Limitations of Network Emulation with Single-Machine and Distributed ns-3

Alberto Alvarez, Rafael Orea, Sergio Cabrero, Xabiel G. Pañeda, Roberto García, David Melendi
Informatics department, University of Oviedo
Campus de Viesques, Gijón, Spain
{alvarezgalberto, orearafael, cabrerosergio, xabiel, garciaroberto, melendi}@uniovi.es

ABSTRACT

Research on large-scale internet services requires an extensive evaluation prior to deployment. A good analysis must include tests over large networks, using real devices and a considerable number of users. However, how to test in these scenarios with many users is an open question. Network emulation can be a good alternative before real deployments, which are complex and expensive. In this paper, we examine the new ns-3 network simulator/emulator in order to determine its capacity in the evaluation of large scale services. For that purpose, a real client/server video service is deployed over an emulated network. The service is progressively scaled up by increasing the number of clients on a single machine. In addition, we have extended ns-3 to support a distributed architecture for network nodes, thus, we repeat the experiments with a distributed set-up. Advantages, disadvantages, possibilities and limitations of both approaches are thoroughly discussed.

Categories and Subject Descriptors

C.2.1 [Network Architecture and Design].

General Terms

Design, Experimentation.

Keywords

Emulation, distributed, network-simulator, ns-3.

1. INTRODUCTION

The number of Internet connections has grown rapidly in recent years. As a consequence, new services oriented to a large number of users have been created. File sharing or video distribution using P2P or other technologies are just two examples. These services are conceived to reach a significant amount of users and to achieve a high degree of availability. Therefore, scalability must be a key aspect in their design. In other words, services must support accesses from a high number of users in order to be successful. Furthermore, it is important to test the maximum stress supported by a system's architecture prior to deployment in the real world; otherwise there is a risk of losing potential clients due to malfunctions. This evaluation may include the testing of the applications, the network architecture or the protocols to maintain the service up and running.

The performance evaluation of new services can be carried out using different approaches. One alternative is the simulation of the whole system with models of the network, equipment and applications. Although it is a scalable method of evaluation, it depends on the accuracy of the models and may require reimplementation of some parts before the real world deployment. Besides, it is not possible to test with real users, because real applications are not used and normally it is not performed in real time. In contrast, test-beds use real equipment and applications to reproduce reality. Hence, they can be extremely costly because they may require complex network architectures. Finally, this may be a big problem if this equipment will not be used for real world deployment, because the goal may be just development of the services, but not its deployment. Emulation is in between these two options. A common emulation method is the combination of real applications, or devices with a virtualized network topology. In addition, if they are real time, human users can be introduced in the equation. As with simulations, emulations still rely on network model accuracy, but the investment in network devices can be saved. Being cheaper than the test-beds and closer to the real world than simulations, emulations present a solid alternative in the evaluation of large-scale services. Nevertheless, the selection of the right emulation platform and the configuration of the surrounding system present difficulties. The emulation framework must be thoroughly studied, in order to detect limitations and obtain precise service performance evaluations.

Nowadays, ns-2 is probably the most extended application for network simulation. In addition, it is also used for emulation thanks to the extension in [7]. Its natural successor, ns-3, was released recently and promises to outperform ns-2 in many aspects. Ns-3 natively supports emulation using a real time scheduler for simulation events and virtual network devices, taps. Taps can be associated with both applications and nodes in a virtual network. Then, applications can exchange traffic through an emulated topology. For example, two processes running on the same computer can communicate as if they are connected by a complex network, but without the need to deploy it. For the evaluation of services with more users, more applications can be introduced and the complexity of the network can be increased. However, this process is not straightforward, because there are constraints imposed by hardware and ns-3. On the one hand, the identification of these limits will help us to build adequate frameworks for our emulations. On the other hand, it will establish boundaries to the validity of our experiment results; in other words, when results are due to real behavior of the analyzed service or a malfunction of the platform.

