Facilitating Focused Internet Measurements*

Zhihua Wen
Case Western Reserve University
Electrical Engineering & Computer Science
10900 Euclid Avenue
Cleveland, OH 44106-7071
zhihua.wen@case.edu

Sipat Triukose
Case Western Reserve University
Electrical Engineering & Computer Science
10900 Euclid Avenue
Cleveland, OH 44106-7071
sipat.triukose@case.edu

Michael Rabinovich
Case Western Reserve University
Electrical Engineering & Computer Science
10900 Euclid Avenue
Cleveland, OH 44106-7071
misha@eecs.case.edu

ABSTRACT

This paper describes our implementation of and initial experiences with DipZoom (for "Deep Internet Performance Zoom"), a novel approach to provide focused, on-demand Internet measurements. Unlike existing approaches that face a difficult challenge of building a measurement platform with sufficiently diverse measurements and measuring hosts, DipZoom implements a matchmaking service instead, using P2P concepts to bring together experimenters in need of measurements with external measurement providers. DipZoom offers the following two main contributions. First, since it is just a facilitator for an open community of participants, it promises unprecedented availability of diverse measurements and measuring points. Second, by offering programmatic access to the entire platform from the experimenter's local computer, DipZoom simplifies staging and execution of complex measurement experiments and lowers the bar for obtaining high-quality measurements.

Categories and Subject Descriptors

C.2.3 [**Computer Systems Organization**]: Network Operations—*Network Monitoring*; C.4 [**Computer Systems Organization**]: performance of systems—*Measurement techniques*

General Terms

Measurement, Experimentation, Performance

*This material is based upon work supported by the National Science Foundation under Grants No. CNS-0520105 and CNS-0551603. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. The presentation of this work was made possible, in part, through financial support from School of Graduate Studies at Case Western Reserve University.

SIGMETRICS'07, June 12–16, 2007, San Diego, California, USA.

Keywords

Network measurements, Internet measurement infrastructures, peer-to-peer systems

1. INTRODUCTION

Network measurements play fundamental role in network design and management. Examples of measurements include a network distance between a given pair of hosts according to some metric, packet loss on the path between them, bottleneck bandwidth of the path, a Web page download time from a given site to a given client, to name just a few. A number of measurement platforms are available to fulfil the need for network measurements. They fall into two broad categories. *A-priori* platforms, such as Skitter [3], AMP [1], IDMaps [10], Surveyor [20], Network Weather Service [40], and M-Coop [35], collect generic measurements irrespective of any specific requests. Other platforms and tools offer *on-demand* measurements [26, 34, 25, 16]. However, the scale and diversity of the Internet make obtaining accurate and representative measurements extremely challenging. For example, most research platforms are deployed on PlanetLab nodes [28] or other well-connected servers. At the same time, Internet-connected devices are increasingly diverse, both in their connectivity (e.g., DSL, dial-up, cable modem, satellite, cellular broadband, Wi-Fi, Wi-Max) and in the device types. Measurements obtained from well-connected servers may not represent this diversity well [2]. Furthermore, with the geographical scale of the Internet, any measurement platform would be hard-pressed to provide a representative sample of the entire Internet. Keynote Systems [21], a commercial measurements provider, attempts to select carefully the location and connectivity of their measuring hosts to reflect typical connectivity of clients, and is rapidly expanding its measurement platform. However, it is a closed system that itself decides what locations, connectivity, and device types constitute a representative sample of the Internet, and which measurements are important to offer. Besides, the cost of Keynote measurements make them inaccessible for researchers.

Another issue is measurement staging. Network measurements are often complex and involve multiple steps. For example, to measure the quality of the server selection of a Web content delivery network (CDN), one might first launch *nslookup* measurements from a number of measurement points to discover a set of CDN servers and the server selected by the CDN for each given measuring point; then

a series of *wget* measurements from each measuring point could be used to compare page download time from the CDN-selected server and from the other discovered servers [36]. In many cases, staging a complex measurement experiment requires an out-of-band discovery of suitable measuring points, negotiating with operators of measurement points involved, and installing measurement scripts at the measurement points. This requires high sophistication on the part of the experimenter, and in many cases professional connections to measuring point operators. As a result, there is a high bar for obtaining high-quality measurements.

This paper proposes an approach to overcome these challenges, concentrating on providing on-demand measurements. Our basic idea is to address the challenge of Internet scale and diversity by utilizing the capacity of Internet users themselves, and to address the challenge of measurement staging by providing a coherent interface for interacting with the resulting (and ever-changing, due to intermittent availability of user hosts) measuring platform. In short, instead of a difficult task of building and maintaining a measurement platform, we implement a *matchmaking service*, which brings together experimenters in need of measurements and external measurement providers.

By making Internet users perform measurements for each other, we in essence implement a peer-to-peer system for network measurements. Our current implementation is in fact analogous to the early Napster approach, with a central index facilitating the discovery of measurement providers.

Our system, which we called DipZoom (for "Deep internet Performance Zoom") supports *focused* (or "zoomed-in"), *on-demand* measurements, where an experimenter can query the system for the measurement points satisfying her specific needs and then perform the measurement from those points. The system also supports an explorative mode, where the experimenter initially executes measurements from a small number of measuring points across a large area and then zooms in on more measuring points from a specific area of interest. DipZoom makes the following main contributions.

- DipZoom drastically lowers the bar for the creation of new measuring points and hence simplifies the recruiting of new MPs. For instance, the developers of the NIMI measurement platform [26] shipped preconfigured measuring hosts to recruit measuring points. DipZoom's MPs are installed by downloading a file and a self-extracting installation script from a Web server. By simplifying the creation of MPs and making them maintenance-free, we hope to be able to attract diverse measuring points and facilitate more accurate measurements than are currently possible.

- DipZoom makes measurements accessible to a casual experimenter. By offering a coherent simple interface to the entire platform, DipZoom makes it possible for the user to discover and request measurements from suitable MPs, using either a graphical DipZoom client or directly from a Java program.

- DipZoom's programmatic access to the system makes it simple to stage complex measurements. A complex measurement is just a Java program, which uses API calls defined by a DipZoom client library to interact with the system. During the experiment, the measurement is coordinated from the program running on the experimenter's computer, which can go through arbitrarily complex steps of discovering MPs, obtaining measurements from them, and obtaining more measurements based on the analysis of the results.

- DipZoom explores two directions in providing incentives for users to become measurement providers. Our current prototype relies on the peer-to-peer approach as an incentive: DipZoom client software is bundled with the MP software, so in order to request measurements from a computer, one must also provide measurements from this computer to others[1]. In the future, we also envision a marketplace of Internet measurements, allowing participants to to charge for their measurement services [7].

We previously presented our broad vision for an open Internet measurements marketplace in a position paper [7]. The current paper describes our implementation of a key aspect of this future marketplace, the matchmaking service that facilitates on-demand measurements. In the rest of the paper, we discuss the related work in more detail in Section 2, describe a high-level architecture of the system in Section 3 and the operation of the main components (measuring points and clients) in Sections 4 and 5. We then present our preliminary experiences with the system in Section 9, first studying the scalability of the centralized core and then presenting preliminary experiments to demonstrate potential benefits of DipZoom. Section 10 gives a summary and outlines our future work.

