The Monitis Monitoring Agent

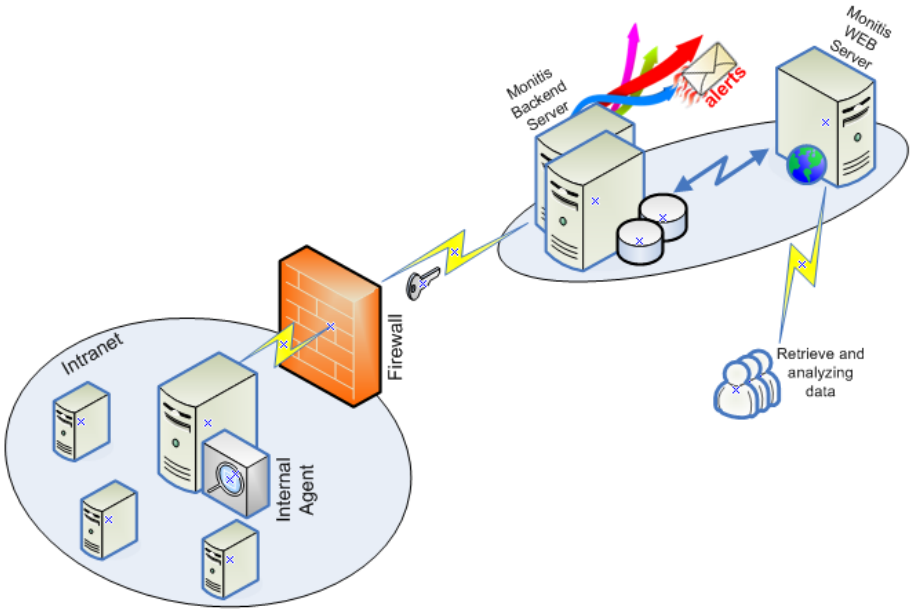
ver. 1.2

General principles, Security and Performance

Monitis provides a server and network monitoring agent that can check the health of servers, networks and applications both within and outside of a customer's firewalls. This agent will deliver the monitoring results in real time to the Monitis Backend and will also trigger alerts. The results can then be viewed in the web based Monitis Dashboard. This whitepaper provides a brief overview of the architecture of the Monitis agents and discusses their security, performance and bandwidth utilization.

