

# CESNET Technical Report 9/2008

## G3 System – Distributed Measurement Architecture

TOM KOŠŇAR

Received 28.11.2008

### Abstract

G3 system aims to be a set of complex tools designed for large scale and continuous network infrastructure measurement visualisation and reporting. After modifications of internal processing scheme and several months of testing we are able to give brief description of G3 system internal architecture that enables to run measurement in distributed infrastructure.

*Keywords:* SNMP, Internet traffic measurement

## 1 Introduction

We started to introduce G3 system from its navigation, visualisation and reporting capabilities in previous technical reports ([1] and [2]). Now it is time to describe how the system behaves while doing the fundamental task – data measurement, processing and storage. We should go back to the history first and mention our motivation and needs at the beginning of the development. Here are our most influential starting points regarding the system measurement, processing and storage architecture:

### *Existing systems experience*

We have tested and/or used many infrastructure monitoring systems based on several concepts years ago and finished with home made system (GTDMSII - developed and used for many years). We have learned that we need a system which is really open in terms of measurement – any imaginable method of data collecting must be adopted (although we expected major role of SNMP – *Simple Network Management Protocol*). We also wanted to design a system which would give the freedom to redefine (or create for new cases) the structure of collected data and structure of objects/devices that are measured – open tree model (no pairs of values limitations no flat data model nor anything similar).

### *Open architecture*

System must be open enough to adopt any monitoring requests that can occur in the future and which may lead to measurement of absolutely new groups of information.

### *Objects identification*

We decided to be absolutely independent on any measurement specific identifiers (SNMP indexes for interfaces, hardware components and others for example). System must keep the possibility to configure the way how any exclusive identifier is constructed.

*Item abstraction*

We wanted to have the possibility to merge all items having the same meaning (but different methods of collecting for example) behind a single virtual item when needed.

*Dynamic non-aggressive behaviour*

There is a lot of items that have to be processed from two consequent steps of data collecting. This is typically the case of SNMP counter type items. System should be able to catch some dynamics in network infrastructure behaviour while not being too aggressive in terms of requesting data.

*Aggregation*

Aggregation “by default” was the initial idea – there has to be adequate support for it at data processing and storage level. For example system should be able to generate single aggregated course of temperature regardless it is based on all temperature sensors of particular device or on specific sensor of all devices in a computer room.

## 2 G3 measurement architecture

### 2.1 Processing framework and items definitions

Requirements mentioned above and our reflections led to specific design. Processing itself is spread between the base two parts of the system: processing framework and items definitions. Processing framework provides the traditional set of running processes – real running system consisting of master controlling process, measurement daemons and optional storage daemons – all of them behaving according to measurement configuration. Item definitions on the other side are not only configurations of real items to be measured. Items are defined as a hierarchy – they are bound into trees containing three levels of items – group level items, section level items and something like object-id items. Item definitions at any level may contain code or references to library functions (is necessary in some cases). While the measurement daemons start to process items in configured order starting at group level and continuing down in hierarchy, any item can also run appropriate processing part of another item regardless of its group or its position in hierarchy. This freedom enables to achieve absolutely open architecture from measurement point of view and the only thing that must be taken in mind is timing and order of actions. On the other side system complexity is the trade off. The advantage for those who need to extend the system with new groups of information are existing predefined groups of items (*System, Interfaces, IP, ICMP, UDP* and others) also including vendor specific SNMP MIB subtrees as well as examples of non-SNMP measured ones. It is advantage especially for *Perl* programmers - whole system as well as item definitions are written in *Perl* (*Practical Extraction and Report Language*).

Here is an example of item tree – extract of currently defined items including internal item identifier and its mapping to OID (*SNMP Object Identifier*):

## Group: HW

## Section: cdspCardStatusEntry

cdspCardLastHiWaterUtilization =>  
 1.3.6.1.4.1.9.9.86.1.1.1.1.4  
 cdspCardResourceUtilization =>  
 1.3.6.1.4.1.9.9.86.1.1.1.1.3  
 cdspCardState => 1.3.6.1.4.1.9.9.86.1.1.1.1.2

## Section: cerentEnvMonTemperatureStatsEntry

cerentEnvMonTemperatureStatsCurrentValue =>  
 1.3.6.1.4.1.3607.2.80.20.1.30  
 cerentEnvMonTemperatureStatsDescr =>  
 1.3.6.1.4.1.3607.2.80.20.1.20  
 cerentEnvMonTemperatureStatsThresholdHigh =>  
 1.3.6.1.4.1.3607.2.80.20.1.40

## Section: cesnet\_cla\_AGC

cesnet\_cla\_claAGC => 1.3.6.1.4.1.8057.7.20.9.1.2  
 cesnet\_cla\_claAGCDescription =>  
 1.3.6.1.4.1.8057.7.20.9.1.3

...  
 ...

## Section: entSensorValueEntry

entSensorPrecision => 1.3.6.1.4.1.9.9.91.1.1.1.1.3  
 entSensorScale => 1.3.6.1.4.1.9.9.91.1.1.1.1.2

...  
 ...

