SHENZHEN STOCK EXCHANGE

Engineering Technical Standard

Multicast Market Data Distribution Protocol of Shenzhen Stock Exchange

(Ver l. 00)

Shenzhen Stock Exchange

August, 2020

Document Description

Revision History		
Date	**Version**	**Revision Description**
2020-03	0.1	Creation
2020-05	0.2	Data type: Add variable length string type and variable length binary type Multicast packet: Adjusted multicast packet checksum algorithm to Adler32 Multicast packet supports management message and application message
2020-08	1.0	Official document

Term Explanation

Abbreviation	Meaning
MDGW	Market Data Gateway
MDDP	Market Data Distribute Protocol. Multicast market data distribution protocol, the market data gateway is configured as the distribution protocol used at the time of multicast
VSS	Vendor Supplied System, the market data receiving system provided by the supplier used to connect to the market data gateway to receive market data

Contents

Multicast Market Data Distribution Protocol of Shenzhen Stock Exchange

I. Preface

The Multicast Market Data Distribution Protocol (MDDP) of Shenzhen Stock Exchange (SZSE) is a set of transmission control protocols applicable to the distribution of stock market data. The main purpose of MDDP is to provide real-time market data distribution services for market participants' multiple business systems in a more efficient and convenient way.

This protocol mainly focuses on how to use MDDP for data transmission. For the relevant definition of application layer message, please refer to the Market Data Interface Specification of Shenzhen Stock Exchange.

II. Type Definition

All integer types used by MDDP are transmitted in network byte order (BIG-ENDIAN).

Type	Description
Int8	8-digit signed integer
Int16	16-digit signed integer
Int32	32-digit signed integer
Int64	64-digit signed integer
uInt8	8-digit unsigned integer
uInt16	16-digit unsigned integer
uInt32	32-digit unsigned integer
uInt64	64-digit unsigned integer
Char[x]	Char[x] represents the character string, and x represents the maximum number of bytes of the string. When the actual length of the string is less than the maximum length of field type, add a space to the right. UTF-8 encoding is uniformly used for the string.
VarChar	Variable-length character string. The length is specified by the leading VarCharLength; UTF-8 encoding is uniformly used for the character string.
VarCharLength	Length of variable-length character string, uInt32
RawData	Variable-length binary data, with the length specified by the leading RawDataLength
RawDataLength	Length of variable length binary data, uInt32
Padding[x]	Unformatted data, padding bit

III. Structure of Packet

This chapter mainly describes the definition of multicast packet in the MDDP protocol. Each MDDP multicast packet consists of three parts: packet header, packet body, and packet trailer.

packet header	packet text	packet trailer

3.1 Packet header

The packet header is transmitted without encryption in the following formats:

Domain name	Type	Length (byte)	Meaning
Protocol	uInt8	1	Protocol ID, fixed as 0xFF
Version	uInt8	1	Version number, currently 0x01
HeaderSize	uInt8	1	The length of the entire packet header, indicating that the packet header consists of several 4Byte words.
SenderId	uInt8	1	Send source ID
MarketId	uInt16	2	Market ID
Channel	uInt16	2	Channel number. The channels in the same market are not repeated, and the data content in the same channel is in order.
SeqNum	Int64	8	Sequence number of packet
MsgCount	uInt16	2	Message count in a packet
Flag	uInt16	2	Flag bit of packet
OriginalSize	uInt32	4	Original data size (optional)
CompressedSize	uInt32	4	Compressed data size (optional)
EncryptedSize	uInt32	4	Encrypted data size (optional)
Padding	Byte[x]	x	Byte aligned padding bit (optional) which ensures that the length of packet header is an integer multiple of 4 bytes, and is meaningless.

1. HeaderSize

HeaderSize refers to the length of the entire packet header. The length is calculated from the first field Protocol of the packet header to the end of the last field Padding (for example, HeaderSize=5 means the total length of the packet header is 5*4Byte=20Bytes).

