libnet Overview

PlayStation®Vita Programmer Tool Runtime Library Release 3.50

Document serial number: 000004892117

Document serial number: 000004892117

Table of Contents

1 Library Overview

Scope of This Document

This document describes libnet, which is used to implement a socket layer. libnet supports an IPv4 protocol stack based on NetBSD, with additional original features. This document also covers the differences from BSD programming as well as internal operations.

Purpose and Features

libnet is a library of modules loaded in the user space and kernel space required to realize a socket layer and supports an IPv4 protocol stack. IPv6 and IPsec features are not and will not be supported.

The programming is similar to BSD network programming because the libnet is based on NetBSD, with additional original features.

Main Features

The following protocols are supported by libnet.

- DNS stub resolver (with cache), DHCP client
- TCP (urgent data not supported), UDP, UDPP2P, TCP over UDPP2P
- IPv4, ARP, ICMP, IGMP end node
- PPP/PPPoE

The following APIs are provided on libnet.

- BSD socket API-compatible functions and extensions

Embedding into a Program

The files required for using libnet are as follows.

Filename	Description
net.h	Header file (Avoid directly including the header file in the net directory.)
libSceNet_stub.a libSceNet_stub_weak.a	Stub library file
libSceNetDebug_stub.a libSceNetDebug_stub_weak.a	Stub library file (These files are for development purposes and cannot be used for master packages)

Include net.h in the source program. Various header files will be automatically included as well.

libnet is only provided in the PRX format. To use libnet, make sure to statically link with either libSceNet_stub.a or libSceNet_stub_weak.a. The PRX modules are in storage managed by the system software and can be loaded/unloaded with libsysmodule API.

For details on the PRX format, refer to "libsysmodule Overview".

libSceNetDebug_stub.a and libSceNetDebug_stub_weak.a can be used only for development purposes as substitutes for libSceNet_stub.a and libSceNet_stub_weak.a.

libSceNetDebug_stub.a and libSceNetDebug_stub_weak.a include the following functions in addition to the functions included by libSceNet_stub.a and libSceNet_stub_weak.a, respectively, to allow network emulation.

- `sceNetEmulationGet()`
- `sceNetEmulationSet()`

Sample Programs

Sample programs for libnet are as follows.

sample_code/network/api_net/console/

This sample shows the basic usage of the network console.

For information on the sample, refer to the sample readme.

2 Using the Library

Basic Procedure

Network communication processes using libnet are based on BSD network programming. (Refer to commercially available references regarding BSD.)

Differences from BSD Programming

Note the following differences from BSD network programming when using libnet.

Starting Up and Terminating the Network

Proper network initialization and termination are required and done by calling `sceNetInit()` and `sceNetTerm()`, respectively.

Starting Up and Terminating the Network Interface

The network I/F is started up and terminated by the system software.

Error Code Handling

With BSD socket API, -1 is returned for errors, and an error code is stored in `errno` to indicate the cause of the error. libnet stores network error codes in `sce_net_errno`. In addition, as a general rule, a negative value returned for the function indicates that an error occurred, and the details of this error can be found in the return value or `sce_net_errno`. The error code returned as a macro takes the form of `SCE_NET_ERROR_EAGAIN`, for example, and `SCE_NET_EAGAIN` for `sce_net_errno()`. Error code macros reference the header file that is exclusively provided for the network.

Sce Error Code Handling

Return values of functions are pursuant to the configuration of Sce error codes. On the other hand, `sce_net_errno` is pursuant to network errors.

Include

Include net.h for all functions.

Naming Conventions and Exceptions for APIs, Structures, and Macros

As a basic policy, BSD system names have a prefix added. The BSD system underscore (_) used as a separator in the names of APIs and structures is removed and the first letter of the word is changed to upper case.

	libnet	BSD System
API names	`sceNet*()`	`*()`
Structure names	`SceNet*`	`struct *`
Macro names	`SCE_NET_*`	`*`

The following are the exceptions.

	libnet	BSD System
Socket close	`sceNetSocketClose()`	`close()`
Multiplexing I/O control	`sceNetEpollControl()`	`epoll_ctl()`
Multiplexing I/O close	`sceNetEpollDestroy()`	`close()`

	libnet	BSD System
DNS resolver	`sceNetResolver*()` (refer to the "DNS Resolver" item)	Example: `gethostbyname()`
_t type definition	Example: `SceNetInAddr_t`	Example: `in_addr_t`

