Internet Backplane Protocol: API 1.0

Alessandro Bassi   Micah Beck   James S. Plank   Rich Wolski

Department of Computer Science
University of Tennessee
Knoxville, TN 37996

[abassi,mbeck,plank,rich]@cs.utk.edu

Abstract

In this document, we present a description of the IBP version 1.0 API. The current implementation of IBP supports only synchronized client requests; all client IBP calls will block pending completion on the server(s) side, or the expiration of the client's timeout. We present the C-language prototype of every call along with a detailed description of the data structures, success behavior and error conditions involved in that call. Failure of an IBP client call is indicated by a return value of **0**, or **NULL**, with a special variable (**IBP_errno**) set to the appropriate error code.

1   IBP Data Structures

A few data structures are used through the IBP world.

1.1   IBP capability

This is the basic building block of IBP. Its format is:

```
ibp://hostname:port/key/WRMKey/WRM
```

- **hostname:port** The two above fields are also called **IBP_depot**:

variable name	variable type
host	char *
port	int

  This table is pretty self-explaining.

- **key** The key can be roughly considered to be the filename.

- **WRMKey** The WriteKey/ ReadKey/ ManageKey is a value that allows the access to the right key for writing, reading, and managing.

- **WRM** This field can have only three values, and is linked to the field above. The values are:

  - WRITE
  - READ
  - MANAGE

1.2 Various tables

1.2.1 IBP_attributes

variable name	variable type
duration	time_t
reliability	int
type	int

- **duration :** specifies the time at which the allocated storage area will be automatically purged from the pool of storage areas managed by the server. Time is specified in seconds since the epoch (as returned by UNIX's **time(2)** function). A value of **0** indicates permanent status for the allocated storage area (it'll be only purged when no more clients have read access to it, otherwise it will be kept alive according to the reliability property).

- **reliability :** is a flag that determines how reliable the allocated storage area will be. The current version of IBP supports two levels of reliability:

  - *IBP_STABLE* which guarantees the existence of the allocated storage area until it is removed due to lack of readers as explained above.

  - *IBP_VOLATILE* which declares the allocated area to be volatile, in the sense that the corresponding IBP server can reclaim storage allocated to this area whenever site administration and/or IBP server policy mandates such move. Stable storage is never reclaimed by IBP server as long as at least one client has read access to that storage.

- **type** is a flag that determines the type of storage allocated. The current version of IBP supports four types of storage:

  - *IBP_BYTEARRAY* which treats the allocated area as a flat byte array. This will have the following implications on future accesses to that storage area:
    * Requests for read to the allocated area will be denied if there are not enough data to satisfy the read request at the time the request is received by the IBP server.
    * Requests to write (append) to the allocated area will be denied if it leads to the total size of the storage area exceeding the maximum allowable size specified in *size*.
    * A maximum of one write operation can be actively writing to the storage area at any given time; other write requests received by the server are queued pending completion of the running write process.
    * No limit is imposed on the number of simultaneous read accesses to the storage area. In addition, due to the use of append-only semantics for write operations, a write operation can be simultaneously active with anynumber of read operations to the same storage area.

  - *IBP_FIFO* which causes the allocated storage area to be treated as a FIFO queue, with the following implications:
    * Read data is removed from storage area once read.
    * Read requests will be blocked if not enough un-readdata is available in the storage area. In addition, noupper limit is placed on the size of data in read requests.
    * Write requests will be blocked if there is not enough space in the storage area to complete the write operation. In addition, there is no upper limit on the size of data involved in a write operation to the storage area.
    * Blocked operations will be un-blocked only when there is more data to read (blocked read operation) or available space to write (blocked write operations)
    * A maximum of one write operation and one read operation can be simultaneously active at any given time. Further requests are blocked pending completion of running operations.

  - *IBP_CIRQ* which causes the allocated storage area to be treated as a Circular Queue, with the following implications:
    * Read data is removed from storage area once read.