## Group: ICMP

## Section: icmp

icmpInAddrMaskReps => 1.3.6.1.2.1.5.13  
 icmpInAddrMasks => 1.3.6.1.2.1.5.12  
 icmpInDestUnreachs => 1.3.6.1.2.1.5.3

...  
 ...

## Group: INTERFACE

## Section: Dot3StatsEntry

dot3StatsAlignmentErrors => 1.3.6.1.2.1.10.7.2.1.2  
 dot3StatsCarrierSenseErrors =>  
 1.3.6.1.2.1.10.7.2.1.11  
 dot3StatsDeferredTransmissions =>  
 1.3.6.1.2.1.10.7.2.1.7  
 dot3StatsExcessiveCollisions =>  
 1.3.6.1.2.1.10.7.2.1.9  
 dot3StatsFCSErrors => 1.3.6.1.2.1.10.7.2.1.3

...  
 ...

## Section: cMsDwdmFECCurrentEntry

cMsDwdmFECCurrentBitErrCor =>

```

    1.3.6.1.4.1.3607.2.40.3.2.1.2
    cMsDwdmFECCurrentUncorWords =>
    1.3.6.1.4.1.3607.2.40.3.2.1.6
    ...
    ...
    Section: ifXEntry
    ifHCInBroadcastPkts => 1.3.6.1.2.1.31.1.1.1.9
    ifHCInMulticastPkts => 1.3.6.1.2.1.31.1.1.1.8
    ifHCInOctets => 1.3.6.1.2.1.31.1.1.1.6
    ...
    ...
    Group: IP
    Section: ip
    ipForwDatagrams => 1.3.6.1.2.1.4.6
    ipFragCreates => 1.3.6.1.2.1.4.19
    ipFragFails => 1.3.6.1.2.1.4.18
    ...
    ...
    Group: SNMP
    Section: snmp
    snmpInASNParseErrs => 1.3.6.1.2.1.11.6
    snmpInBadCommunityNames => 1.3.6.1.2.1.11.4
    snmpInBadCommunityUses => 1.3.6.1.2.1.11.5
    ...
    ...
    Group: SYSTEM
    Section: system
    sysContact => 1.3.6.1.2.1.1.4
    sysDescr => 1.3.6.1.2.1.1.1
    ...
    ...
    Group: UDP
    Section: udp
    udpInDatagrams => 1.3.6.1.2.1.7.1
    udpInErrors => 1.3.6.1.2.1.7.3
    ...
    ...

```

## 2.2 Processing framework configuration

Measurement part of the G3 system follows two level configuration scheme – master configuration and measured devices configurations. Master configuration contains directives for processing framework and consists of two parts. Common part controls mechanism of data storage, contains common measurement parameters, location of item definitions, notification directives and other parameters including even functions when needed (see example below). Measurement processes part

(*'groups\_to\_measure'* in example below) controls number of measurement processes to run, contains location of appropriate devices configurations and also may contain parameters which override relevant settings in common part. Here is an example of such master configuration file – its syntax is *Perl* associative array reference.

```
{
# Item definitions location
'default_item_dir' => '/data/G3/item_dir',
'snmp_port' => 161,
'snmp_max_pdu_len' => 65535,
'snmp_discovery_step' => 3600,
'snmp_max_repetitions' => 1024,
'max_measure_step_time' => 900,
'min_measure_step_time' => 40,
'measure_step_strategy' =>
    ['r','r','S','S','l','l','r','l','l'],
'filter_clean_unmatching_data' => 0,
'store_data' => 1,
'store_do_fork' => 0,
'max_forked_store_processes' => 1,
'store_root_dir' => '/data/G3/results',
'store_base_directory' => 'boxes',
'max_time_to_store' => 300,
'notification_ttl' => 14400,
'notify_errors_to' => 'somebody@somewhere',
'notify_timeout_to' => 'timeouts@somewhere',
'socket_rrd_storage' => 1,
'socket_rrd_storage_path' =>
    '/data/G3/sock/G3_rrd_storage.sock',
'socket_rrd_storage_read_bufsize' => 8388608,
'socket_rrd_storage_write_bufsize' => 524288,
'socket_rrd_storage_step' => '0.1',
'shmem_rrd_storage' => 0,
'shmem_rrd_storage_max_records' => 8192,
'shmem_rrd_storage_wait_before_send_term_signal' => 30,
'shmem_rrd_storage_max_records_check_step' => 10,
'shmem_rrd_storage_step' => '1.5',
'shmem_rrd_storage_size' => 4096000,
'store_label' => sub {
    package G3_1;
    use strict 'refs';
    my($items, $label, $group) = @_;
    return index(lc $label, 'accounting') >= 0
        ?
        ERR() : OK();
};
}
```

```

}
'groups_to_measure' => [
  {
    'min_measure_step_time' => 300,
    'root_object_dir' => '/data/G3/objects-cfg/0'
  },
  {
    'min_measure_step_time' => 300,
    'root_object_dir' => '/data/G3/objects-cfg/1'
  },
  ...
  ...
  {
    'measure_step_strategy' => [60,120],
    'root_object_dir' => '/data/G3/objects-cfg/12'
  },
  {
    'root_object_dir' => '/data/G3/objects-cfg/13'
  }
]
}

