MINER –
A Measurement Infrastructure for Network Research

Christof Brandauer
Salzburg Research
Jakob-Haringer-Str. 5/III
A-5020 Salzburg, Austria
brandauer@salzburgresearch.at

Thomas Fichtel
Salzburg Research
Jakob-Haringer-Str. 5/III
A-5020 Salzburg, Austria
mail@fichtel.at

***Abstract*—Network measurements often require the coordinated use of multiple tools on distributed vantage points. Without any form of supporting command and control system, the management and execution of a systematic measurement study is a tedious mission. In this paper we present MINER which seeks to simplify this task. MINER is a programmable measurement infrastructure that integrates existing measurement tools and provides its users higher-level services on top of them. It enables users to define measurement activities, schedule executions and retrieve their results. The services are accessible via a tool-agnostic and unified programming interface with which measurement applications can be developed. We discuss MINER's requirements and design considerations and present a modular implementation that can be flexibly extended through plugins. Finally, we report on usage scenarios and directions for future work.**

I. Introduction

It is no doubt indispensable to perform monitoring and measurement activities in computer networks. To this end, the community of researchers and network operators has produced a considerable amount of measurement instruments, often called tools. Examples of such tools are ping, traceroute, tcpdump etc. In an attempt to create a taxonomy for measurement and analysis tools, project MOME reports [1] on more than 350 tools.

When a researcher is faced with the task of performing network measurements, she/he typically needs to employ multiple such tools to produce the required metrics. Many types of measurement require that these tools are executed in a coordinated fashion on various vantage points distributed in the network under study [2]. When multiple measurement runs, possibly with varying configurations, are conducted to realize a systematic study, the "manual" execution soon becomes a tedious task. The user has to handle the large diversity related to the configuration of tools and their results. In some way it must be ensured that all tools are available at the measurement points, that they initialize correctly, and that errors arising during the execution are handled gracefully. Without any form of a supporting command and control system the user needs to carry out many (error prone) execution steps and with each new measurement most of this work has to be repeated. This approach clearly fails to scale well.

To simplify the task, and for other reasons, several endeavours have been undertaken [3]–[8] to develop what is often called a *measurement infrastructure*. The common ground among them is that they provide (usually on top of existing tools) measurement facilities on distributed vantage points.

In this paper we present the *Measurement Infrastructure for Network Research* (MINER). The main objective of MINER[1] is to support users in the task of carrying out systematic measurement studies. MINER can easily integrate existing (or new) measurement tools and provides higher-level services on top of them via a programming interface. In a way similar to how simulation scenarios are executed on a simulation platform, measurement scenarios are run by MINER. A scenario can employ an arbitrary number of tools in parallel. A tool is not restricted to just active or passive measurements but it could e.g. be used to configure a testbed component as needed for the particular scenario.

The rest of the paper is structured as follows: in section II we discuss related work and identify how MINER differs from these approaches. In section III we detail the requirements that have been the starting point for our work, design decisions are described in section IV, and implementation aspects are outlined in section V. Sample applications of the framework are described in section VI. Finally, we outline the benefits of using MINER in section VII and conclude the paper with directions for future work in section VIII.

II. Related work

A number of distributed network measurement infrastructures have been developed. They were built with varying objectives and expose very diverse characteristics, e.g. in terms of large-scale applicability, security, flexibility in supporting different types of measurements, applicability to deployments with multiple administrative authorities, reuse of existing tools, programmability, and extensibility.

Several projects have successfully built such infrastructures and collected large amounts of data. ETOMIC [9] enables

This work was partially funded by the European Commission through the IST projects NETQOS and MOMENT.

[1] http://miner.salzburgresearch.at

synchronized active measurements at a high temporal resolution. At the infrastructure level, ETOMIC doesn't provide a user programming interface and is not easily extensible to new types of measurements. The DIMES project [10] created a truly large-scale system with a tight focus on traceroute and ping measurements. Skitter [11] also builds upon a large community and has a similar limited scope of measurement types. Its successor archipelago (ark) [12], [13] is desgined as a fully extensible active measurement infrastructure.

Other approaches - most notably NIMI [7], [14], Scriptroute [6], DipZoom [15] and perfSONAR [3] - head for a generic measurement infrastructure that can be extended and customized to the needs of a specific problem domain.

NIMI is designed for large-scale deployments in multiple administrative domains. Security is addressed with various mechanisms, among them restricting access to authenticated users and encrypting communication messages. Tools are integrated via wrapper scripts that conform to a uniform API, an approach that has been equally chosen for MINER.

Scriptroute takes a completely different approach to achieve goals that are similar to NIMI. The infrastructure is truly public without requiring user accounts. To enable this, a great deal of attention is paid to security issues. The basic approach is to execute measurements in a sandbox environment that imposes very tight limits on resource consumption. Unlike NIMI or MINER, existing measurement tools cannot be wrapped and plugged into vantage points; instead, they have to be (re-)implemented in a scripting language and by using the APIs provided by the sandbox.