2. RELATED WORK

Several existing systems leverage Internet users in achieving their goals. Seti@home [32] is a well-known project that harvests CPU cycles from user desktops for a large scientific computation task. DIMES and Lip6 projects (see [6, 39] and papers listed therein) recruit Internet users to contribute a particular measurement experiment conducted by these projects. DipZoom facilitates on-demand measurements initiated and coordinated by any participant. A commercial performance monitoring service utilizing users' hosts is offered by Gomez [29]. However, Gomez is a closed system, which decides which measuring hosts to accept into the system, which measurements to offer, and which hosts to select for a particular measurement ordered by a customer. DipZoom is an open system that acts merely as a matchmaking service and a facilitator between experimenters and measurement providers. In particular, it allows experimenters to access the totality of available MPs and execute complex or long-running experiments programmatically from their own computer.

A number of measurement platforms and tools supporting on-demand measurements are in operation. Examples of research platforms include NIMI [27] and Scriptroute [34], while Keynote Systems [21] is an example of a commercial platform. By making it simple for end-users to become measurement providers, DipZoom promises to offer greater diversity of measurements and measurement points. DipZoom can also be used as a veneer on top of these systems, giving measurement requesters a convenient way of interacting

[1]Measurement points, however, can exist by themselves to allow their deployment on servers and other devices without direct access to end-users.

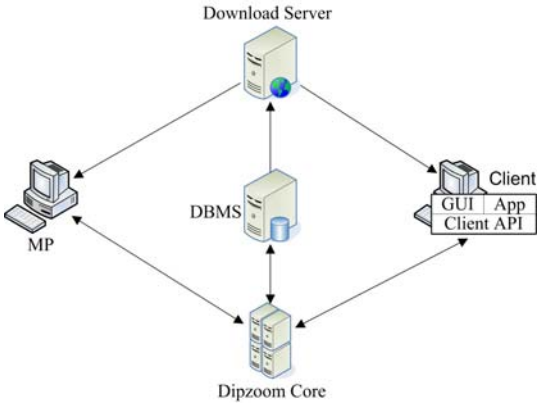

Figure 1: The high-level view of DipZoom architecture

with the entire platform from their own computer using either graphical or programmatic clients.

Many tools are available to measure a variety of metrics, including hop-by-hop bandwidth [17], the bottleneck bandwidth [18, 22, 4], TCP bandwidth [23, 19], latency [24], packet loss [38, 31], and aggregate performance of higher-level operations such as a web page download [12]. We include wget, ping, traceroute, and nslookup as part of the standard MP download. We are currently implementing a "measurement plug-in" mechanism that would allow DipZoom to incorporate arbitrary new measurement tools.

One technique to obtain diverse measurements from end-user perspective is to instrument connections between the client and the server whenever the client requests a service. Specifically, a Web site can send a special Javascript with its pages to run on the client, measure the user experience and send the result back to the server [37]. However, this approach has limited applicability as it can only be used to measure Web page downloads by Javascript-enabled browsers.

3. THE SYSTEM OVERVIEW

The DipZoom platform consists of measuring points (MPs), clients, and the DipZoom core (see Figure 1). The measuring points advertise to the core their capabilities (the platform, the offered measurements, etc.), announce their coming on-line, and perform measurements requested by the core on behalf of the clients. The core maintains the database of the MPs and keeps track of those MPs that are currently on-line. The clients query the database for the MPs that fit their requirements (based on such characteristics as the geographical location, the autonomous system to which the MPs belong, the MP's operating system, the MP's connection bandwidth[2]), submit measurements requests for selected MPs, and download the results.

Thus, DipZoom is in essence a peer-to-peer network for Internet measurements. Our current implementation has a centralized core, similar to Napster, except in our case the core serves as both the central index of peers and the focal point for all communication between clients and MPs. Obviously the core could become a potential bottleneck. While we plan to explore a distributed core architecture in the future (and in fact our vision for DipZoom involves direct interaction between the clients and the MPs [7]), our preliminary scalability study in Section 9 indicates that even the centralized core can serve a large number of DipZoom participants.

To become a measuring point, a user must download and install the DipZoom measuring software, which executes as a daemon and by default starts automatically at boot time. It is essential for some DipZoom applications (see Section 7) to ensure that a host run only one MP instance at a time, and to track individual MP instances for any misbehavior. To this end, the core generates each MP executable instance dynamically for each download, assigning it a globally unique ID (MPID) and embedding into it a unique secret key. All subsequent interactions between the MP instance and the core is encrypted using the instance's secret key. During the installation process, the MP queries for the network interfaces present on the host and invites the user to override the default rate limiting parameters for each interface and each measurement offered. The rate limiting information includes three parameters: the time interval between successive measurements, the bandwidth consumed by an individual measurement (the measurement would be terminated and the partial output returned upon reaching this limit), and the number of outstanding measurement requests the core is allowed to send to the MP. The rate limiting can be specified individually for each measurement type and interface to reflect the fact that different measurements consume different amount of resources, and different interfaces have different capacities (and hence different tolerances for amount of resources consumed by the background measurements).

To become a DipZoom client, a user must download the DipZoom client software, which includes a Java class library and a graphical front-end. In the peer-to-peer spirit, the client software is bundled with an MP instance, so that a computer must serve as a measuring point in order to run a DipZoom client. The client library exposes well-defined application programming interface that the user can utilize to perform complex or long-running measurements programmatically. This ability to script arbitrarily complex measurement experiments is somewhat similar to Scriptroute [34] but with a major difference: while Scriptroute allows the experimenter to specify her own measurement and submit it to a known Scriptroute node, DipZoom allows measurement specifications that include discovering and exploring a number of measuring points in the course of the experiment. For example, the scripted experiment may begin by querying for MPs in Ohio and obtaining a certain measurement from a small sample of the discovered MPs; then based on the results of the above measurement, the experiment may zoom in on MPs in Cleveland and obtain more detailed measurements from this focused MP set. The experimenter specifies these complex experiments by implementing them as a Java program and linking this program with the DipZoom client library.

The graphical front-end is just one example of such a Java program built on top of the DipZoom client library. It allows the human user to actively interact with the system in an explorative mode: discover MPs, select MPs for measurements, analyze the results, and decide on the next steps.

[2]The bandwidth filter is not currently implemented. We plan to adopt the mechanism implemented by Firefox to support this function.

4. THE MP OPERATION

The operation of a measuring point includes the following main stages: the login, idling, and measurements. These stages are discussed in the following subsections.

4.1 Login

The MP initiates the login protocol every time the MP starts. The login process occurs over TCP. It begins with *Hello* message to the core. The Hello message has two parts, the clear text and the cipher text, encrypted using the 128-bits Advanced Encryption Standard (AES) cipher [33][5]. The core uses the MPID sent in the clear to search for the corresponding AES key from the database and to decrypt the rest of the message. The encrypted part includes the MPID again, which the core compares with the clear-text MPID to verify the integrity of the message, a random number `Rand1` generated by the MP (see below), the message type (`ONLINE_NOTIFY` for the Hello message), and the protocol version.