Features That Can Be Replaced

The following features, which can be replaced with a separate feature from among the features supported by the BSD system, are not supported.

	libnet Features	BSD System Features Not Supported
Network multiplexing I/O	`sceNetEpoll*()`	`select(),poll()`
Conversion of IPv4 address from text format to binary format	`sceNetInetPton()`	`inet_addr()`
Conversion of IPv4 address from binary format to text format	`sceNetInetNtop()`	`inet_ntoa()`

The socket ID set operation (such as `FD_SET()`) used with the `select()` system may be used as a bit operation, such as to determine whether or not to use a socket, and therefore, it is supported.

libnet ID (`SceNetId`)

libnet allocates libnet IDs to identify sockets and DNS resolvers.

The following is the range of values. Socket ID set operations (such as `SCE_NET_FD_SET()`) can be used, but it will not meet the future value range, and therefore, a program cannot be written expecting a value range.

	Value Range
Socket	0 - 1023
Network multiplexing I/O	2000 - 2255
DNS resolver	4000 - 4255
Dump	6000 - 6015

The libnet ID use a negative value to indicate an invalid value. IDs increase in the order by which the items are created. When the maximum value of each ID is exceeded, the value returns to the minimum value. Although this is a mechanism to decrease misses in debugging or in development, do not create a program that builds upon this mechanism. The number of IDs that can be created at one time is 128 sockets at the maximum excluding the IDs reserved by the system. However, in actuality, the maximum number of IDs may not be created at the same time because the creation of IDs is limited by the usage status of the kernel work area. Refer also to the "Internal Operations" chapter in this document.

Unblocking Block Functions (Abort)

Unblocking is performed for libnet IDs. Call the API corresponding to the target ID.

	API
Socket	`sceNetSocketAbort()`
Network multiplexing I/O	`sceNetEpollAbort()`
DNS resolver	`sceNetResolverAbort()`
Dump	`sceNetDumpAbort()`

This is applied to blocks including blocks waiting for a timeout. In addition, when the target ID is not blocked at the time the abort API is called, you can select whether to treat that as an abort failure or treat that as aborted when the target ID called the next block-waiting API.

Ending Use of libnet IDs

To end use of libnet IDs, call the API corresponding to each ID.

	API
Socket	`sceNetSocketClose()`
Network multiplexing I/O	`sceNetEpollDestroy()`
DNS resolver	`sceNetResolverDestroy()`
Dump	`sceNetDumpDestroy()`

Timeout Values

With the exception of the linger time, this is handled in microseconds. Areas requiring a BSD system `struct timeval` have been changed to be handled in `int` type microseconds.

The minimum time unit is 10 milliseconds, and all values less than that (not including 0) are handled as though specified at 10 milliseconds.

DNS Resolver

This is an original API feature. It supports hostname to IP address conversion (forward lookup), IP address to hostname conversion (reverse lookup), blocking features, and non-blocking features. End waiting for the non-blocking process uses network multiplexing I/O. In addition, when multiple records corresponding to a name exist, the result of the first record is handled.

Debugging Name

To simplify debugging, a string of up to 32 characters (including NULL end) can be specified for the first argument of the following APIs.

- `sceNetSocket()`
- `sceNetEpollCreate()`
- `sceNetResolverCreate()`
- `sceNetDumpCreate()`

Because this is the debugging name, avoid specifying strings with confidential data, such as a key.

Thread Context Release

libnet automatically releases internal thread information created implicitly, so no special process is needed.

Multithread Support

libnet can call APIs supporting multithreading from multiple threads at the same time.

However, when multiple threads attempt to receive data from the same socket, only one of the threads will be able to receive data, and the result of the whole operation will be undefined. When multiple threads attempt to send data to the same socket, the result is dependent on the order in which threads call the send function. Note that thread scheduling may affect the data order when data is sent from multiple threads in TCP or TCP over UDPP2P.

Log Access Features

libnet provides packet dump access features in tcpdump format as APIs.

The IP/TCP/UDP checksums from the sender of the packet dump may not be calculated depending on device conditions.

MAC Address Settings

A MAC address cannot be set.

Reference Information

Broadcast Sending

When data is sent as a broadcast address to 255.255.255.255 (`SCE_NET_INADDR_BROADCAST`), the broadcast address is converted to a broadcast address calculated from the net mask, and data is sent to the converted broadcast address. For this reason, normal applications do not have a problem with selecting the above address as the broadcast address. To send the data without converting the broadcast address, the `SCE_NET_SO_ONESBCAST` socket option of `sceNetSetsockopt()` can be used.

