Invisible Internet Client Protocol (I2CP)

Revision 0.9, 28 August, 2003

http://www.InvisibleNet.net/ info@invisiblenet.net

jrandom@invisiblenet.net

Table of Contents

1.Protocol Overview

Goals

The Invisible Internet Client Protocol (I2CP) allows applications simplified access to the I2P network without requiring them to deal with the issues involved with the Invisible Internet Network Protocol (I2NP). Specifically, it defines the wire level protocol as well as semantics for the messages passed between clients and the router for communicating with the router over bidirectional TCP/IP sockets. I2CP does not specify how the client libraries are created, what APIs they expose to applications, or even what language they're written in.

Requirements

I2CP requires that the client can access the router over bidirectional TCP sockets. In nearly all cases, these connections will be over to the local machine.

Threat Model

I2CP does not provide data confidentiality or handle lossy connections beyond what TCP has built in. For this reason, I2CP really should only be run between clients and routers on the same machine or over trusted networks. Secured I2CP transports may be added in the future. I2CP however does not expose any private keys across the wire – proof of authorization to collect messages for a Destination is provided by public key signatures.

2.Messages

What follows are the currently defined messages in the Invisible Internet Client Protocol (I2CP), though common data structures are located in the I2P Common Data Structure Specification. For simplicity, all I2CP messages begin with the same structure, though that structure is not listed below. Specifically, all I2CP messages transmitted begin with a 4 byte `Integer` specifying the entire size of the current message's body (the body being what's specified below), followed by a 1 byte `Integer` specifying the type of message (the id field below), after which the rest of the message is formatted according to the type of message, as specified below.

If there is a fatal error causing either the client or the router to desire to cancel sending a message part way through, it should drop its connection. Administrative sessions are not stateful (aka each administrative message from client to router must provide authentication), but normal client sessions are stateful and survive disconnects. Client sessions expire when either the client sends a `DestroySessionMessage` or the router times out the session according to its own configurable timer. If a client sends any message other than a `CreateSessionMessage` when there is no valid session, the router must reply with a `SessionStatusMessage` specifying that the session is not valid.

Note: the contents and bitbuckets for specific `DataStructures` are detailed in the I2P Data Structures Spec.

Client to Router Messages

These messages are sent from typical client applications to the router

Message:	1 (CreateSessionMessage)
Description:	This message is sent from a client to initiate a session, where a session is defined as a single `Destination`'s connection to the network, to which all messages for that `Destination` will be delivered and from which all messages that `Destination` sends to any other `Destination` will be sent through.
Contents:	`SessionConfig`
Responses:	SessionStatusMessage
Notes:	

Message:	2 (ReconfigureSessionMessage)
Description:	This message is sent from a client to update the configuration for a session.
Contents:	`SessionId, SessionConfig`
Responses:	SessionStatusMessage
Notes:	

Message:	3 (DestroySessionMessage)
Description:	This message is sent from a client to destroy a session.
Contents:	`SessionId, SessionConfig`
Responses:	SessionStatusMessage
Notes:	The router at this point should release all resources related to the session

Message:	4 (CreateLeaseSetMessage)
Description:	This message is sent in response to a RequestLeaseSetMessage and contains all of the `Lease` structures that should be published to the I2NP Network Database.
Contents:	`SessionId, SigningPrivateKey, PrivateKey, LeaseSet`
Responses:	None, or SessionStatusMessage if the session is invalid
Notes:	The `SigningPrivateKey` matches the `SigningPublicKey` from within the `LeaseSet`, as does the `PrivateKey` with the `PublicKey`. The Signing keys are necessary to allow the router to destroy the `LeaseSet` if the client goes offline, and the normal keys are necessary for decrypting garlic routed messages. The `LeaseSet` granted may include `Lease` structures for tunnels pointing at another router if the client is actively connected to multiple routers with `Leases` granted to each.

