Journal of Network and Computer Applications 35 (2012) 1402–1414

Contents lists available at SciVerse ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

Unified architecture for network measurement: The case of available bandwidth ☆

Giuseppe Aceto [a], Alessio Botta [a], Antonio Pescapé [a,*], Maurizio D'Arienzo [b]

[a] University of Napoli Federico II, Dipartimento di Informatica e Sistemistica, Via Claudio 21, 80125, Italy
[b] Second University of Napoli, Dipartimento di Studi Europei e Mediterranei, via del Setificio 15, 81100, Italy

ARTICLE INFO

Article history:
Received 31 January 2011
Received in revised form
21 August 2011
Accepted 10 October 2011
Available online 20 October 2011

Keywords:
Distributed measurement platform
Active measurement
Available bandwidth
Network monitoring

ABSTRACT

In the field of network monitoring and measurement, the *efficiency* and *accuracy* of the adopted tools is strongly dependent on (i) structural and dynamic characteristics of the network scenario under measurement and (ii) on manual fine tuning of the involved parameters. This is, for example, the case of the *end-to-end available bandwidth estimation*, in which the constraints of the measurement stage vary according to the use of the final results. In this work, we present UANM (unified architecture for network measurement), a novel measurement infrastructure for the automatic management of measurement stages, tailored to the end-to-end available bandwidth estimation tools. We describe in detail its architecture, illustrating the features we introduced to mitigate the problems affecting available bandwidth estimation in heterogeneous scenarios. To provide evidences of UANM benefits, we present an experimental validation in three selected scenarios deployed over a real network testbed to (i) quantify the overhead introduced by the use of UANM, (ii) show how UANM is able to alleviate the interferences among concurrent measurements, and (iii) illustrate how UANM is capable to provide more accurate results thanks to the knowledge of the network environment. Finally, for the first time in literature, we provide a "fair comparison" of eight available bandwidth estimations tools in terms of accuracy, probing time, and intrusiveness.

1. Introduction

Measuring the performance of current networks is complicated by several factors, such as the increased number of devices connected and their heterogeneity, the presence of overlays, the use of multipath routing, etc. The availability of new measurement infrastructures to support complex measurement experiments has become of paramount importance, mainly because single tools that have obtained good performance in the past can be now unreliable or may not converge. This is the case of end-to-end available bandwidth (AB) estimation tools. Almost every year, the list of these tools is enriched with new proposals: as a non-exhaustive list of active AB estimation tools we can cite IGI/PTR (Hu and Steenkiste, 2003), Pathload (Jain and Dovrolis, 2003), Pathchirp (Ribeiro et al., 2003), Abing (Navratil and Cottrell, 2003), Spruce (Strauss et al., 2003), Assolo (Goldoni et al., 2009), Diettopp (Johnsson et al., 2004), Wbest (Li et al., 2008), YAZ (Sommers et al., 2006), and Traceband (Guerrero and Labrador, 2010). This witnesses the interest of research community towards the measurement of this metric, used by different kinds of applications (P2P, overlay networks, etc.) and for different purposes (billing, QoS, etc.).

The integration of available bandwidth measurement tools in networked systems often relies on the implementation of a new estimation method or on the inclusion of pre-existing code. Unfortunately, despite the interesting proposals coming from both universities and industries, the efficiency and accuracy of such tools is still strongly dependent on structural and dynamic characteristics of the network under measure as well as on manual fine tuning of the parameters related to the adopted measurement method. The measure of available bandwidth is even a more complicated task when the size of the network increases. Link heterogeneity, overlay networks, and multipath routing make the measurement process complex, and single tools can return with unreliable results or cannot converge. Moreover, the measurement constraints (accuracy, time duration, intrusiveness, etc.) vary according to the purpose of the measure, and a tool working fine in certain conditions is often highly unreliable or unsuitable in other ones. In these situations the measurement process requires a detailed design step before its execution, with particular attention to the interference effects when active AB estimations are performed simultaneously (Croce et al., 2010). As a consequence, tools that share a measurement end point can obtain inaccurate measures or can lead to the complete invalidation of the measurement.

☆ Preliminary results within the same framework of this work have been recently presented in Aceto et al. (2010).

* Corresponding author. Tel.: +39 0817683856; fax: +39 0817683816.
E-mail addresses: giuseppe.aceto@unina.it (G. Aceto), a.botta@unina.it (A. Botta), pescape@unina.it (A. Pescapé), maudarie@unina.it (M. D'Arienzo).

1084-8045/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.jnca.2011.10.010

Due to all these reasons, an expert operator is currently needed in order to properly measure the AB in heterogeneous networks. To this end, we designed and implemented UANM (unified architecture for network measurement), a novel measurement infrastructure to automatically manage end-to-end available bandwidth estimation tools. UANM is based on measurement servers capable to provide complex and mutually exclusive end-to-end available bandwidth estimations to any interested applications. It can work with dynamically loadable measurement plugins or with standalone measurement tools; in both cases it can also interact with third-party measurement tools. A decision engine selects and configures the most suitable method according to the measurement environment and the intended use of the measure, e.g. allowing for the mitigation of interference with concurrent network utilization.

UANM is used as a classic client–server application, and can be deployed as a peer-to-peer network of measurement services. The server (in the following, "UANM-daemon" or simply "daemon") acts as a multi-method estimation tool with automatic tuning of parameters, striving to perform the best achievable tradeoff among low-intrusiveness, accuracy, and low latency according to the requests of the clients. The client is available to all the interested applications in form of an easy-to-embed API, and can send requests to the measurement server through the network using a specific protocol. Measurement requests can involve normal hosts (single side methods), hosts equipped with a UANM-daemon instance, or hosts equipped with third-party measurement tools (with at least one edge supporting UANM).

This framework can manage different implementations of the measurement techniques (i) as dynamically loadable modules or (ii) as standalone binaries. This eases the inclusion of already available third-party tools as well as tools not yet implemented, thus widening the availability and the variety of estimation methods supported. This also makes UANM suitable for fair comparisons among different existing techniques in real world scenarios, and can even serve as development and testing framework for the implementation of new estimation methods.

In this paper we describe the architecture of UANM in detail and we present an experimental analysis aimed at illustrating the benefits achievable. UANM is geared towards wide adoption among developers of network applications and researchers in the field of network measurement, and a prototype is released under GPL license and with a LGPL API for the development of measurement plugins. We believe that UANM allows to overcome the main limitations of current tools, providing accurate AB estimation in heterogeneous environments.

2. Background

Consider a network path of N store-and-forward (with FIFO policy) links that connect a *sender* to a *receiver*. The *end-to-end capacity* of the network path is defined as

$$C \equiv \min_{i=1,\ldots,N} C_i$$

being C_i the *capacity* of the i-th link (i.e. the transmission rate at data-link layer). The capacity of a network path is then equal to that of the link with the minimum capacity (called *narrow link* of the path); the end-to-end capacity is the maximum rate attainable on the path when there is no other traffic on it.

For each time instant, the i-th link is either inactive or transmitting at its full capacity, so the average utilization in the time interval $(t-\tau,t)$ is

$$\overline{u_i}(t-\tau,t) \equiv \frac{1}{\tau}\int_{t-\tau}^{t} u_i(x)\,dx$$

and τ is the *averaging timescale*.

The *available bandwidth* in the time interval $(t-\tau,t)$ for the i-th link of the path is

$$A_i(t-\tau,t) \equiv \frac{1}{\tau}\int_{t-\tau}^{t} C_i\cdot(1-u_i(x))\,dx = C_i\cdot(1-\overline{u_i}(t-\tau,t))$$

In other words the available bandwidth of a link is the average of the unused capacity during the considered time interval.

In analogy with the path capacity, the *end-to-end available bandwidth* of the considered path is defined as

$$A(t-\tau,t) \equiv \min_{i=1,\ldots,N} A_i(t-\tau,t)$$

and the link with the least available bandwidth (which limits the whole path) is named *tight link*.

The underlying assumptions are that during the measurement time interval (i) the path is fixed and unique (i.e. it is not subject to routing changes or multipath forwarding), and (ii) the capacity of each link is constant. While both assumptions have been validated in wired paths for time spanning in the order of multiple days (Paxson, 1997), the same is not always valid for wireless and mixed wireless-cum-wired scenarios (Lakshminarayanan et al., 2004).

3. Related work

Many works have presented, analyzed, and compared the available bandwidth estimation tools. For example, a novel technique and the related tool (named traceband) are presented in Guerrero and Labrador (2010). The performance of the tool is compared with that of Spruce and Pathload in a controlled environment and on a real network using different traffic patterns, comprising self similar traffic. Traceband uses a moving average to cope with varying channel conditions in wireless networks. Comparative results show that the tool achieves higher performance than the others in terms of convergence time and intrusiveness, and the same accuracy of Pathload in all the tested conditions. On the other hand, Botta et al. (2005) shows that the composition of different techniques, i.e. capacity estimation joined with available bandwidth estimation, provides an improvement of 15% in the accuracy.