Continuous Sending of UDP Socket

Data that is sent from an application and requires ARP resolution of the sender's address before being sent to a device via libnet is held in the ARP unresolution queue, and after ARP resolution is completed, the data is actually sent from the device. Note that when sending data by UDP socket, the application sending function will end normally even if ARP resolution is not completed.

When data is continually sent by a UDP socket until ARP resolution is completed, this indicates that there is a data overflow of the ARP unresolution queue and this overflow data is being properly sent to the application, but in fact, the data is not being sent but will be dropped.

The most recent sending data is held in this ARP unresolution queue, and there is one instance in the current implementation. Normally, ARP resolution is required at the time of the first communication to the sender's address, but if there is no communication within 20 minutes, ARP resolution must be performed again.

Because UDP is an unreliable protocol, the dropping of packets before being sent to a device due to ARP unresolution and the loss of packets along the path are considered to be essentially the same, and if resending of data is required, this is done on the application side.

UDPP2P and TCP Over UDPP2P Socket Loopback and Broadcasts

The sockets of UDPP2P and TCP over UDPP2P perform loopback like other sockets. For sockets for which either the encryption option or the signature option is specified (including the encryption and signature options specified for `sceNetSend()` and other send functions), refer to the following table.

Internet communication mode/ ad hoc communication mode

	To Local Terminal (*1)	Broadcast
Sockets without the encryption and signature options	Perform	Perform
Sockets with either the encryption option or the signature option	Perform	Do not perform (*2)

(*1) Includes the loopback address and the IP address allocated to the local terminal.

(*2) The local terminal does not receive the data which are broadcasted in the ad hoc communication mode, but the other terminal can receive such data.

3 Internal Operations

Memory Management

libnet has a fixed kernel work area of 1 MiB. In kernel space, memory relating to the network is consumed from this work area. In user space, memory specified with `sceNetInit()` will be used.

Internal Threads

The following threads will be created on the kernel space.

- Interrupt processes
- Soft clock

The interrupt processing thread is activated by network device interrupts, such as, the receiving of data.

The soft clock thread runs in cycles of 10 milliseconds, and is also activated by events. This means it is activated at least once every 10 milliseconds.

Scheduling within the kernel is executed pursuant to the priority order of the thread calling the libnet network function. However, like the TCP processing of resending data, the internal processing of the kernel is executed by the thread assigned to the kernel space.

DNS Addresses and Operation

The maximum number of DNS addresses is two, for the primary and secondary addresses. DNS addresses are used in a round-robin fashion.

If there is a DNS address explicitly specified in network settings, it will take priority over the DNS address obtained with DHCP. Even if a secondary address is not explicitly set, the DNS address explicitly set will be used and the primary DNS address obtained with DHCP will not be used.

Socket Buffer Sizes

A buffer for sending data and another for receiving data exist per socket. The send and receive socket buffers do not necessarily have the size corresponding to the initial value allocated to them. Consider the buffer size indicating the maximum value.

The initial value of each socket buffer is as follows. `SCE_NET_SO_SNDBUF` and `SCE_NET_SO_RCVBUF` of `sceNetSetsockopt()` are respectively used to change the initial value.

Socket Type	Send Socket Buffer	Receive Socket Buffer
TCP, TCP over UDPP2P	65535 bytes	65535 bytes
UDP, UDPP2P	9216 bytes	No upper limit (Note)
RAW	- (No meaning)	No upper limit (Note)

Note: In the current implementation, the size is 120 KiB, but this value cannot be assumed as the actual value depends on the number of packets and the data size that are received.

Normally the socket buffer size need not be changed, but in the following cases, changing it may be considered.

- Improvement of kernel work area consumption
- Bandwidth limit

Particularly as regards the kernel work area consumption, if the amount of communication per unit time during expected communication is large owing to the way the application is designed, perform calculations by assuming that the maximum memory amount of the socket buffers of each socket is consumed.

Kernel Work Area Consumption

The amount of the kernel work area that is consumed is determined, to some degree, by the created number of sockets and received packets. The application can assume that there is enough kernel work area left, if the minimum value for the memory bytes currently remaining [obtainable by `sceNetGetStatisticsInfo()`] after the expected communication operation of the application is approximately 128 KiB. Even if this value is less than 128 KiB, in most cases, it will not be a problem if the amount of memory bytes currently available is approximately 128 KiB.