The core responds to the MP with its own encrypted Hello message, which has the format <MPID; Hello; Rand1; Rand2>. `Rand1` is included to prevent the message replay by a malicious attacker: a replayed message, even if properly encrypted, would highly unlikely include the matching `Rand1`. When MP receives the message, it decrypts and checks the MPID and Rand1 to see if the message can be trusted and if it belongs to the current authentication session.

Assuming the above checks pass, the MP sends all the necessary information to the core with the message <MPID; REGISTER; Rand2; Rand3; MP-Info in XML>. This information includes the operating system of the MP, the MAC address of the current network interface, the IP address, the offered measurement types, and for each measurement type, the measurement rate limiting information for the current interface. It may happen that the current network interface has been added to the host after the MP has been installed. In this case, the MP has no rate limiting information for this interface. Then, as in the initial installation phase, the MP would invite the user to override the rate limiting parameters for this interface before initiating the login process with the core.

After the core receives the above message and checks using `Rand2` that it belongs to the current login session, the core verifies the rest of the information in the message and returns the authentication result, success or failure. Should the authentication fail, the MP will keep retrying the authentication every 5 minutes until success or until the MP is terminated.

Finally, the login protocol ensures that only one instance of any MP software is active at a time, and that every host runs only one instance of the MP. While processing the `ONLINE_NOTIFY` message, the core checks if the MPID is already listed among online MPs, and during the processing of the `REGISTER` message, the core checks if the MAC address included in this message has already been announced by an online MP. In either case, the core sends the `Self-Shutdown` message to the old MP (instructing it to go offline) and purges it from the list of online MPs. Thus, in the case of a duplicate login, DipZoom always replaces the old MP with the new one and therefore enforces the single MP instance and single host conditions at all times.

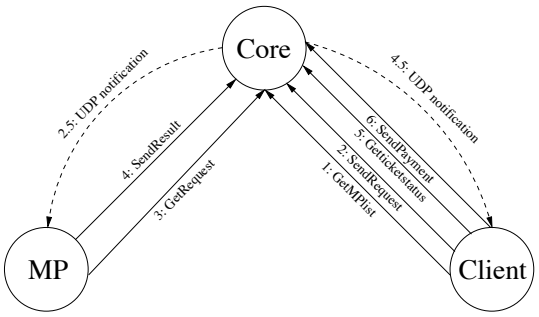

Figure 2: The interaction flows in DipZoom

4.2 Idling

An authenticated MP sends periodic Keep-Alive UDP messages to the core (every 60 seconds). The Keep-Alive message serves two purposes. First, it notifies the core that MP is still online and active. Second, if the MP operates behind a firewall or a network address translation box, it maintains the port mapping in the firewall so that the core can communicate with the MP (see Section 8).

When the core misses three consecutive Keep-Alive messages from an authenticated MP, the core assumes that the MP is no longer active. The core will remove this MP from its list of online MPs and stop including this MP in its responses to client service queries.

Note that Keep-Alive messages may cease not because the MP has terminated but because the network connectivity between the core and MP was lost (e.g., the host left a wi-fi hot spot). In this case, the MP is not aware of the situation, so the messages are still sent but just cannot reach the core. Then, once the network connection is restored, the core will receive an unexpected Keep-Alive message from a presumably off-line MP. To handle this situation, the core responds to any unexpected Keep-Alive message with the Re-authentication message to the MP, forcing the MP to login anew.

4.3 Measurement

The MP-core interactions to perform a measurement are illustrated on the left-hand side of Figure 2. Our general design philosophy is to use UDP communication for core scalability in the common best-effort case, but always have a backup TCP mechanism for reliability and dependable firewall traversal. When the core receives a measurement request from a client (which may include a list of MPs, see Section 5), the core generates separate *measurement tickets* for every MP involved. A measurement ticket is a job assignment for a single MP, e.g., "ping foo.com 10 times from MP with id 101". The core notifies any MP with outstanding tickets to pick up its tickets from the core. The notification is done using a `Measurement_Req` UDP message (step 2.5 in figure 2). In response, the MP opens a TCP connection to the core and obtains all the outstanding tickets (step 3).

In accordance with our general philosophy, the core uses the `Measurement_Req` UDP notification as purely performance optimization to reduce measurement response time: the MP will request any outstanding tickets from the core on its own if it received no notifications for a certain period of time, currently 5 minutes (hence the dashed line corresponding to

step 2.5 in Figure 2). However, by using inexpensive UDP notification, we speed up ticket delivery and also reduce the number of unnecessary TCP connections from the MP to the core when there are no tickets to pick up.

After receiving tickets from the core, the MP executes the measurements and sends the results back over a new TCP connection. The MP does not reuse the TCP connection that it utilized to obtain the tickets: the extra time to open a new connection is negligible compared to the execution time of most measurements, thus the delay reduction from the reused connection would be insignificant. At the same time, not maintaining the TCP connection while the measurement takes place reduces the number of TCP connections at the core and improves the core scalability.

When a client requests measurements of same target from multiple MPs, a possible measurement burst may occur even if the aggregate request rate is within the per-target limits imposed by the core (see Section 7). This may skew measurement results and cause a traffic burst at DipZoom core when the MPs send the results to the core at about the same time. To avoid such a situation, an MP adds a random delay of up to 15 seconds between receiving a new request and executing the measurement. Thus, users who want measurements with coordinated time can only obtain coarse-grained coordination by controlling the time when they send their measurement requests. We plan to investigate options for finer grained control over time of measurement in the future.

We should note that, since MPs are often run on end-user devices, measurement results may be affected by concurrent activities on the MP machine (e.g., due to CPU load or network traffic). It's up to the experimenter to interpret the results carefully by sufficient sampling. Measurement results may also be affected by different versions of the same measurement tool or the version and type of the local operating system. Users can obtain the information about measurement tools and operating systems through DipZoom core to classify the results. However, currently offered measurements seem to be insensitive to measurement tool versions and operating system types.

5. THE DIPZOOM CLIENT OPERATION

DipZoom client operation involves the same stages as the operation of the MPs. The idling operation is similar to the MPs and is not discussed further in this paper. The other modes are discussed below.

5.1 Login

Unlike MPs, the DipZoom clients are allowed to run multiple instances on the same host or copy instances from one computer to another. While running multiple MP instances on a host may skew measurement results and complicate MP cheating detection, multiple client instances do not create these problems. Thus, we do not generate unique client instances for each download: the client software obtained by all users is identical. The main purpose of client login is to authenticate the client (to make sure that the client can only access its own information at the core, e.g., that it cannot delete someone else's measurement results), and to ensure that all its messages are encrypted and cannot be replayed (which is important for some of DipZoom applications that fall outside the scope of this paper).

When a client logins for the very first time, it generates an AES session key S and sands it to the core, encrypted with the core's well-known public key. In response, the core generates a unique client ID and a new AES client key C, and returns this information, encrypted with the client's session key S, to the client. Subsequent communication in the first client session is encrypted with the session key S and carries the client ID, both in the clear and the encrypted part, similar to MPIDs in MP messages.

In subsequent logins, the client also generates a new session key every time for added security, but the login message is encrypted by the client key C, and carries the client ID in both clear and encrypted parts. This allows the core to use the client-specific state previously accumulated from previous sessions in processing client messages such as checking the status of previously submitted measurement requests, or viewing the results from completed requests. Using client-specific key prevents a different client from accessing this state.