```

Measured device configuration file has always name *.configuration* and is placed in device directory. Device directory is dedicated directory which is created in directory specified as *'root\_object\_dir'* in master configuration file. Full path of device configuration file in this example is: */data/G3/objects-cfg/6/10.10.1.1/.configuration*. Device configuration contains parameters needed to measure appropriate device and also may contain other parameters overriding inherited parameters from master configuration. Here is an example of device configuration – its syntax is *Perl* associative array reference again.

```

{
  'snmp_version' => '2c',
  'snmp_community' => 'v2c_some_community',
  'object_description' => 'router-X',
  'snmp_host' => '10.10.1.1',
  'group_topology' => Backbone Part NORTH',
  'group_technical' => 'router',
  'store_object_directory' => '10.10.1.1',
  'group_location' => 'PoP in City Y'
}

```

Device configuration may be extended with filtering configuration. Filtering enables to store some part of measured data (specific interface traffic for example) separately under dedicated name to specific location. Such storage location may be shared by configuration of other devices. It can help for example to keep long term statistics of external line utilisation of multi homed network in cases when line

end points frequently move from device to device. There is sufficient functionality within user G3 user interface and G3 reporter to merge and aggregate such course in any case, but this option makes it independent on erasing measured data of any participating device. Filter configurations are files in *.FILTER* directory which is sub-directory in device directory. Filter configuration filename is the root name under which filtered data will be stored. In case we want to store filtered data only we may set up *'filter\_clean\_unmatching\_data'* parameter to non-zero value (globally set in master configuration above). An example of such configuration follows – same syntax as before.

```
{
# group - base group for which we are currently
#           storing data
'group' => 'INTERFACE',
'store_data' => 1,
# records can be expanded - with key items
#           from 'parent' group
# ...which is SYSTEM in this case (items like sysName,...)
'expand_records' => 1,
# items that should be stored into the data set
# dedicated to this filter
'store_label' => sub {
  my ($items,$label,$group)=@_;
  return OK if ($label eq 'range-of-validity');
  return OK if (index(lc($label),'keyby')>=0);
  return OK if (index(lc($label),'bitrate')>=0);
  return OK if (index(lc($label),'pktrate')>=0);
  return OK if (index(lc($label),'speed')>=0);
  return OK if ($label eq 'sysName');
  return OK if ($label eq 'sysLocation');
  return ERR;
},
# filter matching conditions for one or more instances
'condition' => sub {
  my $rec=shift;
  # $rec is anonymous hash with measured values
  # we want interface to be UP
  return OK if (
    value_from_tree($rec,0,['ifAdminStatus'])==1
    &&
    value_from_tree($rec,0,['ifOperStatus'])==1
    &&
    (
      # must have one of the following addresses configured
      value_from_tree($rec,' ',['ifKeyByIP']) eq '10.10.10.3'
```

```

    ||
    value_from_tree($rec, ' ', ['ifKeyByIP']) eq '10.10.11.21'
  )
);
return ERR;
},
# something like post-processing
'apply' => sub {
  my ($cfg,$data,$items,$filter_group,
      $filter_id,$coordinates)=@_;
  # We require exact count of interfaces that match
  if (scalar @$coordinates != 1) {
    # bad count...
    # we may want to send a message...
    send_filter_match_error_mailmsg($cfg,$data,$items,
                                    $filter_group,$filter_id,$coordinates);
    # we will remove all marks
    # which were set up in 'condition' function
    remove_filter_signs($cfg,$data,$filter_group,
                       $filter_id,$coordinates);
    return OK;
  }
  # We may want to setup administratively interface speed..
  # my ($group,$instance)=@{$coordinates->[0]};
  # my $rec=value_from_tree($data,{},[$group,$instance]);
  # $rec->{IfSpeedAdministrative}=800000000
  #       if exists $rec->{IfSpeedAdministrative};
  return OK;
},
# This filter notifications goes there
'notify' => 'errors@somewhere'
}

```

### 2.3 Items definitions

Items definitions resides in *'default\_item\_dir'* directory. Each item is defined in separate file. Syntax of each file is *Perl* associative array reference. There is no space to describe all options and possibilities here but I'm giving examples of item definitions just to illustrate how is the processing spread among processing framework and items. Here are examples of item hierarchy from the top – 3 items, each representing different level.

Group level example first:

```

'SYSTEM' => {
  'section' => 'group',
  'lbl' => 'SYSTEM',