As for the comparisons, in Goldoni and Schivi (2010) the authors compare several well known tools on a real testbed with 100 Mbps links, and with both constant bit rate (CBR) and Poissonian cross-traffic. The authors evaluate the accuracy, the intrusiveness, and the convergence time of the tools. Reported results show that the highest accuracy is provided by Pathload and YAZ, regardless of the kind of cross-traffic. The intrusiveness and the convergence time of Pathload and YAZ are however significant, especially during the tests on an emulated wide area network. Among the remaining tools, Pathchirp, IGI/PTR, and Diettopp achieve average performance in all the situations, while Assolo exhibits high accuracy, low intrusiveness, and short convergence time. As claimed by the authors, the tool calibration was out of the scope of their contribution. However, in these conditions, the tools may not produce the best results. Angrisani et al. (2006) evaluate the performance of Pathload, Pathchirp, IGI/PTR, and Spruce under different kinds of network traffic – i.e. on/off, bursty, multiple TCP streams – reporting that Pathchirp and IGI/PTR achieve the lowest standard deviation. The works of Shriram et al. (2005) and Murray et al. (2005) present a performance evaluation of different tools on a very high speed network.

Paxson (2004) claims that a better design stage of measurement experiments is of great importance to avoid frequent mistakes, mainly due to the imperfections of tools. Indeed, a perfect measurement tool does not exist, and every tool should be released with an indication of its accuracy, i.e. the deviation from

the true value under different conditions. This does not necessarily mean that a tool is better than another, but rather that a calibration is needed to detect and correct possible errors. The works (Ali et al., 2006; Strauss et al., 2003) test Pathload, Pathchirp, IGI/PTR, and Spruce on real testbeds and report several pitfalls in which they can end. The existence of biased results due to current implementation technology is also shown in Urvoy-Keller et al. (2008) and Shriram and Kaur (2007); the latter also proposes a simulated environment for a fair and unbiased comparison of tools.

Although it is not specifically designed for the measurement of the available bandwidth, NetQuest (Song and Zhang, 2007) is one of the first examples of an architecture that takes into the right account all the variables related to the measurement on a large scale network. It operates in two steps: it first tailors the experiments to the environment, and then builds a global view of the network. The design step relies on Bayesian techniques and on a subset of active measurements to limit network starvation. In the second step, inference algorithms are applied on the available subset of real data to compute a global map of the network status. In UANM, we outline the same design requirements as follows: (i) the use of different techniques on different parts of the network in order to achieve the best result; (ii) the augmented accuracy obtained with additional measurements given existing information; (iii) the support of multiple users who may be interested in different areas of the network. The accuracy of NetQuest is strictly dependent on the amount of data collected. Differently from UANM, NetQuest is not mainly interested in the achievement of the best performance from the measurement of the available bandwidth, since it is more oriented to a wider knowledge of the general network status. Similar to our approach, Sommers et al. (2006) do not propose a new method to measure the available bandwidth, but they propose an architecture called YAZ, whose main goal is to calibrate the existing tools in order to obtain the best results from the measurements. In contrast with UANM, YAZ does not provide support for concurrent measurement experiments, it does not consider the status of the network under study, and it does not support third-party tools.

The analysis of the state of the art reveals that an architecture able to select and calibrate the best tool for the specific measurement and to provide users or applications with a result and its error range is still needed. Such an architecture would also be a good environment for the fair comparison of available bandwidth estimation tools. Although many techniques are already available, they are still not conveyed on a common platform. UANM intends to fulfill this gap thanks to a new paradigm that is described in the following sections.

4. UANM architecture

UANM is a distributed platform whose main features are as follows:

- the support for different measurement techniques and tools in a fair environment;
- a full compliance and open interaction with existing tools;
- a mutual exclusion of concurrent measurements;
- an automatic selection and calibration of the most suitable tool.

It is made up of four different components: daemons, clients, measurement plugins, and third-party probes. The main component is the UANM-daemon that orchestrates the measurement plugins and interacts with the other UANM-daemons and third-party probes in order to fulfill the measurement requests issued by the clients. Each daemon communicates with the other daemons and with the clients by means of a dedicated control protocol (*uanmProtocol*), and also with third-party probes using their specific control protocol. The interaction between clients and daemons happens in the classic client–server fashion, while daemons are arranged on a peer-to-peer basis. All the components are written in C language and run on Unix operating systems, currently supporting Linux platforms.

4.1. UANM workflow

The UANM software platform is designed to provide "smart" available bandwidth estimations between a daemon and a remote host to applications demanding for them. The simplest use case is a client application running on the same host of a daemon that requires an available bandwidth estimation between the local host and a remote host. The client application needs to know that the local host is equipped with a daemon, the port on which it is listening, besides the remote address and the direction of the measure.[1]

When a measurement request is received, the daemon executes a feasibility check, aimed at verifying if it can actually perform the measurement and which tool is best fitted to the current *context*, and how to configure it for such context. The *context* is a model that comprises both a description of the path status and a description of measurement constraints (i.e. an information about the measurement, optionally specified by the client). The path status refers to the characteristics that may affect the measurement process, which can be both structural (e.g. presence of wireless hops or of broadband access links along the path, presence of UANM instances or known third-party estimation tools on the other edge, etc.) and behavioral (e.g. already congested path, rapidly changing routes, etc.). This information is provided by a module called *knowledge base* that will be soon described. The measurement constraints are set by the client depending on the purpose of the measure. As an example, an application may request a group of quick-and-dirty estimations to choose in a set of eligible communication partners (e.g. for server selection), or it could request a non-intrusive sampling (e.g. for network monitoring). The server offers a set of measurement choices, namely *measurement profiles*, each aimed at maximizing some characteristics, trading off the others. If no constraints are specified, the measure is set up to reach a trade-off between accuracy and low-intrusiveness, in compliance with the context. An application can also specify detailed constraints such as the average timescale, the standard deviation, the number of estimation tries, the total probe load, or even the exact measurement technique. More information about measurement profiles and constraints are provided in Section 4.5.

Once the feasibility check is completed and the measurement parameters are configured, the daemon sets up the communication with the other edge of the path under evaluation and schedules the measure. The *scheduler* module processes the measurements according to the policies. The policy currently implemented is a first come first served (FCFS) queue, but measurement constraints and information about convergence time of the selected technique allow for more complex scheduling criteria. Thanks to the *scheduler*, it is possible to avoid interference caused by concurrent measurements, as explained in Section 4.4.

4.2. UANM-daemon

The daemon component is executed on one or both edges of the path under test and waits for the measurement requests

[1] This information is provided by the client API, by means of a configuration file and a runtime check for daemon reachability.

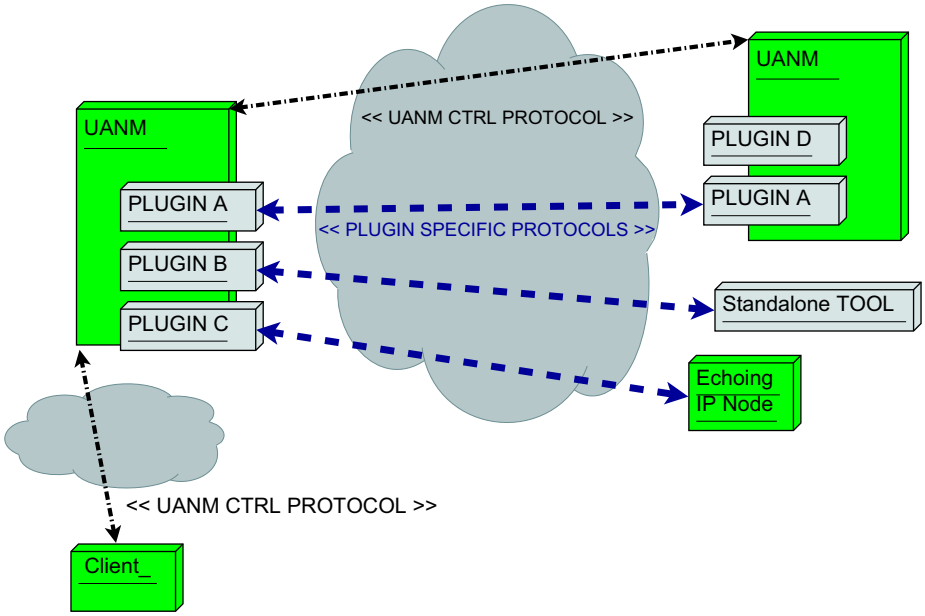

Fig. 1. Deployment diagram.

issued by the clients. It needs little or no interaction with the user since it is possibly started by boot scripts and stays up all the time. To issue a measure request, an application has to connect to the daemon using the *uanmProtocol*, whose API is provided as a C library called *libuamn*. The daemon implements multiple measurement tools in the form of dynamically loadable modules, called "measurement plugins" or, in short, "plugins". The daemon can also interact with the standalone version of third-party measurement tools, and it can use them as one of the measurement edges. Figure 1 depicts a deployment diagram that includes various possible settings.

Since it runs as a daemon, the server is detached from the terminal, and it is controlled either by a client issuing management commands through the network or locally using *posix* signals.

In order to gain flexibility and to ease future incremental enhancements, the daemon is divided in the following modules:

- *Plugin manager* is in charge of controlling the measurement plugins. At startup, and at each configuration-reload request, it scans the plugin directory to seek the available plugins, it loads each of them (updating the *knowledge base* with the related information), and monitors their status.
- *Knowledge base* collects all the information needed by the daemon to process the requests, which are the following:
  - *plugin list*: the list of measurement plugins known to the daemon, each with its own characteristics and current status;
  - *edge list*: the list of known edges (UANM-daemons, third-party probes), each with its own information (type of edge, available methods, time of latest communication, etc.);
  - *network scenario*: information affecting the planning of the measurement (first hop characteristics such as capacity, symmetrical/asymmetrical, wireless/wired, etc.);
  - *client list*: the list of the clients that contacted the daemon.