2. SenderId

SenderId is used to identify different senders on the same multicast address and port.

3. MarketId

MarketId, reserved field, currently fixed as 1.

4. Channel

For market data channels, market data can be divided into different categories according to business (such as by securities), and the market data of each category can be further divided into multiple channels according to the data size.

5. SeqNum

SeqNum. Each sender will maintain a sequence number that starts from 1 in the Channel combination of the corresponding market. Each message in the packet will keep the sequence number increase by 1, and the SeqNum of the next packet will be the SeqNum of the last message of the previous message plus 1. According to the different flag bits of ResendBySeqNum, SeqNum has two encoding mechanisms:

1. ResendBySeqNum=0, the first packet that the sender starts or restarts is numbered from 1.

2. ResendBySeqNum=1, the sender uses the sequence number of the first packet in the message as the sequence number of the packet.

(This mode can only be used when a sequence number exists in the application message and is continuously increasing, such as a tick-by-tick market data)

6. MsgCount

This field indicates the message count in the packet body.

7. Flag

Flag bit, used to identify packet attributes. The first bit from the left is Bit15.

Flag Bit	**Meaning**
Bit 15	PossDupFlag flag bit, possible duplicate flag. Identifies the data in the packet as possible duplicate data. At this time, SeqNum can be less than the expected sequence number. If the data receiver has already received the data, it can be discarded directly. 0: Impossible duplicate packet 1: Possible duplicate packet
Bit 14-13	PacketType flag bit, mark of packet type 00: management packet 01: application packet
Bit 12	Whether the ResendBySeqNum flag bit supports resending by the SeqNum of packet 0: Not support 1: Support
Bit 11-10	Compress flag bit, the mark of compression 00: no compression 01: zlib compression 10: Reserved 11: Reserved
Bit 9-8	Encryption flag bit, the mark of encryption

	00: no encryption 01: user-defined encryption 10: Reserved 11: Reserved
Bit 7	MsgHeader flag bit, leader length flag bit 0: Absence of leader length 1: Existence of leader length
Bit 6-0	Reserved

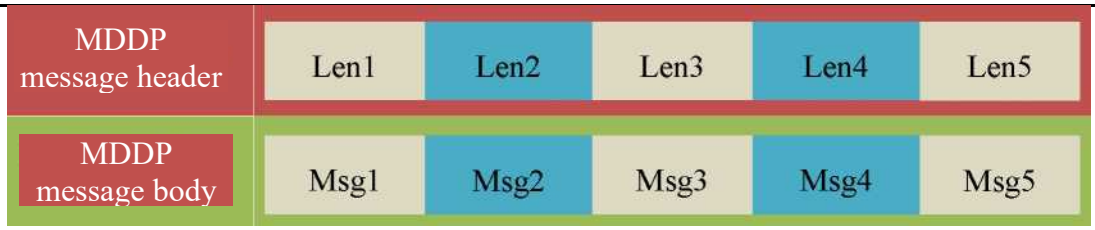
8. OriginalSize
An optional field, the length of packet body before compression and encryption; the packet header contains this field when the packet body is compressed or encrypted.
9. CompressedSize
An optional field, the length of transmitted data after compression; the packet header contains this field when the data is compressed.
10. EncryptedSize
An optional field, the length of transmitted data after encryption; the packet header contains this field when the data is encrypted.
11. Padding
An optional field, and the packet header contains this field when the length of packet header is not an integer multiple of 4Byte.