```

```

'group' => 'OBJECT',
'convert_measured' => sub {
    my ($cfg,$items,$state,$data,$rec,$label)=@_;
    my $val=value_from_tree($rec,{},$label);
    # FIXED instance ID for SYSTEM
    my $instance=(keys %$val)[0];
    # set validity information
    set_value(
        $val->{$instance},
        'range-of-validity',1,
        get_time($val->{$instance},$label,$state)
    );
    return OK;
},
'compare-group' => sub {
    use strict;
    use G3_1;
    my ($cfg,$items,$state,$last_data,$data,
        $last_rec,$rec,$label,$key)=@_;
    my ($instance,$retval,$task);
    foreach $task
        ('compare-instance','postprocess-instance') {
        $retval=item_compare(
            $cfg,$items,$state,$last_data,$data,
            value_from_tree($last_rec,{},$key),
            value_from_tree($rec,{},$key),$label,
            (keys %{
                value_from_tree(
                    $data,
                    {},
                    ['SYSTEM']
                )
            }))[0],
            $task
        );
        return $retval unless $retval==OK;
    }
    return OK;
},
'compare-instance' => sub {
    my ($cfg,$items,$state,$last_data,$data,
        $last_rec,$rec,$label,$key)=@_;
    my $x=value_from_tree($rec,{},$key);
    if (scalar keys %$x) {
        # preset -run_counters
        $x->{-run_counters}=1;
    }
}

```

```

    }
    my $all_fields=
        all_exclusive_keys(
            value_from_tree($last_rec, {}, [$key]),
            value_from_tree($rec, {}, [$key])
        );
    my ($field, $retval);
    foreach $field (@$all_fields) {
        $retval=
            item_compare(
                $cfg, $items, $state, $last_data, $data,
                value_from_tree($last_rec, {}, [$key]),
                value_from_tree($rec, {}, [$key]),
                master_key($field),
                $field, 'prepare-value');
        return $retval unless $retval==OK;
    }
    return OK;
},
'postprocess-instance' => sub {
    my ($cfg, $items, $state, $last_data,
        $data, $last_rec, $rec, $label, $key)=@_;
    my $all_fields=
        all_exclusive_keys(
            value_from_tree($last_rec, {}, [$key]),
            value_from_tree($rec, {}, [$key]));
    my ($field, $retval);
    foreach $field (@$all_fields) {
        next unless exists $items->{$field};
        $retval=
            item_compare($cfg, $items, $state,
                $last_data, $data,
                value_from_tree($last_rec, {}, [$key]),
                value_from_tree($rec, {}, [$key]),
                master_key($field),
                $field, 'postprocess-value');
        return $retval unless $retval==OK;
    }
    return OK;
},
'store-as-parent' => \&other_store,
'store-file' => 'navigation',
'order' => '1'
}

```

Section level example second:

```
'system' => {
    'section' => 'section',
    'lbl' => 'system',
    'group' => 'SYSTEM',
    'index_field' => sub { return 'system' },
    'force_use_iids' => [0],
    'order' => '1',
    'can_create_record' => '1'
}
```

And object-id like item last:

```
'sysName' => {
    'section' => 'system',
    'req' => 1,
    'lbl' => 'sysName',
    'group' => 'SYSTEM',
    'oid' => '1.3.6.1.2.1.1.5',
    'prepare-value' => \&prepare_other,
    'process-value' => \&process_other,
    'store-value-when' => \&is_other_value,
    'store-value' => \&other_store,
    'store-file' => 'navigation'
}
```

## 2.4 G3 system generic architecture

Full scheme of G3 system is in Figure 1. Basic work flow since the system starts is the following:

- system starts its core – *master process* and reads master configuration file (part of *measurement configuration*)
- master process checks and/or sets necessary components (basic directory structure, log mechanism, validity of item definitions path and others)
- master process starts appropriate common storage process when needed (in case of non-zero *socket\_rrd\_storage* or *shmem\_rrd\_storage* parameter)
- master process parses *groups\_to\_measure* section in master configuration and starts child measurement process – *measurement module* for each record found
- measurement process (*measurement module*) checks (periodically) appropriate *root\_object\_dir* for devices configuration files, last measurement results files, measurement state files (parts of *measurement configuration*)
- measurement process (*measurement module*) serialises (after measuring each device) devices configurations according to requested measurement start times and starts to measure first device in the queue

- collected and processed data are stored (using configured mechanism), device state file and last measurement result file are updated and device configuration is inserted into the queue according to next measurement time (fixed or generated - depends on *measure\_step\_strategy* configuration parameter)

There are several mechanisms how measurement process stores collected and processed data – everything depends on combination of basic storage parameters. Following list describes system behaviour when appropriate parameters are set (non-zero value means ON):

#### *store\_data*

setting this parameter is a necessity otherwise NO data will be stored; switching it of may make sense for measurement tuning or for operational monitoring when item definitions may generate alarms or notifications but without any need to store data

#### *store\_do\_fork*

measurement process forks immediately after collecting and processing data (regardless of storage method used); child process starts to store the data while parent process immediately starts to measure next device in the queue (it may be the same device again); this method may be used when a very short time step is requested (very low values of *min\_measure\_step\_time*)

#### *shmem\_rrd\_storage*

measurement process (or its child optionally) writes processed data to shared memory segment and starts to measure next device in the queue (or exits when being a child); storage process (*storage*) periodically checks shared memory segment, writes data found to real storage area and cleans the memory

#### *socket\_rrd\_storage*

measurement process (or its child optionally) writes processed data to *UNIX socket* and starts to measure next device in the queue (or exits when being a child); storage process (*storage*) periodically checks other side of the socket and writes received data to real storage

#### *nothing set*

measurement process (or its child optionally) stores processed data itself; storing data without forking may delay measurement of next device in the queue in this case