When a large amount of the kernel work area is consumed, each network function will return an insufficient memory error to the application. Although memory consumption can be recovered, for example, by closing the applicable socket, it is recommended that memory consumption be controlled so that it does not become too large. Consider the following.

- Reduce the maximum number of sockets
- Reduce the send and receive socket buffer sizes using `sceNetSetsockopt()`
- For TCP, reduce the number of connections entering the TIME_WAIT state

The number of TCP connections entering the TIME_WAIT state can be reduced as follows. Use of the linger option is not recommended.

- Rather than create and destroy TCP connections many times within a short amount of time, reuse the connections, or use UDP instead
- After confirming that FIN has been received first [by confirming that the return value for receiving data on the other end's processing that is equivalent to `sceNetShutdown()` or `sceNetSocketClose()` is 0], perform `sceNetShutdown()` or `sceNetSocketClose()` processing

When the kernel work area becomes insufficient, each function will return the `SCE_NET_ENOBUFS` error.

Hints for Looking into Network Problems

Check Memory Consumption

Check the current status, including the amount of memory consumed, using `sceNetShowIfconfig()`. An output example is given below.

```
lo0 (index=1)
   inet:127.0.0.1/255.0.0.0
   UP MULTICAST MTU:33192(max:33192)
eth0 (index=2)
   Type:Ethernet HWaddr:XX:XX:XX:XX:XX:XX
   inet:0.0.0.0/0.0.0.0 Bcast:0.0.0.0
   UP RUNNING BROADCAST MULTICAST MTU:1500(max:1500)
   RX packets:26(1812 bytes) errors:0 dropped:0(emu 0)
      bcast:1(60 bytes) mcast:1(60 bytes)
   TX packets:24(1448 bytes) errors:0 dropped:0(emu 0)
      bcast:1(32 bytes) mcast:1(32 bytes)
   time: 0:00:12
   ICM: 0/10 prio= 1 STARTED
   emu: seed:ac84f3a4
      send drop:0%(0ms, 0ms) delay:100+-0ms order:0%(0ms) dup:0%
          bps:0 size:0- pattern:ffffffff ffffffff
```

```
      recv drop:0%(0ms, 0ms) delay:0+-0ms order:0%(0ms) dup:0%
          bps:0 size:0- pattern:00000000 00000000
dns addresses
   aaa.bbb.ccc.ddd, 0.0.0.0
statistics
   kernel: free_size= 990368/free_min= 987904
   libnet: free_size=  16252/free_min=  16252
   packet: 0, 3
```

"RX" indicates the receiving side, and "TX" indicates the sending side. "packets:" indicates the number of packets and (in parenthesis) their total byte size, "errors" indicates the number of error packets, and "dropped:" indicates the number of packets dropped and (in parenthesis) the number of packets dropped by network emulation.

"time:" indicates the interface startup time in hours, minutes, and seconds.

"ICM:" indicates the status of the intermittent connection. In this example, the non-communication time during intermittent connection is set to 10 seconds, indicating that 0 seconds have elapsed since the last communication. "prio=1 STARTED" indicates that the intermittent connection mode is enabled and that the current status is the communication start status. If the intermittent connection mode is enabled and the non-communication time has elapsed, the "prio=1 STANDBY" status is indicated.

"emu:" indicates the status of network emulation. Random seed indicates ac84f3a4, and the settings of both the sending side and the receiving side indicate packet loss (rate, loss duration, pass duration), packet delay (delay, jitter), packet order (rate, wait time), packet duplication, bandwidth limit, packet size limit, policy pattern (system side, application side) in sequence.

"kernel:" and "libnet:" indicate the memory status of the kernel work area and the libnet work area, respectively.

"packets:" indicates the numbers of packets held currently in the protocol stack. The difference between the former and the latter is the system calculation method for debugging. For applications, mainly refer to the latter number.

Check Information of the Sockets Being Created

Check the status of sockets using, for example, sceNetShowNetstat(). An output example is given below.

```
ID:  24 (udp,61)  *..*..  R-Q: 0     S-Q: 0     OPENED SceNetDhcp
  Local *:68              Remote *:*
ID:  22 (raw, 0)  **....  R-Q: 0     S-Q: 0     OPENED ping
  Local *:*               Remote *:*
ID:   4 (raw, 0)  *.....  R-Q: 0     S-Q: 0     OPENED ping
  Local *:*               Remote *:*
```

Two lines are displayed for each socket.

