[MS-PCCRR]:
Peer Content Caching and Retrieval: Retrieval Protocol Specification

---

Revision Summary

Date	Revision History	Revision Class	Comments
12/05/2008	0.1	Major	Initial Availability
01/16/2009	0.1.1	Editorial	Revised and edited the technical content.
02/27/2009	0.1.2	Editorial	Revised and edited the technical content.
04/10/2009	0.2	Minor	Updated the technical content.
05/22/2009	1.0	Major	Updated and revised the technical content.
07/02/2009	1.1	Minor	Updated the technical content.
08/14/2009	2.0	Major	Updated and revised the technical content.
09/25/2009	2.1	Minor	Updated the technical content.
11/06/2009	2.2	Minor	Updated the technical content.
12/18/2009	2.2.1	Editorial	Revised and edited the technical content.
01/29/2010	2.3	Minor	Updated the technical content.
03/12/2010	2.3.1	Editorial	Revised and edited the technical content.
04/23/2010	2.4	Minor	Updated the technical content.
06/04/2010	3.0	Major	Updated and revised the technical content.
07/16/2010	3.0	No change	No changes to the meaning, language, or formatting of the technical content.
08/27/2010	3.0	No change	No changes to the meaning, language, or formatting of the technical content.
10/08/2010	3.0	No change	No changes to the meaning, language, or formatting of the technical content.
11/19/2010	3.0	No change	No changes to the meaning, language, or formatting of the technical content.
01/07/2011	3.0	No change	No changes to the meaning, language, or formatting of the technical content.
02/11/2011	3.0	No change	No changes to the meaning, language, or formatting of the technical content.
03/25/2011	3.0	No change	No changes to the meaning, language, or formatting of the technical content.
05/06/2011	3.0	No change	No changes to the meaning, language, or formatting of the technical content.

Date	Revision History	Revision Class	Comments
06/17/2011	3.1	Minor	Clarified the meaning of the technical content.

Contents

1 Introduction

The **Peer Content Caching and Retrieval Framework** is based on a peer-to-peer discovery and distribution model. It is designed to reduce bandwidth consumption on branch-office wide-area-network (WAN) links by having **clients** retrieve content from distributed caches when available instead of the **content servers**, which are often located remotely from branch offices over the WAN links. The **peers** themselves act as caches from which they serve other requesting peers. The framework also supports the mode of using pre-provisioned hosted caches in place of peer-based caching. The main benefit of the framework is to reduce operation costs by reducing WAN link utilization, while providing faster downloads from the local area networks (LANs) in the branch offices.

The Retrieval Protocol defines two protocol message exchanges – one for querying the **server** for the availability of certain content, and the other for retrieving content from a server. The framework incorporates both the Retrieval Protocol and the Discovery Protocol [MS-PCCRD] together to enable a client to discover and retrieve content from multiple peers that have the content instead of the original content server.

1.1 Glossary

The following terms are defined in [MS-GLOS]:

encryption key

The following terms are defined in [MS-PCCRC]:

block
block hash
segment
segment ID (HoHoDk)
segment secret

The following terms are specific to this document:

block range: A set of consecutive **blocks** within a **segment** described by a pair of integers, the first being the index of the first **blocks** in the range, and the second the number of consecutive **blocks** in the range.

client (client-role peer): For the **Peer Content Caching and Retrieval Framework**, a **peer** that is looking for content, either from the **server** or from other **peers** or hosted caches. In the context of the Retrieval Protocol, a **client** is a **peer** that requests a block-range from a server_role_peer.

content server: The original source of the content that **peers** subsequently retrieve from each other.

cooperative mode: A mode of operation for the **client-role peer** in the **Peer Content Caching and Retrieval Framework**, in which it discovers and obtains content **blocks** from other **peers**, and shares content **blocks** it has with other **peers** in the network.

download schedule session: The session invoked by a **client** instance of the **Peer Content Caching and Retrieval Framework** within a **segment retrieval session** that schedules **block** downloads with available **servers**.

higher-layer application: The applications that use the Peer Content Caching and Retrieval: Retrieval Protocol, either by itself or as part of the **Peer Content Caching and Retrieval Framework** or other applications.

hosted cache mode: A mode of operation for the **client-role peer** in the **Peer Content Caching and Retrieval Framework**, in which it obtains and shares content (only) with a single **server** whose location is preconfigured on the **client-role peer**.

index: The **block** number within a **segment**, which can be used to compute the offset of the **block** in the **segment**, once multiplied by the **block** size used for that particular **segment block** number within the target **segment**.

initialization vector: A data block that some modes of AES cipher block operation require as an additional initial data input. Refer to [SP800-38A] for detailed definition and usage.

peer: An instance of the Retrieval Protocol for the **Peer Content Caching and Retrieval Framework** running on a host. A **peer** can be both a **client** and a **server** in the Retrieval Protocol operations.

Peer Content Caching and Retrieval Framework (or Framework): The framework that creates Peer Content Caching and Retrieval Discovery Protocol instances to discover **client-role peers** and download the content **blocks** from either **client-role peers** (**cooperative mode**) or hosted cache (**hosted-cache mode**).

Retrieval Protocol exchange: The request/response communication initiated by a **client-role peer** issuing a request to a given **server-role peer**, and concluded by the **server-role peer** responding to the request.

segment retrieval session: A session that defines a set of operations on a **client-role peer** that use the Discovery Protocol (in **cooperative mode**) and the Retrieval Protocol to discover and retrieve ranges of **blocks** (partial or complete) of a **segment**.

server (server-role peer): A **peer** that listens for incoming **block**-range requests from **client-role peers**, and responds to the requests.

simple download: A GetBlocks request/response that is carried out without an associated GetBlockList request/response.

target segment: The **segment** for which the **client-role peer** is requesting the desired **block range** in a **segment retrieval session**, identified by the **segment ID)**.

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as described in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

References to Microsoft Open Specification documents do not include a publishing year because links are to the latest version of the documents, which are updated frequently. References to other documents include a publishing year when one is available.

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information. Please check the archive site,

http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624, as an additional source.

[SP800-38A] National Institute of Standards and Technology. "Special Publication 800-38A, Recommendation for Block Cipher Modes of Operation: Methods and Techniques", December 2001, http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf

[MS-DTYP] Microsoft Corporation, "Windows Data Types".

[MS-PCCRC] Microsoft Corporation, "Peer Content Caching and Retrieval: Content Identification".

[MS-PCCRD] Microsoft Corporation, "Peer Content Caching and Retrieval Discovery Protocol Specification".

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, http://www.rfc-editor.org/rfc/rfc2119.txt

[RFC2616] Fielding, R., Gettys, J., Mogul, J., et al., "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999, http://www.ietf.org/rfc/rfc2616.txt

1.2.2 Informative References

[FIPS197] National Institute of Standards and Technology, "Federal Information Processing Standards Publication 197: Advanced Encryption Standard (AES)", November 2001, http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[MS-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary".

1.3 Overview

The Retrieval Protocol defines three request/response exchanges between a client and a server on top of an HTTP [RFC2616] transport – to query the supported version range of the server, to query the availability of specific content, and to retrieve specific content. The protocol assumes that the client identifies both the specific content it is looking for and the server it will contact. The discovery of the content information and the server address is outside the scope of the Retrieval Protocol. The request/response exchanges are:

- Content Availability Request: The client initiates a query to the server for the availability of the specified content. The server responds with the ranges (subsets or all) of the requested content it has.
- Content Retrieval Request: The client initiates a request to the server for the specified content. The server either replies with the requested content or with content of zero length when the requested content is not available.
- Version Negotiation Request: The client initiates a request to the server to query the supported Retrieval Protocol version range. The server replies with its supported Retrieval Protocol version range.