5.2 Measurement

The client-core interactions involved in a measurement are shown on the right-hand side of Figure 2. All steps done over TCP are marked in solid lines are those done using UDP are marked in dashed lines. Again, any UDP communication serves as purely performance optimization, and the system will work correctly even if all the UDP messages are lost.

As a first step, the client opens a TCP connection and sends a *service query* to the core to find available online MPs that satisfy the client's requirements, such as location, measurement type, operating system, and autonomous system (step 1 in Figure 2), as well as the number of measurements the client desires from each MP. In response to the service query, the core returns a list of MPs that satisfy the requirements. In computing this list, the core considers the currently online MPs that have enough spare capacity (according to the measurement rate limiting parameters announced by the MPs at login).

The client then uses *another* TCP connection to send its measurement request, which includes the subset of MPs selected form the above list as well as the requested measurement specification (the measurement type, i.e., ping, the measurement target, i.e., cnn.com, the measurement parameters if any, and the number of measurements to be performed). The measurement request uses a new TCP connection because of a potentially long delay between the service query and the measurement request: this delay could be caused for example, by the human user deciding which MPs from the service query response to select for the measurement.

Upon receiving the measurement request, the core generates measurement tickets for individual MPs from the request. The core assigns a unique id to each ticket and returns these ids to the client as the reply to the client's measurement request. The client can use these ticket ids to query the status of submitted tickets and obtain the results in the future.

After generating request tickets, the core executes the measurement protocol with individual MPs involved (see Section 4.3. As results from MPs become available, the core sends a UDP notification to the client (step 4.5). This notification is, again, a pure performance optimization. In the absence of these notifications, the client will periodically (every five minutes) query the core for ticket status over TCP. If any tickets have already been finished by the corresponding

MPs and returned to the core, the client can choose to either wait for more tickets or open a new TCP connection to download the measurements already available (step 6) and thus complete the protocol.

6. DIPZOOM CLIENT APPLICATIONS

We implemented the MP client functionality described in the previous section in a Java class library. The library exposes a simple application programming interface one could use to access DipZoom functionality from an arbitrary Java application. The next subsection describes the DipZoom client API, while the subsequent subsection presents a useful application we ourselves built on top of it. This application implements a graphical front-end to DipZoom, allowing a user to interact with the system in an exploratory mode.

6.1 The Client API

The DipZoom client library provides the following functions for the programmer. Developers can access these functions in their applications by simply linking the applications with the DipZoom library.

- **Login.** The login function can accept as arguments the address and port of the DipZoom core and the login information (the client ID and cipher key). The login function can alternatively accept as the argument the name of an XML file containing the above information, or have no arguments in which case the default file name of "login.xml" is assumed. If the login file does not exist, the system will consider this call to be a first time login, and process it according to Section 5.1.

- **Logout.** This function is called as the last interaction with the core. Any further interaction would have to be preceded with a login call.

- **GetMPlist.** The getMPlist function queries the DipZoom core for a list of MPs satisfying a filter information specified as arguments. The filter information may currently include MPs' country, region, city, AS (autonomous system) number, operating system, measurement type and the number of measurements requested.[3] A successful getMplist call returns the list of MPs that satisfy the filter requirement.

- **Sendrequest** The sendrequest function sends a measurement request to the DipZoom core. The parameters include the measurement type (e.g., "ping") and target (e.g., the hostname to be pinged), the parameter string to be passed to the measurement tool (e.g., "-i 5 -D" for a ping), the number of measurements and a list of selected MPIDs for executing these measurements. These MPs are selected from those provided by a prior getMPlist call. A sendrequest call will return a list of successfully generated measurement ticket IDs, which the core has submitted to the corresponding MPs. Note that tickets may not be generated for some of the requested MPs because of per-MP or per-target rate limits. The sendrequest may also fail altogether, e.g., when the parameter string does not comply with the predefined restrictions for the requested measurement type.

- **Getticketstatus.** This function checks the status of all outstanding requests associated with the client. The getticketstatus function does not take any parameters and returns the ticket number and status for each ticket.

- **Sendpayment.** The confusingly called sendpayment function returns the measurement results of completed tickets, for which a prior getticketstatus call returned a "result received" status. The sendpayment function takes as an argument the list of ticket numbers and returns the list of the corresponding measurement results.

Using the DipZoom client library, an experimenter can script complex and long-running measurements and execute them programmatically. For example, we used this library to implement all the experiments presented in Section 9.

6.2 A DipZoom Graphical Front-End

As an example of an application that can be built on top of DipZoom client library, we have implemented a graphical DipZoom client from-end. It is an intuitive tool to find qualified MPs and submit measurement requests. Figure 3 shows a screenshot of the graphical DipZoom client. In this example, we requested MPs physically located in Cleveland, OH, United States running on Linux and offering single ping measurements. We have found three candidate MPs in Figure 3, and we can select any subset of them to send the ping measurement request. We then can select the measurement target for ping, e.g., google.com, and the measurement parameters, e.g., "-l 1024" (1024 bytes data for each packet). The final command looks just like "ping -l 1024", although it will be run from remote MPs rather than user's local machine. After the request is submitted to core, we can watch the progress of the requests and download the results as they become available at the DipZoom core side.

7. SECURITY

DipZoom is by design an open system: any host can join as an MP, a client, or both. As in any open system, security becomes an important issue. Table 1 lists the main security threats in DipZoom environment, and our counter-measures.

The denial of service attacks against a measurement target and an MP are dealt with in a straightforward way through rate limiting. To prevent an induced distributed DoS attack against a measurement target, DipZoom core uses a leaky bucket algorithm to limit the aggregate rate of measurement requests targeting any given Internet host. The core enforces this aggregate limit across all the MPs by only generating the measurements tickets within this limit. In addition, when generating tickets for a particular MP, the core enforces the limit on the number of outstanding tickets specified by the MP at login. Finally, each MP protects itself by spacing repeated measurements requested in the same ticket and by limiting per-request bandwidth consumption according to the MP configuration file.

[3]We plan to add the bandwidth of MPs' Internet connection in the near future. We will estimate the bandwidth of MP's network connection with the help of MPs' built-in wget measurement, which is similar to the technique used by the Firefox browser for measuring its host bandwidth.