- First line: Socket ID, socket type, network emulation policy number, status flag, data size of receiving queue, data size of sending queue, state, debugging name
- Second line: Local connection information, remote connection information

The state flags have the following meanings.

- Own socket, socket receive wait, socket send wait, epoll receive wait, epoll send wait, wake signal generation for intermittent connection

If the above meanings are true, an asterisk (*) is displayed, and if the above meanings are false, a period (.) is displayed.

epoll wait indicates the state of waiting for a socket event with sceNetEpollWait() and sceNetEpollWaitCB().

4 Network Emulation

Overview

The network emulation feature is provided on the Development Kit and Testing Kit for emulating packet losses and delays that occur on the network for packets that are exchanged by UDP or TCP. Use this feature to test whether the application operates properly when the communication status is unstable.

The following are the methods for setting when to generate a packet loss or delay.

- Call `sceNetEmulationSet()` from within an application program
- "★Debug Settings" in the system software
- psp2ctrl utility path

The frequency by which to generate a packet loss or delay can be set as desired. Target packets can also be specified so as to generate a packet loss for a specific socket.

The conditions under which network emulation is applied are as follows.

	Development Kit Development Mode	Development Kit Release Mode	Testing Kit	Retail Unit
`sceNetEmulationSet()`	Valid	Invalid	Invalid	-
★Debug Settings	Valid	Valid (SDK 1.810 or later)	Valid (SDK 1.810 or later)	-
psp2ctrl Utility	Valid	-	-	-

”Valid” and “Invalid” respectively indicate that network emulation is applied and not applied.
“-“ indicates that the feature is not included, and thus means the same as “Invalid”.

> **Note**
> The **Development Mode** and **Release Mode** settings in the Development Kit can be changed in **Release Check Mode**. For **Release Check Mode**, refer to the "Development Kit Neighborhood Settings Guide" document.

★Debug Settings and psp2ctrl utility can be set at the desired timing, but calling `sceNetEmulationSet()` returns an error under the following conditions, making the network emulation settings invalid (SDK 2.000 or later).

- When using **★Debug Settings** (when not "Disabled")
- When a preset value for network emulation is set with the psp2ctrl utility

Version

The version of the network emulation feature explained below is 1.04 (0x0104 in the 16-bit notation).

The version history of the network emulation feature is as follows.

Version	Corresponding SDK	Description
1.00	0.920	Initial release
1.02	0.995	Added 4 as a specifiable preset value
1.03	0.996	Added bandwidth limit feature of the receiving side
1.04	1.500	Added bandwidth limit option to the parameter flag

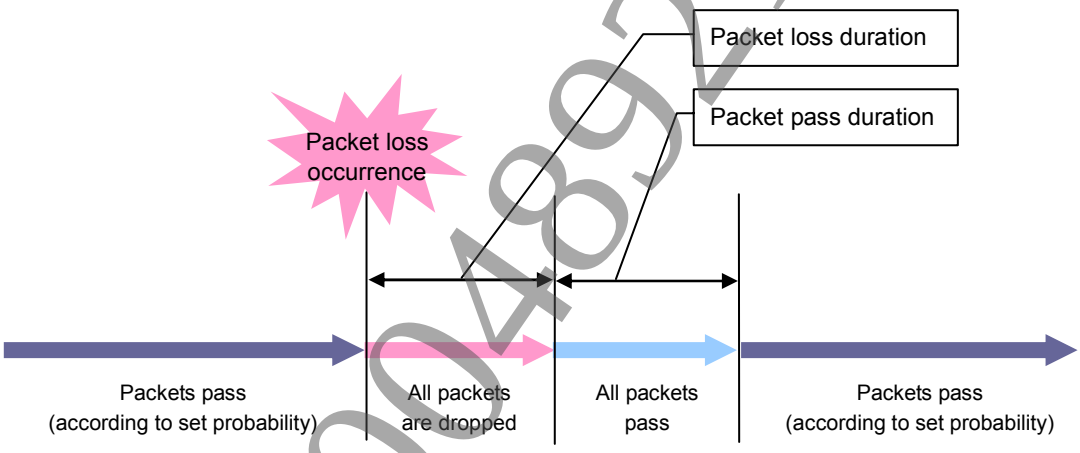
Emulation Parameters

The parameters that can be set for network emulation are listed below. Since the parameter names correspond to each of the members of `SceNetEmulationParam` and `SceNetEmulationData`, refer also to "libnet Reference".