The exchanges can be utilized in conjunction or independently, as described in the following examples:

- The client can query the server for the availability of the content, identify what content the server has, and then retrieve only the available content from the server; or
- The client can query the server for the availability of the content, identify what content the server has, and decide not to retrieve the content; or

- The client can retrieve the content directly from the server without querying for the availability of the content first.
- For all scenarios described earlier, the client can optionally query the server for its supported version range first before querying for content availability or retrieving **blocks**.

The Retrieval Protocol does not mandate the relationship between these exchanges, as shown in the examples. As a result, in the case where they are used in conjunction, the **higher-layer applications** invoking the Retrieval Protocol must be able to retain the availability list from the availability query and use it to retrieve part or all of the available content in the subsequent retrieval request(s).

Peers within the Peer Content Caching and Retrieval Framework use the Retrieval Protocol in one of two ways, depending on whether they are in **cooperative mode**, retrieving content from each other, or **hosted cache mode**, retrieving it only from a single preconfigured server. In the cooperative mode case, a peer uses the framework's Discovery Protocol (see [MS-PCCRD]) to locate peers who have the desired content, and then initiates exchanges with the discovered peers to obtain the content. In hosted cache mode, a peer directly initiates exchanges with the hosted cache to obtain the desired content.

1.4 Relationship to Other Protocols

The Retrieval Protocol uses HTTP [RFC2616] as a transport.

The Peer Content Caching and Retrieval Framework uses the Retrieval Protocol [MS-PCCRR] and Discovery Protocol [MS-PCCRD] to discover peers when in cooperative mode, and query and download content from other peers. The framework also uses the data structures as described in [MS-PCCRC].

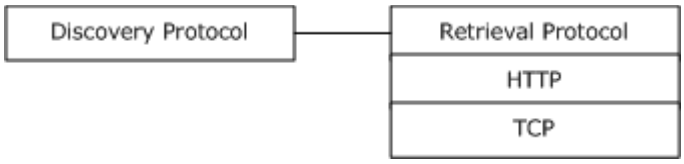


Figure 1: Protocol stack diagram

1.5 Prerequisites/Preconditions

- A higher-layer application using the protocol MUST have the Content Information (see [MS-PCCRC] section 2.3) for the block ranges and **segments** that it is retrieving from the server. The Content Information contains all the relevant information necessary for discovering and verifying the content blocks.
- The client must be able to identify and use the encryption algorithm and key used by the server to encrypt the content.

1.6 Applicability Statement

The Retrieval Protocol is designed to handle the content availability query and content retrieval parts of the operation.<1> It is also suitable for other types of content or object retrieval tasks because it does not assume any characteristics of the content.

The Peer Content Caching and Retrieval Framework, which uses the Retrieval Protocol, is best suited when there is a need to reduce load on a content server or reduce bandwidth usage on the link between the peers and the content server. This is because the protocol enables downloading data

from peers on the high speed link instead of the content server, which may be behind a slow link or may be heavily loaded.

1.7 Versioning and Capability Negotiation

The Retrieval Protocol has no version negotiation or capability negotiation behavior, although it carries a protocol number in its messages.

This document covers versioning issues in the following areas:

- **Supported Transports**: This protocol MUST be implemented on top of HTTP as discussed in section 2.1.
- **Protocol Versions**: The protocol version is 1.0.
- **Security and Authentication Methods**: There is no authentication or authorization in the protocol. The blocks served by the server-role peer, however, are encrypted as described in section 3.1.2.5.3.
- **Localization**: The protocol does not contain locale-dependent information.

1.8 Vendor-Extensible Fields

None.

1.9 Standards Assignments

None.

2 Messages

The Retrieval Protocol is made up of a limited number of fully defined messages sent on top of the Peer Download Transport.

2.1 Transport

2.1.1 Peer Download Transport

The Peer Download Transport is a peer-to-peer transport built on top of HTTP [RFC2616]. The client/server HTTP protocol is turned into a peer-to-peer transport by having each peer implement both a client and a server role. In a given transport session between two peers P1 and P2, the initiator peer acts as client, and the other peer acts as server. If P1 is the initiator of the transport session, P1 sends an HTTP request, and P2 replies by sending an HTTP response. Both the Retrieval Protocol request and response message types are included in the body of the HTTP messages. The payload of each such HTTP request or response consists solely of a single Retrieval Protocol message, with the response message prefixed with an additional length field (as defined in section 2.2.2) for reassembly purposes. A transport session between any two peers spans a single request-response sequence, and no context is kept within the transport across different transport sessions between those two peers.

Each peer implements the server role by reserving the URL under the root path of {116B50EB-ECE2-41ac-8429-9F9E963361B7}/ and listening for POST requests on it.

The initiating/client-role peer P1 at IP address A1 initiates the transport of a given request-type Peer Retrieval Protocol message to peer P2 at IP address A2, by sending an HTTP POST request to the root path of {116B50EB-ECE2-41ac-8429-9F9E963361B7}/.

2.1.2 Transport Security

The Peer Download Transport does not implement any security. There is no peer authentication or authorization, and messages are sent in clear text. At the transport level, peers accept and process all messages coming from any other peer.

2.2 Message Syntax

Messages are formed by headers and a message body. Both headers and body are formed by a sequence of fields. Each field is aligned according to the current protocol version's default alignment, currently 4 bytes.

All Retrieval Protocol messages are variable size messages. The valid range of the total message size MUST be from 16 bytes to 98,304 bytes (or 96 KB).

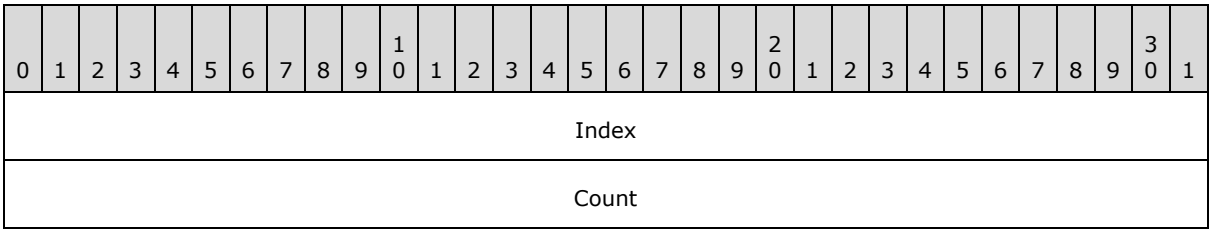
2.2.1 Common Data Types

The protocol supports three field types:

- Integer (**DWORD** fields as defined in [MS-DTYP] section 2.2.9, transmitted in network byte order).
- **BLOCK_RANGE_ARRAY** ((Integer [2])[count], i.e. a count-sized array of BLOCK_RANGE fields).
- **BYTE** array (**BYTE**[count], i.e. a count-sized array of bytes).

2.2.1.1 BLOCK_RANGE

A BLOCK_RANGE is an array of two integers that defines a consecutive array of blocks.



Index (4 bytes): The **index** of the first block in the range.

Count (4 bytes): Count of consecutive adjacent blocks in that range, including the block at the **Index** location. The value of this field MUST be greater than 0.

Index and **Count** are both integer fields in the range of 0x00000000 to 0xFFFFFFFF, but contain a value in the range from 0 to 511 (inclusive) for the **Index** field, and 1 to 511–Index (inclusive) for the **Count** field. For example, a BLOCK_RANGE of [42, 7] represents all the blocks starting from block index 42 to block index 48, including the last one.

2.2.1.2 BLOCK_RANGE_ARRAY

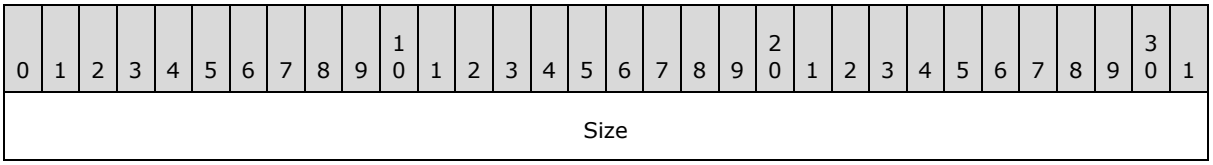
Variable-size array containing BLOCK_RANGE entries.