* Read requests will be blocked if not enough un-readdata is available in the storage area. In addition, noupper limit is placed on the size of data in read requests.
* Write requests will **NOT** be blocked if there is not enough space in the storage area to complete the write operation, but will overwrite the beginning of the queue. In addition, there is no upper limit on the size of data involved in a write operation to the storage area.
* Blocked operations will be un-blocked only when there is more data to read (blocked read operation).
* A maximum of one write operation and one read operation can be simultaneously active at any given time. Further requests are blocked pending completion of running operations.
  - *IBP_BUFFER* which causes the allocated storage area to be treated as a restricted-access flat storage area, with the following properties:
    * Only one process can be actively accessing the storage area for read and/or write operation at any given time. Other requests are blocked pending completion of theone that has access to the storage area at any giventime.
    * All write operations start at the beginning of the storage area, overwriting any data that had been stored there previously (even if it had not been read).
    * The amount of data available to a read operation at any given time is the amount that had been stored by the last write call.

1.2.2 IBP_set_of_caps

variable name	variable type
readCap	IBP_cap
writeCap	IBP_cap
manageCap	IBP_cap

The capabilities included in an **IBP_set_of_caps** object allow the client read access, write access, and management access to a particular storage area, respectively.

1.2.3 IBP_CapStatus

variable name	variable type
readRefCount	int
writeRefCount	int
currentSize	int
maxSize	ulong_t
attrib	IBP_attributes

readRefCount and **writeRefCount** hold the reference count for the read and write capabilities respectively (on return from an *IBP_PROBE* command) and are ignored for the other *IBP_manage()* commands. **currentSize** holds the current size of data stored in the underlying storage area (for storage areas of type *IBP_FIFO* and *IBP_CIRQ* it holds the maximum size of the underlying storage area). **maxSize** holds the maximum size of the storage area, while **attrib** holds the storage area attributes as defined earlier.

1.2.4 IBP_DptStatus

variable name	variable type
StableStor	ulong_t
StableStorUsed	ulong_t
VolStore	ulong_t
VolStoreUsed	ulong_t
duration	long

StableStor and **VolStor** are the Stable Storage size and the Volatile Storage size respectively, while **StableStorUsed** and **VolStorUsed** are the Stable Storage used and the Volatile Storage used. The **Duration** parameter is the max duration.

1.2.5 IBP_timer

variable name	variable type
ClientTimeout	int
ServerSync	int

The two timers have a completely different function. The **ClientTimeout** indicates the time the application is willing to wait for a response from the server. This parameter is used to improve the fault-tolerance of the IBP Client Library, to prevent waiting forever from an answer from a hanged server; or in case the network connection is particularly bad, or a very high latency time. The **ServerSync** is used as an **"or"** condition: i.e., in a IBP_load operation, the application program can ask for **N bytes** or whatever gathered after **ServerSync** time. This can be very helpful when another client is writing on the same media, and the application asking to load the data does not know how many bytes are written, but it's willing to wait for some time before giving up.

2   IBP Allocate

	variable name	variable type
parameters	depot	IBP_depot
	timeout	IBP_timer
	size	ulong_t
	attr	IBP_attributes
return value		IBP_set_of_caps

IBP_allocate() allocates a remote storage area on the host **depot**. The allocated area has a maximum possible size of **size** bytes, and storage attributes, defined by **attr**.

Return values

Upon success, **IBP_allocate()** returns an *IBP_set_of_caps* object. Otherwise, it returns a *NULL* pointer and sets *IBP_errno* to one of the following values defined in **"ibp_protocol.h"**

- **IBP_INVALID_PARAMETER** : One or more of the parameters to the IBP_allocate() call has an invalid value (e.g. NULL *targetHost*, invalid entry in *attr*, ...)

- **IBPE_CONNECTION** : An error has occurred while trying to connect to the IBP server running on **targetHost**.

- **IBPE_SOCK_WRITE** : An error has occurred while trying to write to the socket connection to the IBP server.

- **IBPE_SOCK_READ** : An error has occurred while trying to read response from the IBP server.

- **IBP_BAD_FORMAT** : Response from the IBP server does not have the expected format, or the IBP server received a badly formatted request.

- **IBP_INVALID_CMD** : The IBP server has received a command it does not recognize.

- **IBP_WOULD_EXCEED_LIMIT** : Granting the request would cause the IBP server to exceed the maximum storage limit allocated to the storage category defined in *attr*.

