УДК 004.002
Melnyk V.M.,Melnyk K.V., Zhyharevych O.K.
Lutsk National Technical University

NETWORK EMULATION FOR JAVA-BASED APPLICATIONS THROUGH SOCKET FACTORIES

Melnyk V.M.,Melnyk K.V., Zhyharevych O.K. **Network emulation for java-based applications through socket factories.** *Network emulation discussed in the article provides the capability of evaluating distributed applications on a stand-alone system. Applications can be exposed to adverse repeatable network conditions without requiring complex testbeds. This paper describes the design and implementation of a portable and object-oriented network emulator targeted to the development and test of Java-based Internet applications. The emulator is based on instrumented sockets is able to emulate the behavior of links with a given bandwidth and communication delay. The emulator is organized modularly, so that it is possible to plug-in user-defined classes for bandwidth and delay figures generation. Carrying out experiments with emulated sockets is as simple as running the tested application code on a single host.*
KEYWORDS: *Network, emulator, Internet application, sockets, Java-development.*
Fig. 1., Ref. 6.

Мельник В.М., Мельник К.В., Жигаревич О.К. Мережева емуляція для java-додатків за допомогою об'єднань сокетів. *Емуляція мережі забезпечує можливість оцінки розподілених додатків в автономній системі. Додатки можуть мати схильність виявлення умов повторності мережі, не вимагаючи складних тестових систем. Описується розробка і реалізація портативного об'єктно-орієнтованого емулятора мережі, орієнтованого на створення і тестування додатків для Інтернет, розроблених на Java. Емулятор заснований на інструментальних сокетах і здатний емулювати поведінку зв'язків із заданою пропускною і затримуючою здатністю. В організований за модульним принципом, так що можна підключити його в класах користувачів для генерації пропускної здатності та залежностей затримки. Проведення експериментів з подібними сокетами є простим, як і виконання тестового коду програми на одному хості.*
КЛЮЧОВІ СЛОВА: *мережа, емулятор, Інтернет-додаток, сокети, розробка в Java.*
Рис. 5., Літ. 7.

Мельник В.М., Мельник К.В., Жигаревич О.К. Сетевая эмуляция для java-приложений с использованием сокетных объединений. *Эмуляция сети обеспечивает возможность оценки распределенных приложений в автономной системе. Приложения могут иметь склонность изъявления условий повторимости сети, не требуя сложных систем тестирования. Описывается разработка и реализация портативного объектно-ориентированного эмулятора сети, ориентированного на разработку и тестирование приложений интернет, разработанных на Java. Эмулятор, созданный на инструментальных сокетах и способный эмулировать поведение связей с заданной пропускной и задерживающей способностью. Он организован за модульным принципом, так что можно подключать его в классах пользователей для генерации пропускной способности и зависимостей задержки. Проведение экспериментов с описанными выше сокетами простое, как и исполнение тестового кода программы на одном хосте.*
КЛЮЧЕВЫЕ СЛОВА: *сеть, эмулятор, Интернет-приложение, сокеты, разработка в Java.*
Рис. 5., Лит. 7.

Introduction

The amount of Internet-oriented applications has increased during recent years. This process concerns area of the applications development and the re-building area of many Internet enabled applications needed to be in Internet in order to have their lives prolonged. From the other site, not so many tools have been introduced to support such application development. In many cases, if the final product is developed in an Internet setting, the phases of development and testing are carried out on a single machine and sometimes inside a LAN. However, bandwidth and communication delay differ of some magnitude orders when operating on a single host or in a LAN, rather than through links with long hauls or with dial-up connections. Some big difference between the deployment and the development environment is likely to hide design issues affecting system performances that can be relevant for the user. In this case, because of differences in two environments the debugging phase could be less effective. For the matter let us to implement a complex user interface as an Applet, interacting with a server by means of remote method invocation, the Java object oriented version of RPC. It would be useful to test interface responsiveness and friendliness of use in its operative environment before deploying the product. To cope with this problem, we can test either the application directly on the field or using a network emulator because of sensitivity RPC-based programs to network delay. The solution of course has some drawbacks and advantages: the effectiveness of an application testing in the real operative environment is counterly balanced by the difficulty of setting some testbed. Some different problem is that developer is not able to reproduce or control the test conditions. The guarantee of emulation is a simple set up of experiments and their repeatability. It can be for good solution if it is not required a big amount of details.

A simple flexible network emulator for Java applications is presented for messages intercepting between the application and the network API. It also manipulates the communication to behave as in the network presence with delay and limited bandwidth. The emulator is based on a customized socket (emulation socket) and able to reproduce the Internet links behavior with characteristics of delay and bandwidth for defined user. These customized sockets can operate on a single workstation, and used to test and debug an Internet application on a separated system.