This type is declared as follows:

```
typedef BLOCK_RANGE BLOCK_RANGE_ARRAY[];
```

2.2.2 TRANSPORT_RESPONSE_HEADER

The transport adds the following header in front of response-type protocol messages for reassembly purposes:



Size (4 bytes): Total message size, in bytes, excluding this field. The valid range of the total message size MUST be from 16 bytes to 98,304 bytes (or 96 KB).

2.2.3 MESSAGE_HEADER

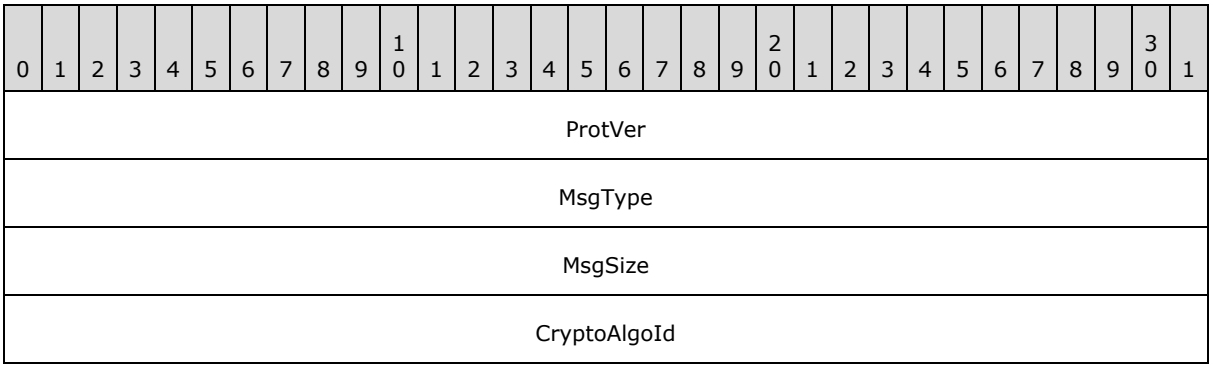
All Retrieval Protocol messages are prefixed by a message header.

Messages can be one of two types: request-type or response-type. Request-type messages include MSG_NEGO_REQ (section 2.2.4.1), MSG_GETBLKLIST (section 2.2.4.2), and MSG_GETBLKS (section 2.2.4.3), and response type messages include MSG_NEGO_RESP (section 2.2.5.1), MSG_BLKLIST (section 2.2.5.2), and MSG_BLK (section 2.2.5.3). Request-type messages initiate a communication

session between two peers. Response-type messages are sent only on response to a Request-type one (see Protocol Details (section 3) for more details).

A request-type message can be delivered only as an HTTP request. A response-type message can be delivered only as an HTTP response to an incoming HTTP request.

The layout of the message header is as follows.



ProtVer (4 bytes): Protocol version number, formed by concatenating the protocol major version number and protocol minor version number, encoded as follows (where MSB is Most Significant Byte and LSB is Least Significant Byte):

1st Byte (Addr: X)	2nd Byte (Addr: X+1)	3rd Byte (Addr: X+2)	4th Byte (Addr: X+3)
Minor version MSB	Minor version LSB	Major version MSB	Major version LSB

The major version number is encoded in the least significant word of the protocol version's **DWORD**.

The minor version number is encoded in the most significant word of the protocol version's **DWORD**.

Both the major and minor version number can express the version range of 0x00000000 to 0xFFFFFFFF. Currently, the protocol version number MUST be set to {major=1 (0x0001), minor=0 (0x0000)}.

MsgType (4 bytes): The type of message in the message body, expressed as a binary integer. MUST be set to one of the following values.

Value	Meaning
MSG_NEGO_REQ 0x00000000	A protocol version negotiation request. The request declares the minimum and maximum version numbers supported by the requesting client-role peer.<2>
MSG_NEGO_RESP 0x00000001	A protocol version negotiation response. It is sent in response to any protocol version negotiation request or to any other request with protocol version not supported by the server-role peer. The response declares the minimum and maximum version numbers supported by the responding server-role peer.
MSG_GETBLKLIST 0x00000002	A request for a list of **block hashes** of blocks in the **target segment** that are possessed by the destination server-role peer (list expressed as a **block range** array), and intersecting the list of block hashes specified in the

Value	Meaning
	request itself.
MSG_GETBLKS 0x00000003	A request for an array of block hashes (specified by a block range array). Since only one block will be returned, a MSG_GETBLKS message SHOULD specify only a single range containing only a single block.
MSG_BLKLIST 0x00000004	A response message containing a list of block hashes of blocks in the target segment that are possessed by the destination server-role peer (list expressed as a block range array), and intersecting the list of block hashes specified in the previous request from the client-role peer.
MSG_BLK 0x00000005	A response message containing the (first) actual block requested by the client-role peer via a block range array in a MSG_GETBLKLIST message.

MsgSize (4 bytes): Protocol message total size including MESSAGE_HEADER, but not including the Transport Header. The valid range of the total message size MUST be from 16 bytes to 98,304 bytes (or 96 KB).

CryptoAlgoId (4 bytes): Encryption algorithm used by the server-role peer to encrypt data. MUST be one of the following values.<3> Refer to [FIPS197] for the AES standard and [SP800-38A] for the supported block cipher modes listed in the following table.

Value	Meaning
0x00000000	No encryption.
AES_128 0x00000001	AES 128-bit, CBC-mode encryption.
AES_192 0x00000002	AES 192-bit, CBC-mode encryption.
AES_256 0x00000003	AES 256-bit, CBC-mode encryption.

2.2.4 Request Message

The Retrieval Protocol defines three request messages sent by the clients to the servers: MSG_NEGO_REQ (section 2.2.4.1), MSG_GETBLKLIST (section 2.2.4.2), and MSG_GETBLKS (section 2.2.4.3). The complete layout of a request-type Peer Content Caching and Retrieval: Retrieval Protocol message is as follows.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
MESSAGE_HEADER																															
...																															
...																															
...																															

MESSAGE_BODY (variable)
...

MESSAGE_HEADER (16 bytes): Message header.

MESSAGE_BODY (variable): Message body, which contains either a GetBlockList (MSG_GETBLKLIST) or GetBlocks (MSG_GETBLKS) request message.

2.2.4.1 MSG_NEGO_REQ

The MSG_NEGO_REQ (Negotiation Request) message is a request for the minimum and maximum protocol version supported by the target server-role peer. The message contains the minimum and maximum protocol version supported by the requesting client-role peer.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
MinSupportedProtocolVersion																															
MaxSupportedProtocolVersion																															

MinSupportedProtocolVersion (4 bytes): Minimum protocol version supported by the requesting peer. The protocol version is encoded identically to the **ProtVer** field defined in section 2.2.3.

MaxSupportedProtocolVersion (4 bytes): Maximum protocol version supported by the requesting peer. The protocol version is encoded identically to the **ProtVer** field defined in section 2.2.3.

2.2.4.2 MSG_GETBLKLIST

The MSG_GETBLKLIST (GetBlockList) message contains a request for a download block list. It is used when retrieving a set of blocks defined by one or more **BLOCK_ARRAY_RANGE** items.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
SizeOfSegmentID																															
SegmentID (variable)																															
...																															
ZeroPad (variable)																															
...																															
NeededBlocksRangeCount																															

NeededBlockRanges (variable)
...

SizeOfSegmentID (4 bytes): Size, in bytes. of the subsequent **SegmentID** field. The syntactic range of this field is from 0x00000000 to 0xFFFFFFFF. The actual value of this field depends on the hashing algorithm used as defined in [MS-PCCRC]. Implementations SHOULD support all allowed **SegmentID** lengths, and MUST support content with 32-byte **SegmentIDs**.<4>

SegmentID (variable): Public Segment Identifier for the target segment of content (also known as **HoHoDk**). See [MS-PCCRC] for a description of contents, segments, blocks, and identifiers.