- **IBPE_FILE_ACCESS** : The IBP server has encountered an error while trying to access one or more of its internal files that control access to the storage area.

- **IBPE_INTERNAL** : The IBP server has encountered an internal error while processing the client's request.

- **IBP_TYPE_NOT_SUPPORTED** : A request to allocate a storage area of type *IBP_FIFO* was made to an IBP server that does not support this type.

3   IBP store

	variable name	variable type
parameters	cap	IBP_cap
	timeout	IBP_timer
	data	char *
	size	ulong_t
return value		ulong_t

IBP_store() stores **size** bytes starting at **data** in the storage area accessed through the IBP capability **cap**. For this call to succeed, **cap** must be a *writecap* returned by an earlier call to **IBP_allocate()**, or imported from the client which made the **IBP_allocate()** call. **IBP_store()** is a blocking call that only returns when the required size of data is successfully stored at the desired storage area accessed through the IBP capability **cap**, or an error causes the call to abort prematurely. The call appends data to the end of any previously stored data at the storage area accessed through **cap** for storage areas of type *IBP_BYTEARRAY*, *IBP_CIRQ* and *IBP_FIFO*. Data written to a storage area of type *IBP_BUFFER* overwrites any previous data (starting at the beginning of the buffer). If a **ServerSync** time is set, the call will return either if all the data has been written, or the time has expired.

Return values

Upon success, **IBP_store()** returns the number of bytes written. Otherwise it returns **0** and sets **IBP_errno** to one of the following error codes:

- **IBP_WRONG_CAP_FORMAT** : The IBP capability **cap** doesn't have the proper format.

- **IBP_CAP_NOT_WRITE** : The IBP capability **cap** is not a write capability.

- **IBPE_CONNECTION** : An error has occurred while trying to connect to the IBP server running on **targetHost**.

- **IBPE_SOCK_WRITE** : An error has occurred while trying to write to the socket connection to the IBP server.

- **IBPE_SOCK_READ** : An error has occurred while trying to read response from the IBP server.

- **IBP_BAD_FORMAT** : Response from the IBP server does not have the expected format, or the IBP server received a badly formatted request.

- **IBP_INVALID_CMD** : The IBP server has received a command it does not recognize.

- **IBP_CAP_NOT_FOUND** : The storage area accessed through **cap** does not exist on the associated IBP server.

- **IBP_CAP_ACCESS_DENIED** : The storage area accessed through **cap** cannot be accessed for write operations.

- **IBP_SIZE_EXCEEDS_LIMIT** : The write operation would cause the aggregate size of the storage area to exceed the maximum size specified in the **IBP_store()** call. This error is only relevant for storage areas of type *IBP_BYTEARRAY*.

- **IBPE_FILE_ACCESS** : The IBP server has encountered an error while trying to access one or more of its internal files that control access to the storage area.

- **IBPE_FILE_WRITE** : The IBP server has encountered an error while attempting to store incoming data to the underlying storage area.

- **IBP_RSRC_UNAVAIL**: A resource used by the IBP server was unavailable to service the request. This error is only relevant when the underlying storage area has type *IBP_FIFO*.

- **IBPE_INTERNAL** : The IBP server has encountered an internal error while processing the client's request.

4 IBP load

	variable name	variable type
parameters	source	IBP_cap
	timeout	IBP_timer
	buf	char *
	size	ulong_t
	offset	ulong_t
return value		ulong_t

IBP_load() reads up to **size** bytes, starting at **offset**, from the storage area accessed through the IBP capability **cap**, into the memory area pointed by **buf**. For storage areas of type *IBP_FIFO* and *IBP_CIRQ*, **offset** is ignored. A **size** value of **-1** causes all currently stored data in an *IBP_BYTEARRAY* type storage area to be read. For storage areas of type *IBP_FIFO*, a **size** value of **-1** causes a read operation for all current contents of the storage area. For this call to succeed, **cap** must be a *readcap* returned by an earlier call to **IBP_allocate()**, or imported from the client which made the **IBP_allocate()** call. **IBP_load()** is a blocking call that returns only when all required data is read, the **ServerSync** expires, or the read operation is prematurely terminated due to an error.