3.2 Packet Body

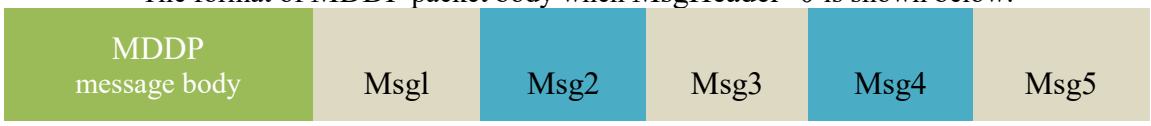
The packet body consists of MDDP message header and MDDP message body. MDDP message header indicates the length of each application message in the packet, and the MDDP packet body stores specific application messages. A MDDP message may contain multiple complete application messages, and messages are not transmitted across packets. Since some protocols use a context-sensitive encoding mechanism and cannot perform message splitting, the MDDP message header is optional and identified by the MsgHeader flag in the packet header.

The format of MDDP packet body when MsgHeader=1 is as follows:

Domain name	Type	Length (byte)	Meaning
Lengths	uInt32[x]	4*MsgCount	MDDP packet header indicates the length of each application message in the packet.
Body	Byte[x]	x designed by Lengths	MDDP message header



The format of MDDP packet body when MsgHeader=0 is shown below:



3.3 Packet Trailer

The packet trailer defines the checksum and is not encrypted for transmission. The calculation range of checksum starts from the packet header (including) to the end of packet body. The checksum algorithm is Adler32.

Domain name	Type	Length (byte)	Meaning
Checksum	uInt32	4	Checksum

IV. Definition of Management Packet

This chapter mainly describes the definitions of management packet used by the MDDP protocol.

4.1 Multicast Heartbeat Packet

The multicast flow is uniquely identified by the multicast address, port and SenderId. When there is no application layer data on the multicast flow, the sender will periodically send multicast heartbeat packet on the multicast flow to help the data receiver detect the status of multicast channel. The transmission interval of multicast heartbeat packet is 5 seconds. Failure to receive any data within 3 heartbeat periods by data receiver indicates that the multicast source is faulty.

When the channel is 0, the packet is a “multicast heartbeat packet”. The multicast heartbeat packet has no packet body and can be discarded directly upon receipt. Packet sample:

Domain name	Type	Value
Protocol	uInt8	Protocol ID, fixed as 0xFF
Version	uInt8	Version number, currently 0x01
HeaderSize	uInt8	5
SenderId	uInt8	Sender ID
MarketId	uInt16	1
Channel	uInt16	Fixed as 0
SeqNum	Int64	0
MsgCount	**uInt16**	**0**
Flag	uInt16	High bit →Low bit 0000000000000000
CheckSum	uInt32	Checksum

4.2 Data Flow Heartbeat Packet

The data flow is uniquely identified by Channel and SenderID. When the ResendBySeqNum flag bit of the packet is 1, the sender will periodically send data flow heartbeat packet on the data flow to inform the data receiver of the sequence number corresponding to the last data message sent on the current data flow.

This protocol sets the SeqNum of the "data flow heartbeat packet" to the SeqNum of the last message sent on the current data flow. When no application message has been sent on the data flow, the SeqNum of the "data flow heartbeat packet" is 0.

When Channel is not 0 and MsgCount is 0, it means that the message is a "data flow heartbeat packet" without packet body. Packet sample:

Domain name	**Type**	**Value**
Protocol	uInt8	Protocol ID, fixed as 0xFF
Version	uInt8	Version number, currently 0x01
HeaderSize	**uInt8**	**5**
SenderId	uInt8	Sender ID
MarketId	uInt16	1
Channel	**uInt16**	**2011**
SeqNum	Int64	9999

MsgCount	**uInt16**	**0**
Flag	uInt16	High bit →Low bit 0000000000000000
CheckSum	uInt32	Checksum

Sample of channel heartbeat message of tick-by-tick market data of Shenzhen Stock Exchange

4.3 Data Flow End Packet

"Data flow end packet" is a special data flow heartbeat packet, used to inform the data receiver that all service data on the current data flow has been sent.