Packet Loss

Parameter	send/recv	Description	Specifiable Values
Packet loss (random) "*drop_rate*"	send/recv	Specifies the probability of packet loss occurrence.	0 to 100% 1% unit Initial value: 0
Packet loss duration "*drop_duration*"	send/-	When packet loss occurs, the packet loss happens for the duration that is specified here. It is possible to reproduce consecutive packet losses. When 0 is specified, this parameter is applied to only that 1 packet.	0 to 8000 ms 1 ms unit Initial value: 0
Packet pass duration "*pass_duration*"	send/-	Specifies the duration for which packet loss does not occur, following the lapse of the packet loss duration.	0 to 8000 ms 1 ms unit Initial value: 0

Figure 1 Packet Loss Parameters



Packet Delay

Parameter	send/recv	Description	Specifiable Values
Packet delay (fixed) "*delay_time*"	send/-	Causes the specified delay for all the packets	0 to 8000 ms 1 ms unit Initial value: 0
Packet jitter "*delay_jitter*"	send/-	Specifies variation in the delay specified with *delay_time*. This causes delay to occur at random in the range of (*delay_time*) ± (*delay_jitter*). However, the packet sequence remains unchanged.	0 to (*delay_time*) ms 1 ms unit Initial value: 0

Packet Order

Parameter	send/recv	Description	Specifiable Values
Packet reordering "*order_rate*"	send/-	Specifies the reordering probability of the packet sequence. The packet at the beginning of the send queue is transferred to the end of the queue with the specified probability. However, reordering does not occur for the next packet after the packet that has been reordered. Moreover, note that reordering is actually performed by setting the *order_delay_time* suitably.	0 to 100 % 1 % unit Initial value: 0
Packet reordering wait time "*order_delay_time*"	send/-	Specifies the maximum wait time to be waited until the next packet arrives for the case when reordering occurred due to the probability but actual packet reordering could not be done because there was no other packet in the send queue. Reordering is done as soon as the next packet arrives, and the wait time is cancelled.	0 to 8000 ms 1 ms unit Initial value: 0

Packet Duplication

Parameter	send/recv	Description	Specifiable Values
Packet duplication "*duplication_rate*"	send/-	Specifies the packet duplication probability. If duplication occurs, the same packet is sent once.	0 to 100 % 1 % unit Initial value: 0

Limit

Parameter	send/recv	Description	Specifiable Values
Bandwidth limit "*bps_limit*"	send/recv	Specifies the bandwidth limit in bps. 0 bps is considered as no limit. If the limit value is exceeded on the sending side, the data being at that time will be stacked in the send queue. On the other hand, on the receiving side, the data (packet) received when the limit value is exceeded will be dropped as it is. Refer also to the "Caution" section in the Network Emulation chapter.	1024 to 1,073,741,824 (=2^30) bps 1 bps unit Initial value: 0

(Supplement) Note that sending and receiving data of an application stops for the communication with sent or received data amount exceeding the bandwidth limit constantly (not momentarily). The time that the application takes to actually recognize the stop of the sending and receiving data varies depending on the increasing tendency of the sent or received bytes, which is used by the system to determine whether the data amount exceeds the bandwidth limit.

(Reference) By enabling the bandwidth limit option of the parameter flag, the operation can be changed to send and receive data at fixed intervals instead of stopping the sending and receiving data completely when the bandwidth limit is exceeded.

Applied Conditions

Parameter	send/recv	Description	Specifiable Values
Policy pattern "*system_policy_pattern*" "*game_policy_pattern*"	send/recv	Applies an emulation feature to the send packets from a socket that has a policy number that has been specified as active. For details, refer to the "Policies" section below.	Bit flag of each 32 bits Initial value: 0
Policy flag "*policy_flags*[64]"	send/-	Not used at present	Reserved (always 0)
Packet size "*lower_size_limit*" "*upper_size_limit*"	send/-	Applies an emulation feature to the packets whose size is larger than or equal to the specified *lower_size_limit* and smaller than or equal to the specified *upper_size_limit*. (Thus, this parameter is usable when the emulation features, such as packet loss, are used concurrently.) The packet size is the size at the driver level and includes the 20 bytes of the IP header and the 14 bytes of the data link (at transmission). When 0 is specified for *upper_size_limit*, an infinitely large size is considered to have been specified.	0 to 65535 for each. Initial value: 0 *lower_size_limit* must be smaller than *upper_size_limit*. (except when 0 is specified for *upper_size_limit*)