Motivated work

Every time we try to except some desirable properties for network emulator: portability, ease of use, and configurability. They become really needed to study Internet applications by using the network emulator. The requirement of portability is particularly significant in considering that Internet applications are to be deployed on different hardware or software platforms. In case of using a specific operation system or hardware it could be particularly bothered.

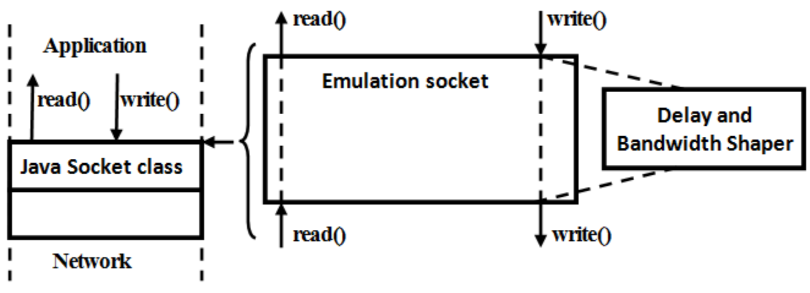

Fig. 1 – The emulation socket layer operates between the application level and the socket level

The emulator for network must be easy in use and highly configurable. In the purpose to observe the application's behavior, the programmer should be required to specify the parameters of communication only, and do not to deal with specific tools of operation system or details of network protocols.

Some network emulators and traffic shapers are described in several articles. The network simulator [1] has been rebuilt for enhancing of emulation facilities [2]. The real traffic can be injected into the simulator for studying of the experimental algorithms behavior during operating with real traffic or testing a protocol implementation with reproducible conditions. The network simulator is a powerful tool with providing of wide variety of protocols for wired and wireless networks. However, network simulator is more targeted to test and estimate protocols with real-traffic sources, but no applications. If there is a want to use network simulator to study the applications behavior in one network conditions, the phase of setup is quite complex. So, few machines may need to run the experiment, one machine is acting as the emulator and the others executing the nodes of application.

Running this experiments is some easier with a simple but effective network emulator [3] which allows to run experiments on a autonomous system. Here the communication between the protocol layers is intercepted to be analyzed. It emulates the real network presence by introducing queues with finite size, bandwidth limitation, delay of communication and loss links. Running on FreeBSD the implementation, operates at the interface between TCP and IP. It suffers from dependency of operating system. It also is not so easy to run experiments with user defined time and varying network conditions. Practically, each time it is needed to change the conditions of network and the system must be reconfigured by means of the command line or some shellscript.

The wide-area network emulation tool is described in [4] running on UNIX machines. A set of hosts are adopted by delay line and connected by a LAN, running a distributed application can appear as if connected through long-haul links with adjustable communication characteristics. But the tool is only able to introduce artificial delay in the communication between two or more processes and is impossible to control the bandwidth. Another network emulator is supporting applications using UDP, is described in [5]. It is intended to facilitate the adaptive application development of operating in wireless networks and subjected to wide network fluctuations.

To accomplish the requirements discussed above here is designed a portable, Java-based object oriented emulator, which can be simply extended by adding user-defined classes to the emulator package. The work has been influenced by both methods described above. The idea of inserting has been taken from these to configurate the layer that emulates the network. In the system the tools for the software stack are intercepted to communicate between the application and the socket APIs used to access the network. The reasonable for this choice is that in Java is possible to use only application level APIs for preserving programs portability.

Assumptions and functionality

A Java-based distributed application for Internet runs on top of a host set interconnected by long-haul links. Each host runs one or more JVMs that represent the application nodes. In accordance with these assumptions, the system is modeled as a directed graph made of nodes and links. They are defined as follows:

The *node* is as a JVM instance which is running a component of application and different JVMs running on the same host are considered as different nodes. The *link* for communication is as a both-ended uni-directional channel between nodes. This channel is intended at the level of *application* and everything under the socket APIs is considered as a black-box with given properties of bandwidth and delay. This presumption has some good impact on the design of our system. The link delay represents the latency from the time the sender node writes a data set; to the time those data become available for a read at the destination node. The single-directional links allow defining asymmetric inter-node communication behavior.

The aim of the tool is to support the execution of all application nodes on a single machine, or in a LAN, emulating the behavior of an Internet environment, where the emulated network configuration is defined through a directed graph like it was above. This functionality can be done by associating each link with a special customized socket able to shape the traffic over the link combined with user's requirements. The way these sockets manipulate the communication behavior saves up the blocking and/or non-blocking semantics of socket read and write operations.