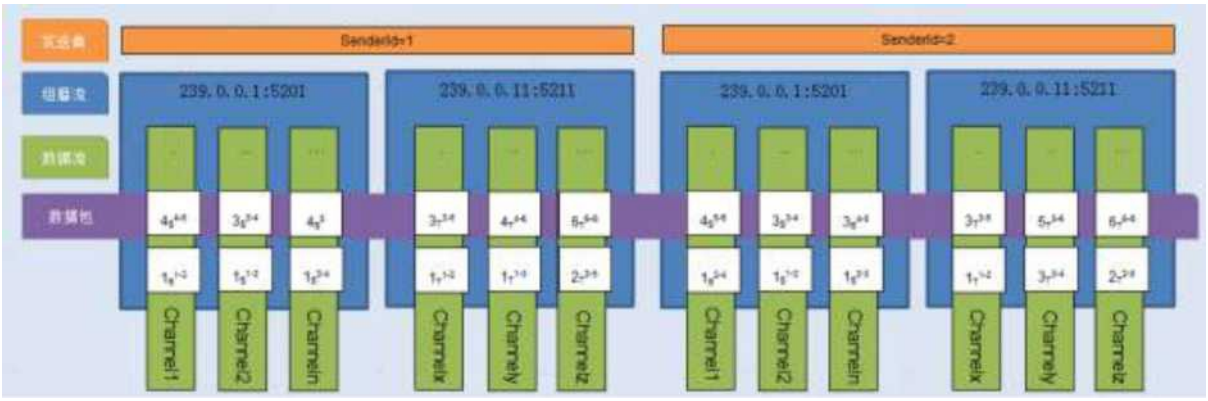
When Channel is not 0 and MsgCount is 0xFFFF (or written as 65535), it means that the packet is a "data flow end packet".

Message sample:

Domain name	Type	Value
Protocol	uInt8	Protocol ID, fixed as OxFF
Version	uInt8	Version number, currently 0x01
HeaderSize	**uInt8**	**5**
SenderId	uInt8	Sender ID
MarketId	uInt16	1
Channel	**uInt16**	**2011**
SeqNum	Int64	9999
MsgCount	**uInt16**	**OxFFFF**
Flag	uInt16	High bit →Low bit 0000000000000000
CheckSum	uInt32	Cheksum

V. Appendix

5.1 Relation Diagram of Sender- Multicast Flow- Data Flow- Data Packet



Note l: Senderl and Sender2 are two senders that are backups for each other

Note 2: Multicast flow is based on multicast address, port and sender

Note 3: Data flow is based on the multicast flow and channel.

Note 4: $N_{S^{x}y}$, N represents the packet sequence number, S the snapshot data flow, x-y the xth to yth snapshot messages on this data flow. The ResendBySeqNum flag is 0

Note 5: $N_{T^{x}y}$, N represents the packet sequence number, T the tick-by-tick data flow, x-y the xth to yth message on this data flow. The ResendBySeqNum flag is 1.

5.2 Application Message Decoding

MDDP message header	Len1	Len2	Len3	Len4	Len5
MDDP message body	Msg1	Msg2	Msg3	Msg4	Msg5

Header	Body

MDDP protocol currently only supports the distribution of SZSE binary market data. For the definition of application messages stored in the MDDP message body, please refer to the “Binary Market Data Interface Specification of Shenzhen Stock Exchange”.

MDDP message header provides the length of each message in the MDDP packet body. The receiver can implement message splitting without parsing the MDDP message body according to the message length information provided by MDDP message header.

Message	Message offset
Msg1	Offset1=0
Msg2	Offset2= Offset1+Len1

Msg3	Offset3= Offset2+Len2
Msg4	Offset4= Offset3+Len3
Msg5	Offset5= Offset4+Len4

In order to improve efficiency, MDDP canceled the checksum at the end of each message in the "Binary Market Data Interface Specification of Shenzhen Stock Exchange", and checked the integrity of the entire packet through the checksum of MDDP packet header. After the receiver performs the integrity check on the MDDP packet, it is allowed to directly complete message splitting and decode the message through MDDP message header, dispense with a separate integrity check on each message.