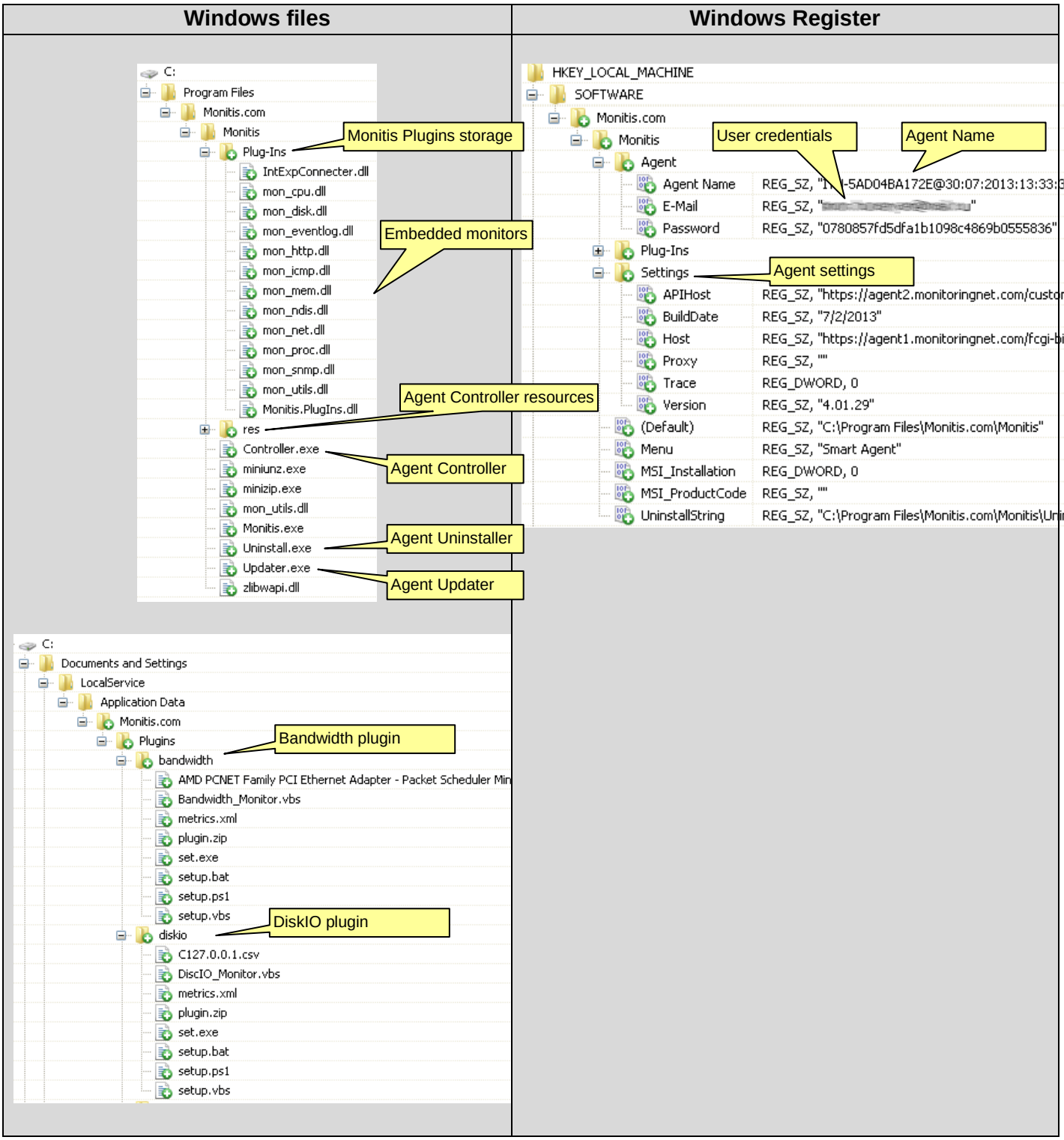
Monitis provides downloadable agents for both Windows and Linux operating systems. Users may deploy an agent for each monitored host/machine. You can also install just one agent within each local network and use it to monitor other servers agentless via SNMP (currently in beta). After deployment of the agents, users will configure specific monitors (e.g. CPU check, memory check, etc.) for agents centrally from the Dashboard. The agent will periodically run the configured checks and will transmit accumulated information to the Monitis Backend using HTTPS protocol. The usage of HTTPS is important as there is no need to open additional ports on your firewalls.

Agents use the Monitis internal protocol for communication. The communication of the agent with the Monitis Backend is depicted in the diagram below.



The Agent runs as a daemon on Unix-like machines or a service on Windows machines and executes each specific check's through internal threads. They are programmed in native C to have a small footprint.

After installation, it will have the predefined folders structure which is located on a user specified path and which is depicted below.



Note: Windows agent files are located in two places.

The Linux version executable module (Agent) has only the command line interface as depicted below (Ver. 4.01.23 or higher)

> **Usage**: *monitis [-C configuration file] [-L log file] [-l lock file] [-D home directory] [-U user e-mail] [-A agent name] [-t loglevel] [-H host server] [-a api url] [-p MD5 of account pass] [-P plugin dir][-V] [-h]*

Where options:

-h, --help	Print detailed help screen
-V, --version	Print version information
-C, --conf-file=PATH	Configuration file path (default: /etc/monitis.conf)
-L, --log-file=PATH	Log file path
-l, --lock-file=PATH	Lock file path
-D, --home-dir=PATH	Home directory path
-U, --user-email=E-MAIL	E-Mail address used to login to Monitis
-A, --agent-name=NAMA	Agent name
-t, --log-level=INT	log level [0-10]
-H, --host-server=URL	Monitis server URL
-a, --api-url=URL	Monitis custom API URL
-p, --api-hash=HASH	Custom API password, MD5 encoded
-P, --plugins-dir=PATH	plugins folder path

