EPOCH Protocol client API C

Endpoint Oriented Channel application programming interface for C language

The header file epoch.h declares all the symbols defined in the next document.

struct epoch_s

epoch_s structure represents a connection state to the server. It has other fields that helps to build that state.

char ipv4[EPOCH_IPV4]

This field is used to set the IP version 4 of the server. If empty, then the IP version 6 is used to that end.

char ipv6[EPOCH_IPV6]

This field is used, if ipv4 field is empty, to set the servers IP version 6 address.

unsigned short port

Indicates the port in which the server is listening.

char key[EPOCH_KEYMAX]

Key field is a random generated session key by the user and will be used to encrypt/decrypt the transmitted data to and from the server.

char *pubkey

This is the public key of the server. It is used to encrypt the random generated session and transmit it.

int rdtmout

In absence of a keepalive or heartbeat mechanism, this field provides a timeout for reading operations. Once the timeout expire, the read function will return an error.

int cntmout

This timeout represents the amount of time that the client is willing to wait for establish the tcp connection before drop the attempt.

struct netconn *nc

This is a handle to an obscure structure that contains fundamental fields for the connection. It must not be manipulated directly.

enum epoch_mode

epoch_mode enumeration has the modes in which the EPOCH manage the flowing data.

SINGLE_THREAD_SNDFIRST

This mode indicate to the EPOCH API that the data will be sent first to the server, and once is done, the client will receive the data sent by the server.

SINGLE_THREAD_RCVFIRST

This mode indicate to the EPOCH API that the data will be received first from the server, and once is done, the client will send its data to the server.

MULTI_THREAD

This mode indicate to the EPOCH API that the data will be sent from the client to the server at the same time the data will be received from the server to the client.

typedef void (*callback_snd**)(void ***sndbuf**, int64_t ***sndlen**)**

Callback function prototype for sending data.

void *sndbuf

This parameter points to an internal buffer for the data to be sent. The client should write this buffer before the callback returns. Once the callback returns, the EPOCH API will send it.

int64_t *sndlen

This parameter points to an internal 64bit signed integer that will be used to know how much data is in the *sndbuf parameter. Before the callback returns, the client must set it to indicate how much data is in the buffer. The length must be less than or equal to the buffer size specified in the epoch_endpoint function call. Once the client set *sndlen to 0 the EPOCH API will consider the upload operation done.

typedef void (*callback_rcv**)(void ***rcvbuf**, int64_t* rcvlen**)**

Callback function prototype for receiving data.

void *rcvbuf

This parameter points to an internal buffer for the data to be received. The client must copy the received data to a second buffer before the callback returns and should continue doing this until there is no more data to receive.

int64_t rcvlen

This parameter is set by the EPOCH API and indicate how much data is stored in the buffer. Once this parameter is set to 0 by the EPOCH API, there is no more data to receive.

int epoch_endpoint**(struct epoch_s ***e**, const char ***endpt**, const char ***auth**, enum epoch_mode* mode**, int64_t** bufsz**, callback_snd** snd_callback**, callback_rcv** rcv_callback**)**

The epoch_endpoint function connect to the server and run the endpoint indicated by the *endpt parameter. If everything goes well, it will keep

calling snd_callback until there is no more data to send, and rcv_callback until there is no more data to receive.

struct epoch_s *e
: A pointer to initialized structured with the parameters to establish the connection to the server.

const char *endpt
: The endpoint ‘the program’ to execute at the server side with its parameters. An example could be like this: “/bin/ls -la” or just “/bin/ls”.

const char *auth
: The authentication token is used by the server to determine which endpoints ‘programs’ and parameters can run.

int64_t bufsz
: Sets the buffer size for communication. It has to be greater than or equal to 65536.

enum epoch_mode mode
: Indicates the mode in which the server and the client will send and receive data.

callback_snd snd_callback
: Specifies the callback for the sending operation. Client to server. If set to null, the EPOCH API will send immediately a 0 to the server indicating the end of sending transmission.

callback_rcv rcv_callback
: Specifies the callback for receiving operation. Server to client. If set to NULL, the EPOCH API will silently discard the arriving data until receive 0 from the server.

The return value is EPOCH_SUCCESS for the correct execution of the entire process. Otherwise it will return any of the following errors:

EPOCH_EBUFNULL
: An error has occurred creating communication buffers.

EPOCH_ECONNECT
: Error trying connecting to the server.

EPOCH_ESEND
: Error sending data and the connection is terminated.

EPOCH_ERECV
: Error receiving data and the connection is terminated.

EPOCH_EENDPTAUTH
: Error in endpoint authentication process. This could be either because the endpoint an its parameter are not allowed or because the authentication token is wrong

EPOCH_EMTINIT
: If mode is set to MULTI_THREAD, this error indicate that an error has occurred in the multi-thread environment initialization.

Note of caution: The client and the endpoint must keep in mine that one write or read at one end is not necessary equivalent to one read or write at the other end. It’s crucial for both to continue reading, in the case of client, until the rcvlen parameter of the rcv_callback function is set to 0 by the EPOCH API; in the case of endpoint, until **read primitive** returns EOF or any signal indicating its standard input closure.

At the endpoint programming is important to close the standard output and the standard error descriptors once writing operations are done. This way we tell the EPOCH server we have finish doing writing.

int resolvhn**(const char ***host**, char ***ip**, int** v6**, int** timeout**)**

This function is capable of resolve and ipv6 or ipv4 from a domain name. If the function success it return 0, otherwise -1.

const char *host

The domain name to resolve, example “google.com”.

char *ip

A buffer greater enough to store an ipv4 or an ipv6 in dotted format. “127.0.0.1” or “::1”.

int v6

If set to non-zero indicates that the resolving must be done in the ipv6 address space. Otherwise, an ipv4 will be resolved.

int timeout

A timeout in seconds indicating the time the client is willing to wait until resolution. If is 0, then there is no timeout.