QOS_AGGREGATE V1.6

Release history

Version	Author	Comments
1.0	Gijsbert Wiesenekker	Initial release.
1.1	Gijsbert Wiesenekker	Supports multiple monitors, source and alarm expressions.
1.2	Gijsbert Wiesenekker	Supports delta qos, delta alarms, speed QoS, speed alarms and receive alarms.
1.3	Gijsbert Wiesenekker	Code cleanup. QoS messages are now prefixed with the name of the probe.
1.4	Gijsbert Wiesenekker	The documentation is now in PDF format.
1.5	Gijsbert Wiesenekker	The probe now supports dynamic source and targets for the QoS messages. The code has been greatly simplified in preparation to support delta, speed and moving average calculations through the expression. AS A CONSEQUENCE THIS VERSION NO LONGER SUPPORTS DELTA AND SPEED CALCULATIONS, THIS WILL BE ADDED BACK IN THE NEXT RELEASE.
1.6		Added a `discard` option to discard strings from the target.

Description
This probe subscribes to QoS messages and performs calculations on the subscribed QoS messages. You can for example take the average of three QoS values. The aggregate will be published as a new QoS message.

Installation
Ensure SDK_Perl 5.04 or greater is deployed to the robot that will run `qos_aggregate`.
Deploy the probe.

Usage
For troubleshooting you can also run the probe from the command-line using:
```
qos_aggregate.bin [-u <username>] -p <password>
```
on Windows or
```
./qos_aggregate.pl [-u <username>] -p <password>
```
on Linux. The username/passsword is required to login to Nimbus. The default username is `administrator`.

Configuration
Double click the probe in Infrastructure Manager to raw configure it or edit the configuration file with a text-editor (recommended):

Name	Optional/Required	Description
`loglevel`	Optional. The default is 1.	Controls the amount of debug information written to the logfile.

Name	Optional/Required	Description
`logfile`	Optional. The default is `qos_aggregate.log`	The name of the logfile.
`logsize`	Optional. The default is `1000000`.	The maaximum size of the logfile. The logfile will be rotated if the size is exceeded.
`interval`	Optional. The default is `600`.	The interval at which the probe should check if all QoS messages have been received for the defined monitors. The interval should be smaller than any of the intervals for all QoS watchers (see below).

The `<monitors>` section defines the monitors. For each monitor you specify:

Name	Optional or required	Description
`description`	Optional	A description of the monitor.
`active`	Optional. The default is `no`.	If set to `yes` the monitor is active, if set to `no` the monitor will be skipped. This allows you to keep monitors for future reference.
`qos_definition`	Required.	The qos_definition of the aggregated QoS consists of the name, the group, the description, the unit and the abbreviation of the unit separated by a ':'. For example: `IOSTAT_TPS:QOS_APPLICATION:tps:number:nr` The QoS name will be prefixed by the (impersonated) probe name.
`qos_float`	Optional. The default is 0.	If equal to 0 the aggregated QoS messages will be published as an integer value. If equal to 1 the aggregated QoS messages will be published as a floating-point value.
`prid`	Optional.	If defined the aggregated QoS messages will be sent as if originating from probe `prid`. This allows you to enhance other probes, and makes it very easy to add these QoS messages to existing list-views and performance charts.
`source`	Required.	The source of the QoS. You can use a static value but also any valid Perl `eval()` expression as described below.
`target`	Required.	The target of the QoS. You can use a static value but also any valid Perl `eval()` expression as described below.

Name	Optional or required	Description
`join`	Optional. The default is `source`.	If regular expressions are used for the QoS messages the probe must be able to determine which QoS messages have to be aggregated. `join` specifies the field that should be the same for these QoS messages. Currently the keywords `source`, `target` and `sourcetarget` are supported.
`expression`	Required.	Any valid Perl `eval()` expression to calculate the aggregated value of the QoS message as described below.
`alarm_eval`	Optional. The default is `undefined`.	A comma-separated list of any valid Perl `eval()` expression that refers to the Perl variable `$e` to determine if an alarm has to be sent. The Perl variable `$e` will be set to the aggregated value. Optionally the expression can be prefixed by the alert level using any of the keywords `INFO:`, `WARNING:`, `MINOR:`, `MAJOR: or CRITICAL:` For example if you want a warning alarm to be sent if the aggregated value is between 1 and 2 and a critical alarm if the value is greater than 3 you use `alarm_eval = WARNING:($e > 1) and ($e < 2),CRITICAL:($e > 3)`
`interval`	Optional. The default is `600`.	The interval in which all of the subscribed QoS messages should occur.