The Linux Agent uses the following libraries:

> libpthread.so, libdl.so, libc.so, libz.so, libmysqlclient.so, libssl.so, libcrypto.so, libsnmp.so, libxml2.so, linux-gate.so(Linux 32), ld-linux.so(Linux 32), linux-vdso.so(Linux 64), ld-linux-x86-64.so(Linux 64)

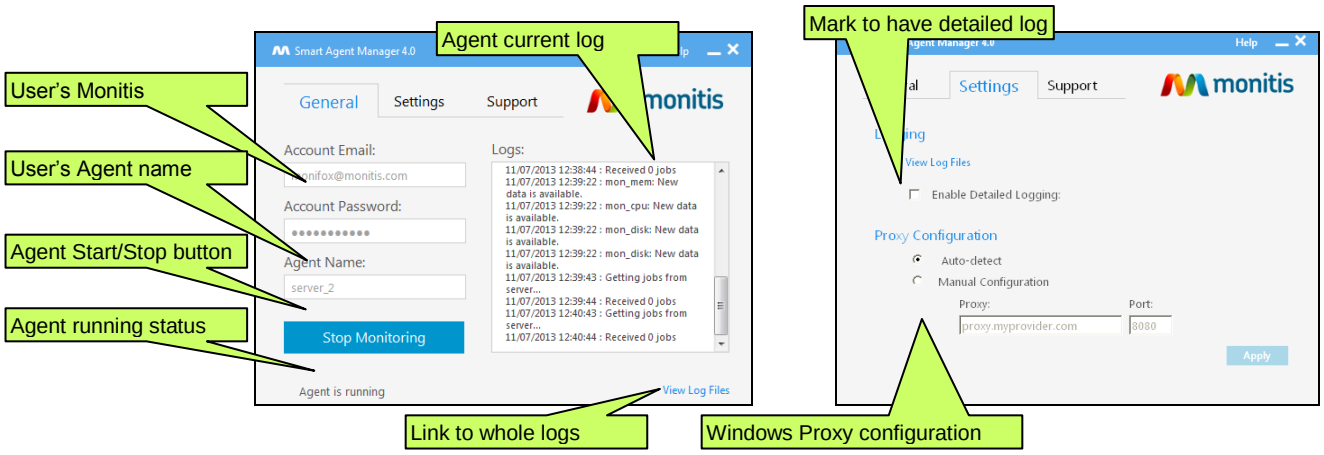
*(*libsnmp.so and libmysqlclient.so are not required if you are not using SNMP (currently in beta) and MySQL monitors*).*

Monitis also provides a simpler way of controlling the agent by using the shell script (monitis.sh) depicted below:

> *monitis.sh {conf|start|stop|restart|status|show|log}*
>
> where:

conf	run configuration wizard
start	start agent
stop	stop agent
restart	stop if running and start again
status	show agent current status
show	show main configuration
log	open log file with 'tail -f'. (default value: 100)

For Windows users, Monitis provides a graphical interface - the Monitis Smart Agent Manager (Controller) – that allows users to: start and stop agent, change password and provides a visual log. It also has a feedback feature that allows users to send comments and feedback to Monitis.

