SD-Mon Tool Description

1 Contents

Chance Log

Version	Date	Comment
1.0	03/02/2015	First version for D5.4 delivery
1.1	12/02/2015	New features: configurable localhost IP address, Makefile, free installation directory
1.2	18/02/2015	Added basic run-time visualization
2.0	07/04/2015	WEB online visualization for inter-group messages

2 Introduction

SD-Mon is a tool aimed to monitor SD-Erlang systems.

This purpose is accomplished by means a *shadow network* of agents, mapped on the running system.

The network is deployed on the base of a configuration file describing the network architecture in terms of hosts, Erlang nodes, global group and s_group partitions.

Tracing to be performed on monitored nodes is also specified within the configuration file.

An agent is started by a master SD-Mon node for each s_group and for each free node. Configured tracing is applied on every monitored node, and traces are stored in binary format in the agent file system.

The shadow network follows system changes so that agents are started and stopped at runtime according to the needs. Such changes are persistently stored so that the last configuration can be reproduced after a restart. Of course the shadow network can be always updated via the User Interface.

Exchanged inter-node and inter-group messages are displayed at runtime. As soon as an agent is stopped the related tracing files are fetched across the network by the master and they are made available in a readable format in the master file system.

3 Overall Architecture

SD-Mon is an Erlang application based on a master-agent architecture.

Currently sub-masters are not supported so each agent is directly linked to the master.

Agents run as Erlang hidden nodes in order not to propagate nodes connection across the network.

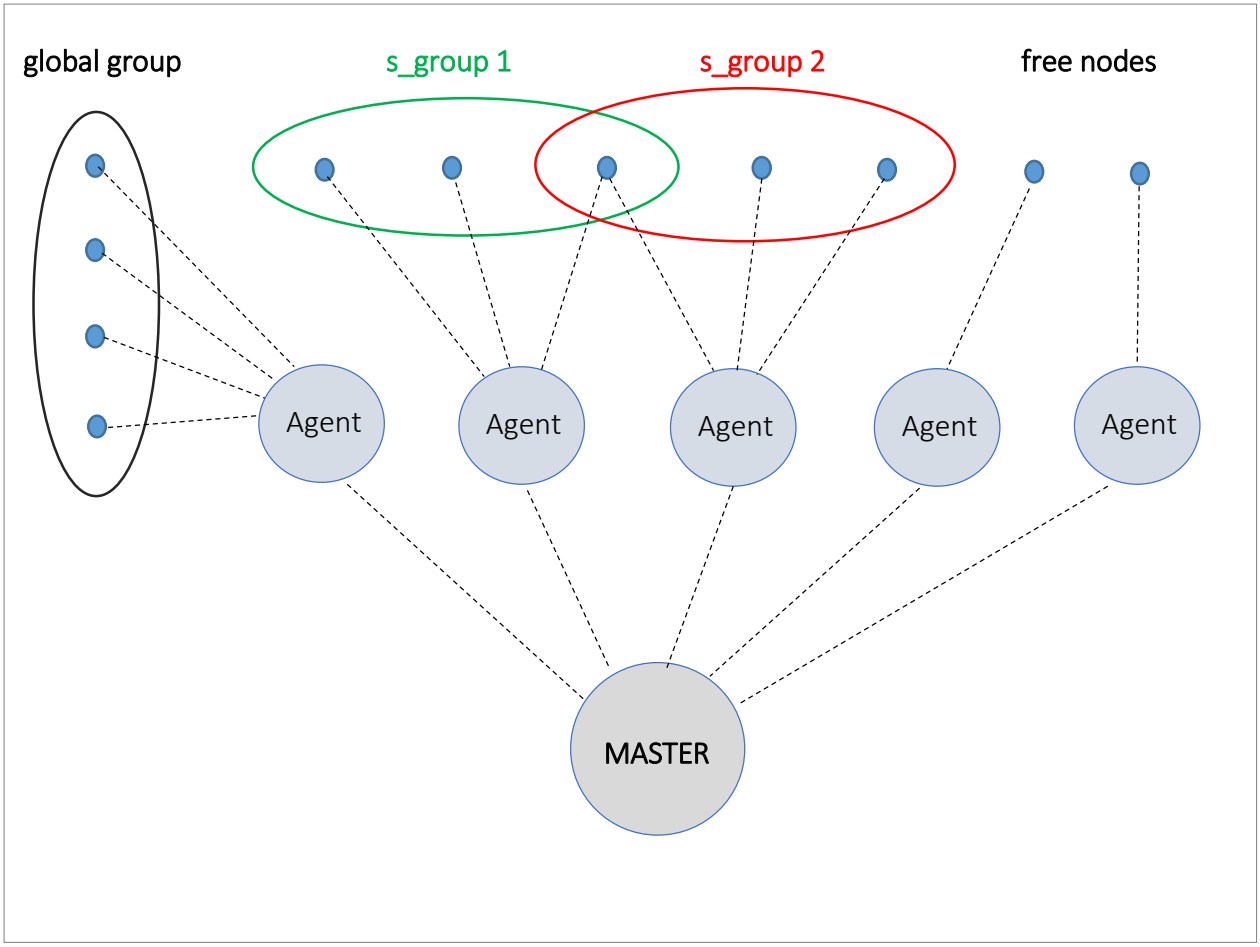


Figure 1: SD-Mon shadow network

3.1 Master

The Master is gen_server started and supervised by a supervisor process at application start-up.

The shadow network is deployed according to configuration. In order to limit the network usage the host running the maximum number of Erlang nodes is chosen to run the agent as well. Once the agents are started they ask for configuration data, consisting in nodes to be monitored and traces to be run on them. Each network change in s_group is cached by agents and notified to the master, which is the only one having a global network view. It takes care to reshape the shadow network accordingly: for instance if a new s_group is created a new agent is started, if an s_group is deleted the related agent is quitted, the tracing files are gathered from its host, and new agents are started for its nodes not controlled by any other agent.

Moreover, changes are dumped in new configuration files (originals are saved) in order to allows the tool to consider the last known configuration in case of restart.

User can start and stop the sdmon application, start and stop single agents and ask for the master status.

Trace files related to an agent, controlling a group or a free node, are gathered each time an agent is closed, at runtime or when the whole application is closed. Binary files are moved at the master host under the `BASEDIR/SD-Mon/traces/` directory and decoded on the fly, from now on being

`BASEDIR` the base directory where the tool is installed. A text file is produced for each controlled node and a summary of statistics is created for each agent and at global system level.

System events are logged by the master under `BASEDIR/SD-Mon/logs`.

3.2 Agents

Each agent takes care of an s_group or of a free node. At startup it tries to get in contact with its nodes and apply the tracing to them as stated by the master. Binary files are stored in the host file system.