  The information in the *knowledge base* is consulted and updated by every other module when it has new information or needs; it can also be explicitly set in the configuration file (providing the daemon with external knowledge).
- *Decision engine-scheduler* checks the incoming requests for feasibility, integrates their specifications, and schedules the measurements. Upon the arrival of a new measurement request, the module refers to the *knowledge base* to check the feasibility of the measurement, which is fulfilled if the following conditions are met: (i) the measurement constraints can be satisfied in the current network context by at least one available plugin; (ii) the plugins selected in the previous step are available at both edges; (iii) the timing constraints can be satisfied.

Since the communications with the clients are performed on the network, the *scheduler* manages also the timing of the control messages. Moreover, it enforces the policy to manage concurrent incoming requests: the measure can start only when both the involved edges are not busy for a measurement. This is needed because, to the best of our knowledge and experience with existing AB estimation tools, these tools are designed to execute independent measurements, and may be unreliable when executed simultaneously from hosts that share links of their connecting paths. In Section 5 we show in our laboratory testbed the effect of the interferences occurring when concurrent measures are performed. More details on the implementation of mutual exclusion of measurements are provided in Section 4.4.

4.3. The measurement plugins

The plugins perform the actual measurements. If the measurement technique implemented by the plugin needs both the edges to be controlled, there will be two components: a `SENDER` plugin, that generates probe packets, and a `RECEIVER` plugin, that receives the probe packets (and usually performs the computation of the measure). In this case, the two components usually play different roles: one of them is executed as a daemon, waiting for orders (we identify this behavior as `WAITER`), and the other is in charge of initiating the measurement (`INITIATOR`). The daemon exploits this information to decide whether the command to start the measure must be issued to the local plugin or requested to the other edge. Some techniques work with an unmanaged edge (e.g. towards an HTTP server), so there is only one plugin

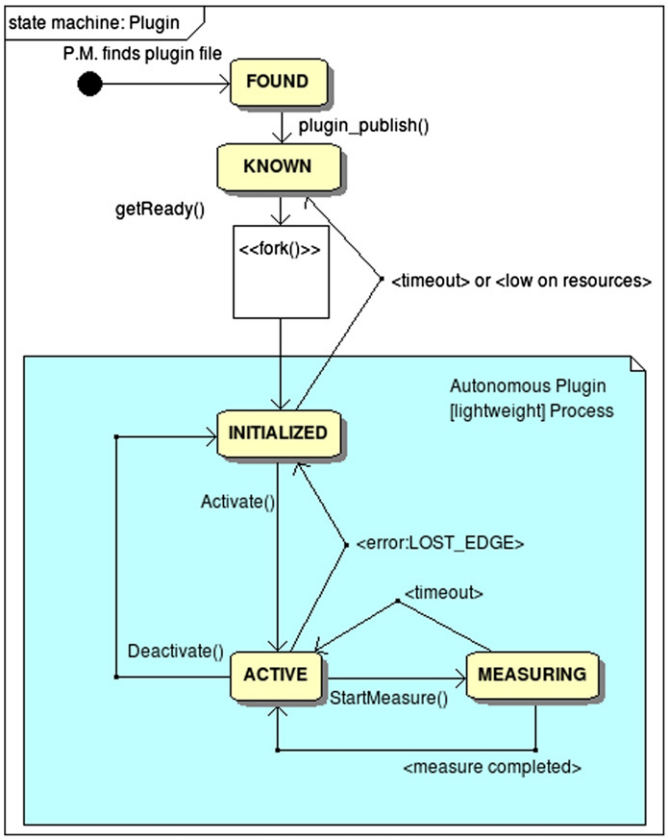

Fig. 2. Plugin state chart: state transitions are triggered by plugin method calls, or by events during measurement process.

under control: the plugins using these techniques are denoted as `SINGLESIDE`. The daemon makes use of this information to check the correct matching of available plugins on the edges of the path while assessing the feasibility of the measurement. In its life under the UANM control, a plugin evolves through different states as depicted in the state chart in Fig. 2 and explained below:

- *FOUND*: the shared object implementing the plugin has been found (the filename agrees with plugin naming conventions).
- *KNOWN*: the shared object has been correctly loaded, its interface has been published to the daemon (the *knowledge base* is then updated with the information about the plugin).
- *INITIALIZED*: the plugin is run in its own lightweight process. If it is a `WAITER`, it is bound to its default port (or a dynamically assigned one, if the default one is not available).
- *ACTIVE*: the plugin has successfully set up the communication with its counterpart and it is waiting the order to start the actual measurement. Synchronization is managed by the *scheduler* module.
- *MEASURING*: the actual measurement process is running. It will terminate returning either the measure or an error, unless interrupted by the daemon.

Each plugin implements three kinds of measurement: fully specified, constraint-based, and profile-based. They are linked to the kind of measurement requested by the clients through the API (see API measurement functions in Section 4.5). For fully specified measurements, the clients have to specify the entire parameter string to be used by the plugin. In the profile-based measurements, clients select a predefined profile for the measurements (e.g. minimally intrusive). The daemon then calculates the parameter values most suitable to this request, using the information in the *knowledge base*. For constraint-based measurements, the clients specify the constraints on the measurements (e.g. the maximum duration). Using this information and the one in the *knowledge base*, the daemon calculates the most appropriate values for the parameters.

A template is provided for the creation of new plugins, including the common interface (file `plugn.h`). Available plugins can be used to understand the changes to be made to a tool in order to put it under the UANM control.

The list of plugin methods that a plugin has to provide is (see Fig. 2) the following:

- `plugin_publish()`—loads the dynamic linking object code of the plugin, making it available to the platform; if successful, it brings the plugin to the state `KNOWN`.
- `getReady()`—executes the initialization code of the plugin, up to the port binding; if successful, it brings the plugin to the state `INITIALIZED`.
- `Activate()`—makes the `INITIATOR` set up the connection with its `WAITER`; if successful, it brings the plugin to the state `ACTIVE`.
- `startMeasure()`—makes the `SENDER` start sending the probe packets; the state of both sides of the plugin will become `MEASURING`, and at the end of the measure, `ACTIVE`.
- `deactivate()`—makes the `INITIATOR` close the control connection with its `WAITER`; it will bring the plugins to the state `INITIALIZED`.
- `abortMeasure()`—makes the `SENDER` stop sending probe packets immediately; the state of both sides of the plugin will be reverted to `ACTIVE`.
- `teardown()`—executes the shutdown code of the plugin, releasing each resource (in particular, closing the listening socket); it brings the plugin back to the state `KNOWN`.

Most of the AB estimation tools have their own control protocol used to synchronize the edges and to set up the measurement parameters. UANM manages the plugins as gray-boxes, using only high level methods such as `Activate()`, `startMeasure()`, and ignoring underlying details. This eases the transformation of third-party tools in plugins with minimal or no changes in the original code. This also leaves full compatibility between the plugin and its original standalone version. For plugins that implement the same measurement technique of a third-party tool, the plugin name and metadata report the version of the third-party tool that is compatible with this plugin. As an example, the plugin Pathchirp_2.4.1_rcv_0.1 is the version 0.1 of an implementation of the Pathchirp method, with the control protocol compatible with Pathchirp version 2.4.1. This way the original tools are fully compliant with the plugins sharing the same version number. The list of the measurement methods currently implemented as plugins is reported in Table 1.

4.4. Mutual exclusion of measurements

The main component of UANM, the daemon, is a multi-thread application, able to concurrently manage several measurement tasks; the platform is distributed, and its nodes interact, with both peer-to-peer or client-server paradigms, in a many-to-many fashion. This notwithstanding, it has been proven that timely packet generation and packet time-stamping, and thus the accuracy of active measurement tools, are affected by mutual interference when sharing a measurement node (Botta et al., 2010) or a network path (Croce et al., 2010). In order to limit this interference, a mutual exclusion mechanism has been included in the design of UANM, namely, a *scheduler* module (see Section 4.2).

Table 1
List of *measurement plugins* implemented.

Plugin name	Protocol version	Measure	Sender/ receiver	Provided profiles
Abing (Navratil and Cottrell, 2003)	2.2.0	ABW	Both	MONITOR
Assolo (Goldoni et al., 2009)	0.9	ABW	Both	DEFAULT
Diettopp (Johnsson et al., 2004)	0.2	ABW	Both	SELECTION, QOS
IGI/PTR (Hu and Steenkiste, 2003)	2.1	ABW	Both	DEFAULT
Pathchirp (Ribeiro et al., 2003)	2.4.1	ABW	Both	MONITOR, DEFAULT
Pathload (Jain and Dovrolis, 2003)	1.3.2	ABW	Both	SELECTION, QOS
Spruce (Strauss et al., 2003)	0.3	ABW	Both	DEFAULT
Wbest (Li et al., 2008)	1.0	ABW	Both	MONITOR, DEFAULT

Thanks to it, each daemon performs at most one single measurement each time, thus avoiding both interference due to local resource sharing, and interference on the path under measurement. When the nodes of UANM are used as vantage points along a path in a WAN scenario, this offers a mean to mitigate mutual interferences among active measurements in a distributed, peer-to-peer fashion.