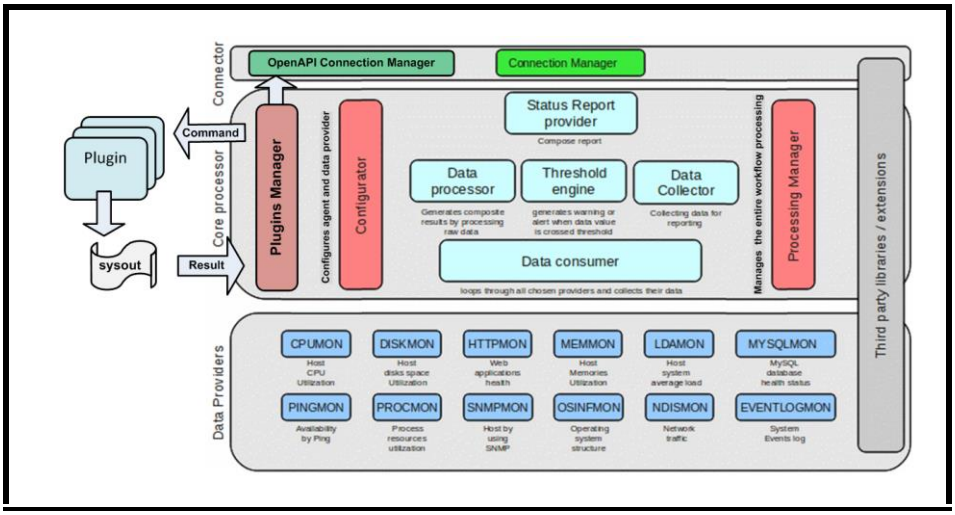


Monitis agent architecture

The Monitis internal agent contains embedded monitors such as; CPU, memory, drive space, etc (see the embedded monitors table below. In addition, users have the ability to add monitors beyond the embedded ones. To do this you will need to take advantage of additional plugins (see the plugins table below) which can be installed automatically from the Dashboard. As soon as an agent receives a request to do additional monitoring, such as Tomcat server monitoring, it downloads the appropriate plugin, then performs initial setup and periodically runs and sends measured data to the Monitis Backend via the Monitis Open API. This approach allows having almost an unlimited number of monitors that can plugin to the agent on demand and we are in constant development to add more feature plugins.

Every plugin is implemented as a standalone console application, run by an agent from a command line and sends measured results back to the agent via standard output ("sysout") stream. Naturally, the input and output formats should conform to the Monitis internal protocol that is used for communication between plugin and agent.

Below is a high level view of the agent internal architecture.



- Connection with the Monitis Backend is provided by two HTTP(S) connectors
    - Connection Manager which gets jobs and sends a measured data provided by predefined Data Providers (embedded monitors).
      The data from all active Data Providers is wrapped into Universal Data Collector (UDC) form, zipped and sent to Monitis as a unit block.
    - Open API Connection Manager which sends out measured data provided by plugins. Data received from any plugin is sent via Monitis API by using special form.

- Configurator periodically sends requests to the Monitis Backend for monitors' configuration changes and stores configurations internally.
- Plugins Manager activates the required plugins at the required intervals and collects testing data. The activated plugins perform the necessary checks by using the required resources/libraries and send back the measured data.

Below are descriptions for the existing Data Providers/Monitors.

Plugin	Description	Used resources	Comments
CPUMON	Monitoring the CPU(s) load/utilization for user (applications level), kernel (system level), nice (applications with nice priority), idle (unutilized CPU(s) part), iowait (CPU(s) idle during I/O request)	/proc/stat	
DISKMON	Monitoring the free and used disk(s) space		

Plugin	Description	Used resources	Comments
HTTPMON	Monitoring the applications/sites by using HTTP(S) connection	libssl.so library	Can be used for intranet and extranet services/sites monitoring from intranet.
MEMMON	Monitoring the free memory and swap sizes.	/proc/meminfo	Note that for remote memory parameters “snmptable.so” library is used
LDAMON	Monitoring the average system load over a period of time (1, 5 and/or 15-minute periods)	/proc/loadavg	Linux only
MYSQLMON	Monitoring MySQL database health status	libmysqlclient.so library	Linux only
PINGMON	Monitoring for intranet and extranet hosts by using PING protocol		ICMPMON in Windows
PROCMON	Monitoring the chosen process CPU utilization, memory and swap sizes usage.	/proc/%ProcID%/stat /proc/%ProcID%/cpu	
SNMPMON	Monitoring host by using host-embedded SNMP engine.	libsnmp.so library	currently in beta
EVENTLOGMON	Monitoring System Events log info		Windows only

Plugins

Any new plugin that is developed by Monitis developers must pass a detailed QA before going into production.