Tracing is internally used in order to get aware of s_group operations (create or delete an s_group, add or remove nodes to an s_group) happening at runtime. An asynchronous message is sent to the master whenever one of these changes occurs.

Since each process can only be traced by a single process at the time, each node (included those belonging to more than one s_group) is controlled by only one agent.

When a node is removed from a group or when the group is deleted, another agent takes over, as shown in Figure 2.

When an agent is stopped, all traces on the controlled nodes are switched off.

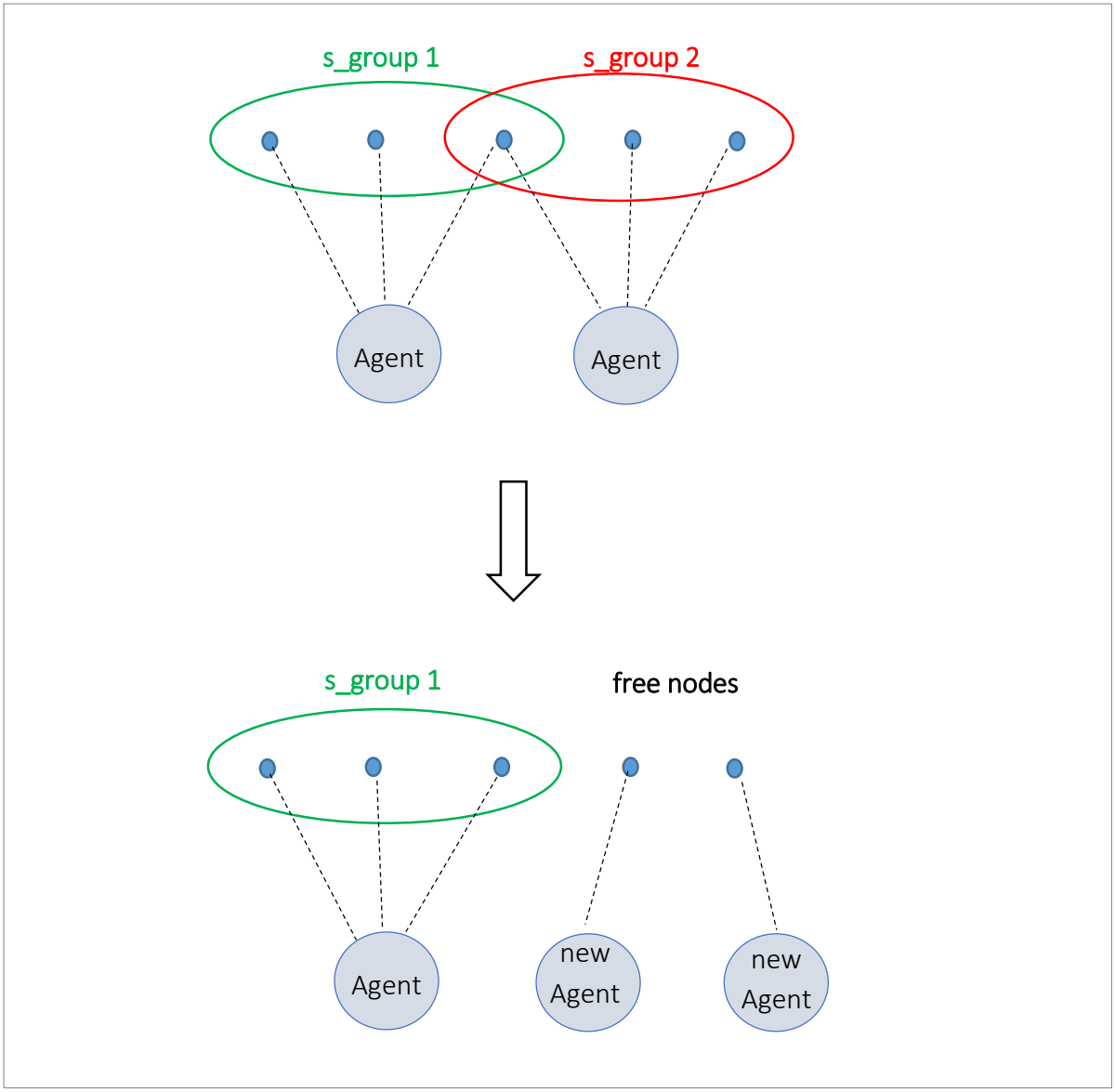


Figure 2: Delete s_group 2

3.3 Robustness

A double robustness mechanism is adopted against network turbulence and failures.

First of all every agent node is monitored so that as soon as a `nodedown` message is received a *direct monitoring* for that node is initiated: a dedicated process is spawned to poll the missing node and when it comes up again a `nodeup` notification is sent to the master who checks the agent status and align the system to any changes performed in the meantime.

This is done with the second mechanism which is based on a *configuration token* (see Figure 3). A token is an integer identifying the current valid configuration: it is sent to the agent (who stores it) every time something changes in the related part of the network. After a `nodeup` event the master sends the current token to the agent and if it does not match, the agent is configured from scratch. This strategy grants a fast recovery from network failures.