An "a priori" analysis of the limitations of network emulation reveals some constraints. First, emulations carried out on a single machine have limited system resources. Some of them are consumed by the emulator, and some are available to introduce real applications or virtual machines in the virtual network. The resource consumption and the number of these applications are a constraint to perform large experiments. The distribution of applications in different computers partially solves this problem, as [7] authors did for ns-2. However, new pitfalls are generated. All traffic exchanged by the applications must pass through the emulator, which introduces an extra delay. Moreover, the amount of traffic managed by the emulation is limited by the physical connection of the computers. For example, the emulation of a gigabit Ethernet may be impossible if they are connected in a 10/100 Ethernet. In conclusion, distributed environments may have more possibilities, but there are still limits. Of course, powerful hardware and high capacity links are a quick solution, but they are normally too expensive. Thus, it is important to study different emulation environments in order to find suitable evaluation frameworks.

This paper studies the effect on service emulation with ns-3 when the number of real applications is increased gradually. We have chosen video distribution over a local area network as a simple, but representative, case study. We perform ns-3 emulations on a single-machine and on a distributed environment. For this purpose, we have developed a distributed client extension. Then, resource consumption and service behavior are thoroughly examined in order to find limitations and accuracy of the results.

The remainder of the paper is structured as follows. Section 2 analyses related works. Goals of this work are set out in Section 3. The distributed emulation extension for ns-3 that we have developed is described in Section 4. Section 5 discusses the performance evaluation and explains our case study. Finally, Section 6 is dedicated to conclusions and future work.

2. RELATED WORK

Emulators are less popular than simulators in network research. Different works [2] focus on listing the most popular emulators, describing their particularities. Despite its native emulation support, ns-3 is not yet included among those classifications. Due to its novelty, there are few papers that study ns-3 or use it in their evaluations, either in emulation or simulation. There are works that deal with ns-3 features and goals or its development roadmap [4]. Some of them help to increase simulator models for experimentation, such as [1] where the implementation of a new model for WiMAX is described. However, few actually make active use of the simulator. [11] implements MANET routing protocols for ns-3 simulation, including up to 90 nodes in their experiments. On emulation, [12] compares a real deployment with an ns-3 experiment, using up to 35 nodes. The experiments in these publications give some clues about the performance of ns-3, although it is a collateral result. To our knowledge, there are no publications that specifically focus on ns-3 constraints.

The research of these issues is more abundant on ns-2. Interesting for our work here is the emulation extensions proposed in [8] and [7]. The former studies how to improve exactness in ns-2 real time emulations focusing on wireless models. The latter proposes the distribution of applications over different computers, alternatively to single-machine emulations, although distributed extension detailed in this solution has several drawbacks. Following the instructions described in [9], we tested some scenarios. We found that only homogeneous topologies were supported. Due to the specific addressing scheme in ns-2 and address manipulating in Magdeburg's model, hierarchical addressing is not allowed, which, on the other hand, is required for mixed topologies such as wired-to-wireless scenarios. The authors of [6] evaluate the performance of these proposals, also finding several additional limitations. They carry out emulations with up to 250 paths between a pair of nodes and focus their evaluation on the distribution layer, rather than on the emulator itself. They show that throughput bottlenecks, packet drops and RTT delays are mainly related with CPU overhead in the simulator and UDP tunnels implementation. The results shown have been a good starting point for our work with ns-3.

Other interesting proposals try to overcome common emulator pitfalls. One of them is the usage of virtual time, instead of real time. Basically, the machine clock is slowed down for applications, so they work slower than in real time. Therefore, emulators can cope with more events using the same resources. Although these can be feasible solutions for some cases, it is not possible to mix them with human users or real devices, which can not be slowed down. Authors in [18] implement synchronization to virtual time mechanism for OMNET++ and they are currently working on exporting their prototype for ns-3. Another feasible method of increasing emulator capacity relies on parallelization techniques coupled with distributed systems. [15] explains this issue. Basically, complex networks are partitioned into simpler subnets and ghost nodes are placed in representation of missing subnets. Authors claim that protocols like these are easy to develop and could be straightforwardly included into any emulator. According to ns-3 documentation, there are proposals to include these features in future ns-3 releases. Other works such as [3] mix both concepts, space parallelization and time virtualization, which could help to reach higher limits for a given emulator.