The `<qos>` section specifies the QoS messages to subscribe to as follows:

Name	Optional or required	Description
`name`	Required	The name of the QoS to subscribe to.
`source`	Required	The source of the QoS to subscribe to. It can be set to a static value, but also to any valid Perl regular expression without the leading and trailing ‘/’.
`target`	Required	The target of the QoS to subscribe to. It can be set to a static value, but also to any valid Perl regular expression without the leading and trailing ‘/’.

Name	Optional or required	Description
`discard`	Required	A Perl regular expression to discard strings in the target so that they can be joined. Some of the vmware targets for example contain not only the disk name but also a description so this allows you to select only the diskname.
`receive_alarm`	Optional	If specified an alert will be generated if a QoS has not been received during (`receive_alarm` * `samplerate`) seconds.

The source of the subscribed QoS messages is stored in a Perl array `@s` in the order specified in the configuration file, so
`$s[0]` will contain the source of the first subscribed QoS message,
`$s[1]` will contain the source of the second subscribed QoS message,
`$s[2]` will contain the source of the third subscribed QoS message,
etc.
This allows you to construct a value for the source of the aggregated QoS using a Perl string expression like `$s[0] . $[s1] . $s[2]`

The target of the subscribed QoS messages is stored in a Perl array `@t` in the order specified in the configuration file, so
`$t[0]` will contain the target of the first subscribed QoS message,
`$t[1]` will contain the target of the second subscribed QoS message,
`$t[2]` will contain the target of the third subscribed QoS message,
etc.
This allows you to construct a value for the source of the aggregated QoS using a Perl string expression like `$t[0] . t[s1] . $t[2]`

The samplesvalues of the subscribed QoS messages are collected in a buffer. The buffer contains the `sampletime` and the `samplevalue` of the QoS messages.
When all slots in the buffer have been filled, the probe checks if the time between the oldest and newest message is less than or equal to the interval time. If so, the aggregate value is calculated as follows:

The samplevalues of the subscribed QoS messages are stored in a Perl array `@v` in the order specified in the configuration file, so
`$v[0]` will contain the samplevalue of the first subscribed QoS message,
`$v[1]` will contain the samplevalue of the second subscribed QoS message,
`$v[2]` will contain the samplevalue of the third subscribed QoS message,
etc.

The QoS expression can refer to the array `@v`. You can use the variable `$e` that will be set to the value of the aggregated QoS in an alarm expression.
Two examples:
If you want to alert if the aggregated QoS value is below 10 you use the alarm expression:
alarm = `$e` < 10

The following configuration file gives an example on how to scale back interface traffic to Kbit/sec:

```
<monitors>
   <0>
      description = scale interface traffic to Kbit
      active = no
      qos_definition = INTERFACE_TRAFFIC_KBIT:QOS_APPLICATION:Kbit:kb
      qos_float = 1
      source = $s[0]
      target = $t[0]
      expression = $v[0] / 1024
      alarm_eval = $e > 2
      interval = 600
      <qos>
         <0>
            name = QOS_INTERFACE_TRAFFIC
            source = .*
            target = .*
         </0>
      </qos>
   </0>
</monitors>
```

The following configuration file gives an example on how to calculate total IOPS for all VMware guests:

```
<monitors>
   <0>
      description = calculate percentage datastore used
      active = no
      qos_definition = TOTAL_IOPS:QOS_APPLICATION:Number:nr
      qos_float = 1
      source = $s[0]
      target = $t[0]
      join = sourcetarget
      expression = ($v[0] + $v[1] )
      interval = 600
      <qos>
         <0>
            name = name-of-qos-for-read-iops
            source = .*
            target = .*
............discard = Disk Average Read Requests Per Second$
         </0>
         <1>
            name = name-of-qos-for-write-iops
            source = .*
            target = .*
............discard = Disk Average Write Requests Per Second$
         </1>
      </qos>
   </0>
</monitors>
```