Message:	5 (SendMessageMessage)

Description:	This is how a client sends a message (the payload) to the `Destination`. The API implementation of this access layer should provide transparent encryption of the payload to the `Destination`'s public key in the same way that it should handle transparent decryption of the payload when received via `MessagePayloadMessage` below - the private keys are never be given to the router. As such, the router shouldn't do any encryption of the message beyond transport layer encryption, but the access libraries should.
Contents:	`SessionId, Destination, Payload`
Responses:	MessageStatusMessage
Notes:	As soon as the SendMessageMessage arrives fully intact, the router should return a MessageStatusMessage stating that it has been accepted for delivery. Later on, based on the delivery guarantees of the session configuration, the router may additionally send back another MessageStatusMessage updating the status

Message:	6 (ReceiveMessageBeginMessage)
Description:	Request the router to deliver a message that it was previously notified of
Contents:	`SessionId, MessageId`
Responses:	MessagePayloadMessage
Notes:	The ReceiveMessageBeginMessage is sent as a response to a MessageStatusMessage stating that a new message is available for pickup. If the message id specified in the ReceiveMessageBeginMessage is invalid or incorrect, the router may simply not reply, or it may send back a DisconnectMessage.

Message:	7 (ReceiveMessageEndMessage)
Description:	Tell the router that delivery of a message was completed successfully and that the router can discard the message
Contents:	`SessionId, MessageId`
Responses:	None
Notes:	The ReceiveMessageBeginMessage is sent after a MessagePayloadMessage fully delivers a message's payload.

Router to Client Messages

These messages are sent from the router to normal client sessions

Message:	20 (SessionStatusMessage)
Description:	Instruct the client as to the status of its session
Contents:	`SessionId, 1 byte Integer status`
Responses:	None
Notes:	Status values include zero for destroyed, one for created, two for updated, and three for invalid session.

Message:	21 (RequestLeaseSetMessage)

Description:	Request that a client authorize the inclusion of a particular set of inbound tunnels
Contents:	`SessionId, 1 byte Integer # tunnels, (RouterIdentity, TunnelId)*, Start Date, End Date`
Responses:	CreateLeaseSetMessage
Notes:	The response back with a `LeaseSet` may not contain all of these tunnels, and it may also contain additional ones.

Message:	22 (MessageStatusMessage)
Description:	Notify the client of the delivery status of a message
Contents:	`SessionId, MessageId, 1 byte Integer status, 4 byte Integer size`
Responses:	ReceiveMessageBeginMessage
Notes:	The known status values are 0 for message is available, 1 for accepted, 2 for best effort succeeded, 3 for best effort failed, 4 for guaranteed succeeded, 5 for guaranteed failed. The last 4 byte Integer specifies the size of the availabe message and is only relevant for status = 0.

Message:	31 (MessagePayloadMessage)
Description:	Deliver the payload of a message to the client
Contents:	`SessionId, MessageId, Payload`
Responses:	ReceiveMessageEndMessage
Notes:	

Client Admin to Router Messages

These messages are sent to the router by applications requiring administrative control of the router's operation. These pieces of functionality are not necessary for most applications.

Message:	8 (GetBandwidthLimitsMessage)
Description:	Request that the router state what its current bandwidth limits are
Contents:	`AuthenticationKey`
Responses:	BandwidthLimitsMessage
Notes:	

Message:	9 (SetBandwidthLimitsMessage)
Description:	Specify new bandwidth limits for the router
Contents:	`AuthenticationKey, BandwidthLimits`
Responses:	None
Notes:	

Message:	10 (GetTrustedRoutersMessage)

Description:	Request that the router state what its trusted routers are
Contents:	`AuthenticationKey`
Responses:	TrustedRoutersMessage
Notes:	

Message:	11 (SetTrustedRoutersMessage)
Description:	Specify new trusted peers for the router
Contents:	`AuthenticationKey, 2 byte Integer # hashes, that many Hashes`
Responses:	None
Notes:	Each Hash is the SHA256 of the RouterIdentity to be trusted.