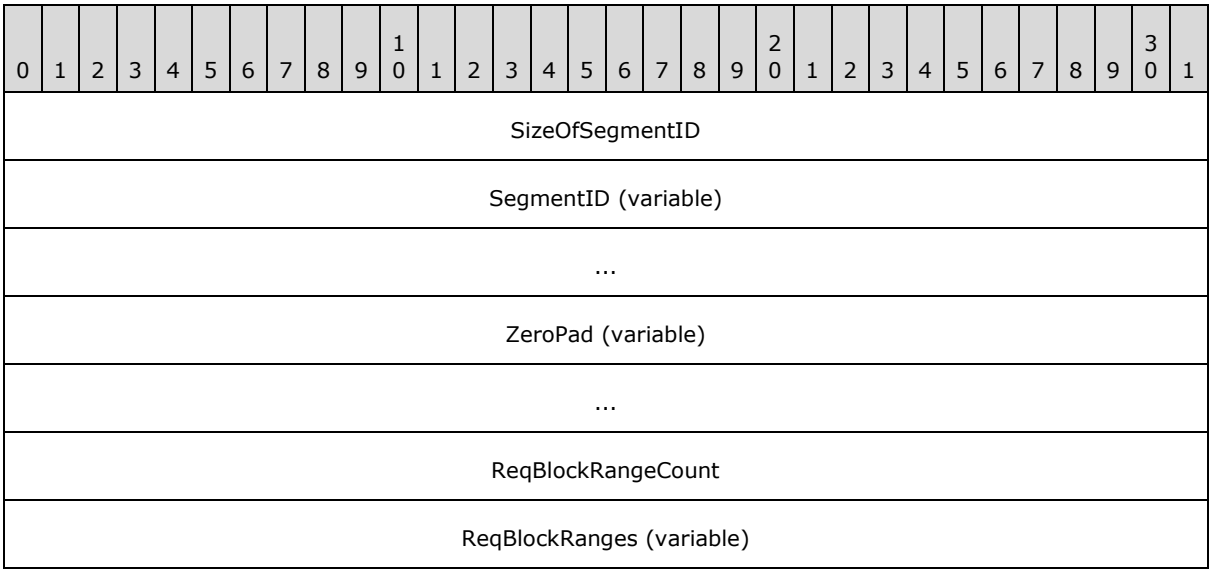
ZeroPad (variable): Sequence of bytes added (as needed) to restore 4-byte alignment, relative to the beginning of this message. The value of each byte MUST be set to zero. This field is 0 to 3 bytes in length, as required.

NeededBlocksRangeCount (4 bytes): Number of items in the subsequent block range array. The syntactic range of this field is from 0x00000000 to 0xFFFFFFFF. The effective range of this field MUST be between 1 and 256 inclusive, because there cannot be more than 256 non-overlapping and non-contiguous ranges in a maximum segment size of 512 blocks.

NeededBlockRanges (variable): Block range array listing the block hashes of the blocks within the target segment that the client-role peer is interested in. The server-role peer will reply with a block range array representing the intersection between the list of block hashes in the **NeededBlockRanges** array and the block range array set of blocks within the target segment currently available for sharing in the local cache of the server-role peer.<5>

2.2.4.3 MSG_GETBLKS

The MSG_GETBLKS (GetBlocks) message contains a request for blocks of content. It is used to retrieve a set of blocks defined by a single **BLOCK_ARRAY_RANGE**.



...
SizeOfDataForVrfBlock
DataForVrfBlock (variable)
...

SizeOfSegmentID (4 bytes): Size in bytes of the subsequent **SegmentID** field. The syntactic range of this field is from 0x00000000 to 0xFFFFFFFF. The actual value of this field depends on the hashing algorithm used as defined in [MS-PCCRC]. Implementations SHOULD support all allowed **SegmentID** lengths, and MUST support content with 32-byte **SegmentIDs**.<6>

SegmentID (variable): Public Segment Identifier for the target segment of content (also known as HoHoDk). See [MS-PCCRC] for a description of contents, segment, blocks, and identifiers.

ZeroPad (variable): Sequence of bytes added (as needed) to restore 4-byte alignment, relative to the beginning of this message. The value of each byte MUST be set to zero. This field is 0 to 3 bytes in length, as required.

ReqBlockRangeCount (4 bytes): Number of items in the subsequent block range array. The syntactic range of this field is from 0x00000000 to 0xFFFFFFFF. The effective range of this field MUST be between 1 and 256 inclusive, because there cannot be more than 256 non-overlapping and non-contiguous ranges in a maximum segment size of 512 blocks.

ReqBlockRanges (variable): Block range array representing the blocks requested for the target segment. **RegBlockRanges** MUST specify a block range containing only one block.

SizeOfDataForVrfBlock (4 bytes): Size in bytes of the subsequent **DataForVrfBlock** field. This field SHOULD be zero.

DataForVrfBlock (variable): Not used by the protocol. This field SHOULD be empty.

2.2.5 Response Message

The Retrieval Protocol defines three response messages sent by the servers in response to client requests: MSG_NEGO_RESP (section 2.2.5.1), MSG_BLKLIST (section 2.2.5.2), and MSG_BLK (section 2.2.5.3). The complete layout of a response-type Peer Content Caching and Retrieval: Retrieval Protocol message is as follows.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
TRANSPORT_RESPONSE_HEADER																															
MESSAGE_HEADER																															
...																															
...																															

...
MESSAGE_BODY (variable)
...

TRANSPORT_RESPONSE_HEADER (4 bytes): Transport response header.

MESSAGE_HEADER (16 bytes): Message header.

MESSAGE_BODY (variable): Message body, which may contain either a MSG_BLKLIST or a MSG_BLK message.

2.2.5.1 MSG_NEGO_RESP

The MSG_NEGO_RESP (Negotiation Response) message is the response message containing the minimum and maximum protocol version supported by the responding server-role peer. The message is sent in response to a Negotiation Request message or to any other request message with a protocol version not supported by the server-role peer.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
MinSupportedProtocolVersion																															
MaxSupportedProtocolVersion																															

MinSupportedProtocolVersion (4 bytes): Minimum protocol version supported by the requesting peer. The protocol version is encoded identically to the **ProtVer** field defined in section 2.2.3.

MaxSupportedProtocolVersion (4 bytes): Maximum protocol version supported by the requesting peer. The protocol version is encoded identically to the **ProtVer** field defined in section 2.2.3.

2.2.5.2 MSG_BLKLIST

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
SizeOfSegmentId																															
SegmentId (variable)																															
...																															
ZeroPad (variable)																															
...																															

BlockRangeCount
BlockRanges (variable)
...
NextBlockIndex

SizeOfSegmentId (4 bytes): The size, in bytes, of the subsequent **SegmentId** field.

SegmentId (variable): The Public Segment Identifier for the target segment of content (also known as HoHoDk). See [MS-PCCRC] for details.

ZeroPad (variable): A sequence of N bytes added (only as needed) to restore 4-byte alignment, where 0 <= N <= 3. Each byte's value MUST be set to zero.

BlockRangeCount (4 bytes): Number of items in the subsequent block range array. The server MUST set the **BlockRangeCount** field to 0 if it does not have any of the requested block ranges.

BlockRanges (variable): A block range array describing the blocks currently available for download from the current server-role peer for the target segment, within the boundaries of the list of block ranges of interest (**NeededBlockRanges**) specified by the client-role peer in the previously received GetBlockList request message (MSG_GETBLKLIST (section 2.2.4.2)). <7>

NextBlockIndex (4 bytes): The index of the first block after the block sent in the current message, currently available for download from this server-role peer. If no such next block is available, this index MUST be zero.