DipZoom [15] seeks to establish a large availability of measurement points by resting upon a peer-to-peer model where each measurement client must also serve as a measurement point. DipZoom is built upon a matchmaking service that allows clients to locate measurement points providing the required facilities. According to [15], DipZoom ships with a very limited number of tools, however work is in progress on an extensibility mechanism that allows for new tools to be plugged in. The rate-limits imposed by the execution environment restrict the spectrum of tools that can be integrated. MINER shares the feature of a programmable platform with DipZoom.

The perfSONAR infrastructure [3], [16] is concerned with the monitoring of multi-domain infrastructures. It is primarily made for network operation centers (NOC) and performance enhancement and response teams (PERT). To them it provides means to observe, analyze and debug performance aspects of end-to-end connections traversing multiple domains. Among other networks, perfSONAR is currently employed in GEANT2 and Internet2 where many instances of measurement points are continuously producing data on fundamental network metrics (e.g. one-way delay, one-way loss, traceroute, interface utilization, TCP throughput). If a user requests data that is currently not available, a corresponding monitoring/measurement activity is triggered. The perfSONAR approch is based on the service oriented architecture (SOA) paradigm. All components are designed as web services and the system can be easily extended by adding new services. In terms of architecture, MINER also relies on a SOA design. However, these services are almost exclusively infrastructure internal and not exposed to the user. The selection and configuration of internal services is done implicitly in a declarative way through the scenario specification.

All infrastructures described above have some features in common with MINER: they all provide monitoring and measurement services in TCP/IP networks on the basis of existing algorithms and/or tools. The primary motivation for using one of the infrastructures is to analyze the behavior of the network under study. To achieve this a number of measurement points is operated to continuously monitor / measure key performance metrics. The results are maintained in (large) data archives. The services to collect, store and analyze the data are of great utility to get an insight into the dynamics of the network, to study performance issues, to analyze network anomalies and conduct various other studies. Although nothing prevents a user from making similar types of measurements with MINER, the above described infrastructures are much more suitable for such studies.

MINER is really designed for a different purpose. The main goal is to support users in evaluating some entity by making systematic measurement studies. A study is characterized by the need to perform multiple measurement runs with various configurations. Each measurement run requires the coordinated use of multiple tools in distributed locations. Naturally, it must be possible to make use of existing tools. The entity under study could e.g. be an end-user application, a network protocol, a QoS mechanism or some networking device. An example application of the infrastructure is shown in section VI.

III. Requirements

A major objective in the development of MINER is to establish a *programmable* measurement infrastructure that enables the combined usage of *arbitrary* measurement tools via a unified application programming interface (user API). The primary services offered by the infrastructure enable a user to:

- specify measurement scenarios;
- schedule the execution of scenarios;
- retrieve results and associated logs of executions.

It is required that the user API is tool-agnostic such that the interface to configure a tool and retrieve its results is unified among all MINER tools. This frees the user from having to know the native configuration/result format of a tool. It must be possible to provide user API implementations in various programming languages so that users can rely on their language of choice.

Another high-level requirement is the clear division between the infrastructure itself and the tools that it integrates. The motivation behind is that the integration of a tool must never require *any* modification at the infrastructure level. The process of tool integration must be made as simple as possible. In principal, there must not be any restrictions as to what a tool can do: active measurements, passive monitoring, or "just" the configuration of a testbed component. A measurement activity is specified in a so called *MINER scenario*, in short scenario. The scenario, its executions, and their results must be stored persistently to provide for traceability in the measurement workflow.

It is a major requirement that the infrastructure is constructed as a *modular* software system where the implementation of a functional component can be extended or completely replaced with a different implementation, respectively. This feature shall enable project specific customizations that do not require any modifications to the core infrastructure.

Measurement specification

A scenario must allow for the combined usage of multiple tools. The activity period of a tool within a measurement can be defined separately for each tool so that one tool can e.g. start later and finish earlier than another tool of the same scenario. The scenario specification must provide facilities to group multiple tools into a single logical unit (e.g. background load) that can be reused by *reference* in another scenario. Thus, if the user makes a modification in this unit (e.g. to fix a configuration error), all dependent scenarios are automatically up-to-date. A user must explicitly define which results a tool has to produce. It must be possible to define conditions on those results (e.g. delay $<$ 200ms). In this case the infrastructure must monitor the delay values in a timely manner and generate an alarm as soon as a condition is violated.

Infrastructure core

A scenario has to be submitted to the infrastructure core where it is verified and – if successful – stored persistently. The verification process should be strict and try to detect as many errors as possible. In particular, we require that the verification of the tool configurations is done already at scenario submission time. This is especially important as the execution of a scenario can be scheduled for a later time. In general, a scenario can be scheduled for execution an unrestricted number of times.

The success of a schedule request is subject to the conflict detection algorithms implemented in a scheduler component. This scheduler must allow for extensibility so that different algorithms can be adopted.

When a scheduled execution is due, the infrastructure core must initiate and coordinate the execution process. First, the execution request (containing the scenario) is distributed to all measurement points where the tools have to be initialized. If one of the tools fails to initialize correctly, the execution process has to be terminated gracefully. When all tools are ready they must be run according to the activity periods defined in the scenario. If errors occur during this active period, they must again be handled gracefully. If no errors occur, a tool typically generates results which are sent to the specified result store. Depending on the configuration this may happen during or after the activity period, respectively. At least the core must implement a result store, however, additional distributed result stores (e.g. close to the data source) should be supported. Users can access the result data through the user API without having to know where the data is actually stored.