Others

Parameter	send/recv	Description	Specifiable Values
Version "*version*"	-	Specifies the version of the work emulation feature to be used.	Any valid version number, or 0, which means the latest version.
Option number "*option_number*"	-	The preset value of the system software corresponding to the option number (1 to 4) can be acquired. In the case of 0, the currently set value is acquired instead of a preset value. The preset values corresponding to each option cannot be changed (set). When an emulation feature is applied with the option number specifying, the preset values are used taking priority over the other emulation parameters.	0 to 4
Parameter flag "*flags*"	-	0x00000001: Applies the bandwidth limit option (as mentioned in Reference of Limit) In addition to the above, there is also a value that can be obtained only during parameter acquisition. For details, refer to the `SceNetEmulationParam` structure of the "libnet Reference" document.	0 or the following bit value: 0x00000001

Parameter	send/recv	Description	Specifiable Values
Random seed "*seed*"	Common to send/recv	Specifies the seed of random number that can be used by the emulation feature. Emulation reproducibility can be expected when a value other than 0 is specified. When 0 is specified, the seed of random number is automatically selected. The selected value can be known when the parameter is obtained.	32-bit value

Preset Values

The preset values of the system software corresponding to the option numbers are as follows. Even in the ad hoc communication mode, network emulation will be applied by selecting one of the following option numbers.

Option number	Description
1	Send delay of 100 msec occurs.
2	Send processing is limited to 128kbps.
3	Send data is dropped with a probability of 5%.
4	3G line is emulated. (*) Parameter flag: 0x00000000 Send delay and jitter: 230 ± 110 msec Send packet loss: 1 % Send packet reordering: 1 % Send packet duplication: 1% Send bandwidth limit: 64 kbps Receive packet loss: 1 % Receive bandwidth limit: 128 kbps

(*) Each parameter value is not necessarily the same as the value of the actual environment. Also, the actual operation depends on the actual environment like in the case of Wi-Fi, etc., and thus the value is used as a guide for the 3G line calculated through the average actual environment.

Policies

Policies in network emulation are a system designed to determine which communication packet an emulation feature is to be applied to. A policy consists of policy numbers that can be set for each socket, a policy pattern that specifies whether to apply the emulation feature to each policy number, and a policy flag (reservation) that specifies the details of the emulation feature to be used for each policy number. By suitably setting all these elements, packet loss, delay, etc., can be made to be generated for specific sockets only.

Policy Number

Any policy number can be allocated for each socket, in the range of 0 to 31 (the default value is 0) for sockets of the application side, and in the range of 32 to 63 (the default value is 32) for sockets of the system software side.

`SCE_NET_SO_TPPOLICY` option of `sceNetSetsockopt()` is used to set a policy number to a socket of the application side, and `SCE_NET_SO_TPPOLICY` option of `sceNetGetsockopt()` is used to obtain a policy number set to a socket.

Policy Pattern

The policy pattern is a 64-bit (system side and application side) bit flag that indicates whether to apply the emulation feature to each of the policy numbers from 0 to 63.

Bit Pattern Arrangement

System Side Policy Pattern		Application Side Policy Pattern
Bits 63…56 always handled as 0	Bits 55…32 undefined	Bits 31…0

Bit Meanings

Bit	Value	Meaning
n	0	Do not apply the emulation feature to (the socket of) policy number *n*.
	1	Apply the emulation feature to (the socket of) policy number *n*.

Policy Flag

A 16-bit policy flag is reserved for each policy number from 0 to 63.

Caution

Packet loss Caused by Send Queue Overflow

When a packet delay occurs on the sender side due to the emulation feature, or when packets that exceed the bandwidth limit are sent on the sender side, these packets are stacked in the send queue until the timing at which they are to be sent. If the application attempts to further send other packets in this state, the maximum number of packets that can be stacked in the send queue may be exceeded, in which case the excess packets are dropped.