3. GOALS

This paper goes one step further than known related works. There is an existent increasing interest in network emulation for service evaluation. However, nobody has thoroughly studied ns-3 limitations on emulation. Thus, we have set several goals that will contribute to the state of the art of the simulation/emulation subject. First, this document contains valuable information for those with the task of evaluating new services through emulation. Specifically, we take the perspective of services that expect a significant amount of clients, due to the current interest in them and their high resources demand. Therefore, researchers or service designers may use the contributions of this paper for their performance evaluations.

Different setups of ns-3 have been tested and analyzed. Inspired in the work of [7] for ns-2, we have developed a *distributed clients extension* for ns-3. It provides the possibility of spreading applications inside the emulation in different computers. This approach has some advantages, because we are not restricted to using a single-machine for the emulation. However, it may have some drawbacks that are also analyzed. Finally, note that it is not a goal of this paper to judge the validity of ns-3 as a tool. Nevertheless, the limitations found may be used by researchers and developers to improve the overall performance of emulation platforms.

4. DISTRIBUTED CLIENTS EXTENSION

The aim of this section is to provide detailed description of the architecture developed in order to allow distributed emulation. The core of this approach is called the *distributed clients* *extension*, where client refers to any real application that is introduced in the emulation. Using this new model, we are able to create ns-3 scenarios with nodes outside the emulator machine, thus, lightening its workload. Because our goal is to provide an extension independent to ns-3, we use UDP tunnels to transport Ethernet frames between the emulator and each application. Therefore, the distribution is transparent to the emulator. Such an environment may be hard to configure. Moreover, network scenarios may vary from simple topologies to complicated designs. Thus, configuration should be automated as much as possible, but still highly flexible.

Next, we describe the design of the extension, some limitations and its configuration process.

4.1 Design and implementation

Ns-3 emulation features offer two different emulation classes, TapBridge and EmulatedNetDevice. The latter is basically intended to work with simulated objects through real test-beds, so the former suits our goal better. TapBridge operation involves several modes. The most relevant here are BridgedDevice mode and LocalDevice mode. For particular details, we refer to [13]. On the one hand, LocalDevice mode allows ns-3 to create virtual interfaces where local processes can be attached. Configuration of interfaces is entirely governed by the emulator itself. On the other hand, BridgedDevice mode uses existing virtual devices, such as Linux bridges and ns-3 connects to them. For example, this mode is specially indicated when using virtual machines, which may want to control the interfaces themselves. This mode is also convenient for our purposes, because we can take control of the interfaces to create our UDP tunnels. The target is therefore to extend ns-3 BridgedDevice scheme to disaggregate virtualized hosts to remote machines as shown in Figure 1.