Probably the best setup for large (>10e5 items) or “aggressive” (frequent measurement) configurations is *socket\_rrd\_storage* without *store\_do\_fork*. Choosing optimal parameters also depends on average device response times. In general the essential parameter is proportion between the number of parallel measurements (number of measurement processes is unlimited) and corresponding amount of data to store on one side and throughput of real storage mechanism including I/O at OS level, databases, data formats, supporting tool efficiency and others. G3 system uses two types of databases. First one is proprietary (*Perl Storable* based) and is used for text items. It is fast and represents small volumes of data only – no

brake in the storage process. Second one is *RRD* (Round Robin Database) based on *rrdtool* packages and *RRDs Perl* API. G3 system is configured to store multiple items (currently 16) into a single *RRD* to speed up the things. Although the *RRA* (Round Robin Archive) structure may be set up in configuration we usually use system default setup. It (single *RRA*) currently consists of 7 aggregation intervals with approximately 3400 rows (summary) and three values per row (minimum, maximum, average) covering 12 year period. There has to be also adequate support (especially for large configurations) at the operating system level to use resources effectively. There are parameters that depend on local conditions and monitoring purposes, but there are common rules that should be taken in mind anytime when setting up monitoring with *RRD* based storage – I can undoubtedly recommend to read papers like [3] and provide step by step testing when preparing critical *RRD* based installations. Our production single host installation currently runs on 4xQuad Core Xeon X7350, 64GB RAM, 8xSAS 15K 73GB drive in RAID10 (64K chunk, adaptive read ahead, write-back cache at the controller) with Debian-etch installed. OS scheduler is set to *noop*, file-system read ahead is set to *16KB*, number of requests in queue is set to *512*. This installation runs G3 system with currently 14 parallel measurement processes running. Data are stored through *UNIX socket* mechanism. Overall number of measured devices is currently around 120 and total number of items measured is around 550000. Average measurement time step is 750 seconds with current limits at 40 and 1200 seconds as you can see in Figure 2.

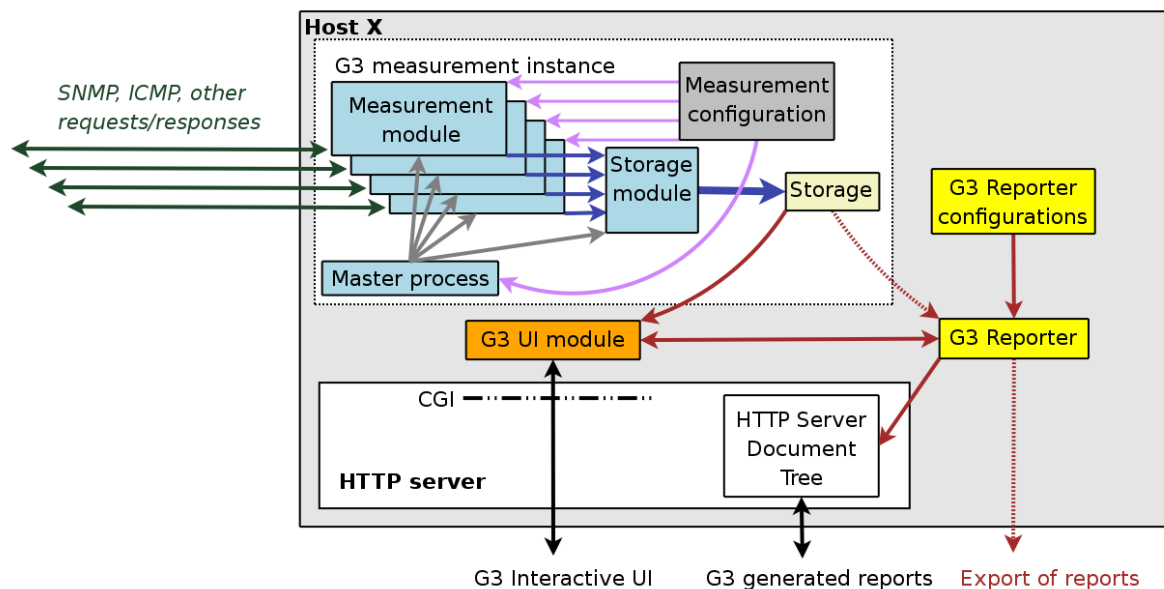
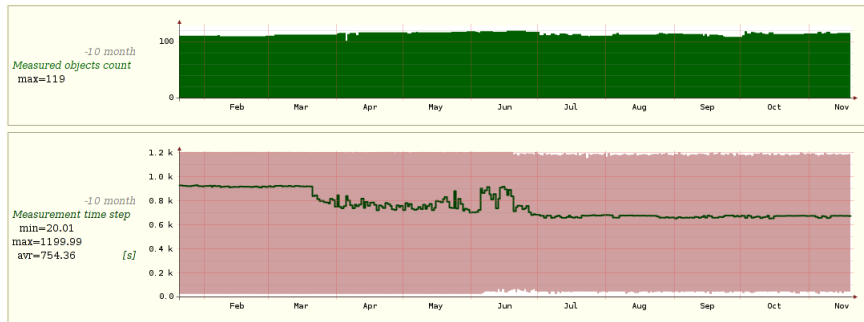


Figure 1. G3 system generic scheme.