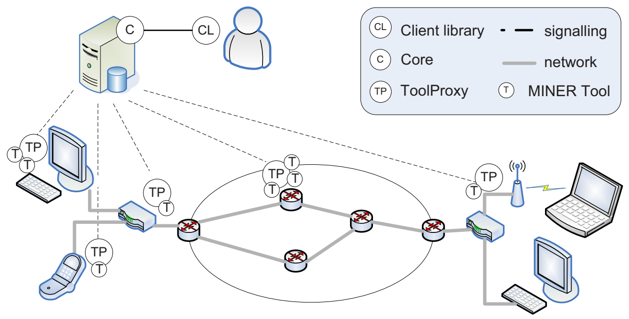

Fig. 1. MINER system overview showing the main subsystems

Measurement point

A measurement point is a device on which a tool is executed. A tool can be used multiple times within the same scenario and it may be acceptable to execute multiple scenarios on a device at the same time. We require that tool instances are dynamically created by the infrastructure upon request and removed afterwards. Otherwise a measurement point would be unnecessarily overloaded with tool instances just to be prepared for the maximum usage case.

We require a mechanism that allows for automated deployment of a tool to a remote measurement point. Additionally, hot installation of a tool into a running measurement point is required. Multiple versions of the same tool can be active at the same time. In general, tools must run as isolated components within a measurement point in the sense that there must not be any namespace of library conflicts. As an example, it must be valid to execute two tools at the same time although they both rely on the same library in different versions. It must be possible to package system libraries (shared object files or DLLs) with a tool and have them activated at runtime.

Finally we require that mobile devices (programmable cell phones, PDAs, etc.) can be used as measurement points.

IV. Design

Four subsystems have been identified in the design of MINER: the ClientLibrary, the Core, the ToolProxy and the MINER Tool. These subsystems are shown in figure 1.

The **ClientLibrary** is a software library that enables a user to utilize the services provided by the infrastructure. With this user API implementation a programmer can conveniently define scenarios, schedule executions, and retrieve their results. ClientLibrary implementations can be provided for various programming languages.

The **Core** is the server component of the infrastructure. It handles scenario submissions and execution schedule requests. It triggers the timely execution and handles all communication with the ToolProxies to which the execution request is deployed. It implements the appropriate error handling procedures in accordance with the execution model such that any failures are handled gracefully. It accepts result data coming from the ToolProxies during and/or after the execution and stores that data persistently along with the scenario specification and execution schedule. If a ToolProxy signals the violation of a user defined condition, notification mechanisms are triggered by the Core. Moreover, the Core maintains a continuously updated registry of all ToolProxies and their MINER Tools.

The **ToolProxy** acts as a mediator between the Core and the MINER Tools. When a ToolProxy is started, it registers itself at the associated Core and informs it about its measurement interfaces and available MINER Tools. The ToolProxy handles all communication with the Core on behalf of the MINER Tools. When the ToolProxy receives a scenario execution request it dynamically instantiates the required MINER Tools and follows the execution plan by invoking each MINER Tool's methods at the right time (according to the activity period). The ToolProxy provides the execution environment to the MINER Tools.

A **MINER Tool** is a component that implements a designated tool interface. There is exactly one such interface that is used for the implementation of any type of tool. In many cases, a MINER Tool is a wrapper to an existing tool but this is not required. There are no restrictions as to what a MINER Tool can do as long as it doesn't need more resources than provided by the ToolProxy's execution context (which doesn't impose any constraints in the default setup). A MINER Tool is plugged into the ToolProxy.

Modularity

In order to enable the required modularity, we tried to identify the main components that shall be extensible. In the Core, this applies at least to the functional units described in the following.

Service access The mechanism by which the Core provides access to its services shall not be hard-coded, instead it must be possible to provide multiple implementations as Core plugins.

Execution scheduling A schedule manager is invoked when the Core receives a request to schedule a scenario for execution. The request has to be rejected if it would lead to a schedule conflict. We can envision various types of schedule policies like e.g. (i) don't allow more than 1 active execution per infrastructure / per ToolProxy / per measurement interface (ii) don't allow the parallel activity of the tools X and Y (iii) don't allow more than 1 active instance of tool X per ToolProxy. We design a schedule manager that accommodates an arbitrary number of schedulers which are provided as plugins. The manager accepts a request if none of the registered schedulers rejects it.

Execution model It can be useful to have different execution models depending on the measurement scenario. In one case, it may be required that an execution is canceled if any of the MINER Tools fails during initialization or runtime. In another case the scenario shall be executed if "enough" MINER Tools don't fail (and of course assuming that the execution log contains all relevant error details).

Result processor When result data arrive at the Core they are passed to a result processor. Before storing the data persistently such a processor can perform various actions on the data, e.g. anonymize or compress it. Alternatively, there may be situations where a MINER Tool consists of a sender and a receiver part, both parts send information to the Core, and the result of interest can only be computed by the result processor from these two data sources.