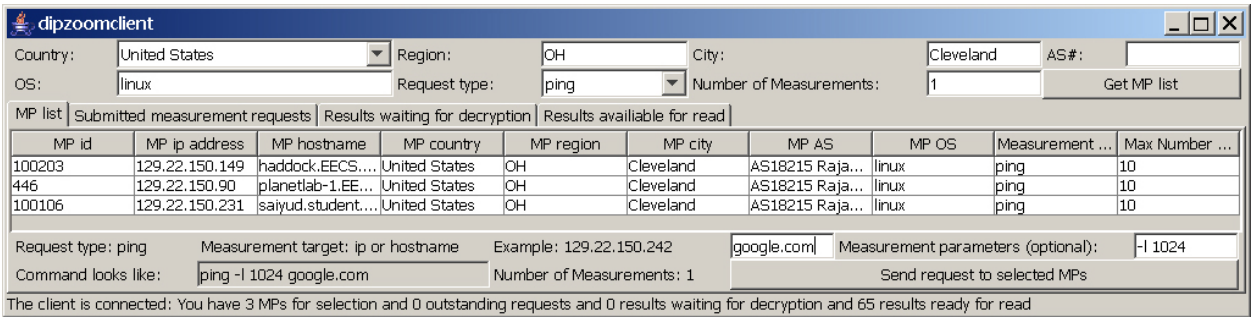

Figure 3: The Graphical DipZoom Client

Security threat	Counter-measure
Induced DoS attack against measurement target	Per-target rate limiting
DoS attack against an MP	Per-MP rate limiting (MP-enforced)
Measurement side-effects	"Trial" measurement by the core
Measurements pollution	Enforcement of single MP instance per host
Fake measurements	Individually generated MP instances with unique embedded keys
Replayed measurements	Using nonce in DipZoom protocol

Table 1: Security threats in DipZoom and counter-measures

The measurement side effects is an interesting threat. As an example, some devices use Web-based management interfaces, and can be configured by invoking a certain URL. An attacker can reconfigure such a device (even if it is behind a firewall!) by requesting an MP behind the same firewall to "measure" the performance of the download of a configuration URL. To prevent this, for wget measurement requests, the DipZoom core will attempt to perform a measurement itself once (using HTTP HEAD request to improve the performance). As long as DipZoom can perform the measurement from outside the MPs network, letting the MP do the same does not increase the vulnerability.

An attacker may attempt to skew measurement results by polluting the set of available MPs with a large number of MP instances that run on a specially configured host that has produces desirable measurement results. The core counters this threat by enforcing at most one MP instance that can login from any given IP and MAC address. Note that a multihomed host can legitimately run multiple MP instances because different interfaces may have different performance characteristics.

A malicious MP may also fake measurement results or lie to the core in announcing some of its characteristics such as operating system. A definitive defense against this threat can only be provided by a trusted computer supporting *program attestation* that Microsoft and Intel are working on [8]. In the meantime, we only raise the bar for implementing this threat by embedding a unique secret key into each downloaded MP instance, and encrypting all interactions between the core and the MP using that key.

Finally, an attacker could replay its own or someone else's messages with measurement results to the core. DipZoom uses a standard solution of including a nonce in every message between the core and an MP and reflecting the received nonce in the next message in the opposite direction (that is, every message carries a reflected nonce received from the other party and a new nonce).

8. FIREWALL AND NAT TRAVERSAL

DipZoom participants, both MPs and clients, are assumed to typically run on end-user devices and other non-dedicated hosts, which are often located behind network address translators (NATs) [4]. NATs hide the IP addresses and port numbers of hosts behind them, and make these hosts unreachable for communication initiated by external hosts. DipZoom needs to be able to penetrate NATs when communicating with the participants. In particular, the DipZoom core needs to notify MPs of new measurement requests and clients of the completed measurement results, while NATs block all externally initiated communication.

A standard technique to address a similar problem in peer-to-peer file sharing systems involves NAT-protected participants initiating and maintaining a full-time persistent TCP connection to a non-firewalled host (which would be the core in our case). However, this technique would create a scalability problem for the centralized core. Instead, we take advantage of recent studies [15, 9] that found most NATs create a temporary port mapping for incoming UDP packets in response to a previous outgoing UDP packet. The external host can send a packet back to the internal address and port by sending the packet to the mapped address and port created by the NAT [30].

Every DipZoom participant already sends periodic heartbeat UDP messages to the core. Since the interval between successive messages, one minute, is less than a typical duration of the port mapping, these messages in effect maintain a full-time port mapping in the NAT[5]. The core takes

[4]The discussion in this section equally applies to firewalls and pure NAT devices, and we use the term NAT to refer to both of them.

[5]Note that, in the case of the firewalls, this port mapping does not appreciably compromise the firewall protection because it only allows UDP messages from the host to which the heartbeats are addressed, i.e., the core, and only to the participant's source port. Even if the attacker obtained the

advantage of this port mapping and sends its UDP notifications through it. Any subsequent data exchanges occur over a TCP connection initiated by the participant from inside, which NATs do not prevent. As a fallback mechanism, if the participant does not receive any UDP notifications from the core for a long time (5 minutes), it opens a TCP connection to the core to check for any missed notifications. This mechanism allows efficient common case processing, where the core does not maintain persistent TCP connections with the participants, and yet in most cases immediately notifies the participants of any action they may need to take, improving the overall responsiveness of the system.

9. PERFORMANCE

This section describes our initial experiences with DipZoom. We first conduct scalability experiments to see how many operations the core can support. We follow with some measurements as examples of the application of the DipZoom system.

9.1 DipZoom Scalability

Our current implementation relies on a centralized core as the tracking index of MPs that are currently online, and as the focal point of communication between the MPs and the clients. Obviously, the core can become the bottleneck. While we have a design that involves a direct communication between clients and MPs and plan in the future to explore a distributed core implementation [7], our preliminary scalability experiments indicate that the core, even when run by a single low-end server, will be able to cope with the load for a sizable number of participants. Our scalability experiments were conducted using a Sun Fire X2100 server with an Opteron 175 2.2GHz CPU and 2G memory. We used a more powerful machine as the load generator (Penguin Altus 1400 server with Opteron 275 CPU and 4G memory) and made sure it was not overloaded during the experiments.

9.1.1 Scalability with Respect to MPs

Our first set of tests concerns the number of MPs the core can support. We study the rate of MP login operations and the operations involved in a measurement the core can handle. For these tests, we disabled the core feature that limits each host to only one MP instance and started a number of instances on the load-generating machine.

To study the login load the DipZoom core can handle, we conducted two experiments, the burst test and continuous test. The burst test examines the limit of concurrent MPs that can login to the core. The continuous test considers the average sustained rate of MP login successes at the core. The burst test is conducted by instructing a number of MPs to login to the core simultaneously and counting the percentage of successful logins and the time to complete the last login operation. In all the experiments (with up to 1000 simultaneous logins), all logins succeeded. Figure 4 shows the time it took the core to process all logins in the burst. It shows that even a backlog of a 1000 logins is cleared in less than 12 seconds, which seems acceptable because this is not an overly time-sensitive operation.

core's IP address and port number, the participant could keep changing the source port number in its heartbeat messages (the feature we have not yet implemented) and could always simply drop any UDP messages without compromising the correct operation as described in this section.