**Figure 2.** Basic single G3 system installation measurement parameters at CESNET.

## 2.5 Distributed G3 system measurement

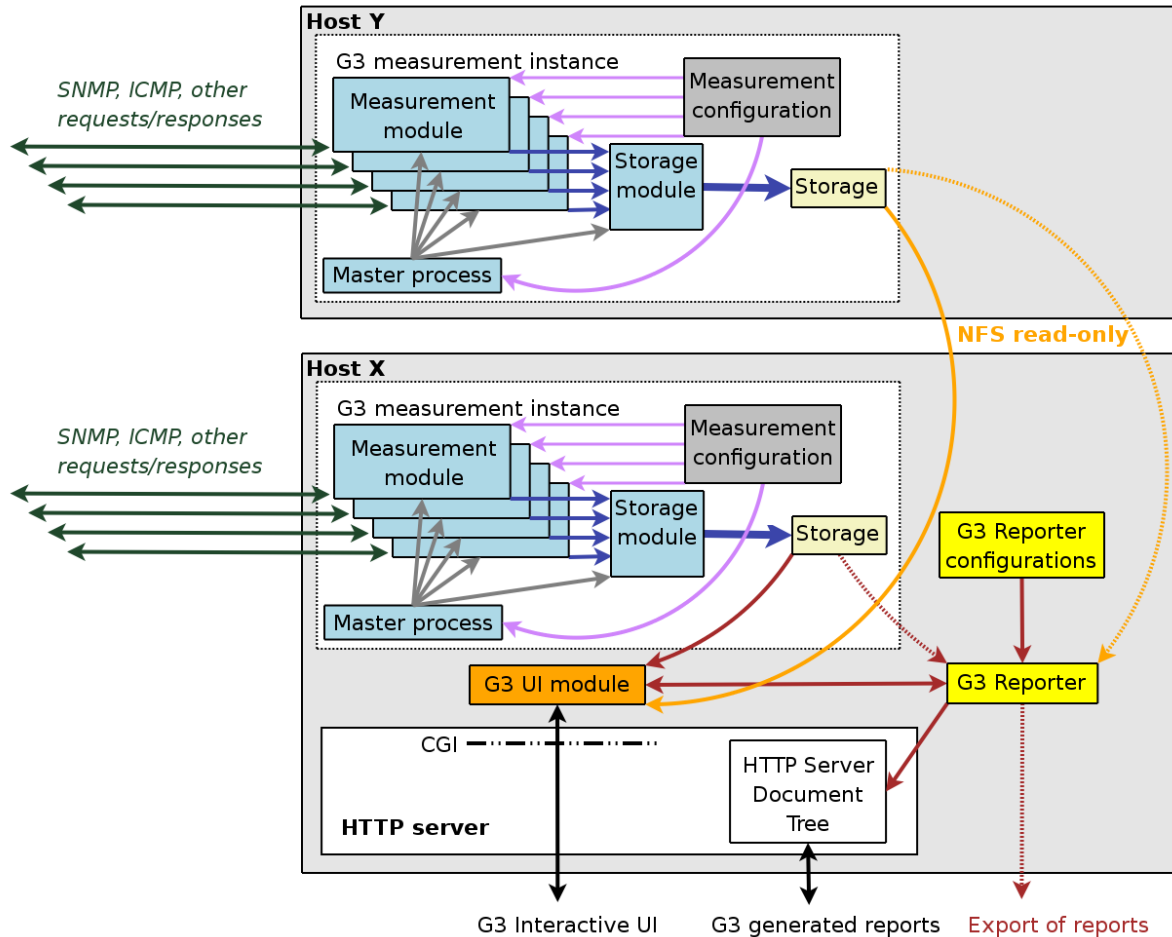
There are cases when single host installation is not suitable or not efficient enough for what we need. Reasons may be for example - no enough powerful hardware available, absolutely different monitoring requirements on devices groups to be measured, requested aggressive and/or separate measurement of some device and others. There is always a chance to install multiple instances of monitoring system but on the other side users probably do not prefer solution with several monitoring hosts each with its own user interface – its confusing. Therefore we improved G3 system to satisfy requirements on separate measurement while keeping single user interface that merges data of all devices regardless of where they were measured. The only limitation was the necessity of at least read-only access to all measured data at file-system level for user interface (*G3 UI module*) – otherwise we would lost native functionality of *rrdtool* packages especially when running aggregated data retrieval (which is something we definitely need). Basic scheme of G3 system in distributed measurement architecture is in Figure 3 – it is just an extension of generic scheme shown in Figure 2.

There are no changes visible in Figure 3 only add-ons. Orange lines show read-only NFS mapping of a part of measuring host storage to host where user interface runs. There was a lot of changes done, but they are hidden deeply in G3 system libraries. Measured data contain full path from G3 storage root at the measuring host – there are many purposes for it – and we had to develop automated path re-mapping mechanisms that translates path derived identifiers on-fly and when needed only. Adequate extensions were done in functions preparing source data for navigation tree generation and other extensions in measurement configuration to give enough flexibility in measurement setup. Current state of G3 system enables to run distributed measurement in both basic modes - separate measurement of groups of devices and separate measurement of device components.