Storage The result storage shall be extensible to new types of data and it shall not depend on any specific technology. The storage component is an abstraction to the physical data store. It can e.g. be implemented via the file system or some form of database system (SQL, RRD, pure XML). Furthermore the result storage should support distributed locations to store large data sets close to the source (e.g. full packet traces).

Result query An extensible result query component can be utilized to implement server side result processing, e.g. to compute statistics from a large volume of stored result data.

Notification When a condition is violated the notification module is invoked. Different forms of notification (log entry, callback to the ClientLibrary, email, command execution etc.) can be useful.

At the ToolProxy we design the following components:

MINER Tools Each MINER Tool is a separate component that is plugged into the ToolProxy.

Execution context Each MINER Tool runs in a well-defined execution context within the ToolProxy. This context shall be customizable by additional components, e.g. to implement a sandbox. The context can be further adapted by providing APIs to components like a packet capture and injection daemon [17] or a MAPI [18] interface.

Scenario specification

The scenario is the entity that fully specifies the required measurement activity, i.e. where to measure, what MINER Tools to use, how to configure them, what results to produce, etc. We design the scenario as a hierarchical composition: a scenario consists of at least one *task*, and each task consists of at least one *action*. A MINER Tool is selected within the action part. To enable the usage of multiple MINER Tools in one scenario, a two-level hierarchy scenario – action would be sufficient. The additional level in the form of a task is introduced to enable logical grouping of actions. Subsequently, a task can be reused in other scenarios by reference, i.e. without having to copy it. In the same way, an existing action can be referenced by several tasks. All units can be annotated with a description and keywords / tags.

Figure 2 shows a graphical representation of a sample scenario which is used in section VI below.

To enable multiple executions of a scenario, the scenario does not hold any execution specific data such as the start time and the duration of the execution. These parameters are specified with the schedule request.

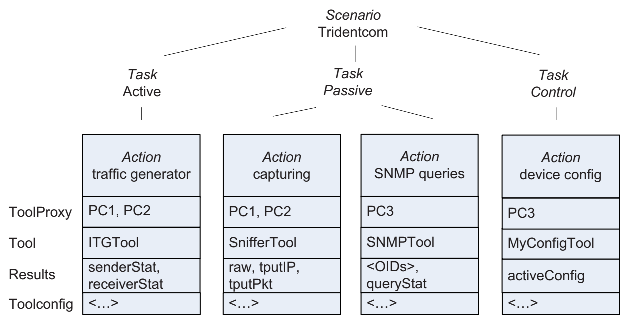

Fig. 2. A sample scenario specification

Scenario verification

To enable the strict and early verification of scenarios at submission time we choose the approach that a MINER Tool must provide two descriptions of itself. The first one is a W3C Schema[2] file that defines a template for a valid configuration. The second one is an XML file that defines all results that the MINER Tool can possibly produce. This file must validate against an XML schema provided to tool implementor. The procedure of MINER tool implementation is outlined in section V-A below.

In the process of the scenario verification we partially rely on XML technology. The Core receives the scenario as an XML document (which is usually created by the ClientLibrary). This document is validated against its XML schema leaving out the tool configurations in the first step. Then, each tool configuration (an XML fragment) is validated against its corresponding schema.

By relying on these well-established XML technologies we can make use of the whole toolchain that is available today. A schema allows for a precise definition of constraints: if e.g. a MINER Tool requires an IP address parameter, the tool developer can use a regular expression that matches only valid IP addresses. This in turn provides great utility to the person writing the MINER Tool configurations as a schema-supporting XML editor will immediately show an error if invalid data is entered.

Besides the XML based validation, the Core makes additional checks like: Do the referenced ToolProxies exist? Do the MINER Tools exist on these ToolProxies? Can the MINER Tools produce the requested results? Are the conditions imposed on results that accept conditions? If these and additional tests pass, the scenario is accepted and saved to the database.

V. Implementation

One key guideline in the implementation is to build upon existing functionality in the form of mature software components and libraries.

MINER is implemented in Java as it offers the power and flexibility of a general purpose programming language, it supports modern software engineering approaches, it is platform independent, powerful IDEs are freely available and a large number of mature libraries exists. Java is considered a "safe" language and has built-in mechanisms for fine-grained access control to system and network resources. A powerful dynamic module system for Java has been specified [19] by the OSGi alliance.

[2]http://www.w3.org/XML/Schema

The OSGi approach is a perfect match for satisfying all the modularity and extensibility needs of MINER. It enables the implementation of the components described in section IV as separate modules (*bundles* in OSGi terminology). Additionally, OSGi comes with full support for installation of bundles from remote locations. Also, bundles can be added, updated, and removed without having to restart the application. OSGi does not constrain usage on (possibly resource limited) mobile devices as it is a lightweight technology that was primarily developed for application in embedded systems.

Our current MINER implementation is based on the following solutions: an OSGi implementation (Eclipse Equinox); a relational database (MySQL) for all persistency needs; an object-relational mapper (Hibernate) that abstracts the concrete database product, manages a connection pool, and provides caching to increase performance; default Java tools to parse, validate, transform, and map (JAX-B) XML documents; software agent technology (JADE) to realize the ToolProxies; Web services (SOAP over HTTP) for communication between the ClientLibrary and the Core. Details of the implementation can be found in project report [20].