Considering the possible plugin status (see Section 4.3 and Fig. 2), it can be noted that mutual exclusion is needed only during the measurement phase: many plugins can be up and ready to start, but the mutual exclusion is only enforced on the transition from ACTIVE and MEASURING states, as described in the following. The different elements composing the UANM architecture with the integration of two levels of control protocols (UANM protocol for the management of experiments and heterogeneous plugin-specific control protocols for the probing) lead to some complexity in the communication sequences.[2] Each daemon can be contacted by a *uanmClient* or by another daemon, the plugin that will be employed can be an INITIATOR or a WAITER, a RECEIVER or a SENDER, and can hold the results of the measurement or not, according to the specific tool, leading to 16 possible cases (all combinations are represented in the current plugin set): we will skip such minor details, focusing only on the sequence that regards the contention of the permission to measure.

In a way similar to the CSMA/CA contention algorithm, synchronization between the INITIATOR and the WAITER sides of a plugin is achieved by means of control messages of types `RequestToStart` and `ClearToStart` or `NotClearToStart`, and by waiting a random `backoff` time. When an INITIATOR in ACTIVE state receives a measure request from the daemon, the measurement is enqueued in the daemon's mutex.[3] When it is popped out from the head of the queue, a `RequestToStart` message is sent to the WAITER and the reply is waited for (with a timeout). If the reply is of type `ClearToStart`, the state of plugin changes to MEASURING, the plugin-specific control protocol is started, and the measurement is performed. At the end of the measurement, the state is reverted to ACTIVE and the mutex is released. If the reply is of type `NotClearToStart`, a random time (uniformly distributed in a `backoff` time interval) is waited, then the `RequestToStart` is sent again; this is iterated until either a `ClearToStart` is received or a given number of retries `max_retries` is reached. On the other side, when a WAITER in ACTIVE state receives a `RequestToStart`, it tries to lock a daemon-scoped mutex. If successful, a `ClearToStart` message is sent back, the state of plugin changes to MEASURING, and the control is handed to the plugin-specific control protocol. If the mutex lock fails (due to another plugin holding it), a `NotClearToStart` message is sent, and no state transition is done; an optional suggested `backoff` can be provided in this message.

An example of this communication sequence is given in Fig. 3, where a SENDER that has the role WAITER is temporarily kept from measuring because of another measurement being on the way.

This mechanism introduces a communication overhead, which implies a delay between the request of measurement and its handling by the plugins. Such delay ranges from 1 RTT to

$$mutex\_overhead = \texttt{max\_retries} \cdot (RTT + \texttt{backoff})$$

This mechanism, however, guarantees the avoidance of concurrent measurements, which translates in higher measure accuracy, as shown in Section 5.2.

4.5. The clients and the UANM API

Every application that is able to communicate with a daemon using the uanm-protocol is a client. A communication API is provided with UANM for this aim. Every application in need of network measurement can benefit from a local centralized orchestration point (daemon) offering enhanced measurement services through a highly abstracted interface. The API provides different measure-request functions ranging from the simplest (e.g. specifying only the endpoints and the type of measure) up to a completely detailed one (e.g. specifying the technique to use and its parameters).

Every application can request a measure that fits its needs using the convenient function. This allows for the use of the UANM servers as probes, to be integrated in distributed measurement infrastructures. In fact, by means of detailed requests, the daemon can also become a transparent layer between the application and the measurement method, so that it can be used as a unifying interface towards different kinds of measurement techniques.

As the main purpose of a daemon is to provide measurement services, the main functions are dedicated to measurement requests. These functions differ in the number of parameters, because the most simple ones make a set of plausible assumptions. The measure-request functions have been designed to provide a simple and flexible interface to the measurement. In the following, we report the kinds of measurement that can be requested by a client.

1. *Measure using profile*: The general use case for the UANM architecture is represented by an application in need of an available bandwidth estimation, with loose constraints, that can just pick one "ready-cooked" *measurement profile* (see Table 2 for a list of available profiles and their characteristics). The *measurement profiles* are an abstract classification of use cases in terms of measurement characteristics,[4] and they are meant as a choice of a preset combination of values for the actual measurement parameters. They are also used by the *decision engine* of the daemon as selection criteria in the choice of the most suitable method. For these reasons, the characteristics of the profiles have been chosen as being useful in

[2] This complexity is hidden by the simple API, which constitutes one of the advantages provided by the use of the UANM platform.

[3] This implements the FCFS policy, at the time being the only supported scheduling.

[4] The measurement profiles have been designed using the available bandwidth estimation as a reference, but the same principles and implementations can be used for every supported type of measure.

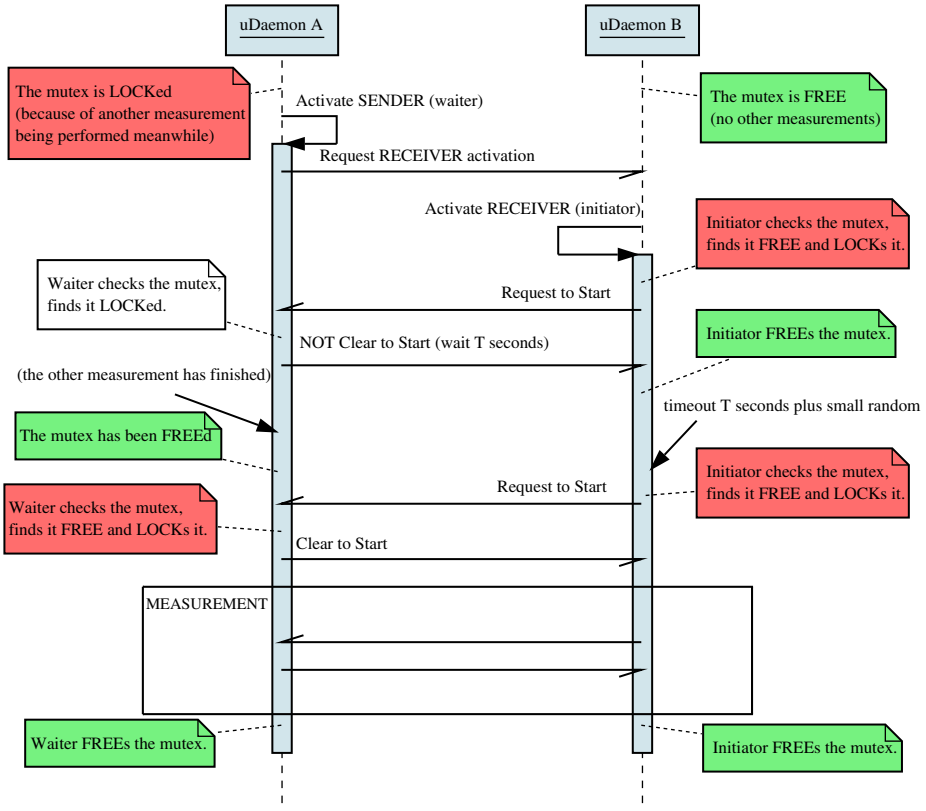

Fig. 3. Measurement mutual exclusion: the case of a SENDER that has the role WAITER and gets deferred because of a LOCKED mutex.

Table 2
List of predefined measurement profiles with characterization.

Profile name	Response time	Accuracy	Repetition frequency	Averaging interval	Probe load
`SELECTION`	Wide	Wide	Once	Wide	High
`MONITOR`	Wide	Wide	Medium	Wide	Low
`QOS`	Strict	Strict	Low	Strict	High
`DEFAULT`	Medium	Medium	Once	Medium	Medium

Table 3
List of supported *measurement constraints*.

Measurement characteristic	Response time	Uncertainty	Averaging interval	Probe load
Type of bound	Upper	Upper	Exact	Upper
Unit	ms	Mbps	ms	Mbps

tool comparison and evaluation (as described in Shriram and Kaur, 2007).

2. *Measure with constraints*: In case the client has specific constraints on the measurement, it is possible to inform the daemon of such constraints by means of this function, specifying the bounding characteristics of the measurement and the value of the acceptable limit. The supported measurement constraints are reported in Table 3.
3. *Fully specified measure*: The maximum control on the measurement process is given to clients through this function, which allows to specify the measurement tool and all the parameters allowed by the tool. The syntax of parameter specification is the same as the command-line syntax of the specific tool, as described in the tool documentation.

To develop "smart" clients and administration interfaces, some administration functions are also provided. According to permissions set for the daemon, some of these functions could be forbidden to some clients. In order to perform additional tasks, ancillary functions are also provided. These functions can be used as building blocks to create a complex client or a manager, or to integrate UANM in another architecture, such as a distributed measurement infrastructure, an anomaly detection system, or a network monitoring application.

5. UANM at work

Thanks to the availability of different tools embedded in UANM, we execute a wide set of comparative tests on a laboratory testbed. In a first set of experiments we validate three important aspects of UANM: (i) the impact of the architecture on the measures with respect to the standalone tools; (ii) the avoidance of the interference of two or more tools in execution on the same measurement path at the same time; (iii) the capability to select the best tool and of properly configuring its parameters for the specific network scenario. In a second set of experiments, we use UANM to compare the accuracy, the intrusiveness, and the measurement time of several available bandwidth estimation tools.