Below are descriptions of existing plugins.

name	Description	Platform	Status	Used Resources
Emulator	Simulates the generic plugin behavior	Win	For internal use only	VBScript
		Linux		Bash
diskio	Monitoring load on specified	Win	In production	WMI, VBScript

name	Description	Platform	Status	Used Resources
	partition (read and written amount of bytes)	Linux	In production	Bash /proc/diskstats
bandwidth	Monitoring load on specified network interface (sent and received amount of bytes)	Win	In production	WMI, VBScript
		Linux	In production	Bash /proc/net/dev
Tomcat	Monitoring health status of Apache Tomcat server	Win	In production	Java, JMX
		Linux	In production	Java, JMX

General principles of Monitis Open API

Generally, access to the Monitis Open API is provided by specifying the API Key and Token values. The API Key will identify exactly who is making an API request and then will allow them to perform the necessary input. This is very important for keeping service security under control.

The Token is valid for a limited time (currently 24 hours) and allows the user to use required parts of the API function. When a token expires it is necessary to ask the Open API to generate a new token. The agent can then obtain a unique Token by request to Open API by providing the API Key and the Secret Key. This pair of keys is unique for every user and provides for stronger user identification.

Both required keys can be obtained online from your Monitis user account. The Secret Key can be renewed at any time and therefore provides strong security.

The measured data is submitted via HTTP(S) POST request which contains permanent and variable parts of data in the request body. The permanent part contains API Key, Token, monitorID, etc. and is used for identification. Its size is normally about 120 bytes.

The variable part contains a measured data and its size (naturally dependent upon count and nature of measurement data) is in normally between 50 – 150 bytes. Thus, the typical API package body is about 200 - 250 bytes.

The Benchmark test

The benchmark test is a simple test that is run to check how the agents consume system resources. The test (below) was done on a low profile host with the following parameters:

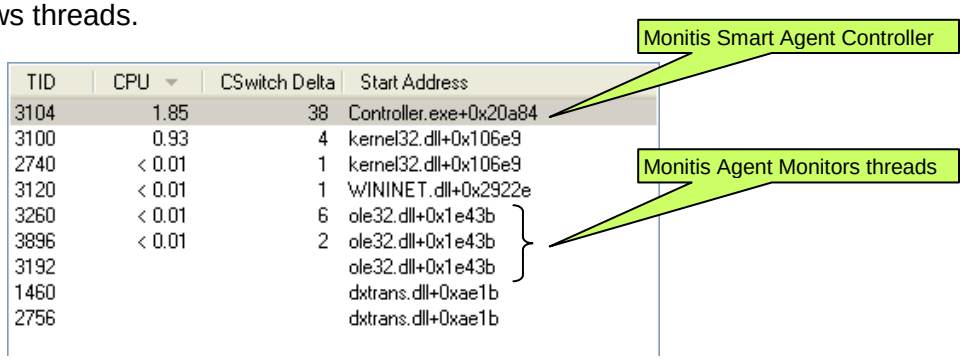
Linux	Windows
• CPU Intel Pentium Dual Core 2.4GHz • RAM 2048 MB • OS Linux-Ubuntu 11.04 (natty) • Kernel 2.6.35-25 • NET Ethernet 100Base-T (100Mbps)	• CPU Intel Pentium Dual Core 2.4GHz • RAM 1024 MB • OS WinXP SP3 • NET Ethernet 100Base-T (100Mbps)

The following monitors (checks) were activated: load average (LDAMON), CPU utilization (CPUMON), memory check (MEMMON) and monitoring for the Monitis agent process itself (PROCMON). In addition, the special test tools named "nethogs" (on Linux machine) and “NetLimits” (on Windows machine) were used internally to test the bandwidth generated by the agent. Note that some other third party applications were active on the monitored machine during the test (Firefox browser, Skype messenger, Text editor, terminal, etc.).

On the Linux machine the Monitis agent ran as a daemon and called the updater and testers as a LWP process (thread).