2.2.5.3 MSG_BLK

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
SizeOfSegmentId																															
SegmentId (variable)																															
...																															
ZeroPad (variable)																															
...																															
BlockIndex																															
NextBlockIndex																															
SizeOfBlock																															

Block (variable)
...
ZeroPad_2 (variable)
...
SizeOfVrfBlock
VrfBlock (variable)
...
ZeroPad_3 (variable)
...
SizeOfIVBlock
IVBlock (variable)
...

SizeOfSegmentId (4 bytes): The size, in bytes, of the subsequent **SegmentId** field.

SegmentId (variable): The Public Segment Identifier for the target segment of content (also known as HoHoDk). See [MS-PCCRC] for details.

ZeroPad (variable): A sequence of N bytes added (only as needed) to restore 4-byte alignment, where 0 <= N <= 3. Each byte's value MUST be set to zero.

BlockIndex (4 bytes): The index in the target segment of the block sent in the current message.

NextBlockIndex (4 bytes): The index of the first block after the block sent in the current message, currently available for download from this server-role peer. If no such next block is available, this index MUST be zero.

SizeOfBlock (4 bytes): The size, in bytes, of the subsequent **Block** field. The server MUST set the **SizeOfBlock** field to zero if it does not have the requested block.

Block (variable): The actual block of data, encrypted according to the cryptographic algorithm specified in the header of the message itself, not including the **initialization vector**.

ZeroPad_2 (variable): A sequence of N bytes added (only as needed) to restore 4-byte alignment, where 0 <= N <= 3. Each byte's value MUST be set to zero.

SizeOfVrfBlock (4 bytes): The size, in bytes, of the subsequent **VrfBlock** field, which SHOULD be zero.

VrfBlock (variable): Currently not used, and SHOULD be empty.

ZeroPad_3 (variable): A sequence of N bytes added (only as needed) to restore 4-byte alignment, where 0 <= N <= 3. Each byte's value MUST be set to zero.

SizeOfIVBlock (4 bytes): The size, in bytes, of the subsequent **IVBlock** field.

IVBlock (variable): The initialization vector used by the server-role peer when encrypting the block of data (**Block** field) sent with this message.

3 Protocol Details

The Retrieval Protocol consists of three types of exchanges: the Negotiation request/response, the BlockList request/response, and the Block request/response. (See section 2 for message formats and field definitions.)

- **Protocol Version Negotiation**: A client-role peer (referred to here simply as "client") initiates a protocol version negotiation with a server-role peer (referred to here simply as "server") by sending a Negotiation Request message (MSG_NEGO_REQ (section 2.2.4.1)), declaring the minimum and maximum protocol versions it supports. The server responds with a Negotiation Response message (MSG_NEGO_RESP (section 2.2.5.1)), declaring the minimum and maximum protocol versions it supports. Implementation of the client side of the protocol version negotiation is optional. The server side of the protocol version negotiation MUST be implemented.
- **BlockList request/response**: A client initiates a GetBlockList request (MSG_GETBLKLIST (section 2.2.4.2)) to a server in order to query the list of content blocks available on the server for a given segment ID, and a list of block ranges within the segment, by sending a MSG_GETBLKLIST request. The server responds with a BlockList response (MSG_BLKLIST (section 2.2.5.2)) containing the list of block ranges for the specific segment ID that are within the ranges of the request. If the server does not support the client's protocol version, it treats the request as a Negotiation Request and responds accordingly (see section 3.1.2.5.1).
- **Blocks request/response**: A client initiates a GetBlocks request (MSG_GETBLKS (section 2.2.4.3)) to a server to retrieve a specific block of a given segment, which is identified by the segment ID and the index of the block in the segment. It does this by sending a MSG_GETBLKS request. The server responds with the requested content blocks in a Block response (MSG_BLK (section 2.2.5.3)). If the server does not support the client's protocol version, it treats the request as a Negotiation Request, and responds accordingly (see section 3.1.2.5.1).

The Peer Content Caching and Retrieval Framework (also referred to as simply "the framework") then uses the Retrieval Protocol to retrieve and assemble complete segments of a content from a combination of sources, including either a set of server peers or a hosted cache, plus the original content server if the former does not have the complete content.

3.1 Retrieval Protocol Details

3.1.1 Client Details

3.1.1.1 Abstract Data Model

The Core Retrieval Protocol client maintains the following data:

- **Outstanding Request List**: A list of request messages sent for which responses have not yet been received, along with the addresses of the peers to which they were sent.

3.1.1.2 Timers

- **Request Timer**: A per-request-message timer set by the client whenever it sends a MSG_GETBLKLIST (section 2.2.4.2) or a MSG_GETBLKS (section 2.2.4.3) request message. When the timer expires before the exchange is completed, the client MUST cancel the current exchange. The default timeout value MUST be set to 2 seconds.<8>

3.1.1.3 Initialization

The Retrieval Protocol requires no explicit initialization for clients.

3.1.1.4 Higher-Layer Triggered Events

3.1.1.4.1 MSG_NEGO_REQ Request

An implementation of the Retrieval Protocol MAY support the sending of a Protocol Version Negotiation Request message (MSG_NEGO_REQ (section 2.2.4.1)), when triggered by a higher-layer application.<9> The following description only applies to the implementations that support this feature.

When initiating a Retrieval Protocol query for the supported protocol versions, the higher-layer applications MUST specify a server address. The Retrieval Protocol implementation MUST:

1. Construct an MSG_NEGO_REQ message (as specified in section 2.2.4.1).
2. Fill in the minimum and maximum protocol versions it supports.
3. Send the message to the server.
4. Store the message in the **Outstanding Request List**.
5. Start the **Request Timer**.

3.1.1.4.2 MSG_GETBLKLIST Initiation

To initiate a Retrieval Protocol query for the list of block ranges on a server, the higher-layer applications MUST specify a server address, a segment ID, and a set of block ranges within the segment identified by the segment ID. The client instance of the Retrieval Protocol instantiation MUST construct and send a GetBlockList message (MSG_GETBLKLIST (section 2.2.4.2)) to the server, store it in the **Outstanding Request List** (3.1.1.1), and start the **Request Timer** (3.1.1.2). The **SegmentID** and **NeededBlocksRanges** fields of the GetBlockList message correspond to the segment ID and the set of block ranges supplied by the higher-layer applications.

3.1.1.4.3 MSG_GETBLKS Initiation

To initiate a Retrieval Protocol request for specific block ranges, the higher-layer applications MUST specify a server address, a segment ID, and a set of block ranges with the segment identified by the segment ID. The client instance of the Retrieval Protocol MUST construct and send a GetBlocks message (MSG_GETBLKS (section 2.2.4.3)) to the server, store it in the **Outstanding Request List** (3.1.1.1), and start the **Request Timer** (3.1.1.2). The **SegmentID** and **ReqBlockRanges** fields correspond to the segment ID and the block ranges of the request.

The Retrieval Protocol SHOULD only request and retrieve one block per exchange of MSG_GETBLKS request and MSG_BLK (section 2.2.5.3) response messages. If the higher-layer applications need to retrieve more than one block, multiple GetBlocks messages MUST be sent with one block per request.

A peer SHOULD perform a **simple download** if it involves a limited number of consecutive blocks in a single block range.<10> This implies that the blocks are consecutive in the segment.

3.1.1.5 Message Processing Events and Sequencing Rules

3.1.1.5.1 MSG_NEGO_RESP Received

On receiving a MSG_NEGO_RESP (section 2.2.5.1) response message from a server, the client MUST first determine if this is a response to a previously sent request by checking the **Outstanding Request List** for the address of the server. If it is a response to either a MSG_GETBLKLIST (section

2.2.4.2) or MSG_GETBLKS (section 2.2.4.3) message, the client MUST compare the ranges of protocol versions and select a protocol version based on the rules described later in this section. It must then resend the original MSG_GETBLKLIST or MSG_GETBLKS message, using the selected version.

If it is a response to an MSG_NEGO_REQ (section 2.2.4.1), the client MUST return the highest protocol version supported by both the client and the server to the higher-layer applications.