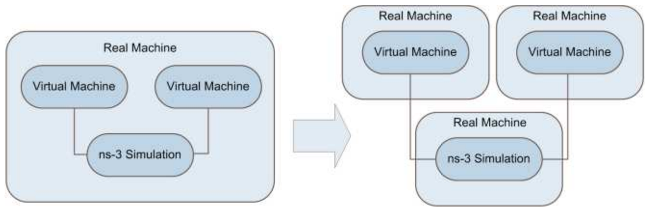

Figure 1. Overview of ns-3 emulation model evolution

In BridgedDevice mode, the ns-3 TapBridge object is linked to an existing host bridge composed of two virtual network interfaces. Virtualized clients are linked to one end of the bridge, while ns-3 ghost node, the one in the virtual topology, is connected at the other end. We intend to export the virtualized client to remote hosts, but we still need to guarantee direct talk between ends. To ensure transparent communications, aware of the client's configuration through lab's real network, traffic generated by virtualized clients is captured in the client's virtual network interface and encapsulated in UDP datagrams. UDP traffic is sent over real network towards emulator host. At their destination, frames are de-encapsulated to be finally delivered to the proper bridge structure connected to the ghost node in the emulator. One instance of our *taptunnel* process, running over each virtual interface, is responsible for this behavior. The concept of this design is inspired by Mahrenholz and Ivanov's work in [8] and has been tuned to match current ns-3 requirements. Unlike the original, it uses single UDP streams for each virtualized client and does not interfere with address translation. In the former work address mapping was mandatory due to the ns-2 addressing scheme, however, the ns-3 addressing scheme is fully ipv4 compliant and therefore, the original source's addresses are now perfectly suitable. Besides, in contrast with the work of Mahrenholz et al., we use identical instances of the tunnel in both ends of the path, not investing efforts in modifications of ns-3 modules source code, which may lead to a lack of compatibility with new or modified versions of ns-3. Therefore, this simplified UDP tunnel module provides us with a clean starting point to which enhancements can be added as long as problems are identified. The extension scheme is illustrated in Figure 2. In such a framework, clients deployed on remote hosts do not notice underlying processes; they act as normal clients using real ipv4 addresses.

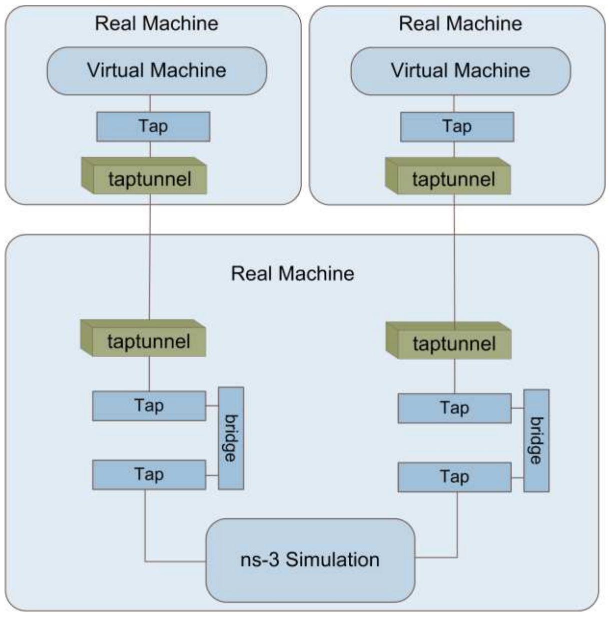

Figure 2. Details on the emulation model extension.

In the physical connection of equipment, each computer hosting client applications should now be locally connected to the emulator machine, preferably, using a network switch. If the resource consumption of evaluated applications allows it, several virtual clients can be deployed on each host. Therefore, we can decrease the number of computers needed without limiting the number of clients in emulations. As this extension is intended to support wide and diverse scenarios, it is not specified what users can use to run their virtualized clients. Users may choose to run their applications directly over native operating systems or use a complete virtualized OS by means of common virtualization tools such as VMWare or User Mode Linux. If the client runs directly over native OS, their applications should be able to select a proper outgoing network interface, among all virtualized devices. This can be achieved by modifying sockets option SO_BINDTODEVICE in the client's applications. However, we might not be able to change the source code of some applications. In that case, we can bind the sockets using a library like *libsocktap*, developed for the NEMAN network emulator [14].

Provided that each remote machine may host multiple virtualized clients, it is necessary to provide rules to avoid direct communication between local interfaces. Here we decide to employ the *sendtoself* kernel patch developed by Ben Greear [16]. The idea is to return the output route via external interfaces, if a path between two local IP addresses is requested and they are configured on different interfaces with the recently created loop flag set to true. The patched kernel will send packets out of the host. Moreover, because a set of different network interfaces share the same host, kernel and routing tables, routes must be set considering source addresses. One rule should match for inbound traffic and select a proper target device based on the destination addresses. Another rule must drive traffic through the right gateway depending on the source address. Thus, policy routing [5] rules will be configured on the client hosts.