A. MINER Tool implementation

To make the infrastructure usable for network measurements, MINER Tools need to be made available. Typically a MINER Tool relies on the functionality of an existing tool but this doesn't necessarily have to be the case. The process of implementing a MINER Tool is outlined in the following. Basically, a MINER tool developer must provide for:

- an implementation providing the required measurement (or other) functionality;
- a configuration schema;
- a specification of available results;
- tool packaging.

1) Functionality: A MINER Tool is realized as a subclass of the abstract Java class *MinerTool*. This class contains an instance of an *IToolContext* interface which is the single means through which the MINER Tool can interact with the ToolProxy, e.g. to send results or to create execution log messages.

With the help of this *IToolContext* a concrete MINER Tool must implement the following 3 methods:

- *boolean init(toolConfig, results)*
- *void run()*
- *void finish()*

The *init()* method is called immediately after the MINER Tool is instantiated. It is used for initialization work and has to make sure that all required resources are available. The method is passed the configuration for the MINER Tool (as an XML document) and a list of the results that are requested. An initialization failure must be signaled by returning false. This information is then spread to all the ToolProxies involved in the scenario and the execution is canceled (unless an alternative execution model is employed).

The *run()* method is called by the ToolProxy when the activity period of the action starts. Inside this method, the "real" work is normally done. If the MINER Tool wraps an existing measurement tool, then this tool executable is typically called now. When results become available, the MINER Tool passes them to its ToolProxy.

The *finish()* method is called after the activity period of the action has ended. It is used to release resources and perform cleanup work.

2) Configuration schema: The developer must provide a W3C XML Schema that defines a template for a valid configuration of this MINER Tool. It is up to the developer in what detail the schema is elaborated. The configuration can be accepted as a simple unspecified String (like a command line). Alternatively, the schema can define (a hierarchy of) elements for each configuration item and specify constraints (e.g. via allowed ranges or regular expressions). With an increasing detail in the schema the scenario verification performed by the Core becomes more powerful.

3) Available results: The developer must provide an XML file that describes all results that this MINER Tool can possible produce. This file must validate against the corresponding schema that is provided to the developer. A result is described by its name and a data type; optionally a specific result processor, unit, description, and configuration parameters can be defined. An example of a configurable result is a throughput that requires an averaging interval. Finally, a result can be marked such that it must not be used in a condition.

4) Packaging: The developer must finally package the compiled Java classes and the 2 XML files into an OSGi bundle which is a normal Java archive (.jar file) enriched with additional meta information. Tool support is available for creating the archive.

B. Implementation status

In middle of 2008 the implementation reached a stable state which we considered as version 1. Most features are implemented as described above. The main deviation is that the Core, while clearly having a modular design, is not yet built on OSGi technology. Therefore, the provisioning of Core modules is not optimal, yet. On the ToolProxy level, more work should be spent on optimizing the run-time behavior with respect to performance and robustness in adverse conditions. As an example, the user may specify that results have to be sent back to the Core only after the execution has finished. Until then, results are buffered at the ToolProxy. There is currently no mechanism to swap out results to a disk if the ToolProxy is running out of heap space.

A lot of effort was put into developing more than 300 unit tests that are executed regularly. We currently have a Java implementation of the ClientLibrary. Although programmability is one of the key features, there are situations where it would be handy to have a graphical user interface (GUI) to the infrastructure. As an example, this applies to browsing executions, their logs and results. To this end we have created a modern web-based proof-of-concept prototype which works well but is in a very early stage. Most of the implementation worked was focused on the infrastructure level. Therefore, only few MINER Tools have been implemented so far:

- SNMPTool to gather information from distributed hosts via SNMP.
- IPPMTool to actively measure one-way delay, packet loss and jitter according to the IPPM specifications[21]–[23].
- SnifferTool to capture network traffic with *libpcap*.
- NTPTool to monitor the state of time synchronization of a host via NTP.
- ITGTool for active traffic generation with the flexible D-ITG generator from the University of Naples [24].
- BARTTool to measure the available bandwidth with the BART tool [25].
- NETEMTool to configure the linux kernel-level network emulator netem [26].
- ETPTool to query performance statistics from the Enhanced Transport Protocol (ETP) [27].

VI. Application

To practically illustrate how MINER can be applied we first show a sample use case and subsequently list how MINER has already been employed in some other real-world cases.

A. Sample scenario

In this section we discuss an exemplary application of the framework in an imaginary use case. In this discussion we disregard whether the setup in terms of trial topology, number of traffic generators, tool implementation etc. is reasonable. The goal is merely to illustrate how MINER can be conceptually applied to approach a given problem.

Let's assume that we need to evaluate several aspects of a networking device. A number of different device configurations shall be tested under various load levels and traffic patterns. Results of interest could be the packet forwarding performance, the correctness of packet filters, the accuracy of the traffic shaping implementation, the response time for SNMP queries, etc.

As a first step, the appropriate MINER Tools have to be provided. Several tools may already be available, like the ITGTool, SnifferTool, and SNMPTool described above. Missing MINER tools have to be implemented. In the given case we create a tool MyConfigTool which configures the device under test.