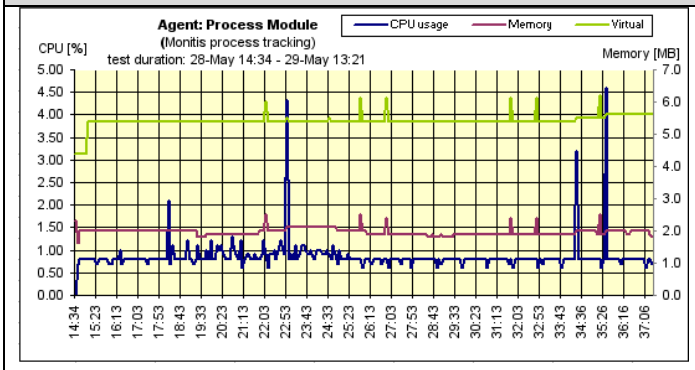
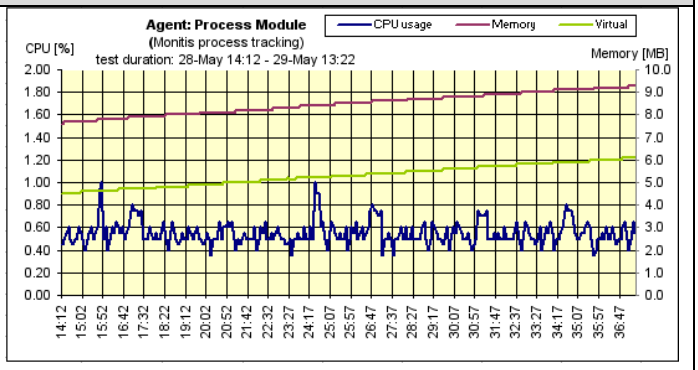
```
PID  PPID   LWP  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
3768    1     -  1   -   - -  1115 -      ?        00:00:11 monitis
   -    -  3768  0  80   0 -     - futex_ -        00:00:00 -
   -    -  3770  0  80   0 -     - hrtime -        00:00:00 -
   -    -  3772  0  80   0 -     - futex_ -        00:00:00 -
   -    -  3773  0  80   0 -     - futex_ -        00:00:00 -
   -    -  3774  0  80   0 -     - futex_ -        00:00:00 -
   -    -  3775  0  80   0 -     - futex_ -        00:00:00 -
   -    -  3776  0  80   0 -     - futex_ -        00:00:00 -
   -    -  3777  0  80   0 -     - futex_ -        00:00:00 -
   -    -  3778  0  80   0 -     - futex_ -        00:00:00 -
   -    -  3779  0  80   0 -     - futex_ -        00:00:00 -
   -    -  3780  1  80   0 -     - futex_ -        00:00:11 -
```

On the Windows machine the Monitis agent was activated as a simple application and therefore the chosen tests ran as Windows threads.



Monitis agent resources utilization

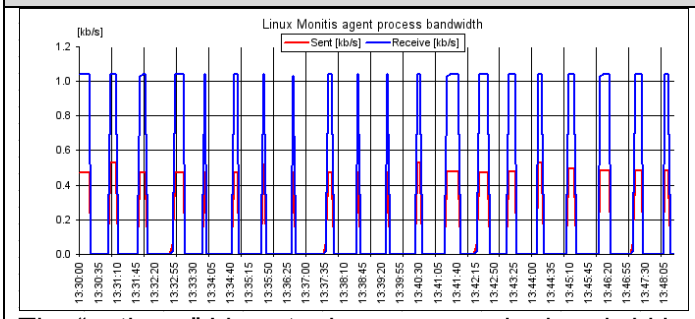
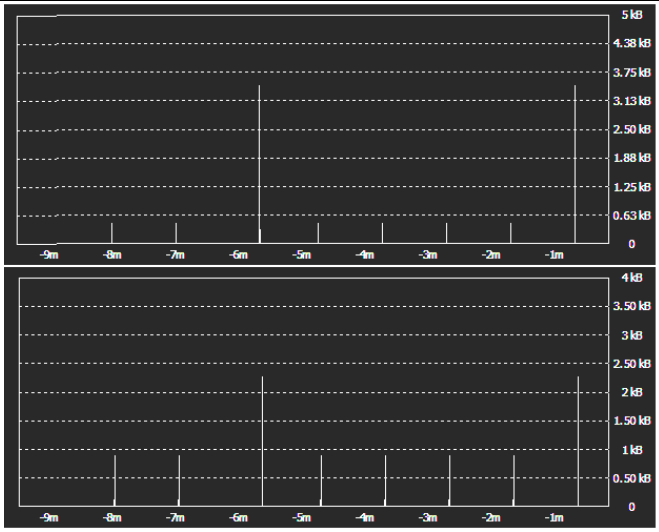
Normally the Monitis agent uses minimum resources on the monitored machine.