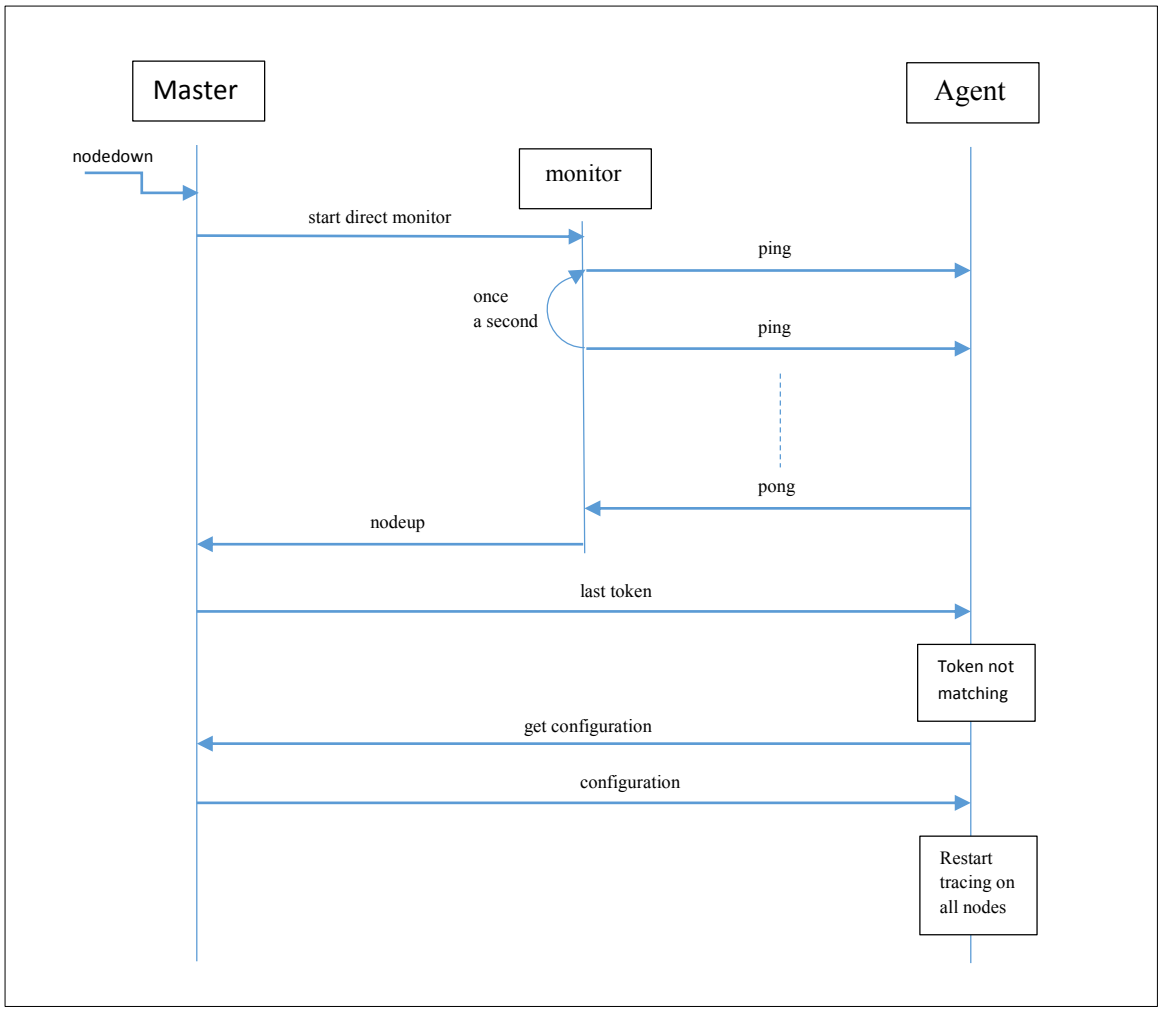


Figure 3: nodedown use case

Direct monitoring is also applied by the agents on the controlled nodes when a nodedown event occurs. Tracing is re-established at `nodeup`.

3.4 Configuration and Trace

The tool can be launched on a running system by means of 2 configuration files:

- group.config
- trace.config

The first one (also used by s_group processes) is a kernel environment and defines the group partition of the system in terms of s_group names and related Erlang nodes.

The second one specifies the tracing to be applied on each group of nodes.

Both files are placed under `BASEDIR/SD-Mon/config/`.

They can be compiled by hand or can be generated with the test command `gen_env` starting from a unique, high level configuration file named `test.config` which is placed in `BASEDIR/SD-Mon/test/config/`

This file consists of a set of Erlang terms defining the system to be observed. It is mainly composed of 5 sections:

1) First section is optional. Here you can state the IP address to be used for all Erlang nodes running on the localhost. If left empty, the address will be deduced from the operating system.
2) The second section just lists the hosts where the system is running.
3) The third section is an integer stating how many Erlang nodes (virtual machines) should be started for each host.
4) In this part groups are listed
5) The last part describes traces to be applied on each group

Currently a few tracing options are supported but more can be added in future.

- *Inter_node*
  Enables tracing of inter-node and inter-group messages, i.e. messages sent by a process to a destination outside its own node and group.
  N.B. this feature is always active since it is internally used by the tool. It is reported here for the sake of completeness.

  For each message sent by a monitored process out of its own Erlang node two entries will be reported in the related trace file:

  - The original trace message in the form:
    `{trace, FromPid, send, Msg, ToPid}`
  - The following tuple:
    `{trace_inter_node, FromNode, ToNode, MsgSize}`

  In the event that the destination node `ToNode` is not part of the same group of the sender node `FromNode`, a third entry will be reported:

  - `{trace_inter_group, FromGroups, ToGroups}`

  where the last two elements are the lists of groups of which the sender and the receiving nodes are part of. The empty list is used for free nodes.

  Example of inter_group message:

  ```
  {trace,<2922.112.0>,send,{vertex,3955,2597,45},<2918.125.0>}.
  {trace_inter_node,'node4@129.12.3.211','node4@129.12.3.176',24}.
  {trace_inter_group,[group3],[group2]}.
  ```

- *garbage_collection*
  Enables tracing on garbage collection.
  Original trace messages will be reported in tracing files (see official Erlang documentation).

- *scheduler*
  Produces information about process scheduling.
  Original trace messages will be reported in tracing files (see official Erlang documentation).

In section 4 it is also possible to define Erlang commands to be executed on the monitored system through a MFA tuple.

3.5 Post-Mortem analysis

When agents are stopped a new directory (whose name is a timestamp in the form `yyyymmdd_hhmmss`) is created in the master file system under the `traces` directory.

For each closed agent a sub-directory with its name is also created and all binary files related to the agent are moved in it. Then a readable version (.txt) of them is created for each controlled node and stored in the same directory together with a summary of statistical data for that agent (file `STATISTICS.txt`).

Lastly, statistics at system level are also created and stored in GLOBAL_STATISTICS.txt.

The following graph gives an example of directory structure.

```
traces/
└── 20150202_141045/
    ├── GLOBAL_STATISTICS.txt
    ├── sdmon_group1@127.0.1.1/
    │   ├── sdmon_group1@127.0.1.1_traces_0_node1@127.0.1.1
    │   ├── sdmon_group1@127.0.1.1_traces_0_node1@129.12.3.176
    │   ├── sdmon_group1@127.0.1.1_traces_0_node1@129.12.3.211
    │   ├── sdmon_group1@127.0.1.1_traces_1_node1@127.0.1.1
    │   ├── …
    │   ├── sdmon_group1@127.0.1.1_traces_node1@127.0.1.1.txt
    │   ├── sdmon_group1@127.0.1.1_traces_node1@129.12.3.176.txt
    │   ├── sdmon_group1@127.0.1.1_traces_node1@129.12.3.211.txt
    │   └── STATISTICS.txt
    ├── sdmon_group2@129.12.3.176/
    │   ├── sdmon_group2@129.12.3.176_traces_0_node1@129.12.3.176
    │   ├── …
    │   └── STATISTICS.txt
    └── sdmon_group3@129.12.3.211/
        ├── sdmon_group3@129.12.3.211_traces_0_node1@129.12.3.211
        ├── …
        └── STATISTICS.txt
```