Even with policy routing configured and *sendtoself* flag active, there is still a problem when clients that belong to different IP networks are in the same host. System network procedures establish strict packet flow through the kernel routing tables, which means that the local table is consulted first. The local routing table is maintained by the kernel. One common use of this table is to keep an entry for each locally configured interface. When a packet is sent from one local interface to another, the gateway that was previously configured is ignored. As a result, packets reach emulated network asking ARP requests for foreign addresses that none of its network neighbors know. Therefore, it is necessary to bypass the local routing table. This table is normally not meant to be manipulated, so it is necessary to patch again kernel sources and establish higher priority value for the local table rule. By doing this, we are able to place policy based rules before local ones. Therefore, when outbound packets search for their route, they are able to find the expected gateway. On the other hand, inbound traffic must be redirected to the local routing table, again, by means of policy based rules.

4.2 Known issues

This section describes some known issues that may be interesting when using this extension.

4.2.1 Dynamic Routing protocols

Dynamic routing protocols are required in some scenarios, for example in mobile ad-hoc networks. In such scenarios, routes are constantly added and removed from the routing table, which may interfere with the routing policies established for our extension. Therefore, distributed clients and hosts can not share the same routing table. It is recommended to create multiple independent network stacks, one per client. After that, routing protocols should be executed in the virtualized independent environment. Virtualization can be achieved by using any means available, from complete virtual environments such as the already mentioned VMWare or User Mode Linux (UML), to simpler network stack virtualization tools like VirtNET.

4.2.2 Shared Channel

There are network topologies where several nodes share the communication channel, such as wireless networks or several computers connected to a hub. Due to the UDP tunneling mechanism implemented in this extension, an effect is produced which is worth mentioning. In the real world, when a node sends a message, all nodes in its range (broadcast domain) receive it, but the channel is occupied only once. However, in the emulation platform, if a node sends a packet, it is first sent to the emulator that forwards a copy to every node in the same broadcast domain. Because every Ethernet frame is encapsulated in a UDP datagram, one datagram is sent to each node. In other words, traffic generated by one node is multiplied by the number of neighbors. This collateral effect of our extension must be taken into account in the dimensioning of the emulation framework. Bandwidth of the network connecting emulator and host machines must be sufficient to support the traffic of the emulated network and the overhead produced. Finally, note that this issue could be solved checking every packet or frame and sending it only to its destination, similar to how it is done in [7]. However, we have not considered this, because it modifies the behavior of the real network. For example, a wireless node would not hear packets from its neighbors and would sense the channel as free.

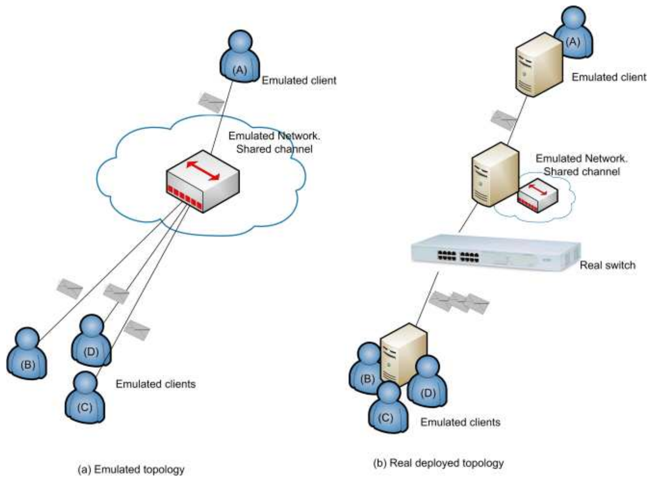

Figure 3. One message sent in the emulated (a) network implies several messages in the real network supporting the emulation (b)