The configuration schema of MyConfigTool contains the following elements:

- the IP address of the device
- the port number of the configuration service
- a public key used for authentication
- the device configuration

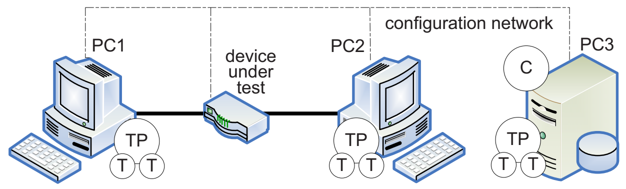

Fig. 3. Testbed used for the example application

The only result that MyConfigTool can possibly produce is called *activeConfig* and is of type *String*. It reports the running configuration of the device after it has been configured.

The implementation of MyConfigTool is straight forward. For this tool it is useful to implement the functionality in the *init()* method. If the device configuration fails, *init()* returns false and the whole execution is canceled. Alternatively, the device configuration could be implemented in the *run()* method. In that case the user would have to be aware that the action containing this tool must be scheduled for earlier execution than the other actions so that the device is configured as expected before the other MINER Tools become active. The first variant is clearly preferable in this case.

In the *init()* method a connection to the device is established, the device configuration is reset and the new configuration is applied. If any errors occur, the error messages are logged and *init()* returns false. The log messages are saved to the execution log which can be retrieved by the user. If the user requested the result *activeConfig*, then the configuration is queried from the device and sent back as a result (via the *addResult()* method provided by the *IToolContext*, see section V-A1). The *run()* and *finish()* methods of MyConfigTool can be empty.

A minimalistic testbed is shown in figure 3. Each node is connected to a second network shown as dotted lines. It is used for signalling traffic. The MINER Core is located on host PC3, ToolProxies are running on hosts PC1, PC2, and PC3. Figure 2 shows an example of a generic scenario tree. The grouping of actions into tasks is an arbitrary choice. Some results of interest are delivered directly by the MINER Tools (e.g. statistics of SNMP query time, packet throughput) while others are computed offline on the basis of the raw capture files (e.g. accuracy of shaping).

It is assumed that all combinations of traffic loads (low/medium/high), traffic patterns (small/medium/large packets), two different sets of SNMP queries (Q1/Q2) and four different device configurations (A/B/C/D) shall be tested. This makes up for 72 different scenarios. Note that for any scenario, several actions can be reused by reference. For example, the action containing MyConfigTool with device configuration A is used in 18 scenarios.

For the given study it may be useful to create an application that enables a user to define the whole set of test cases via a single configuration file. From this file all MINER scenarios are created, submitted, and scheduled for successive execution.

When the whole study has to be repeated at a later time (e.g. to analyse a new firmware release), the set of scenarios is simply rescheduled for another execution.

Figure 4 shows a code excerpt of an application that is built on top of the Java ClientLibrary. The application defines a MINER scenario, submits it to the Core and schedules it for immediate execution. Only the definition of the action containing MyConfigTool is shown in the code, the definition of the other tasks and actions is left out as indicated in line 18. The classes *Scenario*, *Task*, *Action*, and *ResultRequest* are provided by the ClientLibrary. By using them the programmer can easily create the scenario as depicted in figure 2. The tool configuration *mytool.xml* is shown in the lower part of the figure. It is defined in an external file by using a W3C Schema aware editor instead of using an error prone inline approach. It is obvious how different scenarios can be easily created by loading different configuration files. Once the scenario specification is complete, it is submitted to the Core (line 21). If the submission succeeds, the scenario can be scheduled for execution. The call in line 23 triggers the immediate execution.

B. Current MINER usage

So far, MINER has been primarily used in the European ICT project NETQOS [28] which is concerned with autonomous policy-based networking. There, an application written on top of the ClientLibrary automatically maps operational policies which are related to measurable metrics into MINER scenarios. Performance criteria, if expressed in the conditions parts of the policy, are mapped into conditions of the MINER scenario. A scenario is submitted to the Core and executed immediately. If a condition is violated then an alarm is emitted by the Core (using a project specific notification plugin) and the policy adaptation module takes care of this situation. Typically, an adaptation module queries the Core for detailed results (e.g. the history of delay values over some time interval) and uses this data as input to its adaptation model.

A German research project uses MINER since summer 2008 to measure the network performance under novel cooperative algorithms in next generation cellular networks [29].

Researchers from LAAS/CNRS have used MINER to setup an infrastructure that performs automated regression tests of the Enhanced Transport Protocol (ETP) [27].

In an industrial use case, MINER is employed at a local Austrian ISP to continuously measure the perceptual VoIP quality. If measurements indicate problems in (some part of) the network, detailed monitoring activities and measurements are triggered in order to provide the data necessary for analyzing the problem.