The most relevant information included in the global statistic file are organized in 4 tables:

1. Node Tab
   Represents sent inter-node messages by any node.
   The format of each entry is:

   ```
   {FromNode, [{ToNode, TotalSize, NumOfMessages} | ..]}
   ```

   where:

- `FromNode` and `ToNode` are the sender and the receiving nodes
- `TotalSize` is the total size in bytes of all sent messages
- `NumOfMessages` is the total number of sent messages

2. Sent Tab

   Reports inter-node and inter-group messages sent by any node.
   The format of each entry is:

   ```
   {FromNode, {Inter_node, Inter_group, IG/IN}
   ```

   where:
   - `FromNode` is the sender node
   - `Inter-node` is the number of inter-node messages sent
   - `Inter-group` is the number of inter-group messages
   - `IG/IN` percentage of inter-group messages

3. Flow Tab

   Reports incoming and outgoing inter-node messages for any node.
   The format of each entry is:

   ```
   {Node, Incoming, Outgoing}
   ```

   where:
   - `Incoming` is the number of incoming inter-node messages
   - `Outgoing` is the number of outgoing inter-node messages

4. Bridge Flow Tab

   Bridges are nodes belonging to more than one group, which are supposed to vehicle all traffic crossing adjacent groups. Reports inter-node and inter-group messages sent by any node.
   The format of each entry is:

   ```
   {Node, B_Incoming, B_Outgoing}
   ```

   where:
   - `B_Incoming` is the number of incoming inter-node messages sent by nodes belonging to at least one of `Node`'s groups
   - `B_Outgoing` is the number of outgoing inter-node messages sent to nodes belonging to at least one of `Node`'s groups

NOTE: in the actual Erlang implementation Sent Tab, Flow Tab and Bridge Flow Tab also includes message size or average size for each field, not shown here for the sake of clearness.

An example of global statistic data can be found in Appendix 6.2.

3.6 Run-time visualization

A basic function has been introduced in order to follow system evolution at run-time for both sent inter-node and inter-group messages.

When inter-node messages are detected by the tracing processes, they sent a message to their own agent in the form:

```
{in, FromNode, ToNode}
```

The agent just forwards the message to a process named `sdmon_db` on the Master node.
In case the process is not started this message is lost and everything keeps going on as usual.

When, instead, the process is started (by calling `sdmon_db:start()`) it creates an ets table, which is an ordered set having the tuple `{FromNode, ToNode}` as key and associated to a counter.

After that the process enters the receiving loop, waiting for inter-node notifications from agents: every message just increases the counter for the involved nodes.

Moreover, every 0.1 seconds the whole table is dumped on the text file `/tmp/in_tab.txt`.
In a system composed of N nodes the table will hold Nx(N-1) elements in the worst case, represented within the file as a list of tuples `{{FromNode, ToNode}, Counter}`.

In this way it is possible to follow run-time updating for instance by means of the OS command:

```
tail -f /tmp/in_tab.txt
```

For demo purposes it is also possible to use the following commands:

`watch_internode` starts a terminal who follows the file (max. 45 entries)

`watch_internode2` starts 2 terminals (max. 45 entries each). See snapshot below.

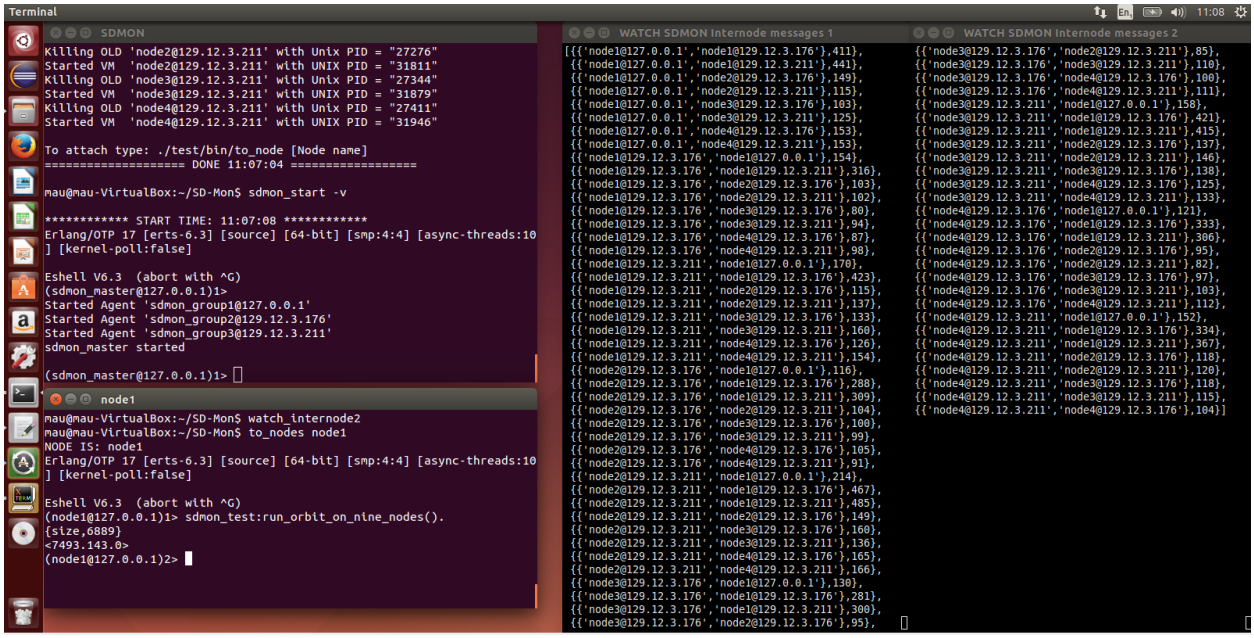


Figure 4. Run-time visualization

Inter-group messages are handled in a similar way, but they can be observed via a WEB interface.

When inter-group messages are detected by the tracing processes, they sent a message to their own agent in the form:

```
{ig, FromNode, ToNode}
```

As before, the agent forwards the message to the `sdmon_db` process which increases the related counter in a dedicated ETS table, but instead of dumping the table in a file, its contentment is sent to a Cawboy WEB server via a WEB socket. The WEB server is developed as an independent Erlang application and can be started and stopped with the bash scripts:

`sdmon_web_start` and `sdmon_web_stop`

Once the server is started the increasing counters of the sent inter-group messages are shown on the WEB page `localhost:8080` at the Master node host (Figure 5).