5.3 Sender Id Encoding Mechanism

The SenderId encoding mechanism is as follows:

1. When the trading day starts for the first time, SenderId is set to the index of Sender in the cluster;

2. Every time you restart, SenderId = Sender index+ (start times-1) in the cluster * the number of Senders in the cluster

3. When SenderId>255, SenderId is reset to the index of Sender in the cluster.

Examples of SenderId encoding mechanism are as follows:

Cluster	Sender index	The 1st start	The 2nd start	The nth start
Sender1	0	**0**	2	0+(N-1)*2
Sender2	1	1	3	1+(N-1)*2

Notes:

If, in the trading day, there are too many restarts or hardware failure causes the SenderId cannot be recovered from the disk after the restart, the SenderId is reset to the index of the Sender in the cluster at this time.

5.4 SeqNum Detection Mechanism

The sequence number of each multicast packet on the data flow increases continuously. Because multicast will cause disorder, packet loss, and repacketization, the data receiver should check the sequence number of multicast packet on the data flow.

1. SeqNum<NextExpectedSeqNum, the expected sequence number of the next packet. At this time, the data packet is considered to be expired data and discarded directly.

2. SeqNum=NextExpectedSeqNum, then process the message and set expected sequence number of the next packet NextExpectedSeqNum= SeqNum+MsgCount

3. SeqNum>NextExpectedSeqNum, indicating that out-of-order or packet loss may have occurred. Two processing mechanisms can be used:

a) It is directly judged as the network packet loss, and the subsequent packets will be processed from the current data packet. When the network infrastructure becomes stable and out-of-order rarely occurs, this mechanism can be considered.

b) First determine that the network is out of order and cache the data. When the cached data exceeds the preset buffer size or the expected message is not received after the timeout, network packet loss can be confirmed. It is required to use the packet with the smallest sequence number in the buffer as NextExpectedSeqNum, and continue to process the buffered and subsequent packets.

Note 1: The receiver's out-of-order buffer needs to be set according to the network out-of-order situation. If the network infrastructure becomes stable and out-of-order rarely occurs, it can be set to a relatively small value (such as 16).

Note 2: In case the retransmitted data is lost, the application layer retransmission mechanism provided by MDGW can be used to recover the data.

5.5 Data Source Fault Detection

After the data source fails and restarts, the SeqNum of the snapshot multicast packet (ResendBySeqNum flag bit is 0) will be renumbered from 1, causing the SeqNum of multicast packet to fall back. MDDP provides two failure detection mechanisms to facilitate receivers to identify and recover from data source failures.

1. SenderId has changed

When the receiver finds that the SenderId on the data flow has changed, indicative of data source restart or transaction day switching, the message sequence number in the latest data packet should be used to reset the next expected sequence number of multicast packet.

2. SenderId has not changed, but SeqNum has fallen back

In this scenario, the SenderId cannot be restored due to hardware failure, causing the SenderId to remain unchanged after the restart but the SeqNum has fallen back. Because the multicast may be out of order, the receiver can configure a threshold. When SeqNum + threshold < NextExpectedSeqNum, it is considered that the data source is restarted, and the message sequence number in the latest data packet should be used to reset the next expected sequence number of multicast packet. Otherwise, the packet is treated as an out-of-order packet.

5.6 Announcement

MDDP protocol supports the announcement function, which can be enabled or disabled through the announcement switch provided by MDGW. After the announcement function is enabled, MDDP divides the announcement into two parts: announcement summary and announcement content. Announcement summary is sent through multicast packets. The message definition of announcement summary is consistent with the announcement message in the binary data interface specification. Since the announcement summary does not carry announcement content, RawDataLength is fixed to 0. The content of announcement is stored in the directory designated by the recipient in the form of a file.