Return values

Upon success, **IBP_load()** returns the number of bytes actually read, otherwise it returns **0**and sets *IBP_errno* to one of the following error codes:

- **IBP_INVALID_PARAMETER** : One or more of the parameters to the IBP_load() call has an invalid value (e.g. negative *size*, ...)

- **IBP_WRONG_CAP_FORMAT** : The IBP capability **cap** doesn't have the proper format.

- **IBP_CAP_NOT_READ** : The IBP capability **cap** is not a read capability.

- **IBPE_CONNECTION** : An error has occurred while trying to connect to the IBP server running on **targetHost**.

- **IBPE_SOCK_WRITE** : An error has occurred while trying to write to the socket connection to the IBP server.

- **IBPE_SOCK_READ** : An error has occurred while trying to read response from the IBP server.

- **IBP_BAD_FORMAT** : Response from the IBP server does not have the expected format, or the IBP server received a badly formatted request.

- **IBP_INVALID_CMD** : The IBP server has received a command it does not recognize.

- **IBP_CAP_NOT_FOUND** : The storage area accessed through **cap** does not exist on the associated IBP server.

- **IBP_CAP_ACCESS_DENIED** : The storage area accessed through **cap** cannot be accessed for write operations.

- **IBPE_FILE_ACCESS** : The IBP server has encountered an error while trying to access one or more of its internal files that control access to the storage area.

- **IBPE_FILE_READ** : The IBP server has encountered an error while attempting to read incoming data to the underlying storage area.

- **IBP_RSRC_UNAVAIL**: A resource used by the IBP server was unavailable to service the request. This error is only relevant when the underlying storage area has type *IBP_FIFO*.

- **IBPE_INTERNAL** : The IBP server has encountered an internal error while processing the client's request.

5 IBP copy

	variable name	variable type
parameters	source	IBP_cap
	target	IBP_cap
	timeout	IBP_timer
	size	ulong_t
	offset	ulong_t
return value		ulong_t

IBP_copy() copies up to **size** bytes, starting at **offset**, from the storage area accessed through the IBP read capability **source**, and writes them to the storage area accessed through the IBP write capability **target**. For storage areas of type *IBP_FIFO* and *IBP_CIRQ*, **offset** is ignored. A **size** value of **-1** causes all currently stored data in an *IBP_BYTEARRAY* type storage area to be copied. For storage areas of type *IBP_FIFO* and *IBP_CIRQ*, a **size** value of **-1** causes a copy operation for all current contents of the storage area. As in other read operation to an *IBP_FIFO* or *IBP_CIRQ* storage area, data read from the storage area will no longer be available for future reads. For this call to succeed, **source** must be a *readcap* returned by an earlier call to **IBP_allocate()**, or imported from the client which made the **IBP_allocate()** call, and **target** must be a *writecap* returned by a similar call.

IBP_load() is a blocking call that returns only when all required data is read, the **ServerSync** expires, or the read operation is prematurely terminated due to an error.

Return values

Upon success, **IBP_copy()** returns the number of bytes actually read, otherwise it returns **0**and sets *IBP_errno* to one of the following error codes:

- **IBP_INVALID_PARAMETER** : One or more of the parameters to the IBP_copy() call has an invalid value (e.g. negative *size*, ...)

- **IBP_WRONG_CAP_FORMAT** : The IBP capability **cap** doesn't have the proper format.

- **IBP_CAP_NOT_WRITE** : The IBP capability **source** is not a write capability.

- **IBP_CAP_NOT_READ** : The IBP capability **target** is not a read capability.

- **IBPE_CONNECTION** : An error has occurred while trying to connect to the IBP server running on **targetHost**.

- **IBPE_SOCK_WRITE** : An error has occurred while trying to write to the socket connection to the IBP server.

- **IBPE_SOCK_READ** : An error has occurred while trying to read response from the IBP server.

- **IBP_BAD_FORMAT** : Response from the IBP server does not have the expected format, or the IBP server received a badly formatted request.

- **IBP_INVALID_CMD** : The IBP server has received a command it does not recognize.

- **IBP_CAP_NOT_FOUND** : One (or both) storage area accessed through **cap** does not exist on the associated IBP server.