### 2.5.1 Separate measurement of groups of devices

This distributed measurement mode is the simple one. Measurement configuration may be in principle the same as in single host measurement – no extensions are needed. Device distribution scheme as well as data merging is shown in Figure 4.

There are only two things that must be done to make the whole system functional in this configuration:



**Figure 3.** G3 system distributed measurement generic scheme.

- appropriate part (results of measurement) of measuring only host storage must be mapped to proper place (directory where tree with results begins) under any name at host running user interface
- mapped directory must be added to configuration of user interface (to be used when constructing navigation tree or when accessing stored data)

File-system mapping may look with relation to Figure 3 and configuration examples above like this:

```
Y:/data/G3/results/boxes -> X:/data/G3/results/boxes.Y
```

And configuration file for interface modules at host X may look like this:

```
{
dirs => {
  # results root directory
  '/data/G3/results/boxes.Y' => {
    # shall be used ?
    active => 1
  },
  '/data/G3/results/boxes' => {
    active => 1
  }
}
```

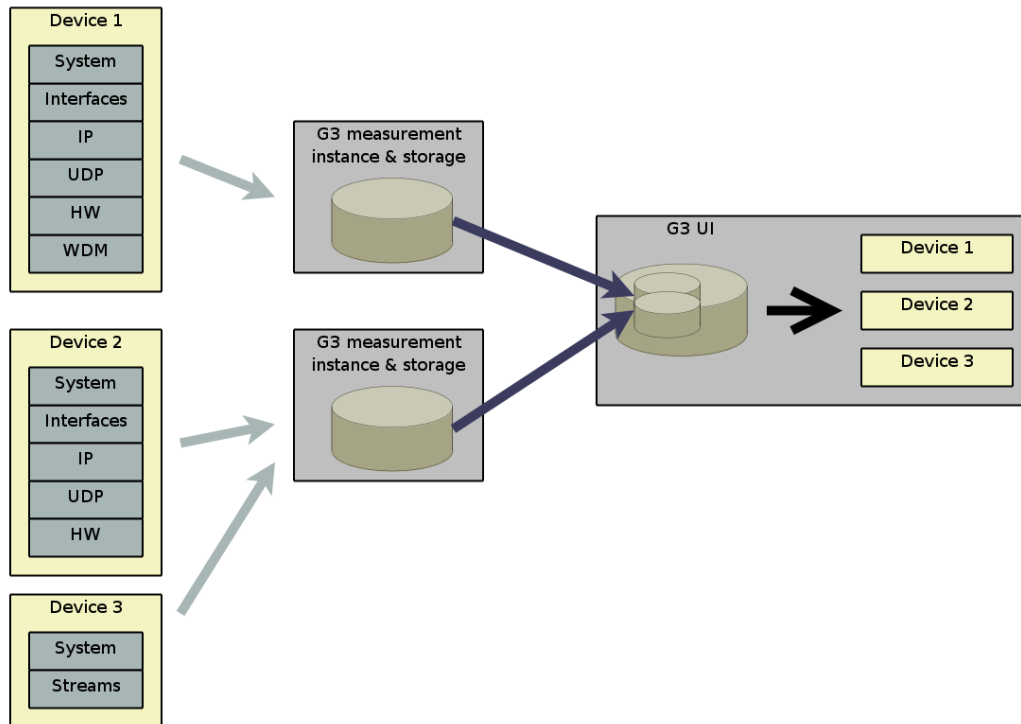


Figure 4. G3 system distributed measurement – groups of devices.

```

    },
    # shared memory key
    shmem_base_key => 'KEYX'
}

```

### 2.5.2 Separate measurement of device components

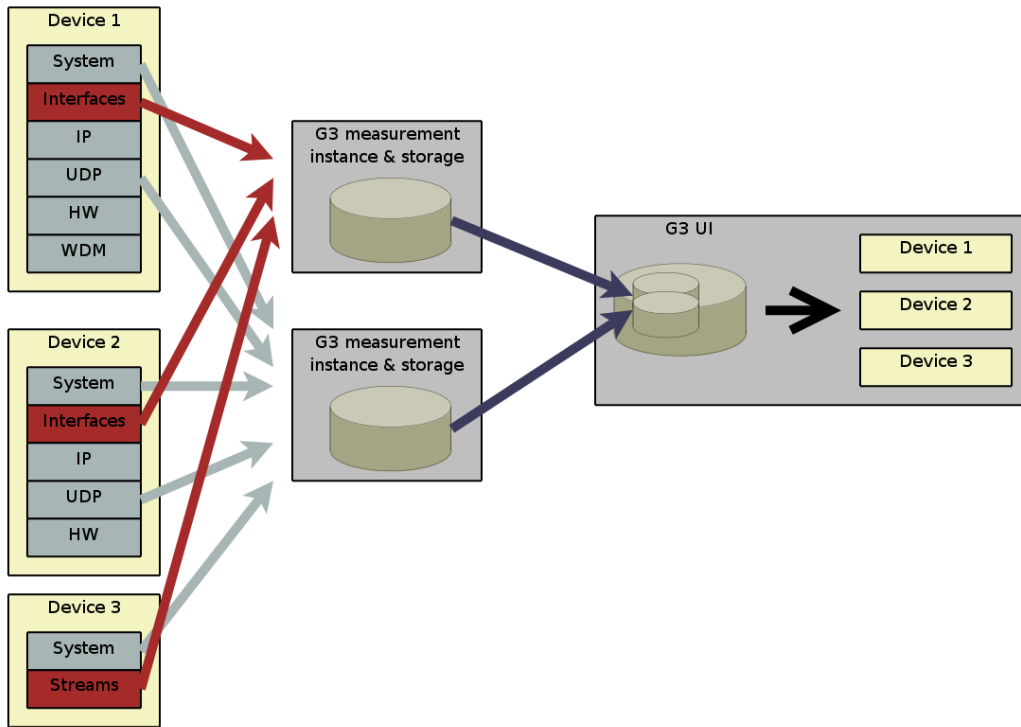
This distribution is more interesting from the configuration point of view. Distribution scheme is shown in Figure 5.