5.1. Testbed and tools

For our experiments we set up a laboratory testbed composed of eight Linux-based hosts (see Fig. 4). More details on the setup

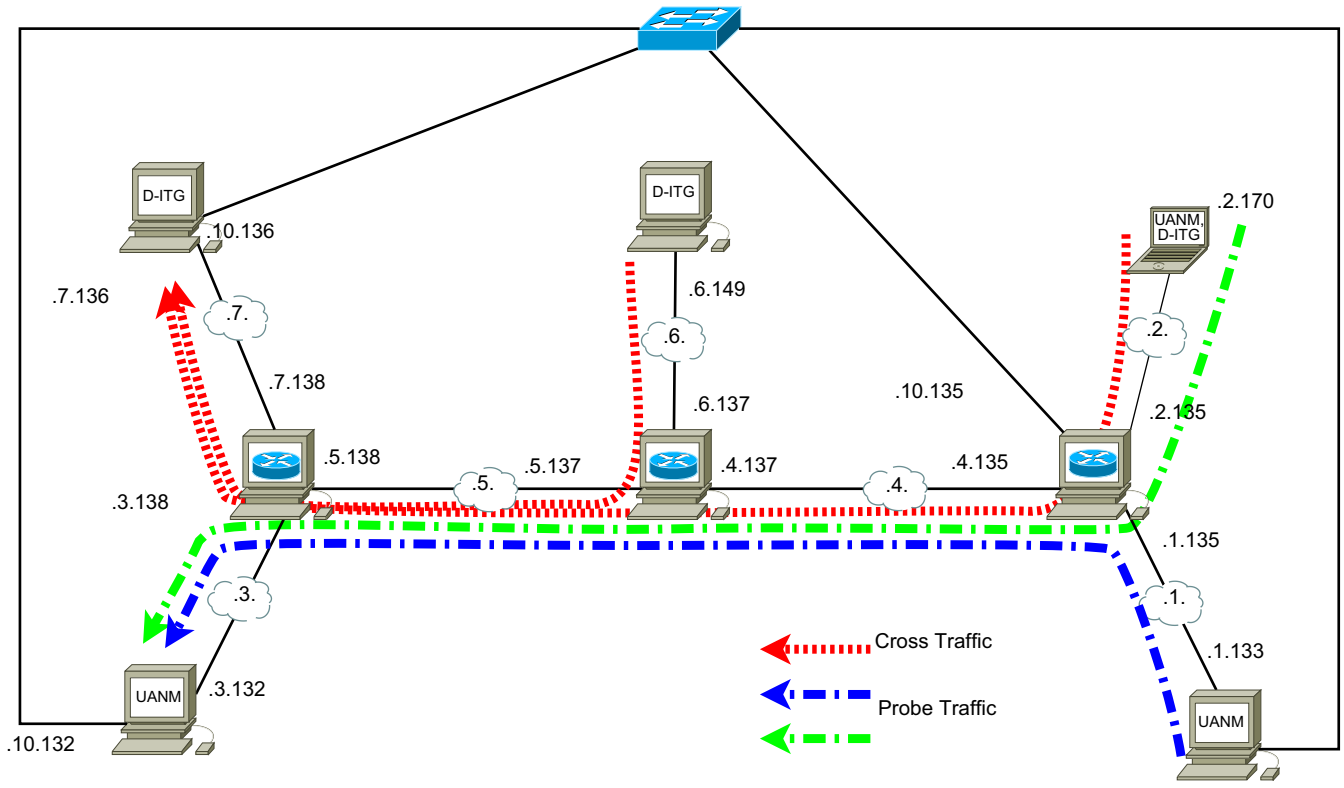

Fig. 4. Testbed used for the experiments.

Table 4
Details of network interfaces installed on the testbed nodes (see Fig. 4 for the testbed layout).

Node	Net	Device	Driver	Driver version	Adapter model
132	ctrl	eth1	e1000	7.3.21-k3-NAPI	Intel 82541GI (r05)
132	3	eth2	sky2	1.22	Marvell 88E8050 ASF (r17)
133	1	eth0	e1000	6.1.16-k2-NAPI	Intel 82541PI (r05)
133	ctrl	eth1	e1000	6.1.16-k2-NAPI	Intel 82541GI/PI (r05)
135	2	eth0	e1000	7.3.20-k2-NAPI	Intel 82541PI (r05)
135	4	eth1	e1000	7.3.20-k2-NAPI	Intel 82541GI (r05)
135	1	eth2	sky2	1.14	Marvell 88E8050 ASF (r17)
136	ctrl	eth0	igb	2.1.0-k2	Intel 82576 (r01)
136	7	eth1	igb	2.1.0-k2	Intel 82576 (r01)
137	6	eth3	sky2	1.21	Marvell 88E8050 (r17)
137	5	eth4	e1000	7.3.20-k2-NAPI	Intel 82541PI (r05)
137	4	eth5	e1000	7.3.20-k2-NAPI	Intel 82541GI/PI (r05)
138	5	eth0	e1000	7.3.20-k2-NAPI	Intel 82541GI/PI (r05)
138	3	eth1	sky2	1.14	Marvell 88E8050 (r17)
138	7	eth2	e1000	7.3.20-k2-NAPI	Intel 82541PI (r05)
149	6	eth0	r8169	2.3LK-NAPI	Realtek RTL-8169 (r10)
149	ctrl	eth2	asix	N/A	ASIX AX88772
170	2	eth0	e1000e	0.3.3.3-k6	Intel 82573L

Table 5
CPUs and Linux kernels installed on testbed nodes.

Node	Cores	CPU	Kernel version
132	2	Intel Xeon 3.2 GHz	Linux 2.6.27.24-mfwd SMP
133	2	Intel Xeon 3.2 GHz	Linux 2.6.15-mpls-1.950 SMP
135	2	Intel Xeon 3.2 GHz	Linux 2.6.22.5-49.fc6.mpls.1.958 SMP
136	8	Intel Xeon E5520 2.27 GHz	Linux 2.6.32-24-server SMP
137	2	Intel Xeon 3.2 GHz	Linux 2.6.26-1-686 SMP
138	2	Intel Xeon 3.2 GHz	Linux 2.6.22.5-49.fc6.mpls.1.958 SMP
149	2	Intel Xeon 3.2 GHz	Linux 2.6.32-24-generic SMP
170	2	Intel Xeon 3.2 GHz	Linux 2.6.27.56-0.1-default SMP

of each machine can be found in Tables 4 and 5. The three intermediate hosts *135*, *137*, and *138* act as routers, and the end systems are provided with *Pathchirp* and *Pathload*, as well as with UANM. To emulate different load conditions, we also provide the three hosts *136*, *149*, and *170* with the D-ITG (Botta et al., 2007) traffic generator. In order to validate the measures provided by the tools and the load conditions enforced by the traffic generator, traffic is captured through `tcpdump` on router *137*. Both the probing packets and the cross traffic are generated from right towards left, along the paths highlighted with arrows. This setup presents a RTT from host *133* to *132* (sender and receiver of probe traffic, respectively) of about 1 ms in average, with about 0.1 ms of standard deviation, measured with 1000 packets sent with zero inter-packet time (ping flooding). The results reported in graphs are collected from client output and daemon logging facilities, and have been validated through the analysis of the traffic traces captured on the router *137*.

5.2. Platform validation

These experiments aim at demonstrating the basic benefits introduced by UANM with respect to a trivial use of available bandwidth estimation tools.

In this section, we focus the attention on the following issues:

1. the evaluation of the overhead introduced by the adoption of UANM with respect to the regular AB estimation tools;
2. the avoidance of the effect of the interference among concurrent measurement processes;

3. the results achieved by UANM when autonomously selecting a tool and its parameters with respect to a basic use of the AB estimation tools.

5.2.1. Overhead introduced by UANM

In the following test we show how the adoption of UANM does not introduce significant overhead with respect to the original tools. Figure 5 reports the results of two series of measurements conducted under different load conditions. This figure shows a diagram with the cross traffic on the *x*-axis and the measured available bandwidth on the *y*-axis, both expressed in Mbps. To increase the readability of such a diagram, we report a line related to the ideal values. All the results are calculated as the average of the outcomes of 10 experiments. We use the same outline for all the validation experiments. In the first measurements we execute Pathload, whose results are returned as usual in a range (PL min and PL max), and Pathchirp (PC). In the other measurements we use the same tools but embedded in UANM under a *fully specified* profile (UANM PL, as the average of min and max, and UANM PC). As shown in Fig. 5, the difference between the outcomes in the two experiments is negligible in all the cases.

5.2.2. Effect of the interference among available bandwidth estimation tools

In this experiment we show the inaccuracy of a measure performed when more measurement processes share even one single part of the network for a long or a short time interval. This problem may occur since the current techniques and tools do not provide coordination among measurement stations or any kind of alert feedback from the network. We set all the links of the testbed to 100 Mbps, and we start two series of concurrent measurement processes from senders *170* and *133* towards receiver *132*. We also generate cross traffic from *149* to *136* at different rates of 10, 20, 50, and 80 Mbps so that there is a bottleneck link between router *137* and *138*.

