Problem

BDNT configuration options

Solution

- throttling(MBytesPerSec)
- enableMulticast and multicastAddress
- unicastPort, baseUnicastPort and useIncrementUnicastPort
- participantPerStream
- (base)QosLibrary
- cbReceiveProcessTimeoutSec and cbReceiveAvgProcessTimeoutSec

`throttling(MBytesPerSec)`

Default: `0.0`. (no throttling)

Availability: SenderFlow; C++ API, CDB.

Description: Bulk data NT flow sender sends out data as fast as it goes - as infrastructure allows.. This can lead for example to exaggerated the network, what can result in a lots of lost data, and re-transitions, slowing down other network activity including sending from other flow senders... If we want to limit the data rate of sending we can do this by specifying `throttling(MBytesPerSec)`. If 0.0 is specified, what is the default value, means no throttling, send as fast as possible. The trotting is applied just for `sendData`. With throttling we can achieve two things::

- prevent exaggeration of the network infrastructure: switches/routers, receivers' NICs
- better distribution of traffic in case of parallel arrays where data are sent from multiple computers

`enableMulticast` and `multicastAddress`

Default:

- `enableMulticast`: true (multicast is used)
- `multicastAddress`: 225.3.2.1

Availability: ReceiverFlow; C++ API, CDB.

Description: If `enableMulticast` is set to `true` (what is the default value) it is used multi-cast to send/receive the data, if the value is `false` the receiver for particular flow listens on unicast address. The multicast address can be set by specifying: `multicastAddress`. The default multi-cast address is 225.3.2.1. The value of multi-cast address is taken just just if multi-cast is enabled! It is not recommended to use multli-cast in combination with TCP.

`unicastPort`, `baseUnicastPort` and `useIncrementUnicastPort`

Default:

- `baseUnicastPort` 48000
- `useIncrementUnicastPort` true
- `unicastPort`: 0

Availability:

- `baseUnicastPort` `useIncrementUnicastPort`: ReceiverStream; C++ API, CDB.
- `unicastPort`: ReceiverFlow; C++ API, CDB.

Description: The unicast port, in case if multicast is disabled, is used in the following way. If `unicastPort` is set to value different than 0 then the value is taken, otherwise if `useIncrementUnicastPort` is true then unicast port is calculated (incremented by one for each flow) from `baseUnicastPort`. If neither condition (`unicastPort` is not set, and `useIncrementUnicastPort` is set to false) is met DDS choose the port. By default incremental ports are used.

In case if the port is set a new DEBUG message is logged: `Stream#Flow: ABC#Flow123 going to listen on unicast address, port: 4800`

`participantPerStream`

Default: `false`. (use global DDS participant)

Availability: SenderStream, ReceiverStream; C++ API, CDB.

Description: It regulates if a DDS participant is created for each stream, or a global one is used. Each instance of DDS participant means also more resources like memory and threads, but on the other side also better performance. So if there is a problem with number of threads/memory it should be used a global participant that is created with the creation of the fists stream. Default configuration is to use global (shared) participant, so no participant per stream, but rather one global for all the streams in a process.

A new log message appears if global(shared) participant is in use: `Going to use global participant for stream: MyStream`

`(base)QosLibrary`

Default: empty (`BulkDataQoSLibrary` QoS library will be used)

Availability: SenderStream, ReceiverStream SenderFlow ReceiverFlow; C++ API, CDB.

Description: By specifying `(base)QosLibrary` we can set which QoS library shall be used. The given QoS librarian has to be defined in: `$ACSDATA/config/bulkDataNTDefaultQosProfiles.xml`. If `(base)QoS library` is not specified, it is used default QoS library: `BulkDataQoSLibrary`.

Please see also: How to configure BDNT to use TCP ?

`cbReceiveProcessTimeoutSec` and `cbReceiveAvgProcessTimeoutSec`

Default:

- `cbReceiveProcessTimeoutSec` 0.010 sec
- `cbReceiveAvgProcessTimeoutSec` 0.050 sec

Availability:

- `cbReceiveProcessTimeoutSec`: ReceiverFlow; C++ API, CDB.
- `cbReceiveAvgProcessTimeoutSec`: ReceiverFlow; C++ API, CDB.

Description: With this options we set the upper limit (timeout) in sec for single frame (cbReceiveProcessTimeoutSec=) and average frame (`cbReceiveAvgProcessTimeoutSec`) processing time. The (**single**) processing time is measured by execution time of a single `cbReceive` callback method of the receiver. The **average** processing time is the average of processing time between `cbStart` and `cbStop`. The size of single frame correspond to 64000 bytes (~64kBytes), so it is measured processing time of a 64000 bytes. If either of limits are exceeded the BDNT logs an error message.

-- BogdanJeram - 2013-07-17

Related articles

- How can more people do development with ACS on the same machine without disturbing each other?
- Which ports are used by ACS?
- Problems connecting to ACS servers on a remote machine: bad /etc/hosts
- Why does the getComponent method of ZLegacy/ACS.ContainerServices return an object of type None?
- Why are some of my print statements not showing up in the container output section of acscommandcenter?