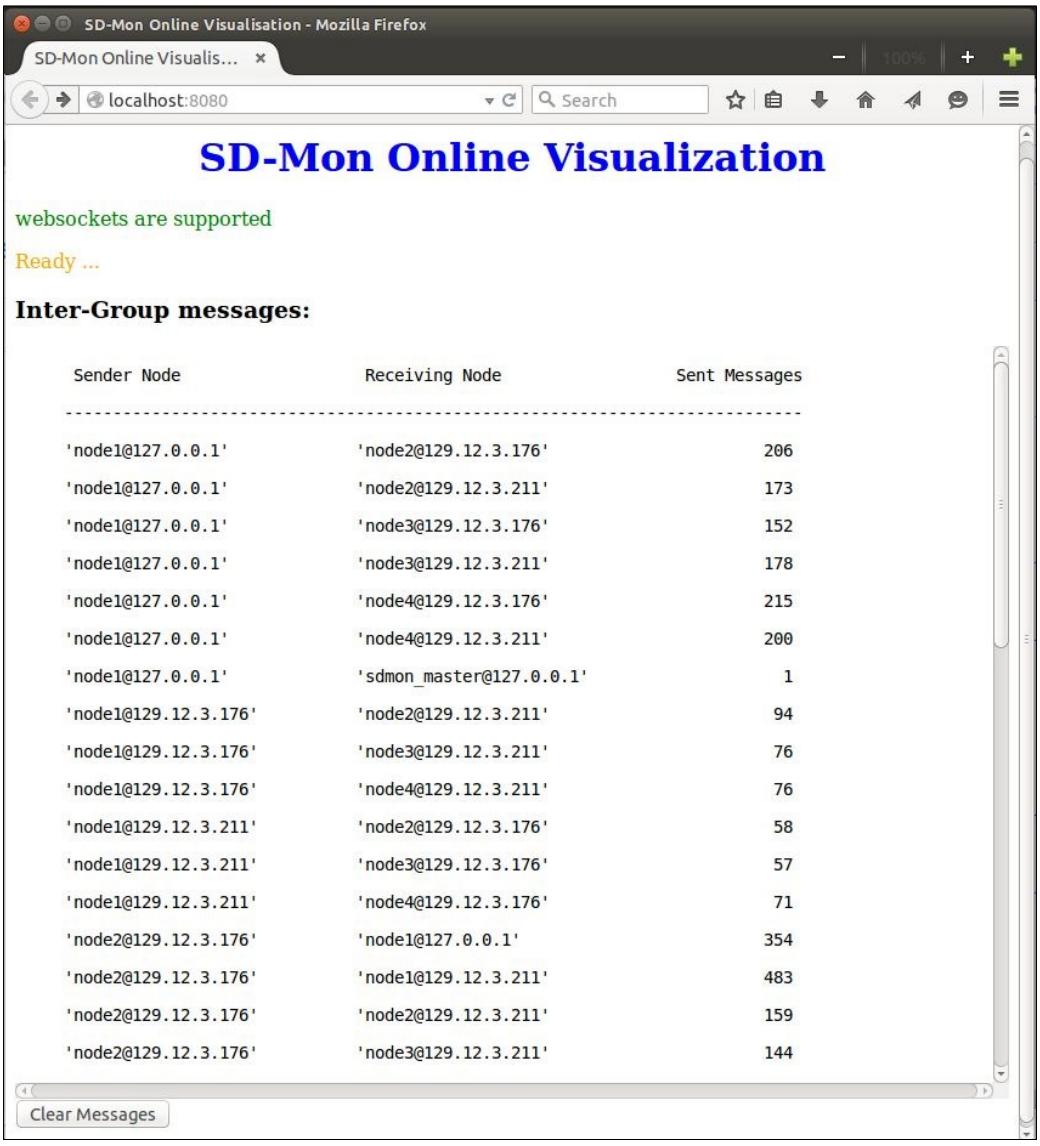


Figure 5: WEB Online Visualization

4 How to run SD-Mon

The following prerequisite applies to be able to run the tool:

- The user must be able to execute SSH commands on target non-local hosts without the needs to provide a password (use ssh-keygen if needed).
  The userid granted to access remote nodes via SSH without password must be defined in test.config file ('uid' tag).
- rebar must be in the command PATH

To install the tool, execute the following commands from a terminal:

```
git clone https://github.com/RefactoringTools/SD-Mon
cd SD-Mon
make
make web
```

The last command imports and installs in the deps/ directory all dependencies needed for WEB visualization.

SD-Mon is started by executing from the installation directory (BASEDIR/SD-Mon) the bash script:

```
> sdmon_start
```

configuration files are read and the shadow network is started. SD-Mon is normally executed in detached mode, without a shell. For debugging purposes a '-v' option is available: the Erlang shell on the master node is opened and the user can interact with the master or simply follows the system evolution.

By executing:

```
> sdmon_stop
```

SD-Mon is stopped: all tracing is removed, agents are terminated and all tracing files are downloaded in the master file system (/traces directory).

4.1 Example 1: SD-ORBIT on single-host

In this example SD-ORBIT is run on a system composed of 5 nodes, all running on local host and distributed in two s_groups, as depicted in Figure 6.

ORBIT parameters are:

- Generators: bench:g124/1
- Size of space: 50000
- Processors: 8

To execute this test open a terminal and type
(after replacing <BASEDIR> with the actual base directory):

```
export PATH=<BASEDIR>/SD-Mon/bin/:<BASEDIR>/SD-Mon/test/bin/:$PATH
cd <BASEDIR>/SD-Mon
cd test/config
rm test.config
ln -s test.config.orbit test.config
cd ../../
run_env
sdmon_start -v
```

Open a new terminal and type:

```
export PATH=<BASEDIR>/SD-Mon/bin/:<BASEDIR>/SD-Mon/test/bin/:$PATH
cd <BASEDIR>/SD-Mon
watch_internode
sdmon_web_start
```

The terminal started by the `watch_internode` command will show internode message counters updating at runtime, each entry in the form `{{FromNode, ToNode}, SentMessages}`.

To observe inter-group messages as they are sent, open a web browser and navigate to `localhost:8080`.

Now attach to node1 Erlang shell and start the Orbit test from there:

```
to_nodes node1
sdmon_test:run_orbit_on_five_nodes().
```

When Orbit run is completed go back on the first terminal and type:

```
application:stop(sdmon).
sdmon_web_stop
```

Find tracing and statistics in `<BASEDIR>/SD-Mon/traces`.