Figure 6 reports the results of these experiments. The situations considered are as follows: two concurrent Pathload (PL upon PL), Pathchirp upon Pathload (PC upon PL), and two concurrent Pathchirp (PC on PC). Moreover, we also report the results related to a standalone use of Pathload (PL standalone) and Pathchirp (PC standalone). As reported, there is an effect due to the interference that affects both the tools, as the difference between the results of concurrent and non-concurrent measurements is significant. We also notice that, in some experiments with Pathload upon Pathload, the tool did not converge. This result is due to the approach

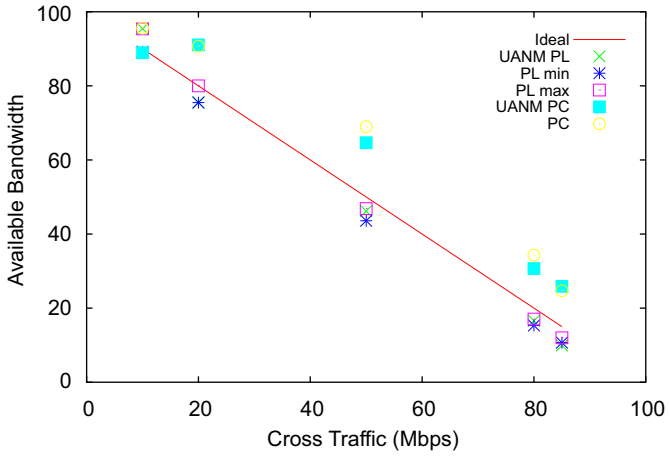

Fig. 5. Overhead introduced by the UANM architecture.

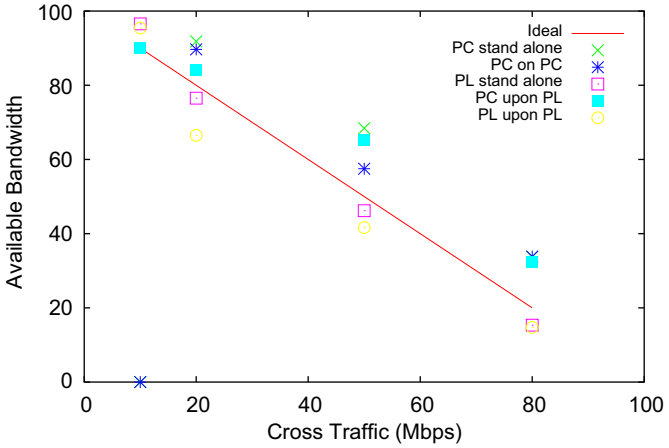

Fig. 6. Effect of the interference among available bandwidth estimation tools.

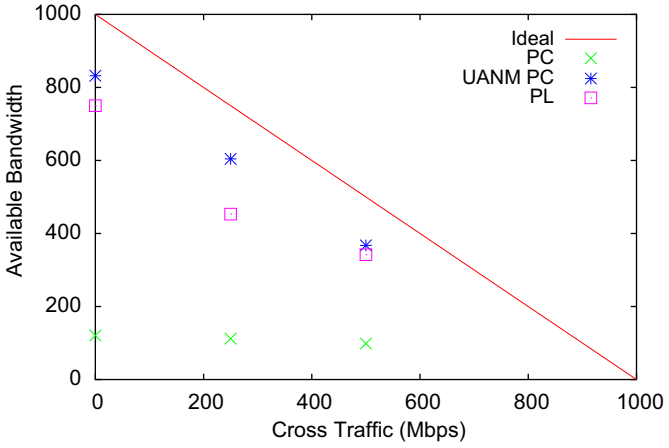

Fig. 7. Selection and trim of the best available bandwidth estimation tool.

adopted in Pathload, which leads the network towards congestion for a short time interval. The design of UANM daemon avoids the interference effect thanks to the *Scheduler* that coordinates the different clients and activates the measurements on a FCFS basis, as reported in Section 4.4.

5.2.3. Selection and trim of the best available bandwidth estimation tool

The *decision engine* in UANM is able to automatically select the best tool related to the particular network configuration and to recognize some basic configurations in order to set up the most appropriate parameters for the required measure. In this experiment we show the difference between a straightforward use of Pathchirp or Pathload with respect to the adoption of UANM onto the same testbed with all the links set to 1 Gbps. We made three series of experiments with cross traffic set to 0, 250, and 500 Mbps. As shown in Fig. 7, a basic usage of Pathchirp returns the worst results, and Pathload still underestimates the ideal values. In such a situation, the UANM *decision engine* selects Pathchirp according to the strict accurate QoS measurement profile, thus preparing the measure with a different configuration of the packet trains. The results of this last series of experiments appear to be the best in all the tested load conditions.

5.3. Use case: comparing available bandwidth estimation techniques

In this section we present the results of a fair comparison of the performance of several AB estimation tools, performed by

means of UANM. All the tools currently available as plugins (see Table 1) are run with the default parameter values, overriding UANM *decision engine* by means of *fully specified* measure requests. While the original Pathload tool returns an upper and a lower bound for the estimated measure, its plugin version returns also the average of those two values. The original IGI/PTR tool returns two values of available bandwidth: one calculated by the IGI algorithm and the other by the PTR algorithm; they slightly differ, except for high cross traffic rates, in which case IGI results are heavily inaccurate. The plugin version of IGI/PTR returns only the value provided by PTR.

No specific settings have been enforced for the operating systems on the testbed nodes (see Table 5), and no change is made on standard settings for the network adapters, except the 100 Mbps set up for the `speed` parameter on all the network adapters involved in the experiments (see Table 4). CBR cross traffic is generated by the host *170* towards the host *136* at rates of 25, 50, 75, and 100 Mbps, and with IP packet size of 1000 Byte. A batch of 10 subsequent requests is performed with each plugin and for each cross traffic rate, as well as in absence of cross traffic. The data collected by the controlling client and the two running UANM-daemons are averaged over the 10 samples, and they are shown and analyzed in the following.

5.3.1. Accuracy

In Fig. 8 the outcomes of accuracy tests are reported for different cross traffic values. Some tools achieved low accuracy, namely Abing (Fig. 8c), Wbest (Fig. 8g), and Spruce (Fig. 8h). In a recent work Goldoni and Schivi (2010) perform a similar analysis, against cross traffic values of 8, 16, 32, and 64 Mbps, and they obtained higher accuracy. In order to verify the cause of this difference, we performed different experiments.[5] Thanks to these experiments, we verified that the difference can be attributed to the fact the whole testbed is equipped with Gigabit Ethernet network adapters, whose interrupt mitigation parameters have been not modified from the default (only the `speed` has been forced to 100 Mbps, while interrupt coalescence has *not* been disabled, differently from Sommers et al., 2006); such condition is known to affect the performance of some available bandwidth estimation tools (Sommers et al., 2006; Jin and Tierney, 2003; Prasad et al., 2004). Besides those reported above, other results deserve some comments. Wbest (Fig. 8g), not considered in Goldoni and Schivi (2010), is tuned by default to operate on IEEE 802.11 wireless networks, so its performance are probably affected by a full wired Fast Ethernet scenario. Diettopp (Fig. 8a) shows the highest performance in terms of accuracy, achieving its best result with 25 Mbps of cross traffic (-2.5 Mbps of average error) and the worst one with fully saturated path (reporting 27 Mbps); it has also the lowest standard deviation, i.e. less than 1 Mbps. Pathload (Fig. 8e) is also quite accurate, with an average error comprised in between -12 and -9 Mbps, but the standard deviation is among the highest ones, mostly in correspondence of low values of cross traffic (almost 28 Mbps for the unloaded path). Pathchirp (Fig. 8f) shows intermediate performance, with an average error comprised in between 3 and 18 Mbps, and the largest error (-35 Mbps) in the absence of cross traffic; its standard deviation is about 5 Mbps, with more stability at higher cross traffic rates. IGI/PTR (Fig. 8f) presents performance comparable to Pathload, with an average error comprised in between -5 and -12 Mbps, and a small standard deviation (in between 1 and 6 Mbps).

[5] We also performed the same measurements with the original version of the tools, obtaining similar results, which excluded UANM from the possible causes.

5.3.2. Probing time

The *probing time* is defined as the difference between the timestamps of the last and the first probe packet. With the inclusion of usually negligible computation time, and with the addition of control communication overhead, this is related to *measurement latency* (time needed by the tool in order to provide a response). For the tools that iteratively probe the network a number of times and provide a final response by filtering intermediate results (Pathchirp, Assolo), we consider the whole measurement period, for three reasons: (i) the filtering process is part of the algorithm; (ii) the value returned by the algorithm refers to the whole probing period; (iii) with this definition, *probing time* is a good index for *measurement latency*. For tools that run continuously, providing a running estimation of the available bandwidth, we forced a time limit of 10 s (we empirically found this value to be enough to get stationary results).

It can be seen in Fig. 9a that the *probing time* of each plugin is almost constant with the cross traffic rate, with the exceptions of IGI/PTR and Pathload. IGI/PTR presents a probing time that is increasing with cross traffic, starting from 74 ms with unloaded path, requiring 0.7 s with 75 Mbps of cross traffic, and 36 s with fully saturated path. In this last case it also shows a standard deviation of 10 s. Pathload shows almost constant probing times, of about 6 s, and a standard deviation of about 2.5 s, for almost all the values of cross traffic. Only at 100 Mbps, it needs about 27 s on average, with a standard deviation of 8 s. Assolo, Diettopp, Pathchirp, and Spruce show probing times comparable with those of Pathload (in between 6 s and 8 s) with very low standard deviation.

