies an incoming packet, is as follows:



Copy

function verifyPacketCommitment(

connection: ConnectionEnd,

height: Height,

proof: CommitmentProof,

portIdentifier: Identifier,

channelIdentifier: Identifier,

sequence: uint64,

commitmentBytes: bytes

) {

clientState = queryClientState(connection.clientIdentifier)

path = applyPrefix(

connection.counterpartyPrefix,

packetCommitmentPath(portIdentifier, channelIdentifier, sequence))

return verifyMembership(

clientState,

height,

connection.delayPeriodTime,

connection.delayPeriodBlocks,

proof,

path,

commitmentBytes)

}

synopsis

To summarize, this section has explored:

* How each chain communicating through IBC will have a client of the other chain in its own IBC stack, which tracks the consensus state and proof specs of the other chain.
* How these on-chain "light clients" avoid the need for one chain to hold a complete copy of another in order to create a trustless IBC connection, saving space and increasing efficiency.
* How the packets, acknowledgements, and timeouts that off-chain relayers send back and forth can be verified by proving that the packet commitments exist inside of these clients on each chain.
* How relayers perform vital functions such as submitting the initial messages to create a new client, keeping client states updated on each chain, sending the handshakes that establish connections and channels between chains, and submitting evidence of attempts to fork or other malicious behavior.
* How the IBC protocol does not need to know who the chains are provided that their IBC clients are kept in sync with valid and verifiable updates and messages, meaning clients can be created for any number of machine types.

previous

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

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

up next

**[IBC Token Transfer](https://ida.interchain.io/academy/3-ibc/7-token-transfer.html)**

[[](https://ida.interchain.io/academy/3-ibc/7-token-transfer.html)](https://ida.interchain.io/academy/3-ibc/7-token-transfer.html)

Rate this Page

icon smile

icon meh

icon frown

Would you like to add a message?

Submit

Thank you for your Feedback!

On this page

[Creating a client](https://ida.interchain.io/academy/3-ibc/4-clients.html#creating-a-client)

[Updating a client](https://ida.interchain.io/academy/3-ibc/4-clients.html#updating-a-client)

[Verifying packet commitments](https://ida.interchain.io/academy/3-ibc/4-clients.html#verifying-packet-commitments)

#### **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)