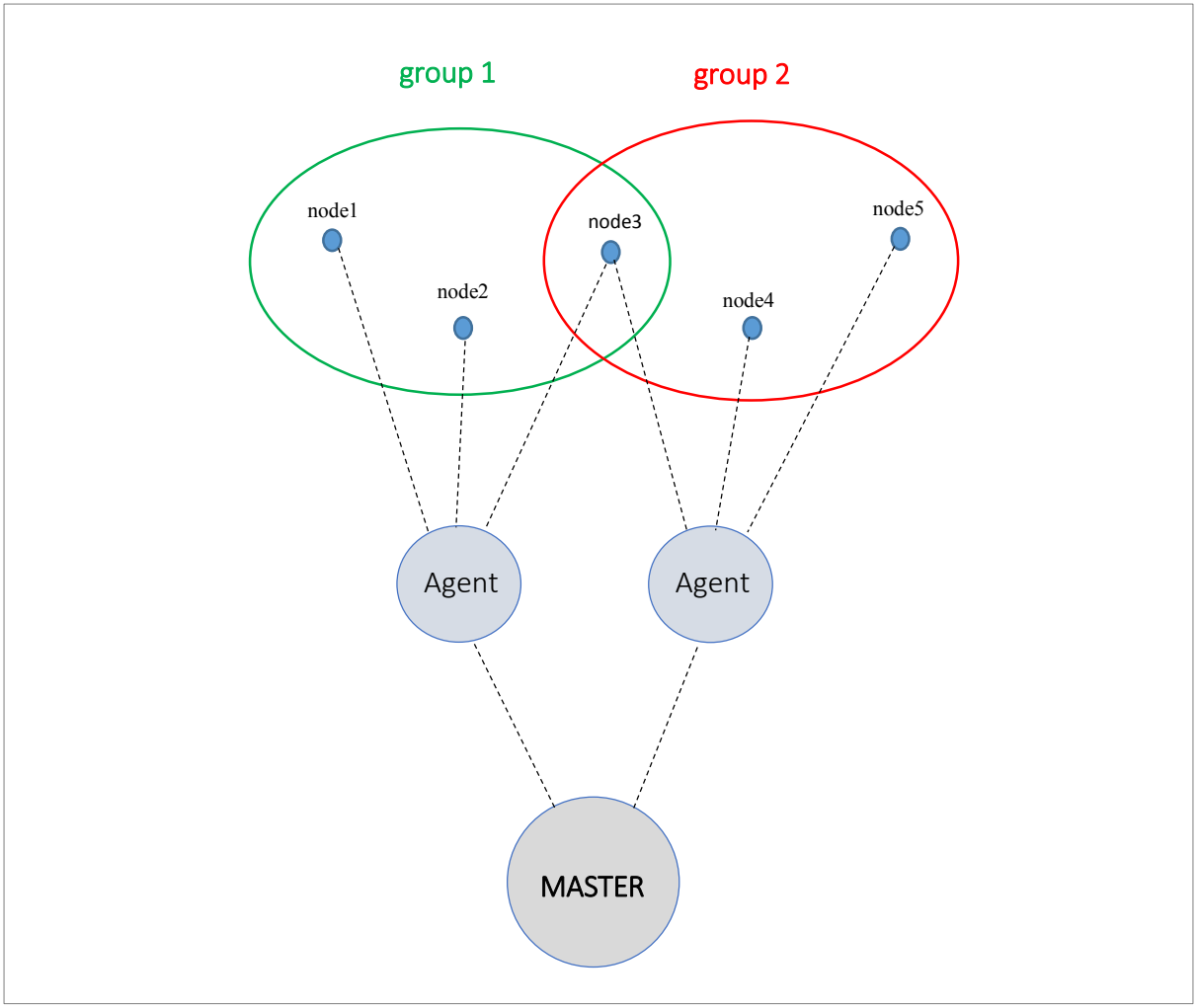


Figure 6: SD-ORBIT on five nodes

4.2 Example 2: SD-ORBIT on multi-host

In this case SD-Orbit is run on localhost and on two remote hosts: `myrtle.kent.ac.uk` (129.12.3.176) and `dove.kent.ac.uk` (129.12.3.211).

The orbit parameters are the same as before, with the exception of the size of space, decreased to 10000.

The system is composed of nine nodes grouped in two s_groups, as shown in Figure 7.

To execute this demo, edit the file `BASEDIR/SD-Mon/test/config/test.config.orbit_3h` and replace the string "md504" with the proper user id (see above).

Now open a terminal and (after replacing `<BASEDIR>`) type:

```
export PATH=<BASEDIR>/SD-Mon/bin/:<BASEDIR>/SD-Mon/test/bin/:$PATH
cd <BASEDIR>/SD-Mon
cd test/config
rm test.config
ln -s test.config.orbit_3h test.config
```

```
cd ../../
run_env
sdmon_start –v
```

Open a new terminal and type:

```
export PATH=<BASEDIR>/SD-Mon/bin/:<BASEDIR>/SD-Mon/test/bin/:$PATH
cd <BASEDIR>/SD-Mon
watch_internode2
sdmon_web_start
```

This time message counters will be displayed on 2 terminals.

To observe inter-group messages as they are sent, open a web browser and navigate to `localhost:8080`.

Now attach to node1 Erlang shell and start the Orbit test from there:

```
to_nodes node1
sdmon_test:run_orbit_on_nine_nodes().
```

When Orbit run is completed go back on the first terminal and type:

```
application:stop(sdmon).
sdmon_web_stop
```

Find tracing and statistics in `<BASEDIR>/SD-Mon/traces`.