5.3.3. Intrusiveness

Figure 9b shows the total amount of probe traffic generated in average by the plugins during each measurement session. For iterative and continuously probing techniques, we use the same conventions used in the *probing time* analysis. It can be noted that almost all the plugins offer a probing load constant with the rate of cross traffic, with the exception of IGI/PTR and Pathload. IGI/PTR generates an amount of probe traffic that is increasing with the cross traffic, starting from 410 kB with unloaded path, up to 3.4 MB when the path is fully saturated, and the standard deviation is increasing from 46 kB to 980 kB. Pathload shows a probe traffic volume that is decreasing with the cross traffic rate. It starts at 6 MB with unloaded path, down to 1.3 MB in saturation, also the standard deviation decreases from 2.8 MB down to 208 kB. This is consistent with the nature of the technique, that has to congest the path, employing more probe traffic when cross traffic contribution is lower. Diettopp generates the same amount of probe traffic in all the cases (5.4 MB), only for the saturated path it reduces the traffic to 2.2 MB showing also higher variability (78 kB of standard deviation). Together with Pathload, it is the tool with the highest intrusiveness. Abing is the least intrusive, with a fixed offered traffic volume of 59 kB in every condition. Similarly, Wbest generates always 134 kB of probe traffic. The volume of probing traffic of Assolo, Pathchirp, and Spruce is almost constant with the cross traffic rate, and all of them generate an amount of probe traffic close to 300 kB.

5.3.4. Discussion

Thanks to this fair comparison, a few considerations can be done. The most accurate plugins are Diettopp and Pathload, both trading off this characteristic with the highest level of intrusiveness, and also high probing time and thus high measurement latency; these properties make them suitable for measurement requests in profile SELECTION or in some applications of profile

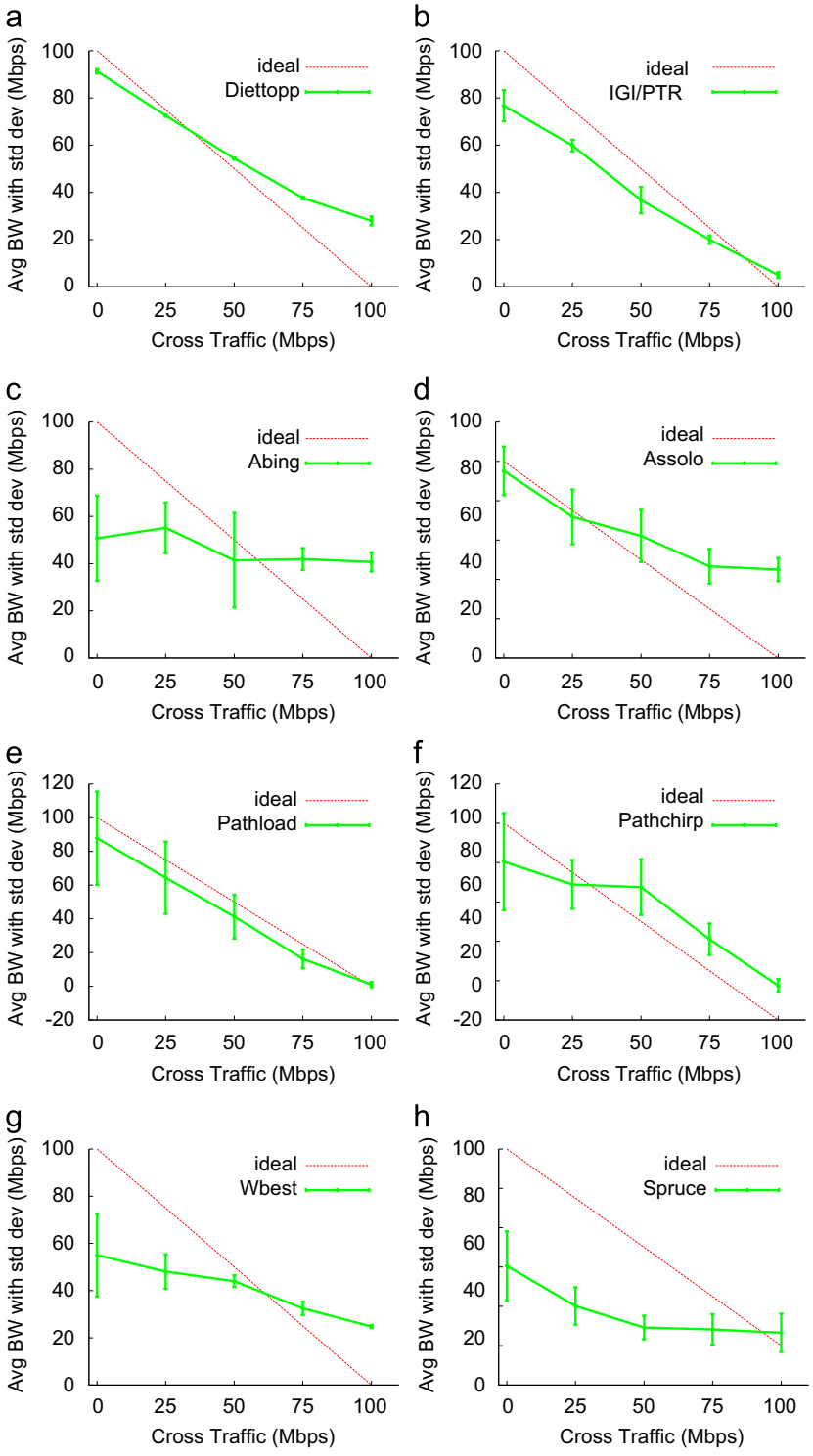

Fig. 8. Experimental results: accuracy of different estimation techniques in the presence of different values of CBR cross traffic with fixed packet size (1000 Byte). (a) Diettopp, (b) IGI/PTR, (c) Abing, (d) Assolo, (e) Pathload, (f) Pathchirp, (g) Wbest and (h) Spruce.

QOS but not for MONITOR or DEFAULT (see Table 2). IGI/PTR is the quickest in all the cases except with the saturated path, and has a good accuracy, but its probing time can range in more than 2 orders of magnitude. For this reason, it is unsuitable for applications that require time-bound measurement (see *Measure with constraint* type of request, Section 4.5); the same is true for its variable probe load. Pathchirp, while being less accurate for low cross-traffic values, has a probing time comparable with Pathload and Diettopp, but significantly less injected traffic; this makes it suitable for MONITOR profile of measurements. The same can be said for Assolo, that shows performance similar to Pathchirp, but higher accuracy for lower cross traffic values.

Even if we used only the default settings of the plugins and a basic network scenario (varying only the cross traffic rate), the experimental results showed that there is no tool that suits the requirements of all the applications. Therefore, an informed choice of the tool and of its parameters are necessary for the effective and efficient estimation of the available bandwidth.

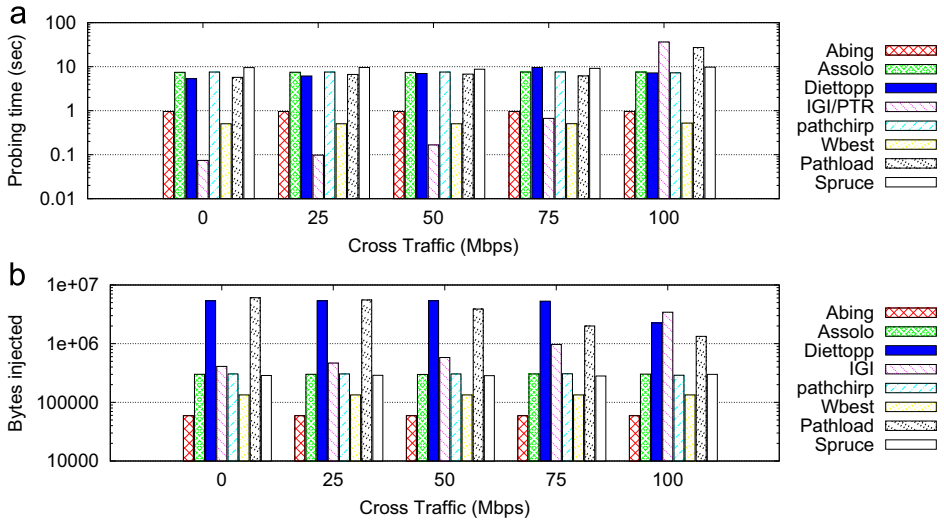

Fig. 9. Experimental results: (a) *probing time* (between first and last probing packet) and (b) *intrusiveness* (total *amount of probe traffic* injected). Cross traffic is CBR with fixed packet size (1000 Byte). Reported values are averaged on 10 runs, in logarithmic scale; standard deviation is not easily readable in the plot, and is reported in the discussion for the few non-negligible cases.