Note that even if the parameter settings are such that no packet loss occurs, send packets may be dropped for the above reason. Whether packets have been dropped can be known from the number of sent and dropped packets using `sceNetShowIfconfig()` described in the "Hints for Looking into Network Problems" section in chapter 3. Since memory for the packets stacked in the send queue is consumed from the kernel work area, there is a possibility that the kernel work area will be insufficient.

Packet Drop and tcpdump Log

When a packet is dropped through the emulation feature, the operations for the log, which is obtained using `sceNetDump*()`, are as follows:

	Operations
Sending side	Not logged
Receiving side	Logged The number of dropped packets is counted as both the number of received packets and the number of received and dropped packets of `sceNetShowIfconfig()`.

Policy Application

It is necessary to set policies appropriately to apply the emulation parameters. For example, when setting the bandwidth limit of the receiving side, the policies of the receiving side must be set in addition to the bandwidth limit parameters.

5 Notes

System Reservation Port Numbers

The following port numbers are reserved by the system and cannot be used. An attempt to perform binding returns the `SCE_NET_EACCES` or the `SCE_NET_EINVAL` error.

This limitation will not be applied for all reserved port numbers during the **Development Mode**, which is set for the entire system. Note, however, that the limitation must be applied when the title is released. This setting can be changed in the **Release Check Mode**. For the **Release Check Mode**, refer to the "Development Kit Neighborhood Settings Guide" document.

UDP port number TCP port number TCP over UDPP2P port number	1 to 1023, 9293 to 9308, 40000 to 65535
UDPP2P virtual port number	32768 to 65535

Note: The UDPP2P (local, remote) port numbers and UDP (local, remote) port numbers of TCP over UDPP2P use respectively predetermined values. For details of "UDPP2P" and "TCP over UDPP2P", refer to the "Network Overview" document.

Ephemeral Port Numbers

An ephemeral port number is assigned when a port is set to port number 0 and is bound. Ephemeral port numbers are assigned in the following ranges.

UDP port number TCP port number TCP over UDPP2P port number	49167 to 65535
UDPP2P virtual port number	32768 to 49999

Suspend/Resume Function

This performs the procedure for returning the required IP address before suspend or standby operations.

System Suspend

The socket operation when system suspend is executed is as follows.

Socket	Operation at System Suspend	Application Processing at System Resume
TCP TCP over UDPP2P UDPP2P	The socket end processing is performed. Because no data is transmitted to the outside, the communication does not appear to have been cut to the other party.	The processing at the sockets cannot be continued. Call `sceNetSocketClose()`.
Connected UDP	The socket end processing is performed. Connected UDP is a UDP socket that calls `sceNetConnect()`.	The processing at the sockets cannot be continued. Call `sceNetSocketClose()`.
Other	Nothing in particular.	The processing at this socket can be continued.

The processing at a socket can be continued if the system determines that the socket does not need to be closed even when the above processing cannot be continued at the socket. Whether the processing can be continued at a socket or not depends on the error returned to the socket. In short, the IP address is released when system suspend occurs while the socket at which the processing cannot be continued will receive `SCE_NET_ERROR_EIPADDRCHANGED(=SCE_NET_EINACTIVEDISABLED)`.

Process Suspend

The socket operation when process suspend is executed is as follows.

Socket	Operation at Process Suspend	Application Processing at Process Resume
All sockets	Nothing in particular.	The processing at the sockets can be continued.

However, if the kernel resources are insufficient, end processing may be executed by the system for those sockets that require ample resources. In the case of sockets whose resources have been recovered, since `SCE_NET_ERROR_ERESUME` is returned and the processing cannot be continued, call `sceNetSocketClose()`. Moreover, if an event that releases the IP address occurs while process suspend is executed (intermittent disconnection, etc.), processing equivalent to system suspend will be executed.

In the Case of Releasing an IP Address for the Intermittent Disconnection, etc.

When an IP address is released for the intermittent disconnection or other reasons, sockets will perform the same processing as in the case of system suspend.

Error Handling

IP Address Change, System Suspend/Process Suspend

In general, in the case of releasing and reallocating an IP address for the reason of system suspend or Wi-Fi communication switching, an application does not need to specifically handle `SCE_NET_ERROR_EIPADDRCHANGED(=SCE_NET_EINACTIVEDISABLED)` or `SCE_NET_ERROR_ERESUME` returned from a socket. Thus, it is recommended to handle the error for the socket in the same way as for a general communication error by executing `sceNetSocketClose()`. A different handling way for the error for the socket should be implemented as an exceptional case only when such processing has a particular meaning.