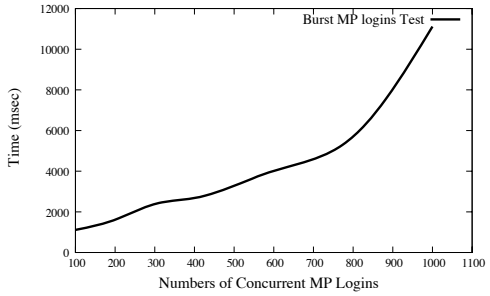

Figure 4: The time to complete a burst of MP logins

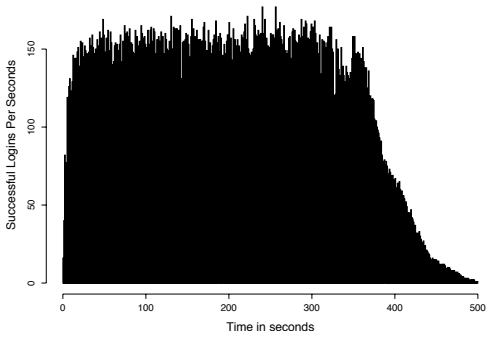

Figure 5: The sustained rate of MP logins

For the continuous test, we start 350 MPs on the load generating machine, with each MP logging in and out every second. The test was run until 70,000 MP login successes have been reached. Figure 5 plots the histogram showing the number of login successes observed in each second of the experiment. We see the sustained rate of around 150 logins/sec, with the tail showing the clearance of the backlog of the extra operations. Note that each login recorded in this test actually corresponds to two operations, a login and a logout (we do not report them as separate operations because login is twice more expensive). Thus, on average the core can support around 300 logins and logouts per second. Assume that on average a regular user turns on his computer 3 times per day (once in the morning at work, another time in the afternoon after lunch, and the last time at home). Then 150 logins/sec could support up to 4.3 million MPs assuming their logins are uniformly distributed over a 24 hour period. Although uniform logins are unrealistic, clearly the core can support a large number of logins.

Now let us turn to the interactions between the MP and the core to perform a measurement. There are two operations involved: getting the tickets and sending back the results. For this test, we start 250 "stub" MPs, which login, wait for the core to generate a large number of tickets for all the MPs, and then launch a loop in which they repeatedly get a ticket and immediately respond with a pre-recorded "result" without any delay between iterations. To maximize the stress on the core, we modified it to always return a single ticket in each iteration, although in reality outstanding tickets to the same MP can be batched. We record, at the core side, the timestamps of successful insertions of the measurement "results" into the database. Each MP terminates

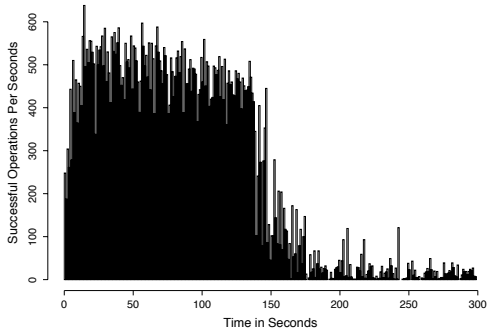

Figure 6: The sustained rate of MP operations involved in a measurement

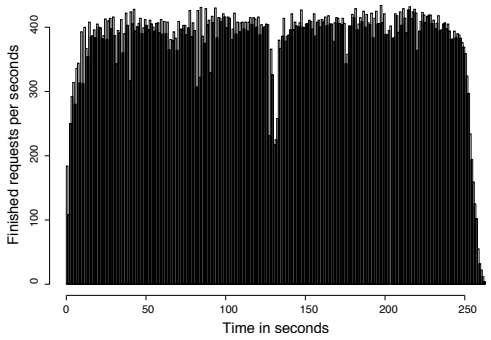

Figure 7: Sustained rate of concurrent client operations involved in requesting a measurement

after processing 300 tickets. Figure 6 shows the number of operations completed in each second. It shows a sustained rate of about 550 operations per second; the long tail indicates clearing of the backlogged operations as a growing number of MPs reach their processed ticket limit and terminate.

9.1.2 Scalability with Respect to Clients

In order to examine the scalability of DipZoom core for concurrent clients requests, we have conducted both burst test and continuous test. In the first test, we let 500, 1000, 1500, 2000 and 2500 different clients login at the same time and check how many clients succeeded. The results are shown in table 2, we can see the success rate is 100% even when we have 2000 clients login at the same time. We should mention that this is a single-attempt login success rate; in reality a DipZoom client will try several times before giving up. This result shows that the DipZoom clients can login successfully even at a very high concurrency rate.

No. of concurrent clients	No. of success	Success rate
500	500	100 %
750	750	100 %
1000	1000	100 %
1500	1500	100 %
2000	2000	100 %
2500	2493	99.72 %

Table 2: Number of concurrent client logins and success rate

In the continuous test, we let 250 clients login, then repeatedly run getmplist and sendrequest functions for 200 times, sleeping for one second after each pair of operations, and then logout. We recorded the finish time for each successful getmplist and sendrequest operation at the core. Figure 7 shows the finished rate at each second. It indicates that DipZoom core can finish client requests at a sustained rate of at least 400 requests per second.

9.2 Demonstration Experiments

To show the capabilities of DipZoom, we conducted experiments testing the accuracy of two recently proposed tools for measuring the distance between a pair of hosts: King [16] and GNP [25]. The evaluation reported in the respective papers describing these tools was limited to well-connected servers as one of the hosts in each pair, because the authors only had publicly available traceroute servers at their disposal. With DipZoom, we were able to use residential measuring points that our colleagues downloaded at our request, thanks to the simplicity of joining the DipZoom platform. Thus, we were able to test the accuracy of these tools separately for well-connected servers (using MPs that we deployed on PlanetLab nodes) and for residential hosts. We implemented all the experiments as Java programs on top of the DipZoom client library.

9.2.1 The Accuracy of King Measurements

King measures the distance between arbitrary pair of hosts by cleverly tricking their respective DNS servers to query each other and approximating the distance between the hosts by the distance between their DNS servers. In our experiments, we select one of the hosts in each pair to be a computer running a DipZoom MP and then compare the distance from this computer to the other host with the measured distance from that MP to the other host. For the well-connected MPs, we selected three PlanetLab nodes in Italy, Ohio, and New York. For the residential MPs connected by DSL lines, we used three hosts in Ohio, Texas, and Michigan. We also performed this measurement using an MP connected via high-speed DSL (6Mbps), and an MP using a recently introduced Verizon Broadband Cellular connection (advertised bandwidth of 500Kbps). For the other host in the pairs, we selected over 1500 pingable nodes from a set of 200,000 IP addresses from a Gnutella network snapshot [13][14].

For each pair, we measure the distance using King four times, and then directly using DipZoom's ping measurements 10 times. Following the format of the King paper, we represent the accuracy of King measurements as a ratio of the King result to DipZoom's ping.

Figure 8 presents the CDF of the ratios for all MPs (*All Users*), as well separate CDFs for the well-connected MPs, residential DSL-connected MPs, and the MP using cellular wireless. The more the CDF curves jump vertically along the ratio = 1, the more accurate King estimates are. The values of the CDF function left of x=1 point show the percentages of under-estimated results. Figure 8 shows that considering all MPs together, over 80 percents of King results tend to underestimate the distance (matching closely the corresponding result from the King study) and almost half of all estimations have values less than half of the ac-

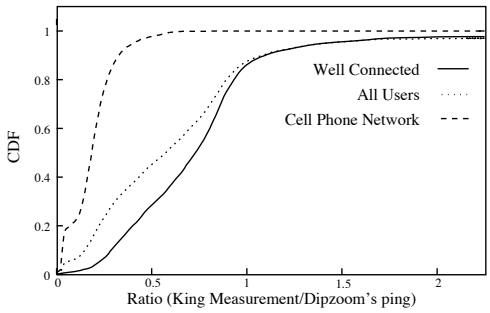

Figure 8: The accuracy of King measurements

tual latency. The latter result is significantly worse than measured in the King study.