Another important consideration is related to the scalability of the proposed architecture. In this paper, we have performed experiments on a small-scale testbed. The question arises how the platform scales, i.e. what we can expect in large deployments. As also recently discussed in the literature (Croce et al., 2010), for what concerns the techniques for AB estimation, the main issues we can expect on large-scale deployments are the intrusiveness of the probing traffic in terms of both traffic rate and profile, leading to high overhead, and the interference between probing packets, leading to inaccurate estimations. Moreover, the experiments reported in Section 5.2.2 and in Croce et al. (2010) show that each tool has its own characteristics with respect to the issues above. Therefore, it is important to have a platform that allows to manage different kinds of techniques and to choose the best one depending on the network scenario (e.g. for large scale deployments a PGM technique with few iterations; Croce et al., 2010). In particular, to make an informed decision, the tool should take into account:

- The overhead in terms of volume of probing traffic injected into the network. The platform should choose the technique that minimizes the overhead.
- The interference among concurrent measurements. The platform has to avoid concurrent measurements when possible (see Section 4.4), or it has to choose the technique least impacted by the interference, when concurrent measurements are necessary.

Besides, another important issue arises in large-scale deployments. The complexity of the network increases, and this undermines the possibility to perform network-aware measurements. In heterogeneous scenarios it is difficult to discover many things about the network, such as, the number of hops, or, more in general, the topology of the network, the kind of links traversed, etc. A specific analysis is necessary to cope with these aspects. We aim at performing it in our future work.

6. Conclusions and future work

In this paper we presented UANM, a novel platform for supporting and orchestrating complex measurements of end-to-end available bandwidth. Thanks to an open API, existing tools or new ones can be easily pluginized in this platform, still keeping a full compliance with original tools. UANM is able to manage concurrent measurement and to avoid interference, thus increasing the accuracy and reliability of any measurement. We described in detail the architecture of the platform, illustrating all the features we introduced in order to mitigate the problems arising in available bandwidth estimation over heterogeneous networks.

The platform is made up of server nodes (the measurement edges, named UANM-daemons) that offer possibly complex estimations of available bandwidth to applications (named UANM-clients) that query for them. Different estimation techniques are available to the measurement server in the form of dynamically loadable modules, the measurement plugins. The offered service is "smart" in the sense that the knowledge and the choice of the measurement details (which technique to use, and how to set its parameters) are entrusted to the daemon, that carries it according to the request of the client and to the known status of the network path under measure. Measurement requests from clients can be very simple, just specifying the address of the other edge of the measurement path, or fully detailed (the daemon in this case offers scheduling of measurement, in order to avoid mutual interferences). The measurements can be made on the path between a server (UANM-daemon) on one edge, and on the other edge (i) another UANM-daemon, (ii) a supported third-party measurement tool, and (iii) a common host (by means of *single-ended plugins*). In order to validate UANM, we performed an extensive set of comparative experiments carried out on a laboratory testbed, and we showed how UANM can actually provide accurate results in typical situations in which the other tools fail. Moreover, thanks to our platform, we provided for the first time in literature a "fair comparison" of eight available bandwidth estimations tools in terms of accuracy, probing time, and intrusiveness. In our ongoing work, we are extending the set of plugins implemented in UANM, widening both the set of supported measures and the number of estimation techniques for each measure, in order to test the platform in a more challenging scenario that also includes wireless links. Moreover, the scheduling policy will be extended from FCFS to other criteria including time constraints or the availability of independent paths on multi-homed daemons, and the mutual exclusion mechanism will be modified in order to allow its enabling on a per-plugin, instead of per-daemon, granularity. Finally, we plan to

increase the capabilities for network awareness of the platform. UANM is geared towards wide adoption among developers of network applications and researchers in the field of network measurement, and a prototype is released under GPL license and with a LGPL API for the development of measurement plugins. We believe that UANM allows to overcome the main limitations of current tools, providing accurate AB estimation in heterogeneous environments.

Acknowledgments

This work has been partially funded by Seven One Solution srl, LINCE project of the FARO programme jointly financed by the Compagnia di San Paolo, and by the Polo delle Scienze e delle Tecnologie of the University of Napoli Federico II.

References

Aceto G, Botta A, Pescapé A, D'Arienzo M. UANM: a platform for experimenting with available bandwidth estimation tools. In: IEEE symposium on computers and communications; 2010. p. 174–9.

Ali AA, Michaut F, Lepage F. End-to-end available bandwidth measurement tools: a comparative evaluation of performances. In: 4th international workshop on internet performance, simulation, monitoring and measurement. Austria; 2006.

Angrisani L, D'Antonio S, Esposito M, Vadursi M. Techniques for available bandwidth measurement in IP networks: a performance comparison. Computer Networks 2006;50:332–49.

Botta A, D'Antonio S, Pescapé A, Ventre G. BET: a hybrid bandwidth estimation tool. in: Proceedings of the 11th international conference on parallel and distributed systems, vol. 2; 2005. p. 520–4.

Botta A, Dainotti A, Pescapé A. Multi-protocol and multi-platform traffic generation and measurement. In: IEEE INFOCOM 2007 DEMO session; May 2007.

Botta A, Dainotti A, Pescapé A. Do you trust your software-based traffic generator? IEEE Communications Magazine 2010;48:158–65.

Croce D, Mellia M, Leonardi E. The quest for bandwidth estimation techniques for large-scale distributed systems. SIGMETRICS—Performance Evaluation Review 2010;37:20–5.

Goldoni E, Schivi M. End-to-end available bandwidth estimation tools, an experimental comparison. in: Proceedings of TMA '10; 2010. p. 171–82.

Goldoni E, Rossi G, Torelli A. Assolo, a new method for available bandwidth estimation. in: International conference on internet monitoring and protection. Los Alamitos, CA, USA: IEEE Computer Society; 2009. p. 130–6.

Guerrero CD, Labrador MA. Traceband: a fast, low overhead and accurate tool for available bandwidth estimation and monitoring. Computer Networks 2010;54:977–90.

Hu N, Steenkiste P. Evaluation and characterization of available bandwidth probing techniques. IEEE Journal on Selected Areas in Communications 2003;21:879–94.

Jain M, Dovrolis C. End-to-end available bandwidth: measurement methodology, dynamics, and relation with tcp throughput. IEEE/ACM Transactions on Networking 2003;11:537–49.

Jin G, Tierney BL. System capability effects on algorithms for network bandwidth measurement. in: Proceedings of the 3rd ACM SIGCOMM conference on internet measurement, IMC '03. New York, NY, USA: ACM; 2003. p. 27–38.

Johnsson A, Melander B, Björkman, M. Diettopp: A first implementation and evaluation of a simplified bandwidth measurement method. In: Proceedings of the 2nd Swedish national computer networking workshop; 2004.

Lakshminarayanan K, Padmanabhan VN, Padhye J. Bandwidth estimation in broadband access networks. in: IMC '04: Proceedings of the 4th ACM SIGCOMM conference on internet measurement. New York, NY, USA: ACM Press; 2004. p. 314–21.

Li M, Claypool M, Kinicki R. WBest: a bandwidth estimation tool for IEEE 802.11 wireless networks. In: 33rd IEEE conference on local computer networks, LCN 2008; 2008. p. 374–81.

Murray M, Smallen S, Khalili O, Swany M. Comparison of end-to-end bandwidth measurement tools on the 10GigE TeraGrid backbone. in: The 6th IEEE/ACM international workshop on grid computing; 2005. p. 300–3.

Navratil J, Cottrell RL. Abwe: a practical approach to available bandwidth estimation. In: Passive and active measurement (PAM) workshop 2003 proceedings. La Jolla.

Paxson V. End-to-end internet packet dynamics. in: Proceedings of the ACM SIGCOMM '97 conference on applications, technologies, architectures, and protocols for computer communication. New York, NY, USA: ACM; 1997. p. 139–52.

Paxson V. Strategies for sound internet measurement. in: Proceedings of the 4th ACM SIGCOMM conference on internet measurement, IMC '04. New York, NY, USA: ACM; 2004. p. 263–71.

Prasad R, Jain M, Dovrolis C. Effects of interrupt coalescence on network measurements. in: Proceedings of PAM; 2004.

Ribeiro V, Riedi R, Baraniuk R, Navratil J, Cot L. Pathchirp: efficient available bandwidth estimation for network paths. in: Passive and active measurement workshop; 2003.

Shriram A, Kaur J. Empirical evaluation of techniques for measuring available bandwidth. in: IEEE INFOCOM; 2007. p. 2162–70.

Shriram A, Murray M, Hyun Y, Brownlee N, Broido A, Fomenkov M, et al. Comparison of public end-to-end bandwidth estimation tools on high-speed links. in: Proceedings of PAM, vol. 3431; 2005. p. 306–20.

Sommers J, Barford P, Willinger W. A proposed framework for calibration of available bandwidth estimation tools. in: Proceedings of the 11th IEEE symposium on computers and communications, ISCC '06; 2006. p. 709–18.

Song H, Zhang Q. Netquest: a flexible framework for large scale network measurements. IEEE/ACM Transactions on Networking 2007;17:106–19.

Strauss J, Katabi D, Kaashoek F. A measurement study of available bandwidth estimation tools. in: Proceedings of the 3rd ACM SIGCOMM conference on internet measurement, IMC '03. New York, NY, USA: ACM; 2003. p. 39–44.

Urvoy-Keller G, En-Najjary T, Sorniotti A. Operational comparison of available bandwidth estimation tools. ACM SIGCOMM—Computer Communication Review 2008;38:39–42.