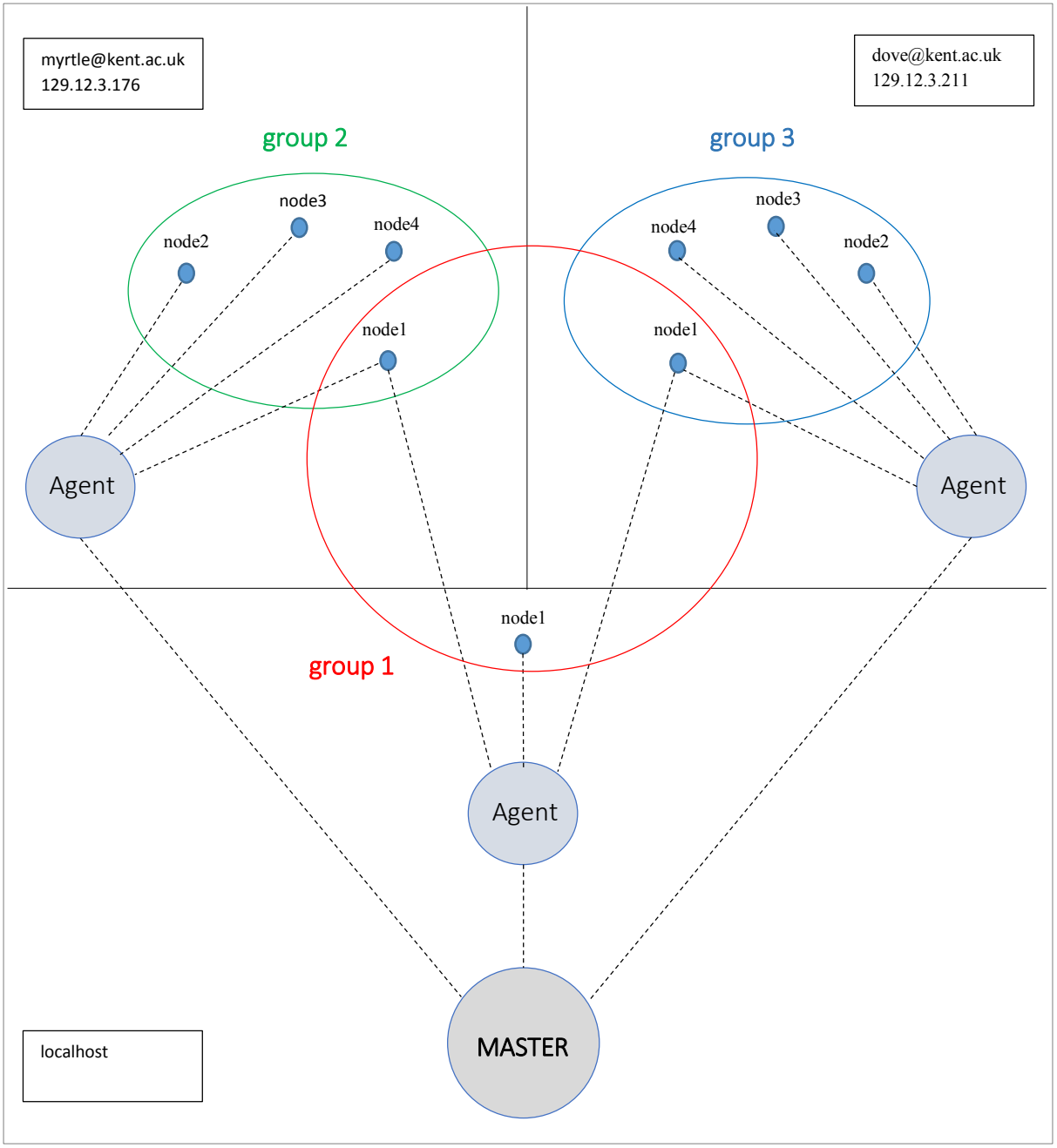


Figure 7: SD-ORBIT on nine nodes

Only `inter_node` tracing is enabled in this example, therefore only inter_node and inter_group messages are included in the resulting tracing files.

Here are some statistics reporting sent inter-node messages during a run of this example:

Node	Inter-node	Inter-group	IG/IN
node1@localhost	2271	1133	49.9%
node1@myrtle	1877	431	23.0%
node2@myrtle	2667	1775	66.6%
node3@myrtle	2556	1683	65.8%
node4@myrtle	2481	1596	64.3%
node1@dove	1874	388	20.7%
node2@dove	2583	1680	65.0%
node3@dove	2505	1606	64.1%
node4@dove	2345	1405	59.9%

Total number of inter-node messages	**21159**
Max inter-node messages node sender number of sent messages	node2@myrtle 2667
Total number of inter-group messages	**11697**
Max inter-group messages group sender number of sent messages	group2 5485
inter-group messages / inter-node messages	**55.3%**

The following table summarizes the message flow through "bridge nodes", which are the nodes belonging to more than one group and through which all communication toward and from external groups is supposed to pass.
The incoming messages reported in the table are those sent to the bridge node by any other node that is member of at least one of its groups. For instance, the incoming messages considered for `node1@myrtle` are those sent by `node1@localhost` and `node1@dove` (members of group1) and by `node2@myrtle, node2@myrtle, node3@myrtle` and `node4@myrtle` (members of group2).
The outgoing messages are instead those sent by the bridge node to any other node that is member of at least one of its groups. For instance, the outgoing messages considered for `node1@myrtle` are those sent to `node1@localhost` and `node1@dove` (members of group1) and to `node2@myrtle, node2@myrtle, node3@myrtle` and `node4@myrtle` (members of group2).
Inter-node messages not reported in the table are sent or received by external nodes, bypassing the bridging role of the bridge nodes.

Bridge Flow Tab	TOTAL INCOMING	TOTAL OUTGOING
node1@myrtle	**2540**	**1446**
node1@dove	**2752**	**1486**

Global statistic data for this example can be found in Appendix 6.2.