Message:	12 (GetAddressVisibilityMessage)
Description:	Request that the router state whether its addresses are publicly known
Contents:	`AuthenticationKey`
Responses:	AddressVisibilityMessage
Notes:	

Message:	13 (SetAddressVisibilityMessage)
Description:	Specify whether the routers addresses should be publicly known
Contents:	`AuthenticationKey, Boolean`
Responses:	None
Notes:	If the Boolean is true, then the addresses should be visible

Message:	14 (GetAddressRotationPeriodMessage)
Description:	Request that the router state after how long addresses are rotated
Contents:	`AuthenticationKey`
Responses:	AddressRotationPeriodMessage
Notes:	

Message:	15 (SetAddressRotationPeriodMessage)
Description:	Specify whether the routers addresses should be publicly known
Contents:	`AuthenticationKey, 2 byte Integer`
Responses:	None
Notes:	The Integer specifies the number of seconds to wait before rotating, or 0 for never rotate.

Message:	16 (ForceAddressRotationMessage)

Description:	Instruct the router to rotate its addresses as soon as possible
Contents:	`AuthenticationKey`
Responses:	LastAddressRotationMessage
Notes:	

Message:	17 (GetPeersMessage)
Description:	Request that the router state what its known peer routers are
Contents:	`AuthenticationKey`
Responses:	PeersMessage
Notes:	

Message:	18 (SetPeersMessage)
Description:	Configure the set of peers the router knows about and will communicate with
Contents:	`AuthenticationKey, 2 byte Integer # peers, then that many RouterInfo structures`
Responses:	None
Notes:	The Integer specifies the number of seconds to wait before rotating, or 0 for never rotate.

Message:	19 (SetPasswordMessage)
Description:	Set the new password to control the router
Contents:	`AuthenticationKey, new AuthenticationKey`
Responses:	None
Notes:	

Router to Client Admin Messages

These messages are sent by the router to clients running administrative sessions in response to their requests.

Message:	23 (BandwidthLimitsMessage)
Description:	Tell the client what the bandwidth limits are
Contents:	`BandwidthLimits`
Responses:	None
Notes:	

Message:	24 (TrustedRoutersMessage)
Description:	Tell the client what the trusted routers are
Contents:	`2 byte Integer # hashes, then those Hash structures`

Responses:	None
Notes:	The Hash structures are SHA256 calculated against the router's RouterIdentity

Message:	25 (AddressVisibilityMessage)
Description:	Tell the client whether its contact addresses are publicly visible
Contents:	`Boolean`
Responses:	None
Notes:	True for publicly visible

Message:	26 (AddressRotationPeriodMessage)
Description:	Tell the client how often its addresses are rotated
Contents:	`2 byte Integer`
Responses:	None
Notes:	Specifies the number of seconds after which the router is obliged to rotate its contact addresses. If this number is all zeroes, then the router is never obliged to rotate its contact addresses.

Message:	27 (LastAddressRotationMessage)
Description:	Tell the client when the last address rotation was
Contents:	`Date`
Responses:	None
Notes:	

Message:	28 (PeersMessage)
Description:	Provide the client with all of the routers that the local router is communicating with
Contents:	`2 byte Integer specifying the number of routers, then that many RouterInfo structures`
Responses:	None
Notes:	

Bidirectional Messages

These messages can be sent by routers or by clients operating in either administrative or normal mode.

Message:	29 (ReportAbuseMessage)
Description:	Tell the other party that they are under attack, potentially with reference to a particular messageId. If the router is under attack, the client may decide to migrate to another router, and if a client is under attack, the router may rebuild its routers or shitlist some of the peers that sent it messages delivering the attack.
Contents:	`SessionId, AbuseSeverity, AbuseReason, MessageId`

Responses:	None
Notes:	MessageId is 0 if irrelevant

Message:	30 (DisconnectMessage)
Description:	Tell the other party that there are problems and the current connection is about to be destroyed. This does not necessarily end a session.
Contents:	`String`
Responses:	None
Notes:	String specifies a description of why the disconnect occurred.

3. References

- Invisible Internet Network Protocol
- I2P Common Data Structures Specification