If the client and server are incompatible, then the client MUST silently discard the MSG_NEGO_RESP message and MUST abort any current exchange with the server, including exchanges for previously sent MSG_GETBLKLIST, MSG_GETBLKS, or MSG_NEGO_REQ messages.

The rules for determining compatibility and selecting a version are listed below:

1. The client's and the server's major version ranges are calculated from the MSG_NEGO_REQ and MSG_NEGO_RESP messages, respectively. In both cases, they are defined as the inclusive range between the major version from the **MinSupportedProtocolVersion** field and the major version from the **MaxSupportedProtocolVersion** field. The highest common major version is the highest value that is included in both ranges. If these ranges do not contain any common values, then no highest common major version exists.

2. The minor versions within the same major version do not affect protocol compatibility. For instance, a client sending a version 3.2 request message and a server replying with version 3.0 message are fully compatible. The client and the server MUST each select their own highest minor version supported within the highest common major version. For example, if the client supports protocol version range [3.2, 5.0] and the server supports protocol version range [2.0, 4.3], then the highest common major version is 4, and the client will be sending messages with version 4.8 (assuming the highest minor version number for major version 4 is 4.8), whereas the server will be replying with messages with version 4.3. Another example: a client with a supported version range of [1.0, 2.1] and a server with a supported range of [2.5, 2.9] will result in a highest common major version of 2, with the client using version 2.1 and the server using version 2.9.

3. If no highest common major version exists, then the client and the server are incompatible.

If there is no existing request message previously sent to the server stored in the **Outstanding Request List**, the client MUST silently discard the received message.

3.1.1.5.2 MSG_BLKLIST Response Received

On receiving a MSG_BLKLIST (section 2.2.5.2) response message from a server, the client MUST verify that it is well-formed and corresponds to a GetBlockList request message (MSG_GETBLKLIST (section 2.2.4.2)) in its **Outstanding Request List**. The client then performs the following checks:

- The client SHOULD verify if the segment ID matches any request in the **Outstanding Request List**. If the client performs the segment ID check, it MUST silently discard the MSG_BLKLIST message and abort the exchange if the segment ID does not match the segment ID of any request.<11>

- The client MUST check if the block ranges overlap with the ranges specified in any request with a matching segment ID in the **Outstanding Request List**. The client MUST silently discard the MSG_BLKLIST message and abort the exchange if the check fails.

If this verification is successful, then the peer MUST:

- Delete the corresponding request message from the **Outstanding Request List**, and cancel its **Request Timer**.

- Return the segment ID and block range from the MSG_BLKLIST message, as well as the server address, to the higher-layer applications.

Otherwise, the response message MUST be silently discarded.

3.1.1.5.3 MSG_BLK Response Received

On receiving a MSG_BLK (section 2.2.5.3) response message from a discovered peer, the client MUST verify that it is well-formed and corresponds to a GetBlocks request message (MSG_GETBLKS (section 2.2.4.3)) in its **Outstanding Request List** (the segment ID and block index would match that of an outstanding GetBlocks request). The client MUST silently discard the message if this verification is unsuccessful. Otherwise, it MUST:

- Delete the corresponding request message from the **Outstanding Request List**, and cancel its **Request Timer**.
- If an encryption algorithm (**CryptoAlgoID** ≠ 0) is specified in the MSG_BLK message, decrypt the block using the pre-provisioned key.
- Pass the segment ID, block index, and (encrypted) block up to the higher-layer applications.

Otherwise, the response message MUST be silently discarded and the exchange aborted.

3.1.1.5.4 Other Messages Received

All malformed messages received by the client and messages of unknown type sent to the Retrieval Protocol URLs specified in section 2.1.1 MUST be silently discarded.

3.1.1.6 Timer Events

3.1.1.6.1 Request Timer Expiration

When the **Request Timer** expires before the exchange (GetBlockList (MSG_GETBLKLIST (section 2.2.4.2)), GetBlocks (MSG_GETBLKS (section 2.2.4.3)), or Negotiation Request (MSG_NEGO_REQ (section 2.2.4.1))) is completed, the client MUST abort the current exchange.

3.1.1.7 Other Local Events

None.

3.1.2 Server Details

3.1.2.1 Abstract Data Model

- **Content Cache**: This is the local content cache on the server. It consists of a list of segment IDs and associated block ranges, along with their Content Information (see [MS-PCCRC] section 2.3) and corresponding content blocks that the client or server has previously obtained either from other peers or from the content server. The server replies to client queries with the information and content blocks stored in its content cache; the client retrieves the content from the server using the Core Retrieval Protocol.
- **Active Client Count**: This counter keeps the number of active clients the server is currently serving. The counter is incremented by 1 when the server receives a request (GetBlockList (MSG_GETBLKLIST (section 2.2.4.2)) or GetBlocks (MSG_GETBLKS (section 2.2.4.3))), and is decremented by 1 when the server sends back a response or discards the request. This counter is

used to limit the number of concurrent clients for a server to a maximum value. The default maximum threshold SHOULD be set to 64<12><13>, and it MUST be configurable. The system administrators should configure this value based on the processing capability of the server. If this counter reaches the threshold, the server will send back an empty response (empty block range in BlockList (MSG_BLKLIST (section 2.2.5.2)) or empty block in Block (MSG_BLK (section 2.2.5.3))) to the client.

3.1.2.2 Timers

- **Upload Timer**: A per-instantiation timer set by a server when the protocol is instantiated. The server MUST abort the protocol instance when the timer expires before the request/response exchange is completed. The default timeout value MUST be set to 15 seconds.<14>

3.1.2.3 Initialization

The server is initialized by starting to listen for incoming HTTP requests on the URL specified in section 2.1.1. The server MUST set the **Active Client Count** to zero.

3.1.2.4 Higher-Layer Triggered Events

There are no explicit higher-layer triggered events for the server, other than waiting for the client messages as enabled by the initialization.

3.1.2.5 Message Processing Events and Sequencing Rules

3.1.2.5.1 MSG_NEGO_REQ Received

On receiving a valid MSG_NEGO_REQ (section 2.2.4.1) message from a client, the server MUST construct a MSG_NEGO_RESP (section 2.2.5.1) message with the maximum and minimum protocol versions that it supports, set the **Upload Timer**, and send the response message back to the client.

3.1.2.5.2 MSG_GETBLKLIST Request Received

On receiving a valid MSG_GETBLKLIST (section 2.2.4.2) request message from a client, the server MUST perform the following actions in the order specified:

1. The server MUST first check if the protocol version is supported, based on the version range comparison rules specified in section 3.1.1.5.1. If the major protocol version is outside the range of the server implementation, the server MUST construct an MSG_NEGO_RESP (section 2.2.5.1) message, fill it in with the maximum and minimum protocol versions it supports, and send the MSG_NEGO_RESP message back to the client.

2. If the major version is supported by the server, the server MUST select a compatible protocol version based on the same rules specified in section 3.1.1.5.1 for the following reply message.

3. The server MUST check if its **Active Client Count** is greater than or equal to the maximum number allowed. If the server is already serving more than or equal to the maximum number of clients, the server MUST reply to the client using a MSG_BLKLIST (section 2.2.5.2) message with an empty block range.

4. Otherwise, the server MUST increment the **Active Client Count** by 1, set the **Upload Timer**, and compute the intersection of the block ranges (for the segment specified) in the MSG_GETBLKLIST request with the block ranges for the same segment in the server's **Content Cache**. The server MUST then send the client a MSG_BLKLIST response message containing the

segment ID listed in the MSG_GETBLKLIST request message, and the computed intersection block ranges (possibly empty).

5. Once the MSG_BLKLIST response message is sent, the server MUST decrement the **Active Client Count** by 1. If the resulting value is negative, the server MUST set the counter to zero.