We find an explanation to the discrepancy by considering individual classes of MP. The well-connected MPs show very similar accuracy to that reported in King study. Thus, the discrepancy is due to the slower-connected residential MPs. At the extreme, the cellular MP shows extreme inaccuracy (under-estimation) of King estimations.

We should mention that, during the experiment, we found that King was unable to measure distances between many pairs of IP. Approximately 20 percents of all residential MPs we tried consistently could not be measured by King. In particular, this problem was found among nodes in Europe and East Asia.

In summary, our experiments confirm the accuracy of King as the distance estimator for well-connected nodes. DipZoom also allowed us to show the nuanced differences in King accuracy for hosts with diverse connectivity.

9.2.2 The GNP Accuracy

Our other experiment considers the GNP system [25]. GNP selects a set of *landmark* hosts, which measure distances between each other and place themselves into a multi-dimensional geometric coordinate space. Then, to estimate the distance between two hosts, one measures the distance from both hosts to each of the landmarks, uses these distances to place each host into the same coordinate space, and computes the distance between the hosts as the Euclidean distance between the two points. If we also have the measured ping distance between these two hosts, we can use compare it with the predicted distance to check the accuracy of the GNP approach.

In order to measure GNP accuracy, we let around 100 MPs each ping a list of about 1000 pingable IP addresses selected from the Gnutella list using DipZoom client library. To improve accuracy, we repeated the test several times and obtained more than 4 million ping results. We used the minimum ping result value between two hosts as their distance. Following the GNP study, we used 15 landmarks and 8 dimensions. Because GNP requires the availability of pings between each measured host and the 15 landmarks, we limited our study to the US hosts. We chose 15 well connected geographically distributed MPs on PlanetLab nodes as landmarks; our landmarks are shown in table 3. We then use the distances to these landmarks to compute the coordinates for all the hosts using the tools obtained from the GNP website [11]. We compare the measured ping distance

IP address	State	City
216.165.109.79	NY	New York
128.114.63.14	CA	Santa Cruz
141.213.4.201	MI	Ann Arbor
128.193.33.7	OR	Corvallis
164.107.127.13	OH	Columbus
129.10.120.112	MA	Tewksbury
128.135.11.152	IL	Chicago
128.4.36.12	DE	Newark
144.216.2.53	NE	Kearney
128.220.247.28	MD	Baltimore
128.112.139.97	NJ	Princeton
198.133.224.145	WI	Madison
156.56.103.62	IN	Bloomington
128.83.122.179	TX	Austin
152.3.138.2	NC	Durham

Table 3: The IP addresses and locations of the landmarks

IP address	State	City
71.231.18.182	WA	Tacoma
67.186.35.152	PA	Pittsburgh
70.238.242.222	OH	Cleveland
68.40.208.255	MI	Waterford
72.72.35.155	MA	Rockland
71.157.135.213	OH	Cleveland
75.10.128.127	MI	Lansing
65.43.173.197	OH	Cleveland
72.72.98.62	MA	Rockland
70.239.25.197	OH	Cleveland

Table 4: The IP addresses and locations of the US residential MPs

and the distance predicted by GNP. As in the King experiment, we separated the results into two groups, the distances from well-connected PlanetLab-based MPs and the ones from residential MPs. We had 81 well-connected MPs and 10 residential MPs. Our residential MPs are listed in Table 4. Admittedly, the distribution of these MPs is skewed towards Northeastern and Midwestern US, and Cleveland is disproportionably represented. Thus, these results should be viewed only as an indication of potential capabilities of the DipZoom platform.

Following the GNP study [25], we used relative error to quantify the accuracy of GNP estimates in this experiment:

$$\frac{|predicteddistance - measureddistance|}{min(measureddistance, predicteddistance)}$$

Figure 9 shows the cumulative probability distribution functions of the relative error in distance predictions, separately for the well-connected and residential MPs. We see excellent results for both MP types. About 95 percent of all the results have relative error less or qual than 50 percent, and remarkably, there is little difference between the residential and well-connected MPs. In these experiments, the distance between each landmark and each measured host was computed as the minimum of four DipZoom measurements on average, and each DipZoom measurement was a minimum ping delay from 10 ping probes. Interestingly, during the initial phase of our experiment, when we had

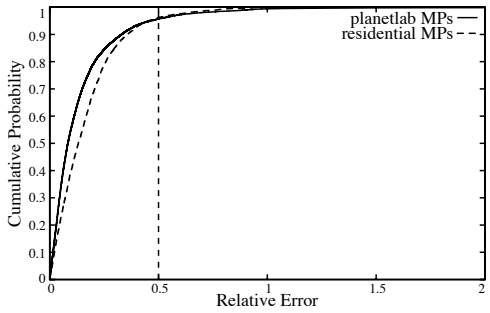

Figure 9: Relative error of GNP distance estimates

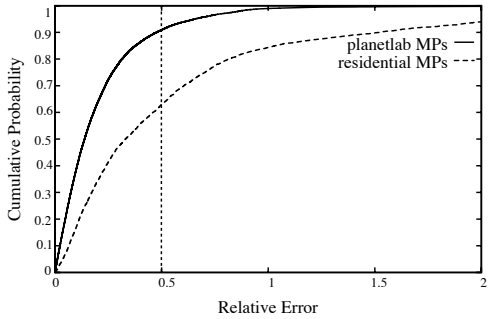

Figure 10: Relative error of GNP distance estimates using the average of 2.5 measurements of landmark-to-host delay

on average only 2.5 DipZoom measurements per landmark-host pair, the quality of the results from residential MPs was much worse while the well-connected hosts were much less affected (see Figure 10). This indicates much higher delay variability of DSL lines, and suggests the need for higher number of repeated measurements when reasoning about these connections.

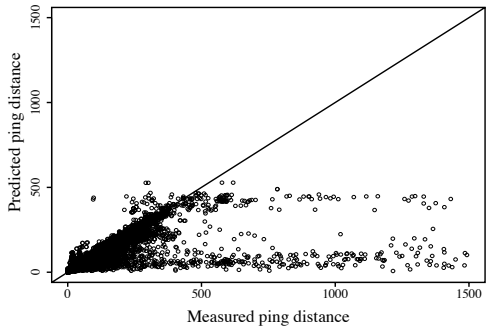

Figure 11: Measured vs. GNP-predicted distances between well-connected MPs and Gnutella peers

Finally, because relative error does not indicate under and over estimation, we plot the measured ping distance between a host pair against GNP-predicted distance. In ideal case, all the points should fall on the center 45 degree line. Figure 11 presents the distances between well-connected MPs and Gnutella peers, and Figure 12 shows the

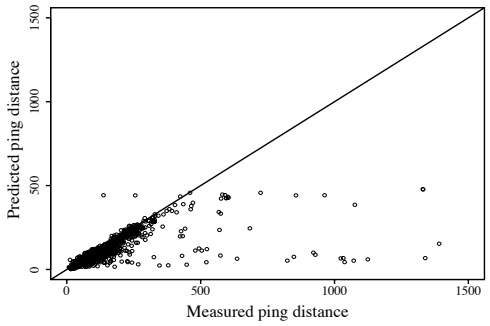

Figure 12: Measured vs. GNP-predicted distances between residential MPs and Gnutella peers

distances between residential MPs and Gnutella peers. Both graphs show that most points are close to the center line, reflecting the the accuracy of GNP prediction. However, we also see that the number of under-estimations (indicated by the points below the 45-degree line) is much larger than the number of over-estimations. Furthermore, most over-estimations happen when the ping distance is small while most under-estimations happen when the ping distance is large.