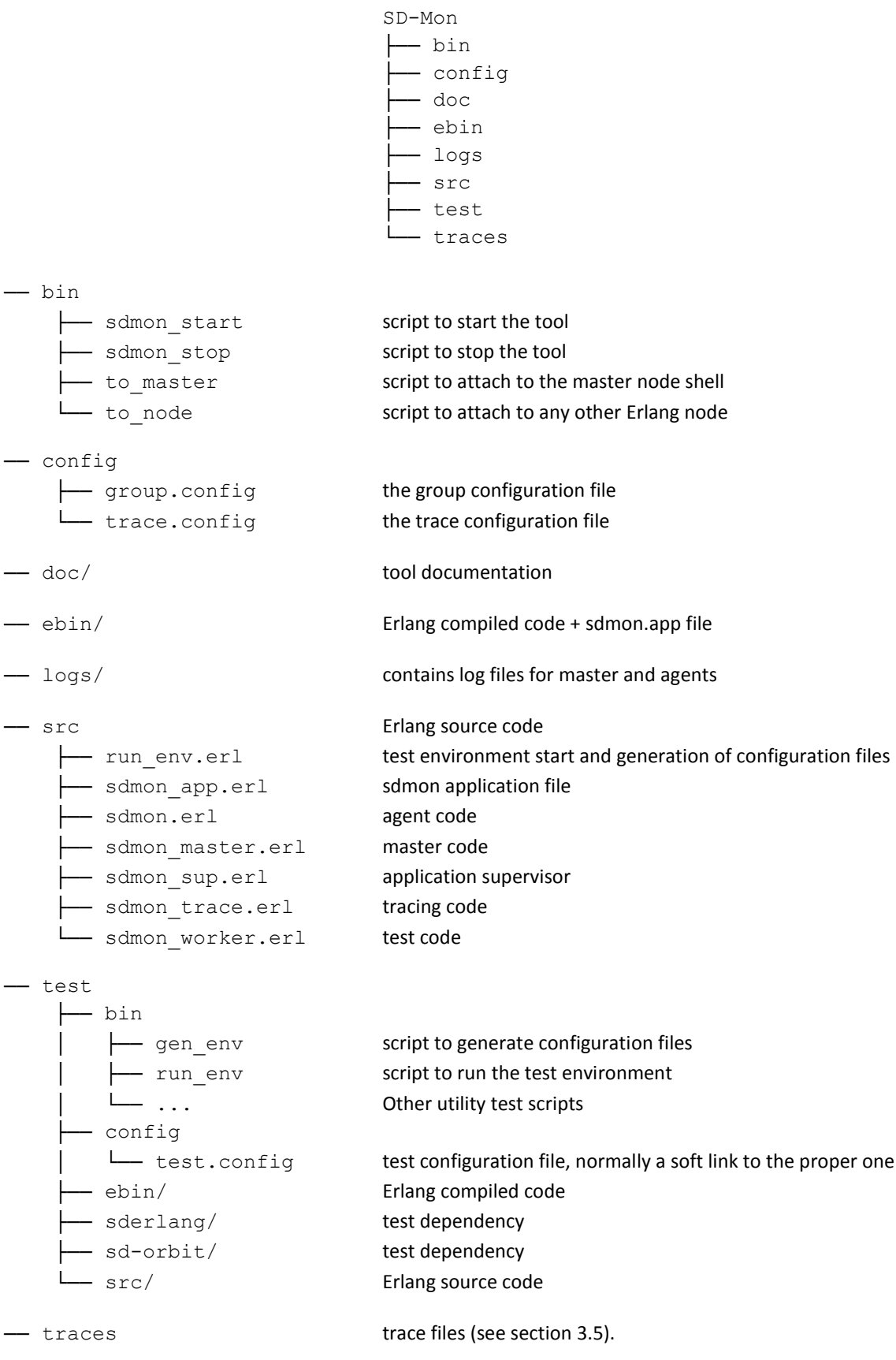
5 Future Plans

Rationale	Priority	Effort
1. TRACES: add more options (system profiles, …)	*	*
2. SCALABILITY and Nonfunctional requirements		
a- Submaster level	****	****
b- Efficient data handling (ETS on master)	***	***
3. Wombat Integration	?	***

6 Appendix

6.1 Directory Structure

The SD-Mon directory structure is represented in the schema below.



6.2 Global Statistics for Example 2

Node Tab (NxN)	From Node	To Node	Tot. size	inter-node
	node1@127.0.1.1	node1@129.12.3.176	39170	539
		node1@129.12.3.211	39311	599
		node2@129.12.3.176	32198	206
		node2@129.12.3.211	31472	173
		node3@129.12.3.176	31043	152
		node3@129.12.3.211	31580	178
		node4@129.12.3.176	32384	215
		node4@129.12.3.211	32039	200
		to_node1@127.0.1.1	1167	9
	node1@129.12.3.176	node1@127.0.1.1	17905	607
		node1@129.12.3.211	8667	456
		node2@129.12.3.176	3378	159
		node2@129.12.3.211	3258	155
		node3@129.12.3.176	2199	104
		node3@129.12.3.211	2718	129
		node4@129.12.3.176	2535	120
		node4@129.12.3.211	2535	120
		sdmon_group1@127.0.1.1	13574	27
	node1@129.12.3.211	node1@127.0.1.1	19181	694
		node1@129.12.3.176	8211	389
		node2@129.12.3.176	2424	114
		node2@129.12.3.211	2799	133
		node3@129.12.3.176	2631	125
		node3@129.12.3.211	3054	144
		node4@129.12.3.176	2778	132
		node4@129.12.3.211	2664	126
		sdmon_group1@127.0.1.1	7054	17
	node2@129.12.3.176	node1@127.0.1.1	17696	667
		node1@129.12.3.176	11670	555
		node1@129.12.3.211	10986	577
		node2@129.12.3.211	3870	183
		node3@129.12.3.176	3102	146
		node3@129.12.3.211	3843	180
		node4@129.12.3.176	4032	191
		node4@129.12.3.211	3540	168
	node3@129.12.3.176	node1@127.0.1.1	16301	584
		node1@129.12.3.176	11013	523
		node1@129.12.3.211	10482	553
		node2@129.12.3.176	3447	164
		node2@129.12.3.211	3402	161
		node3@129.12.3.211	3765	178
		node4@129.12.3.176	3903	186
		node4@129.12.3.211	4344	207
	node4@129.12.3.176	node1@127.0.1.1	15637	544
		node1@129.12.3.176	11232	534
		node1@129.12.3.211	10551	553
		node2@129.12.3.176	3636	172
		node2@129.12.3.211	2880	136
		node3@129.12.3.176	3783	179
		node3@129.12.3.211	3795	179
		node4@129.12.3.211	3879	184
	node2@129.12.3.211	node1@127.0.1.1	16344	597
		node1@129.12.3.176	11253	536
		node1@129.12.3.211	10716	563
		node2@129.12.3.176	3711	176
		node3@129.12.3.176	3939	186
		node3@129.12.3.211	3441	163
		node4@129.12.3.176	3897	185
		node4@129.12.3.211	3747	177
	node3@129.12.3.211	node1@127.0.1.1	15743	555
		node1@129.12.3.176	11121	529
		node1@129.12.3.211	10329	542
		node2@129.12.3.176	3870	183
		node2@129.12.3.211	4089	193
		node3@129.12.3.176	3774	177
		node4@129.12.3.176	3408	162
		node4@129.12.3.211	3447	164
	node4@129.12.3.211	node1@127.0.1.1	13202	342
		node1@129.12.3.176	11193	532
		node1@129.12.3.211	11283	592
		node2@129.12.3.176	3813	181
		node2@129.12.3.211	3744	177
		node3@129.12.3.176	3681	173
		node3@129.12.3.211	3636	171
		node4@129.12.3.176	3714	177

Sent Tab

Node	inter-node	inter-group	IG/IN
node1@127.0.1.1	2271	1133	49.9%
node1@129.12.3.176	1877	431	23.0%
node2@129.12.3.176	2667	1775	66.6%
node3@129.12.3.176	2556	1683	65.8%
node4@129.12.3.176	2481	1596	64.3%
node1@129.12.3.211	1874	388	20.7%
node2@129.12.3.211	2583	1680	65.0%
node3@129.12.3.211	2505	1606	64.1%
node4@129.12.3.211	2345	1405	59.9%

Flow Tab

Node	Incoming	Outgoing
node1@127.0.1.1	4590	2271
node1@129.12.3.176	4137	1877
node2@129.12.3.176	1355	2667
node3@129.12.3.176	1242	2556
node4@129.12.3.176	1368	2481
node1@129.12.3.211	4435	1874
node2@129.12.3.211	1311	2583
node3@129.12.3.211	1322	2505
node4@129.12.3.211	1346	2345