3.1.2.5.3 MSG_GETBLKS Request Received

On receiving a valid MSG_GETBLKS (section 2.2.4.3) request message from a client, the server MUST perform the following actions in the order specified:

1. The server MUST first check if the protocol version is supported, based on the version range comparison rules specified in section 3.1.1.5.1. If the major version is outside the range of the server implementation, the server MUST construct a MSG_NEGO_RESP (section 2.2.5.1) message, fill in the maximum and minimum protocol versions it supports, and send the MSG_NEGO_RESP message back to the client.

2. If the major version is supported by the server, the server MUST select a compatible protocol version based on the same rules specified in section 3.1.1.5.1 for the following reply message.

3. The server MUST check if its **Active Client Count** (Abstract Data Model, section 3.1.2.1) is greater than or equal to the maximum number allowed. If the server is already serving more than or equal to the maximum number of clients, the server MUST reply to the client using a MSG_BLK (section 2.2.5.3) message with an empty block.

4. Otherwise, the server MUST increment the **Active Client Count** by 1, set the **Upload Timer** (Timers, section 2.2.5.3), and construct and send the client a MSG_BLK response message containing a block that is selected based on the following rules:

   - If the block ranges in the MSG_GETBLKS request message contain only one block, the server MUST select the requested block.
   - If the block ranges contain more than one block, the server SHOULD select the first (smallest-index) block from the block ranges and the segment that is specified in the request message.

   The server then MUST check whether the selected block exists in the server's **Content Cache** (Abstract Data Model section 2.2.5.1). If it does, then the server MUST include this block in the MSG_BLK response message it sends. Otherwise, the response MUST contain an empty MSG_BLK response message. The **SegmentID** field in the response message MUST be set to the segment ID of the request, and the **BlockIndex** field MUST be set to the index of the block sent in this message. The server MUST also calculate the value of the **NextBlockIndex** field (section 2.2.5.3).

   The server MUST apply the encryption algorithm chosen by the upper-layer application to the block in MSG_BLK response message. The list of permissible encryption algorithms is given by the **CryptoAlgoID** value table in section 2.2.3.

5. Once the MSG_BLK message is sent, the server MUST decrement the **Active Client Count** by 1. If the resulting value is negative, the server MUST set the counter to zero.

3.1.2.5.4 Other Messages Received

All malformed messages received by the server and messages of unknown types sent to the Retrieval Protocol URLs specified in section 2.1.1 MUST be silently discarded.

3.1.2.6 Timer Events

3.1.2.6.1 Upload Timer Expiration

When the **Upload Timer** expires, the server-role peer MUST abort the protocol instance.

3.1.2.7 Other Local Events

None.

4 Protocol Examples

4.1 Download with GetBlockList and GetBlocks Exchanges

Scenario: Peer P1 is trying to download blocks BN0 -:- BN1 and BN2 -:- BN3 of segment S1 from peer P2.

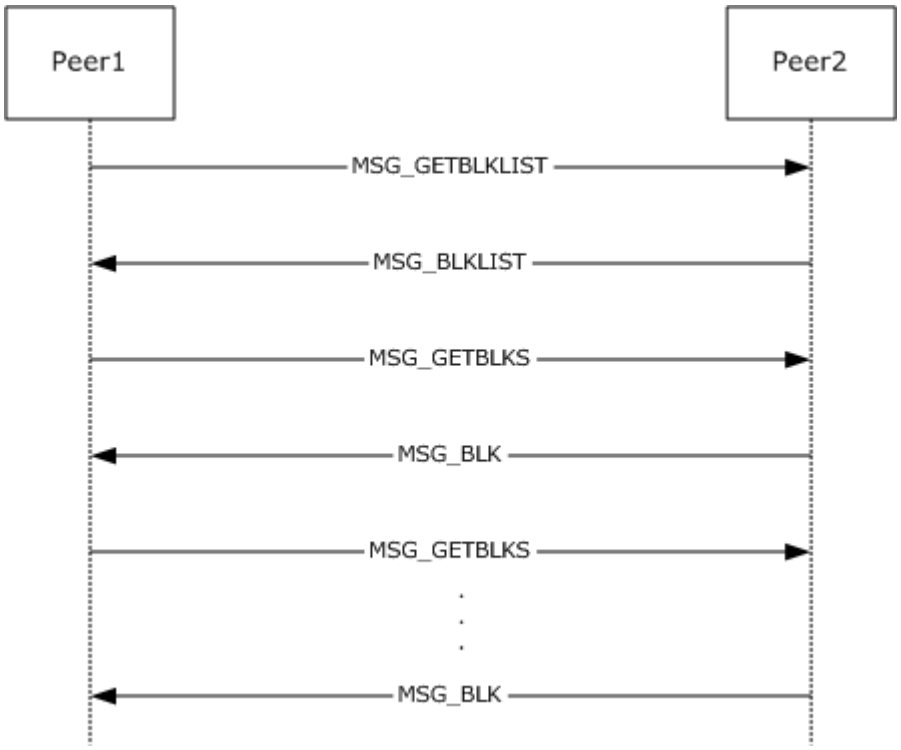


Figure 2: Download using GetBlockList and GetBlocks request/response pairs

Sequence of events:

1. Peer P1 sends a GetBlockList message (MSG_GETBLKLIST (section 2.2.4.2)) to P2, specifying two block ranges of interest, one for BN0 -:- BN1 and one for BN2 -:- BN3.

2. Upon receiving the GetBlockList message, P2 gathers the list of blocks it currently has for the target segment S1, it intersects that with the list of needed blocks specified by P1 in the GetBlockList message, and it sends back to P1 a BlockList message (MSG_BLKLIST (section 2.2.5.2)) containing the set resulting from the previous intersection.

3. Upon receiving the BlockList message from P2, P1 starts downloading blocks by sending a GetBlocks message (MSG_GETBLKS (section 2.2.4.3)) for one block at a time.

4. Upon receiving the GetBlocks message for a given block, P2 replies with a Block message (MSG_BLK (section 2.2.5.3)) containing the actual block of data encrypted using the crypto algorithm selected locally on the server-role peer P2. The **encryption key** MUST be provisioned on both P1 and P2. Refer to section 5.1 for the detailed requirements on encryption and initialization vector generation.

The encrypted block and initialization vector are added to the Block message and sent back to the client-role peer. The requesting peer will be able to decrypt the data only if it knows the hash of data of the segment.

4.2 Simple Download with GetBlocks Download Sub-Sessions only

Scenario: Peer P1 is trying to download two consecutive blocks, BN and BN + 1, of segment S1 from peer P2.

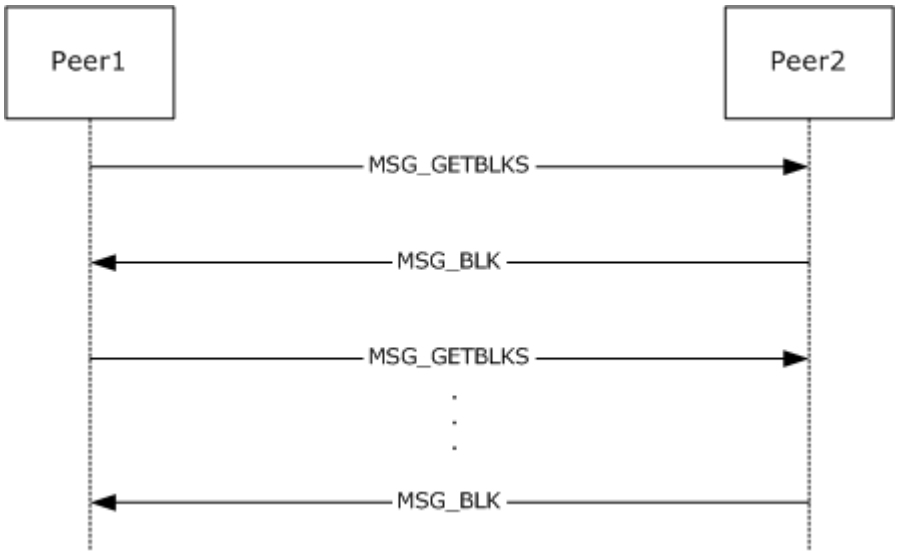


Figure 3: Simple Download using GetBlocks request/response

Sequence of events:

1. Since the download involves only two blocks, P1 decides to skip the GetBlockList message (MSG_GETBLKLIST (section 2.2.4.2)). It sends a GetBlocks message (MSG_GETBLKS (section 2.2.4.3)) for BN, and later for BN+1.