Overall, we observe that GNP has higher accuracy than King and most importantly, is much less sensitive to the connectivity type of measured hosts. (Of course, King's advantage is much faster-produced results.) Both tools, however, tend to under-estimate the inter-host distances.

10. CONCLUSION

This paper presents our implementation and initial experiences with DipZoom, a system for facilitating diverse on-demand network measurements. The system addresses several key needs of networks researchers and designers, as well as IT specialists. First, it addresses the challenge of creating a measurement platform that would be representative of the scale and diversity of today's Internet. Instead of deploying its own platform, DipZoom makes it easy for Internet users to join the network of measurement providers. Second, it significantly lowers the bar for measurement requesters to stage an experiment. An experimenter has a consistent view of the totality of all the measuring points currently available and has a coherent interface to discover the MPs that suits her needs. Further, she can script a complex and long-running experiment, launch it, and collect the results from her own computer using a general programming language (Java). By simplifying the creation of measuring points and improving accessibility of measurements, we hope DipZoom will increase the use of sound measurements in the networking arena and thus reduce the scope for decisions based on guesswork and intuition.

Our immediate future plans include designing and implementing an extensibility mechanism, so that new measuring tools can be plugged in the exiting MPs. This would allow third-party developers to incorporate their measurements into the general DipZoom framework. In particular, it will allow DipZoom to be used as a thin veneer on top of existing measurement platforms, and let these existing platforms benefit from the improved accessibility to experimenters due to DipZoom's simple client interface. Longer term, we plan

to include incentives for measurement providers and incorporate a ranking and reputation system for measuring points.

Acknowledgements

We would like to thank Oliver Spatscheck and Shubho Sen of AT&T Labs – Research for fruitful discussions. We thank our colleagues and friends for deploying MPs on their computers to make our experiments possible. We are grateful to the anonymous referees for their insightful comments.

11. REFERENCES

[1] Active measurement project. http://amp.nlanr.net/.

[2] S. Banerjee, T. G. Griffin, and M. Pias. The interdomain connectivity of PlanetLab nodes. In *Passive and Active Measurement Workshop*, April 2004.

[3] CAIDA. Skitter. http://www.caida.org/tools/measurments/skitter/.

[4] R. Carter and M. Crovella. Dynamic server selection using bandwidth probing in wide-area networks. In *IEEE Infocom*, 1997.

[5] J. Daemen and V. Rijmen. *The design of Rijndael: AES — the Advanced Encryption Standard*. Springer-Verlag, 2002.

[6] Dimes (distributed internet measurements & simulations). http://www.netdimes.org/.

[7] Reference removed for double-blind review.

[8] P. England, B. Lampson, J. Manferdelli, M. Peinado, and B. Willman. A trusted open platform. *Computer*, 36(7):55–62, 2003.

[9] B. Ford, P. Srisuresh, and D. Kegel. Peer-to-peer communication across network address translators. In *USENIX Annual Technical Conf.*, pages 179–192, 2006.

[10] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang. IDMaps: a global internet host distance estimation service. *IEEE/ACM Trans. Netw.*, 9(5):525–540, 2001.

[11] Gnp. http://www.cs.rice.edu/ eugeneng/research/gnp.

[12] GNU wget. http://www.gnu.org/software/wget/wget.html.

[13] Gnutella. http://www.gnutella.com.

[14] Gnutella Topology Snapshots. http://mirage.cs.uoregon.edu/P2P/info.cgi.

[15] S. Guha and P. Fransis. Characterization and measurement of tcp traversal through nats and firewalls. In *Internet Measurement Conference 2005*, Berkeley, CA, October 2005.

[16] K. Gummadi, S. Saroiu, and S. Gribble. King: Estimating latency between arbitrary internet end hosts. In *Proceedings of the Second SIGCOMM Internet Measurement Workshop*, 2002.

[17] V. Jacobson. Pathchar. ftp://ftp.ee.lbl.gov/pathchar.

[18] M. Jain and C. Dovrolis. End-to-end available bandwidth: Measurement methodology, dynamics, and relation with tcp throughput. In *Proceedings of SIGCOMM*, Pittsburgh, PA, August 2002.

[19] R. Jones. Netperf. http://www.netperf.org/.

[20] S. Kalidindi and M. J. Zekauskas. Surveyor: An infrastructure for internet performance measurements. In *INET'99*, 1999.

[21] Measurement and monitoring: Web site perspective. www.keynote.com/solutions/website_perspective.html.

[22] K. Lai and M. Baker. Nettimer: A tool for measuring bottleneck link bandwidth. In *USITS*, 2001.

[23] M. Mathis. Diagnosing internet congestion with a transport layer performance tool. In *INET'96*, 1996.

[24] M. Muuss. Ping. http://directory.fsf.org/network/misc/ping.html.

[25] T. S. E. Ng and H. Zhang. Predicting internet network distance with coordinates-based approaches. In *INFOCOM*, 2002.

[26] V. Paxson, A. Adams, and M. Mathis. Experiences with NIMI. In *Proceedings of Passive and Active Measurement Conf.*, 2000.

[27] V. Paxson, J. Mahdavi, A. Adams, and M. Mathis. An architecture for large-scale internet measurements. *IEEE Communications*, 36(8):48–54, 1998.

[28] Planetlab. http://www.planet-lab.org.

[29] http://www.porivo.com/.

[30] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy. R. rfc 3489: Stun – simple traversal of user datagram protocol (udp) through network address translators (nats), March 2003.

[31] S. Savage. Sting: a TCP-based network measurement tool. In *USITS*, 1999.

[32] Seti@home. http://www.seti.org/.

[33] Specification for the advanced encryption standard (`AES`). Federal Information Processing Standards Publication 197, 2001.

[34] N. Spring, D. Wetherall, and T. Anderson. Scriptroute: A public internet measurement facility. In *Usenix Symp. on Internet Technologies and Systems*, 2003.

[35] S. Srinivasan and E. W. Zegura. Network measurement as a cooperative enterprise. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 166–177. Springer-Verlag, 2002.

[36] A.-J. Su, D. R. Choffnes, A. Kuzmanovic, and F. E. Bustamante. Drafting behind akamai (travelocity-based detouring). In *SIGCOMM*, pages 435–446, 2006.

[37] Symphoniq. http:/www.symphoniq.com.

[38] A. Tirumala, F. Qin, J. Dugan, and J. Ferguson. Iperf, 2002. http://dast.nlanr.net/Projects/Iperf/.

[39] Traceroute@home, Laboratoire lip6 – CNRS. http://tracerouteathome.net/.

[40] R. Wolski, N. T. Spring, and J. Hayes. The network weather service: a distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, 15(5–6):757–768, 1999.