- **IBP_CAP_ACCESS_DENIED** : One (or both) storage area accessed through **cap** cannot be accessed for write operations.

- **IBP_SIZE_EXCEEDS_LIMIT** : The write part of the copy operation would cause the aggregate size of the storage area to exceed the maximum size specified in the **IBP_store()** call. This error is only relevant for storage areas of type *IBP_BYTEARRAY*.

- **IBPE_FILE_ACCESS** : The IBP server has encountered an error while trying to access one or more of its internal files that control access to the storage area.

- **IBPE_FILE_WRITE** : The IBP server has encountered an error while attempting to store incoming data to the underlying storage area.

- **IBPE_FILE_READ** : The IBP server has encountered an error while attempting to read incoming data to the underlying storage area.

- **IBP_RSRC_UNAVAIL**: A resource used by the IBP server was unavailable to service the request. This error is only relevant when the underlying storage area has type *IBP_FIFO*.

- **IBPE_INTERNAL** : The IBP server has encountered an internal error while processing the client's request.

6   IBP mcopy

	variable name	variable type
parameters	DM_op	dtmv_op
	DM_param	dtmv_params *
	CapCount	int
	SourceCap	IBP_cap
	TargetCap[]	IBP_cap[]
	timeout	IBP_timer[]
	size	ulong_t
	offset	ulong_t
return value		void *

IBP_mcopy() copies up to **size** bytes, starting at **offset**, from the storage area accessed through the IBP read capability **source**, and writes them to the storage area(s) accessed through the IBP write capability(ies) **target[]**. The number of target depots is stored in **CapCount**. At the moment, only the point-to-point copy is implemented, so the first two parameters must be NULL and the third one must have value **1**. For storage areas of type *IBP_FIFO* and *IBP_CIRQ*, **offset** is ignored. A **size** value of **-1** causes all currently stored data in an *IBP_BYTEARRAY* type storage area accessed through **source** to be copied. For source storage areas of type *IBP_FIFO* and *IBP_CIRQ*, a **size** value of **-1** causes a copy operation for the maximum size specified in **IBP_allocate()** to be initiated. As in other read operations to an IBP_FIFO or IBP_CIRQ type storage area, data read from the storage area will no longer be available for future reads. For this call to succeed, **source** must be a *readcap* returned by an earlier call to **IBP_allocate()**, or imported from the client which made the **IBP_allocate()** call and **target** must be a *writecap* returned by a similar call.

It is worth mentioning that this call needs (1 + **CapCount**) **IBP_timers**, the first one for the source, the other ones for the each target. **IBP_mcopy()** is a blocking call that returns only when all required data is successfully copied from the source IBP server to all the target IBP server, the highest ServerSync value expired or the operation is prematurely terminated due to an error.

Return values

As this call is heavily dependent on the DataMover operation selected, the return value depends from the particular operation selected.

7   IBP manage

	variable name	variable type
parameters	manCap	IBP_cap
	timeout	IBP_timer
	cmd	int
	capType	int
	info	IBP_CapStatus
return value		int

IBP_manage() allows an IBP client to perform certain management operations on an IBP storage area. Any client that can present the management capability can issue any of the management commands described below. **cap** is an IBP management capability that is returned in the **IBP_allocate()** call or imported from the client which made that call (except when **cmd** = **IBP_PROBE**, where any capability can be used). **cmd** can take one of the following values (defined in the file **"ibp_protocol.h"**

- **IBP_INCR** increments the reference count to the capability associated with the management capability **cap**, and whose type is specified in the parameter **capType**. The parameter **info** is ignored for this command.

- **IBP_DECR** decrements the reference count to the capability associated with the management capability **cap** and whose type is specified in the parameter **capType**. Decrementing the reference count the read capability associated with a storage area to **0** causes the IBP server to delete that storage area from its managed pool. Further requests to that area will fail, while requests currently in progress will be allowed to progress to completion.

  The parameter **info** is ignored for this command.