Linux	Windows
	
The internal Linux agent produces less than 1% of CPU utilization and occupies less than 1.4MB of RAM and 4 MB Virtual memory.	The internal Windows agent produces less than 0.5% of CPU utilization and occupies about 1.7MB of RAM and 1MB Virtual memory.

Note: we do not measure here resource utilization for every specific plugin because their behavior is not differing from the embedded monitors.

Monitis agent bandwidth utilization

Generally, the agent uses the internet connection quite efficiently as it is predefined that the agent requests the Monitis Backend for current tests configurations every 1 minute. The accumulated test results are sent every 5 minutes to the Dashboard.

Linux	Windows
	
The “nethogs” Linux tool measures the bandwidth of any process in real time. The simple calculation of average bandwidth shows that the received average bandwidth is near 0.25 kb/s and the sent bandwidth is near 0.15 kb/s. The average amount of received/sent data is about 112KB/hour and 67 KB/hour respectively.	The “NetLimits” Windows tool provides the network tracking per processes only (not bandwidth). So, the amount of measured sent data is about 96 KB/hour and received about 92 KB/hour. Therefore, the necessary bandwidth is near 0.22 kb/s for download and 0.2 kb/s for upload.

We do not measure the performance and bandwidth for every particular plugin because the stream of bytes which flows to the Monitis API is dependent on the nature of the plugin and on how many active plugins are assigned by a user to the agent. Please take into account that the pure size of the information that is generated by each plugin is about 200 - 250 bytes and data is sent to the Monitis Backend every 5 min. So, typically the output bandwidth that is consumed by plugins is about N*K*S/300 bytes/sec (where N is the number of active plugins, K is the factor that is defined as a ratio of the size of the entire HTTPS package to the body size (usually = 1.5) and S is the output bandwidth required by one plugin (normally approximately 200 bytes)).

In the table below we are displaying what is normally the required (approximately) output bandwidth size in dependence of the number of active plugins.

Count of active plugins	Required output bandwidth [kb/s]	Comment
1	0.001	K = 1.5 S = 200
5	0.005	
50	0.05	
100	0.1	

“As you can see, even a significant number of plugins only require very minimal bandwidth consumption to update”.

The agent's security

The Monitis agent provides security by using the following:

- Encrypted HTTPS protocol used for the connection with Monitis Backend.
- Client initiated encrypted connection (HTTPS) also eliminates altering firewalls, thus reducing risk for possible attacks.
- Heartbeat checks performed to ensure that agents are alive and connected with the Monitis Backend.
- The tests configuration and necessary parameters are kept on the Monitis Backend and sent periodically to the client by using an encrypted channel (HTTPS). The Client stores the current configuration internally in the memory (not in the file itself).
- Monitis performs daily and weekly backups, including backups to storage outside of the Monitis Backend data center.
- Offline agent alert is sent after 20 minutes of inactivity or if no data is being reported by the agent to the monitors. Upon customer's request this interval can be set as low as 5 minutes.

Summary

Thus, it can be assured that the internal agent;

- Produces very low network traffic
- Low CPU footprint
- Utilizes tiny amounts of RAM and Virtual memory
- Provides Offline Agent alerts for critical processes