According to example above we may want to measure *Interfaces* and *Streams* (correspond with *INTERFACE* and *VSTREAM* classes of objects) from one host and the rest from another host. We must configure both measurements carefully to avoid duplications. For example multiple measurement of items that are stored into *RRD* databases may cause multiplication of values at user interface level because system provides automated aggregations. Therefore we must explicitly configure which items to measure at each host – in master configuration. Important part of master measurement configuration file for Host X may look like this:

```

{
  ...
  ...
  # - sign (standalone) switches everything off
  # + sign with item identifier enables items
  # (enabling group level items enables all their children)

```



**Figure 5.** G3 system distributed measurement – groups of device components.

```
'wanted_items_condition' => '-,+INTERFACE,+VSTREAM',
...
...
'store_label' => sub {
    package G3_1;
    use strict 'refs';
    my($items, $label, $group) = @_ ;
    # some special items are always on
    # (cannot be switched off)
    # we may choose which of them to store
    return OK() if index($label, 'sysMeasureStep') >= 0;
    return OK()
        if index($label, 'sysMeasureTimeSpent') >= 0;
    return OK()
        if index($label, 'sysMeasureDelay') >= 0;
    # we don't want to run exact interface accounting
    return ERR() if index(lc $label, 'accounting') >= 0;
    # finally we disable everything
    # except 'INTERFACE' and 'VSTREAM' groups
    return ERR() if $group ne 'INTERFACE'
        &&
        $group ne 'VSTREAM';
    # OK for 'INTERFACE' and 'VSTREAM' groups
    return OK();
},
```

```

...
...
}

```

Configuration for Host Y is complementary and will look like this:

```

{
...
...
'wanted_items_condition' =>
    '-,+HW,+SYSTEM,+UDP,+ICMP,+SNMP,+IP',
'store_label' => sub {
    package G3_1;
    use strict 'refs';
    my($items, $label, $group) = @_ ;
    return ERR() if $group eq 'INTERFACE'
        ||
        $group eq 'VSTREAM';
    return OK();
},
...
...
}

```

File-system mapping as well as configuration of interface modules will be the same as in previous case.

We tested this setup (separate measurement of device components) for three months (after stabilising extended system). We tested not only interactive user interface functionality but also G3 reporter in this configuration. Experimental maps of network utilisation and network health were periodically generated from this distributed configuration during that period.

### 3 Conclusion

G3 system is used for large scale infrastructure monitoring of the CESNET2 network. As the backbone network becomes more hybrid containing L1-L3 devices we observe growth of items that are requested to be measured. Especially number of optical parameters of DWDM systems (including our at home developed optical amplifiers) or number of monitored sensors is evidently growing. We have to prepare scenarios when no enough powerful single hardware will be available (with acceptable price to performance ratio) to measure all required items from the whole set of devices. Distributed architecture of infrastructure monitoring system is a step towards the solution.

## References

- [1] KOŠŇAR, T. *G3 System User Interface Prototype*. Technical report 9/2005<sup>1</sup>, Praha: CESNET, 2005.
- [2] KOŠŇAR, T. *G3 System – Reporter*. Technical Report 14/2007<sup>2</sup>, Praha: CESNET, 2007.
- [3] PLONKA, D.; GUPTA, A.; CARDER, D. Application buffer-cache management for performance: running the world's largest MRTG. *In 21st Large Installation System Administration Conference (LISA '07)*, Dallas (TX), 11-16 November 2007. Available online<sup>3</sup>.

---

<sup>1</sup> <http://www.cesnet.cz/doc/techzpravy/2005/g3/>

<sup>2</sup> <http://www.cesnet.cz/doc/techzpravy/2007/g3-reporter/>

<sup>3</sup> [http://www.usenix.org/events/lisa07/tech/full\\_papers/plonka/plonka.pdf](http://www.usenix.org/events/lisa07/tech/full_papers/plonka/plonka.pdf)