2. Upon receiving the GetBlocks message, P2 replies with a Block message (MSG_BLK (section 2.2.5.3)) containing the encrypted block of data and the initialization vector used during the encryption.

3. Once received the Block message, P1 decrypts and stores it, then it proceeds asking for the second block by sending a new GetBlocks message.

5 Security

5.1 Security Considerations for Implementers

A higher-layer application provides the server-role peer with the encryption algorithm, key size and the encryption key. The choice of the encryption algorithm and key size MUST be one of the **CryptoAlgoId** field values specified in section 2.2.3.

The server-role peer generates an initialization vector suitable for the chosen encryption algorithm and uses the encryption key to encrypt the block using the chosen encryption algorithm. The server-role peer then records the chosen algorithm and the initialization vector in the message, as described in section 2.2.5.3.

Server-role peers and client-role peers never exchange/share/send each other the encryption key.

The **client-role peer** MUST have a-priori knowledge of the encryption key. Using the encryption algorithm and initialization vector it received from the server-role peer, it decrypts the block.

There is no other explicit authentication or authorization built into the protocol, except for the Utility Index strategies above described that may lead to deny service to peers currently considered untrustworthy.

5.2 Index of Security Parameters

None.

6 Appendix A: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs:

- Windows Vista® operating system
- Windows Server® 2008 operating system
- Windows® 7 operating system
- Windows Server® 2008 R2 operating system

Exceptions, if any, are noted below. If a service pack or Quick Fix Engineering (QFE) number appears with the product version, behavior changed in that service pack or QFE. The new behavior also applies to subsequent service packs of the product unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms SHOULD or SHOULD NOT implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that the product does not follow the prescription.

<1> Section 1.6: For Windows Vista and Windows Server 2008, support for the client-side elements of this protocol is available only via the optional installation of the Background Intelligent Transfer Service via Windows Management Framework. Support for the server-side elements of this protocol is not available for Windows Vista or Windows Server 2008.

<2> Section 2.2.3: This message is never sent by Windows, but it is handled by the code if received, by responding with an MSG_NEGO_RESP (section 2.2.5.1) message.

<3> Section 2.2.3: Windows uses AES_128 as the default encryption algorithm. Windows supports: no encryption, AES_128, AES_ 192, and AES_256.

<4> Section 2.2.4.2: By default, Windows implementations use SHA-256 as the hashing algorithm to generate the **SegmentID**, which corresponds to a **SegmentID** length of 32 bytes. Windows Server 2008 R2 is capable of generating **SegmentIDs** using SHA-384 and SHA-512 in addition to SHA-256, but the Windows implementation of the Retrieval Protocol only supports **SegmentIDs** generated using SHA-256.

<5> Section 2.2.4.2: Windows implementations normalize the ranges in the array of block ranges in the MSG_GETBLKLIST and MSG_BLKLIST messages, using the following rules:

- Ranges in the array never overlap with each other.
- Overlapped or adjacent ranges in the array are always combined into a single range.
- Ranges in the array are always sorted by **Index**.

Windows implementations always send block ranges normalized with these rules, but can accept non-normalized ranges in received messages.

<6> Section 2.2.4.3: By default, Windows implementations use SHA-256 as the hashing algorithm to generate the **SegmentID**, which corresponds to a **SegmentID** length of 32 bytes. Windows Server 2008 R2 is capable of generating **SegmentIDs** using SHA-384 and SHA-512 in addition to SHA-256, but the Windows implementation of the Retrieval Protocol only supports **SegmentIDs** generated using SHA-256.

<7> Section 2.2.5.2: Windows implementations normalize the ranges in the array of block ranges in the MSG_GETBLKLIST and MSG_BLKLIST messages, using the following rules:

- Ranges in the array never overlap with each other.
- Overlapped or adjacent ranges in the array are always combined into a single range.
- Ranges in the array are always sorted by **Index**.

Windows implementations always send block ranges normalized with these rules, but can accept non-normalized ranges in received messages.

<8> Section 3.1.1.2: Windows uses a 2 second timeout for each request message. The timeout is configurable between 1 millisecond and 1 minute.

<9> Section 3.1.1.4.1: Windows does not implement this Negotiation Request.

<10> Section 3.1.1.4.3: Windows performs a simple download when it involves less than 4 consecutive blocks in a single block range. When Internet Explorer is used for content retrieval, it reads into 64K buffers. Therefore, in general each read generates a **segment retrieval session** for a single block; in some cases the read could span two blocks if it is not block aligned. This results in a simple download, and no MSG_GETBLKLIST is generated.

<11> Section 3.1.1.5.2: Windows implementations do not perform the segment ID verification for any MSG_BLKLIST message received. Windows implementations rely on the binding handle of the transport from which the MSG_BLKLIST is received in order to identify which request (and, implicitly, the corresponding segment ID) the MSG_BLKLIST is for.

<12> Section 3.1.2.1: By default the server-role peer Windows implementation serves up to 64 simultaneous Upload Sessions per serving-role peer; this limit is configurable between 1 and 16,384.

<13> Section 3.1.2.1: By default the server-role peer Windows implementation serves up to 1,024 simultaneous Upload Sessions per hosted cache server; this limit is configurable between 1 and 4,294,967,295.

<14> Section 3.1.2.2: Windows uses a 15 second timeout for each incoming request. The timeout value is configurable between 100 milliseconds and 1 hour.

7 Change Tracking

This section identifies changes that were made to the [MS-PCCRR] protocol document between the May 2011 and June 2011 releases. Changes are classified as New, Major, Minor, Editorial, or No change.

The revision class **New** means that a new document is being released.

The revision class **Major** means that the technical content in the document was significantly revised. Major changes affect protocol interoperability or implementation. Examples of major changes are:

- A document revision that incorporates changes to interoperability requirements or functionality.
- An extensive rewrite, addition, or deletion of major portions of content.
- The removal of a document from the documentation set.
- Changes made for template compliance.

The revision class **Minor** means that the meaning of the technical content was clarified. Minor changes do not affect protocol interoperability or implementation. Examples of minor changes are updates to clarify ambiguity at the sentence, paragraph, or table level.

The revision class **Editorial** means that the language and formatting in the technical content was changed. Editorial changes apply to grammatical, formatting, and style issues.

The revision class **No change** means that no new technical or language changes were introduced. The technical content of the document is identical to the last released version, but minor editorial and formatting changes, as well as updates to the header and footer information, and to the revision summary, may have been made.

Major and minor changes can be described further using the following change types:

- New content added.
- Content updated.
- Content removed.
- New product behavior note added.
- Product behavior note updated.
- Product behavior note removed.
- New protocol syntax added.
- Protocol syntax updated.
- Protocol syntax removed.
- New content added due to protocol revision.
- Content updated due to protocol revision.
- Content removed due to protocol revision.
- New protocol syntax added due to protocol revision.

- Protocol syntax updated due to protocol revision.
- Protocol syntax removed due to protocol revision.
- New content added for template compliance.
- Content updated for template compliance.
- Content removed for template compliance.
- Obsolete document removed.

Editorial changes are always classified with the change type **Editorially updated.**

Some important terms used in the change type descriptions are defined as follows:

- **Protocol syntax** refers to data elements (such as packets, structures, enumerations, and methods) as well as interfaces.
- **Protocol revision** refers to changes made to a protocol that affect the bits that are sent over the wire.

The changes made to this document are listed in the following table. For more information, please contact protocol@microsoft.com.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
1.2 References	Added explanatory statement regarding the removal of the publishing year from Microsoft Open Specification document references.	N	Content updated.

8 Index