NAME

BIO_s_datagram, BIO_new_dgram, BIO_ctrl_dgram_connect, BIO_ctrl_set_connected, BIO_dgram_recv_timedout, BIO_dgram_send_timedout, BIO_dgram_get_peer, BIO_dgram_set_peer, BIO_dgram_get_mtu_overhead - Network BIO with datagram semantics

SYNOPSIS

```
#include <openssl/bio.h>

BIO_METHOD *BIO_s_datagram(void);
BIO *BIO_new_dgram(int fd, int close_flag);

int BIO_ctrl_dgram_connect(BIO *bio, const BIO_ADDR *peer);
int BIO_ctrl_set_connected(BIO *bio, const BIO_ADDR *peer);
int BIO_dgram_recv_timedout(BIO *bio);
int BIO_dgram_send_timedout(BIO *bio);
int BIO_dgram_get_peer(BIO *bio, BIO_ADDR *peer);
int BIO_dgram_set_peer(BIO *bio, const BIO_ADDR *peer);
int BIO_dgram_get_mtu_overhead(BIO *bio);
```

DESCRIPTION

BIO_s_datagram() is a BIO implementation designed for use with network sockets which provide datagram semantics, such as UDP sockets. It is suitable for use with DTLSv1.

Because **BIO_s_datagram()** has datagram semantics, a single **BIO_write()** call sends a single datagram and a single **BIO_read()** call receives a single datagram. If the size of the buffer passed to **BIO_read()** is inadequate, the datagram is silently truncated.

When using **BIO_s_datagram()**, it is important to note that:

- This BIO can be used with either a connected or unconnected network socket. A connected socket is a network socket which has had **BIO_connect**(3) or a similar OS-specific function called on it. Such a socket can only receive datagrams from the specified peer. Any other socket is an unconnected socket and can receive datagrams from any host.

- Despite their naming, neither **BIO_ctrl_dgram_connect()** nor **BIO_ctrl_set_connected()** cause a socket to become connected. These controls are provided to indicate to the BIO how the underlying socket is configured and how it is to be used; see below.

- Use of **BIO_s_datagram()** with an unconnected network socket is hazardous hecause any successful call to **BIO_read()** results in the peer address used for any subsequent call to

BIO_write() being set to the source address of the datagram received by that call to **BIO_read()**. Thus, unless the caller calls **BIO_dgram_set_peer()** immediately prior to every call to **BIO_write()**, or never calls **BIO_read()**, any host on the network may cause future datagrams written to be redirected to that host. Therefore, it is recommended that users use **BIO_s_dgram()** only with a connected socket. An exception is where **DTLSv1_listen**(3) must be used; see **DTLSv1_listen**(3) for further discussion.

Various controls are available for configuring the **BIO_s_datagram()** using **BIO_ctrl**(3):

BIO_ctrl_dgram_connect (BIO_CTRL_DGRAM_CONNECT)

This is equivalent to calling **BIO_dgram_set_peer**(3).

Despite its name, this function does not cause the underlying socket to become connected.

BIO_ctrl_set_connected (BIO_CTRL_SET_CONNECTED)

This informs the **BIO_s_datagram()** whether the underlying socket has been connected, and therefore how the **BIO_s_datagram()** should attempt to use the socket.

If the *peer* argument is non-NULL, **BIO_s_datagram()** assumes that the underlying socket has been connected and will attempt to use the socket using OS APIs which do not specify peer addresses (for example, **send**(3) and **recv**(3) or similar). The *peer* argument should specify the peer address to which the socket is connected.

If the *peer* argument is NULL, **BIO_s_datagram()** assumes that the underlying socket is not connected and will attempt to use the socket using an OS APIs which specify peer addresses (for example, **sendto**(3) and **recvfrom**(3)).

BIO_dgram_get_peer (BIO_CTRL_DGRAM_GET_PEER)

This outputs a **BIO_ADDR** which specifies one of the following values, whichever happened most recently:

- The peer address last passed to **BIO_dgram_set_peer()**, **BIO_ctrl_dgram_connect()** or **BIO_ctrl_set_connected()**.
- The peer address of the datagram last received by a call to **BIO_read()**.