Links can be parameterized in term of delay and bandwidth, in the following ways: *Fixed* way means that the delay and/or the bandwidth of the link does not change during application's lifetime; *Trace-driven* way gives the figures of delay and/or bandwidth which are expressed by a set of samples included in a file. Each sample is hold for numbered seconds given as specified in the system file of configuration. This option is particularly useful to feed the emulator with real network delay and bandwidth traces (collected as the `ping` utility).

User-defined way contains some pre-defined interfaces which allow us to build user-defined classes that form delay and/or bandwidth figures. Such classes implement a method that generates delay on the fly and values of bandwidth, respectively, which are used by the emulation socket associated to a link for manipulating its communication behavior according to the desired figure. So, developers can specify link parameters as random variables with customized mean value, disperse, etc.

Emulation Sockets

An Emulation Socket is acting as a filter. The application writes some data into the socket each time, and the data flow is aroused according to given bandwidth and delay characteristics. Fig. 1 shows emulation sockets inserting between the application layer and the java.net.Socket system class. Each method call to write() is intercepted by the emulation socket layer and forwarded to the network according to a shaping policy defined in an external module. Shaping is performed only on the stream of the outgoing data, and no operation is performed on the incoming stream of the data. This means that dual directional communication is formed always on the side of the sender.

Each emulation socket is characterized by D and B parameters, representing the delay and the bandwidth associated to the link, respectively. D and B are functions depended of time.

To support traffic shaping, each emulation sockets includes the following structures of the data and threads (fig. 2). *PendingData* is a data structure used to emulate the delay, where data written by the application are stored. To preserve sequential ordering of socket-based streams, PendingData is a circular array managed with FIFO ordering. SendingData is a structure of the data used to emulate the bandwidth. Mover is a thread which extracts data contained in Pending Data at the appropriate time, as described below, and inserts them into Sending Data. Sender is a thread which extracts data, with FIFO order, contained in Sending Data and inserts them into the network through the plain socket. Data are extracted at a B rate.

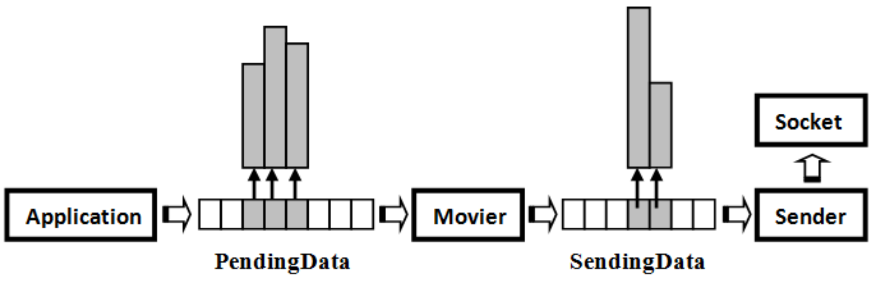

Fig. 2 – Emulation socket structure

The emulation socket behaves as follows:

– If an application, at time t1, writes some bytes into an emulation socket, then data are inserted in Pending Data together with their extraction time (t1 + D), so data remains in Pending Data for a time not lower than D.

– The Mover is thread which sleeps until the next extraction time (t1+D). It moves *all* data having an extraction time lower than t1+D from Pending Data to Sending Data. To get some clearance in how the data order is preserved, lets suppose that another write-operation is performed at time t2 > t1. This may happen when data written at time t2 are subject to a lower delay, and as a consequence they may have an extraction time lower than t1 + D. However, since streams associated to sockets have FIFO semantics, data written at time t2 may be delivered to the application only after all previous data has been delivered, that is at t1 + D time.

– The Sender thread wakes up and in Every T seconds extracts from Sending Data an amount of bytes A = B · T (the bandwidth accumulated during the time of sleeping) and inserts them into the network.

The emulation socket class is defined as a subclass of the java.net.Socket system class. Since it inherits all methods and fields from the Socket class, it can replace plain sockets in an application with little changes. For RMI-based applications, the developer can use Java socket factories to develop the instrument code [6][7]. Socket factories allow a client-server application to communicate by using customized sockets. When the application creates a socket, it actually delegates the creation to the installed socket factory. The factory returns an instance of socket which can implement a behavior that is different from the standard one, while preserving the same interface towards the application. For example, the socket factory can return sockets for data compressing or encrypting, or using a protocol different from TCP.