VII. Benefits gained from MINER usage

MINER is very well usable for cooperative research projects where a group of researchers needs to carry out systematic measurement studies in a project testbed. From our own experience and the feedback of early users we feel that MINER provides a solid set of features that – in its combined form – is not readily available elsewhere. The infrastructure is fully programmable by utilizing a single API which can be

MyApp.java

```
Scenario s = new Scenario("Tridentcom");
Task t = s.addTask(new Task("Control"));

Action a = t.addAction(new Action("config"));
a.addProxy("PC3");
a.setTool("com.example.MyConfigTool");
a.setToolconfig(loadDoc("myconfig.xml");
a.addResultRequest(
    new ResultRequest<String>("activeConfig"));

// create other tasks and actions
...

// submit the scenario to the Core
s.submit();
// schedule the scenario for immediate execution
s.run();
```

myconfig.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<MyConfigTool>
    <IP>10.0.1.1</IP>
    <Port>8080</Port>
    <AuthKey>ssh-rsa AAAAB...</AuthKey>
    <DeviceConfig>commands...</DeviceConfig>
</MyConfigTool>
```

Fig. 4. Code excerpt of a Java application and a tool configuration file

made available in various programming languages. MINER is a very modular software system that allows for far-reaching extensibility using state-of-the-art plugin technology. Existing measurement tools can easily be integrated into measurement points which can run on any Java platform including mobile devices (of course, this doesn't mean that any existing tool runs on any device).

MINER provides a fully documented and thus traceable measurement workflow due to the linked storage of scenario specifications, their executions and the individual results (and logs) of each execution. The result is that a whole measurement study can be repeated at almost no effort. Parts of a scenario can be reused and this allows for efficient management of scenarios and modification of erroneous configurations. Scenarios are strictly validated as soon as they are submitted to the infrastructure. This is very valuable given that a large number of scenarios can be scheduled for execution at a later time. MINER provides a foundation on which measurement applications can be built. As an example, it enables the development of reactive measurements similar to what has been recently proposed in [30] (with the main difference being that the REM instance would not be conducted locally – something that *can* be enabled by plugins).

Practically speaking, MINER can save its users a lot of manual (and repeated) work. Trial executions can be split among different people, for instance a group of people can prepare the various scenario specifications, one person runs the scenario executions, and yet other people analyze the results.

Finally, with MINER it is very straight forward to share a private or project internal testbed. The administrator can install MINER in the (private) testbed. Other colleagues of the project only need access to the web service that is exposed by the Core component. As the end user communication only takes place between the ClientLibrary and the web server, there is no need to enable direct IP access from the end users to the measurement points where the ToolProxies are located.

VIII. Conclusions and future work

In this paper a Measurement Infrastructure for Network Research (MINER) is presented. The main objective of this work is to support users in the task of carrying out systematic measurement studies. The infrastructure integrates existing tools and provides higher-level services on top of them. The user can specify measurement activities that comprise an arbitrary number of tools in a coordinated fashion. Such measurement scenarios are submitted to the infrastructure and can be scheduled for execution at a later time. As soon as an execution has started, results can be retrieved. All services of the infrastructure are available via a tool-agnostic programming interface.

The paper presents the requirements that have driven the work on MINER. Then, the key design aspects are presented and the current implementation of the system is outlined. It is shown how MINER can be applied to a specific problem. The current usage in real-world scenarios is reported and it is summarized what benefits the user can gain from using MINER.

The MINER implementation has reached a first stable version. From the current state the options for future work are manifold. On the infrastructure level, the modularization of the infrastructure Core using OSGi technology is yet to be finished. The ability to extend / adjust the functionality of Core modules to project specific requirements will greatly enhance the applicability of the framework. On the ToolProxy level, more work should be spent on optimizing the run-time behavior with respect to performance and robustness in adverse conditions. Some activities related to carrying out systematic measurement studies would clearly benefit from a graphical user interface.

The applicability of MINER to existing problems is directly related to the availability of MINER Tools. Therefore, it is clearly needed to wrap up many more existing measurement tools as MINER Tools. Obviously, this process can be sped up and optimized if the usage of the infrastructure becomes more widespread and a user community emerges. To this end we have setup a publicly accessible MINER playground. It allows interested people to define scenarios on their desktop and execute them in a small demo testbed hosted by us.

References