BIO_dgram_set_peer (BIO_CTRL_DGRAM_SET_PEER)

Sets the peer address to be used for subsequent writes to this BIO.

Warning: When used with an unconnected network socket, the value set may be modified by future calls to **BIO_read**(3), making use of **BIO_s_datagram()** hazardous when used with unconnected network sockets; see above.

BIO_dgram_recv_timeout (BIO_CTRL_DGRAM_GET_RECV_TIMER_EXP)
: Returns 1 if the last I/O operation performed on the BIO (for example, via a call to **BIO_read**(3)) may have been caused by a receive timeout.

BIO_dgram_send_timedout (BIO_CTRL_DGRAM_GET_SEND_TIMER_EXP)
: Returns 1 if the last I/O operation performed on the BIO (for example, via a call to **BIO_write**(3)) may have been caused by a send timeout.

BIO_dgram_get_mtu_overhead (BIO_CTRL_DGRAM_GET_MTU_OVERHEAD)
: Returns a quantity in bytes which is a rough estimate of the number of bytes of overhead which should typically be added to a datagram payload size in order to estimate the final size of the Layer 3 (e.g. IP) packet which will contain the datagram. In most cases, the maximum datagram payload size which can be transmitted can be determined by determining the link MTU in bytes and subtracting the value returned by this call.

  The value returned by this call depends on the network layer protocol being used.

  The value returned is not fully reliable because datagram overheads can be higher in atypical network configurations, for example where IPv6 extension headers or IPv4 options are used.

BIO_CTRL_DGRAM_SET_DONT_FRAG
: If *num* is nonzero, configures the underlying network socket to enable Don't Fragment mode, in which datagrams will be set with the IP Don't Fragment (DF) bit set. If *num* is zero, Don't Fragment mode is disabled.

BIO_CTRL_DGRAM_QUERY_MTU
: Queries the OS for its assessment of the Path MTU for the destination to which the underlying network socket, and returns that Path MTU in bytes. This control can only be used with a connected socket.

  This is not supported on all platforms and depends on OS support being available. Returns 0 on failure.

BIO_CTRL_DGRAM_MTU_DISCOVER
: This control requests that Path MTU discovery be enabled on the underlying network socket.

BIO_CTRL_DGRAM_GET_FALLBACK_MTU

Returns the estimated minimum size of datagram payload which should always be supported on the BIO. This size is determined by the minimum MTU required to be supported by the applicable underlying network layer. Use of datagrams of this size may lead to suboptimal performance, but should be routable in all circumstances. The value returned is the datagram payload size in bytes and does not include the size of layer 3 or layer 4 protocol headers.

BIO_CTRL_DGRAM_MTU_EXCEEDED

Returns 1 if the last attempted write to the BIO failed due to the size of the attempted write exceeding the applicable MTU.

BIO_CTRL_DGRAM_SET_NEXT_TIMEOUT

Accepts a pointer to a **struct timeval**. If the time specified is zero, disables receive timeouts. Otherwise, configures the specified time interval as the receive timeout for the socket for the purposes of future **BIO_read**(3) calls.

BIO_CTRL_DGRAM_SET_PEEK_MODE

If **num** is nonzero, enables peek mode; otherwise, disables peek mode. Where peek mode is enabled, calls to **BIO_read**(3) read datagrams from the underlying network socket in peek mode, meaning that a future call to **BIO_read**(3) will yield the same datagram until peek mode is disabled.

BIO_new_dgram() is a helper function which instantiates a **BIO_s_datagram()** and sets the BIO to use the socket given in *fd* by calling **BIO_set_fd()**.

RETURN VALUES

BIO_s_datagram() returns a BIO method.

BIO_new_dgram() returns a BIO on success and NULL on failure.

BIO_ctrl_dgram_connect(), **BIO_ctrl_set_connected()**, **BIO_dgram_get_peer()**, **BIO_dgram_set_peer()** return 1 on success and 0 on failure.

BIO_dgram_recv_timedout() and **BIO_dgram_send_timedout()** return 0 or 1 depending on the circumstance; see discussion above.

BIO_dgram_get_mtu_overhead() returns a value in bytes.

SEE ALSO

DTLSv1_listen(3), **bio**(7)

COPYRIGHT