Using emulation socket with RMI-based applications is straightforward. Instead of using the standard constructor of a remote object, the programmer has to create the object by means of an alternative constructor which lets her specify the socket factories provided by developed package: EmuSocketClientFactory and EmulationServer-SocketFactory. The first of them is used by the client code to establish a server connection; the latter is used by the server code to accept incoming connections. This procedure requires the availability of source code.

A more combined instrumentation is required by those applications that directly use sockets. Each creation of socket must be replaced by the creation of an emulation socket class instance. Now we are investigating on how to use the SocketImplFactory pattern. This would require a couple of code lines to be added to the application and leaving unchanged the rest of the code. To deal with the problem of source code unavailability, we developed another version of the emulation socket package which exchanges the java.net.Socket class of the Java Runtime Environment with the shaping version. For the purpose of executing the application, the JVM must be launched with the -Xbootclasspath option. This way the JVM settings for the system classes' location are overridden.

```
#Node section
node node0 127.0.0.1 ports from 1000 to 1999
node node1 127.0.0.1 ports from 2000 to 2999
node node2 127.0.0.1 ports from 3000 to 3999

#Link section
link from node0 to node1 bandwidth 1000B/s
link from node0 to node2 delay 200ms bandwidth
     1000B/s
link from node1 to node0 bandwidth class:
     MyBandwidthShaper
link from node1 to node2 delay file:
     node1-node2-delay.data seconds 1
link from node2 to node0 delay 150ms
link from node2 to node1 delay file:
     node2-node1-delay.data seconds 1
```

Fig. 3 – A network configuration file example

Network configuration is collected in a file shared by all JVMs. Syntax of the network configuration file is straight, as shown in fig. 3. The file of configuration is composed of a node section and a link section. The node section is used to associate an IP address and a range of port numbers to each node. The reason is that, when the application creates a new socket, the emulator must identify the target node, return an instance of emulation socket which has the specified bandwidth and delay properties. For example, when running all JVMs on the same workstation, all nodes share the same address. But since they use different port ranges, the emulator can still identify the correct node. When the application is started, each JVM must be launched setting a special property (node-Name) to indicate to the JVM the node name which is associated with.

Fig. 3 shows the three nodes application configuration. The three JVMs must be launched on the same host, with the nodeName proper to node0, node1, and node2, respectively. The first one will use ports numbered from 1000 to 1999, the second one will use port numbers from 2000 to 2999, and so on. The link section describes delay and bandwidth characteristics of two-end links, e.g. link from node0 to node2 is characterized by constant delay and bandwidth respectively equals to 200 ms and 1000 B/s; the bandwidth link values from node1 to node0 are generated by MyBandwidthShaper class; communication delay from node1 to node2 is conveyed by samples contained in a file, and each sample must be in 1 second hold.

When two Java programs, a client and a server, communicate by means of emulation sockets, the following is happening:

1. The Application of the server creates an EmulationServer-Socket instance and it is waiting on a appointed port.
2. To create a server connection, the client should create a socket, passing as constructor parameters of the server hostname and the number of the port. The emulation socket verifies the constructor parameters to determine the target node and understand from the network configuration which are the bandwidth and delay that must be associated with the communication between the client to the server.
3. The EmulationServer-Socket determines the node identity to request the connection (using the hostname and the number of the port for the client socket), and returns to the server process an emulation socket instance which shapes the traffic passing from the server to the client.
4. From this moment, the both programs can exchange information by invoking standard methods of write() and read().

Performance

Now a partial evaluation of the emulator is present. The experiments were made on Athlon 1600+PC running JDK1.4 over Linux. The tests were performed with an improved resolution of time, obtained by re-building the Linux kernel in order to set the operating system clock resolution to 1 ms (default time is 10 ms).

There is tested the emulation accuracy over a range of different bandwidth and delay values. The scalability of system has also been investigated by repeating the experiments with a node number increasing. The tests concerning bandwidth and delay were executed independently. While testing bandwidth, delay was set to 0, and while testing delay, the bandwidth was set to very high values.

To test the emulation accuracy of delay, there was been developed a simple client-server application, where one client sends 1-byte messages repeatedly to the server. The emulator was configured to introduce a communication delay going from the client to the server, and both of them were measured the elapsed time between sending the message and receiving it. To stress the system scalability, there was repeated a similar test with 20 clients and 1 server. Fig. 4 shows the measured delay in comparison with the configured delay for values going from 0 to 500 ms. The tests for bandwidth were carried out accuracy by making the client communicate with large amounts of bytes to the server. The emulator was configured to introduce a limitation of the bandwidth in the client to the server communication, and both of them in the process of communication measured the observed bandwidth. In this case also the test was repeated with 20 clients and 1 server to evaluate the scalability of system. Fig. 5 shows the observed bandwidth in comparison with the configured one for values increasing from 64KB=s to 512KB=s.