- **IBP_CHNG** changes one or more of the attributes of the storage area accessed through the management capability **cap**. The new values are specified through the parameter **info** (described below). The current version of IBP allows changes to one (or more) of the following attributes:

  - *maxSize* changes the maximum storage size of the underlying storage area. Changing the size of a storage area of type *IBP_FIFO* or *IBP_CIRQ* is currently not allowed. Decreasing maximum size of a storage area of type *IBP_BYTEARRAY* does not affect data already stored there, it will only affect future requests to that storage area.

  - *duration* changes the duration property of the storage area (see description of the **IBP_allocate()** call for further details on the possible values and implications for this parameter.)

- **IBP_PROBE** checks the current state of the storage area accessed through the management capability **cap**. The current state is returned through the parameter **info**, which is defined below.

capType determines the type of the capability affected by the two commands *IBP_INCR* and *IBP_DECR*. It can have one of two values, *IBP_READCAP* and *IBP_WRITECAP*. It is ignored for the two commands *IBP_CHNG* and *IBP_PROBE*.
info is a pointer to a structure of type *IBP_CapStatus*.
The following table summarizes the use of different parameters with every command.

	capType	readRefCount	writeRefCount	currentSize	maxSize	attrib
IBP_INCR	In	Not used	Not Used	Not Used	Not Used	Not Used
IBP_DECR	In	Not used	Not Used	Not Used	Not Used	Not Used
IBP_PROBE	Not Used	Out	Out	Out	Out	Out
IBP_CHNG	Not Used	Not Used	Not Used	Not Used	In	In

Return values
Upon success, **IBP_manage()** returns **0**, otherwise it returns **-1** and sets *IBP_errno* to one of the following error codes:

- **IBP_INVALID_PARAMETER** : One or more of the parameters to the IBP_manage() call has an invalid value (e.g. negative *capType*, ...)

- **IBP_WRONG_CAP_FORMAT** : The IBP capability doesn't have the proper format.

- **IBP_CAP_NOT_MANAGE** : The IBP capability is not a write capability.

- **IBPE_CONNECTION** : An error has occurred while trying to connect to the IBP server.

- **IBPE_SOCK_WRITE** : An error has occurred while trying to write to the socket connection to the IBP server.

- **IBPE_SOCK_READ** : An error has occurred while trying to read response from the IBP server.

- **IBP_BAD_FORMAT** : Response from the IBP server does not have the expected format, or the IBP server received a badly formatted request.

- **IBP_INVALID_CMD** : The IBP server has received a command it does not recognize.

- **IBP_CAP_NOT_FOUND** : The storage area accessed through **cap** does not exist on the associated IBP server.

- **IBP_INVALID_MANAGE_CAP** : The management cap does not match the management cap associated with the storage area.

- **IBP_WOULD_DAMAGE_DATA** : Trying to change the size of a storage area of type *IBP_FIFO*.

- **IBP_WOULD_EXCEED_LIMIT** : Trying to increase the maximum size of an *IBP_BYTEARRAY* type storage area leads exceeding the maximum storage space allocated for its class of storage.

- **IBPE_INTERNAL** : The IBP server has encountered an internal error while processing the client's request.

8   IBP status

	variable name	variable type
parameters	depot	IBP_depot
	StatusCmd	int
	timeout	IBP_timer
	Password	char *
	StableStor	ulong_t
	VolStor	ulong_t
	Duration	long
return value		IBP_DptStatus

IBP_status() allows an application to perform a query over a particular IBP depot and to modify some general storage properties. **depot** is the particular IBP depot the application would like to query. **StatusCmd** can have two values

- **IBP_ST_INQ** queries the IBP depot for its stable storage and the used amount, its volatile storage and the used amount, and the duration. When this command is used, the following 4 parameters are not used.

- **IBP_ST_CHANGE** changes the stable amount, the volatile amount and duration property of the IBP depot. Note the difference between this command and the **IBP_manage()** one. It is not possible to destroy the data already present in an IBP depot, so the changes only take effect if they are equal or bigger than the currently allocated area.

password is the IBP depot password. **StableStor** is the new total Stable Storage of the IBP depot. It must be equal or bigger than the current Stable Storage used; otherwise, it's ignored. **VolStor** is the new total Volatile Storage of the IBP depot. It must be equal or bigger than the current Volatile Storage used; otherwise, it's ignored. **duration** is the new maximum duration allowed. It is not retro-active.