[1] C. Schmoll, J. Quittek, A. Bulanza, S. Zander, M. Kundt, E. Boschi, and J. Sliwinski, "State of interoperability," EU IST Projects MOME, Deliverable D11, 2004. [Online]. Available: www.ist-mome.org/deliverables/D11.pdf

[2] V. Paxson, "Strategies for sound internet measurement," in *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*. New York, NY, USA: ACM, 2004, pp. 263–271.

[3] A. Hanemann, J. W. Boote, E. L. Boyd, J. Durand, L. Kudarimoti, R. Lapacz, D. M. Swany, J. Zurawski, and S. Trocha, "PerfSONAR: A service oriented architecture for multidomain network monitoring," in *Proceedings of the Third International Conference on Service Oriented Computing*, Amsterdam, The Netherlands, 2005.

[4] S. Kalidindi and M. Zekauskas, "Surveyor: An Infrastructure for Internet Performance Measurements," in *Proceedings of INET'99*, San Jose, CA, USA, 1999.

[5] B. Krishnamurthy, H. Madhyasta, and O. Spatscheck, "ATMEN: a triggered network measurement infrastructure," in *Proceedings of International World Wide Web Conference (WWW)*, 2005. [Online]. Available: citeseer.ist.psu.edu/krishnamurthy05atmen.html

[6] N. Spring, D. Wetherall, and T. Anderson, "Scriptroute: A public internet measurement facility," in *4th USENIX Symposium on Internet Technologies and Systems*, 2002. [Online]. Available: citeseer.ist.psu.edu/spring02scriptroute.html

[7] V. Paxson, A. Adams, and M. Mathis, "Experiences with NIMI," in *Proceedings of Passive and Active Measurement Conference*, 2000. [Online]. Available: citeseer.ist.psu.edu/article/paxson00experiences.html

[8] EU IST Project EuQoS, "End-to-end quality of service support over heterogeneous networks," http://www.euqos.eu/, 2007.

[9] D. Morato, E. Magana, M. Izal, J. Aracil, F. Naranjo, F. Astiz, U. Alonso, I. Csabai, P. Haga, G. Simon, J. Steger, and G. Vattay, "The European traffic observatory measurement infrastructure (ETOMIC): A testbed for universal active and passive measurements," in *Proceedings of TRIDENTCOM '05*. Washington, DC, USA: IEEE Computer Society, 2005.

[10] Y. Shavitt and E. Shir, "DIMES: Let the internet measure itself," *SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 5, 2005.

[11] B. Huffak, D. Plummer, D. Moore, and k. Claffy, "Topology discovery by active probing," in *SAINT-W '02: Proceedings of the 2002 Symposium on Applications and the Internet (SAINT) Workshops*. Washington, DC, USA: IEEE Computer Society, 2002.

[12] CAIDA, "Archipelago measurement infrastructure (ark)," http://www.caida.org/projects/ark/, 2007.

[13] K. Claffy, Y. Hyun, K. Keys, M. Fomenkov, and D. Krioukov, "Internet Mapping: from Art to Science," in *Cybersecurity Applications and Technologies Conference for Homeland Security (CATCH)*, Washington, USA, March 2009.

[14] V. Paxson, J. Mahdavi, A. Adams, and M. Mathis, "An architecture for large-scale internet measurement," *IEEE Communications*, vol. 36, no. 8, August 1998.

[15] Z. Wen, S. Triukose, and M. Rabinovich, "Facilitating focused internet measurements," *SIGMETRICS Perform. Eval. Rev.*, vol. 35, no. 1, pp. 49–60, 2007.

[16] J. W. Boote, E. L. Boyd, J. Durand, L. Hanemann, A.and Kudarimoti, R. Lapacz, N. Simar, and S. Trocha, "Towards multidomain monitoring for the european research networks," in *Selected Papers from the TERENA Networking Conference*, 2005.

[17] J. M. Gonzalez and V. Paxson, "pktd: A packet capture and injection daemon," in *Proceedings of Passive and Active Measurement Conference*, 2003. [Online]. Available: citeseer.ist.psu.edu/574091.html

[18] M. Polychronakis, K. G. Anagnostakis, E. P. Markatos, and A. Oslebo, "Design of an application programming interface for IP network monitoring," in *Proceedings of the 9th IFIP/IEEE Network Operations and Management Symposium (NOMS04)*, 2004.

[19] The OSGi Alliance, "OSGi service platform core specification, Release 4," http://www.osgi.org/Specifications, 2007.

[20] C. Chassot and al, "Implementation of policy-oriented measurement, learning and context identification," The NETQOS consortium, Tech. Rep., 2007.

[21] G. Almes, S. Kalidindi, and M. Zekauskas, "RFC 2679: A One-way Delay Metric for IPPM," Sep. 1999.

[22] C. Demichelis and P. Chimento, "RFC 3393: IP Packet Delay Variation Metric for IP Performance Metrics (IPPM)," Nov. 2002.

[23] G. Almes, S. Kalidindi, and M. Zekauskas, "RFC 2680: A One-way Packet Loss Metric for IPPM," Sep. 1999.

[24] S. Avallone, S. Guadagno, D. Emma, A. Pescape, and G. Ventre, "D-ITG distributed internet traffic generator," in *QEST '04: Proceedings of the The Quantitative Evaluation of Systems, First International Conference*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 316–317.

[25] S. Ekelin, M. Nilsson, E. Hartikainen, A. Johnsson, J.-E. Mngs, B. Melander, and M. Bjrkman, "Real-time measurement of end-to-end available bandwidth using kalman filtering," in *IEEE NOMS 2006*, 2006.

[26] S. Hemminger, "Network emulation with NetEm," in *Linux Conf Au*, April 2005. [Online]. Available: http://linux-net.osdl.org/index.php/Netem

[27] M.Diaz, E.Exposito, and P.Snac, "FPTP: the XQoS-aware and fully programmable transport protocol," in *11th IEEE International Conference on Networks ICON'2003*, Sydney, Australia, September 2003.

[28] S. Avallone, P. D. Gennaro, I. Miloucheva, M. Roth, and S. Rao, "NETQOS: Policy based management of heterogeneous networks for guaranteed QoS," in *Proceedings of QoS 2006 Workshop*, Coimbra, Portugal, 2006.

[29] G. Fettweis, "EASY-C project," http://www.easy-c.de.

[30] M. Allman and V. Paxson, "A reactive measurement framework," in *Proceedings of Passive and Active Measurement Conference*, 2008.