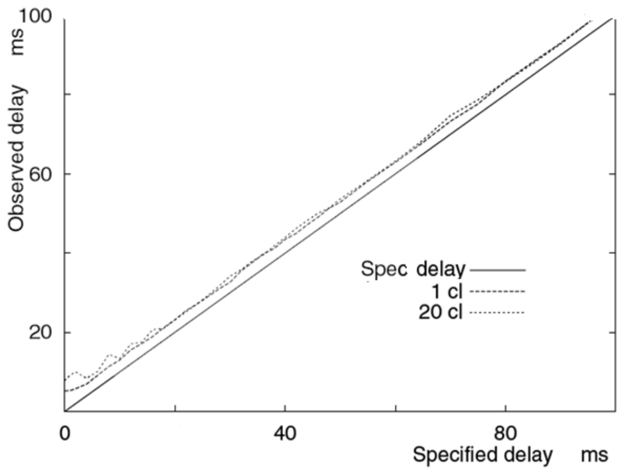

Fig. 4 – The accuracy of delay emulation

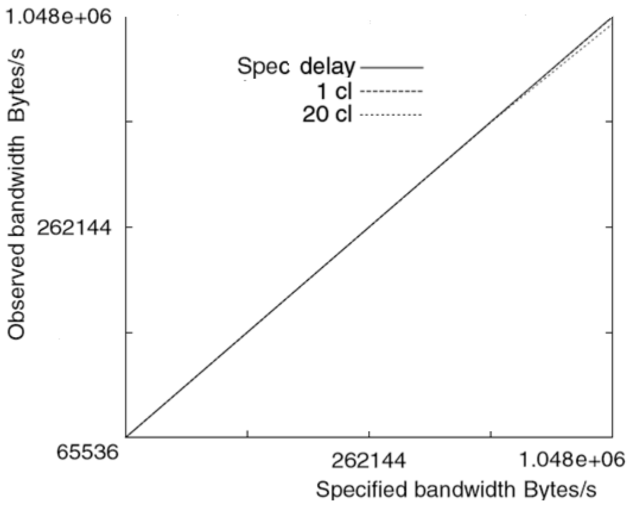

Fig. 5 – Bandwidth emulation accuracy

Discussion

Emulation sockets exhibit a good behavior for both bandwidth and emulation of delay. The observed values of bandwidth are really near the configured values, and even when 20 clients run together the system scales good. The figure of bandwidth starts scattering from the ideal value to values closing to 1MB=s. That is an unrealistic value if it is referred to Internet applications. In agreement with the delay, the observed values are slightly higher than the configured value. For almost of the values this difference is nearly constant, so that it can be removed by subtracting the measured value from the configured one. When the configured delay approximates to zero, system performances start to decrease because of the overhead introduced by emulation sockets. In this case the performances of the system remain good, as with Internet applications delay values near zero are not significant.

Performances of Java-based programs are lower than compiled languages, because of interpretation overhead. For this reason, it can be expected that scalability, overhead and time resolution achievable with emulation sockets will be worse than those achievable. However, Java offers byte code portability across different operating systems and hardware platforms. Emulation sockets are Java-implemented and they can be used as an emulation facility without being constrained to use a particular operation system or hardware.

On the other hand it is considered, that our solution provides a good tradeoff between functionality and performance. Emulation sockets are easy to use and allow great customizability which are unavailable with other tools. When limitations on time resolution are not critical when emulating the Internet setting, the very small values of delay are unrealistic.

References

[1] The ns simulator. http://www.isi.edu/nsnam/ns.
[2] K. Fall. Network emulation in the vint/ns simulator. In *Proceedings of IEEE International Symposium on Computers and Communications*, 1999.
[3] L. Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *Computer Communication Review*, Jan. 27, 1997.
[4] D.B. Ingham and G.D. Parrington. Delayline: a wide-area network emulation tool. *USENIX Computing Systems*, 7 (3): 313 – 332, 1994.
[5] N. Davies, G.S. Blair, K. Cheverst, and A. Friday. A network emulator to support the development of adaptive applications. *In Proceedings of 2-nd USENIX Symposium on Mobile and Location Independent Computing*, A. Arbor, U.S.A, 1995.
[6] Sun Microsystems. *Creating a Custom RMI Socket Factory*. http://java.sun.com/ j2se/1.3/docs/ guide/rmi/ rmisocketfactory.doc.html.
[7] W. Grosso. *Java RMI*. O'Reilly, 2001.