4.3 Environment configuration

The extension configuration does not involve complex operations by researchers; however, some data is extremely delicate to slight mistakes. Each host implicated in a model should have, first of all, the right version of the kernel. For those machines hosting applications, the kernel has been customized as detailed in previous sections. Computer hosting ns-3 emulator can be running in standard kernel version. In order to create an emulated environment several steps must be followed. First, XML files contain configuration details of both hosts and topology. These files can be centrally stored in one computer and accessed by our configuration scripts. One configuration file per machine plus one global file that keeps track of the whole topology are required. The files contain information required to create and configure all virtual interfaces, bridges and routes that are needed for connecting all distributed clients to the emulator machine. Once the framework configuration is finished, standard ns-3 emulation scenario can be instantiated. Configuration files generation is currently done manually. However, a graphical tool for design framework architecture has been designed, as part of the future work. This application should ease the process by allowing users to describe virtual and real topologies. In other words, it should help in the allocation of hosts in the environment and of applications in hosts.

5. PERFORMANCE EVALUATION

In this section, we describe the performance evaluation of different ns-3 emulation setups. We have selected a simple case study that is described in the next subsection. This case study has been implemented on single-machine based emulations and distributed emulations. In subsequent subsections, results are presented and discussed.

5.1 Case study: video distribution service

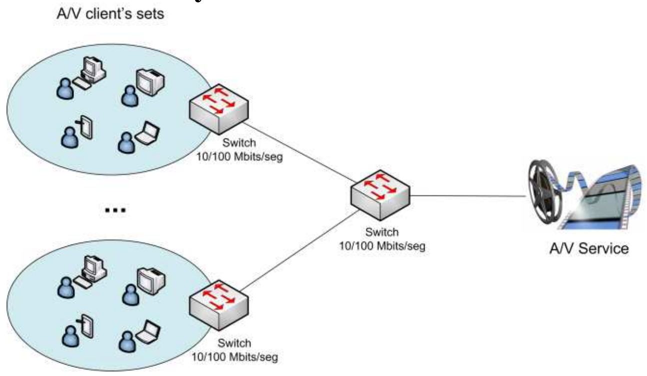

Figure 4. Case Study topology.

We have selected a simple case study to carry out the performance evaluation of different emulation setups. Our experiments reproduce audio/video (a/v) streaming over a local area network (LAN). Specifically, there is one a/v server, connected to a variable number of a/v clients through two levels of switches and all links are 10/100 Mbps Ethernet, see Figure 4. In ns-3 we model these links as CSMA channels with a delay of 16 ns and 100 Mbps throughput, which represents a 3 meter UTP 5e Category wire.

A strong reason for selecting this case study is that there can be real implementations of services similar to this one. For example, a business may wish to show training videos to its employees. If the number of employees is high, the service becomes large-scale, therefore it is important to know whether the network will support it or not. Hence, network managers may desire to emulate it before deployment. Apart from its feasibility, there are many other reasons to choose this scenario. First, a/v is a demanding service that consumes many network and system resources. Thus, it is faster to find limitations on the emulation framework than using other services. Second, it is feasible to deploy a real test-bed to compare real service behavior with emulations. In addition, resource consumption can be easily scaled up by increasing the number of clients. Provided that all clients consume similar resources, we can increase them gradually to find limitations. The scaling can be done in two ways, either by including more clients in each switch or by including more switches. Finally, the measurement of different parameters can be done easily, because of the clear topology of the network.

In this paper, this case study is emulated using two different approaches. First, a single-machine setup is used to carry out experiments with an increasing number of clients. Second, our distributed client extension is used to configure a new emulation platform. Again, the number of clients is increased gradually. It is expected that the former will outperform the latter, because resource consumption is shared among more computers. In both, resources are monitored and a/v transmission analyzed. The software used for the experiments is ns-3 (v3.5) as network emulator, openRTSP as a/v client and live555 as server. All three of them are open source and accessible to anyone who wants to reproduce the experiments. In addition, client and server work using the standard protocols RTSP/RTP, which facilitate service analysis. We have used two different sample videos, as a result of the codification of a single raw video source using MPEG-2 video codec and without audio stream. Hence, later analysis is simplified by the existence of one single stream. The video source sample is *highway_qcif.yuv* from a well known samples library [17]. The first video has a bit rate of 250Kbps and is referenced as *low quality* video hereinafter. The second video is a 1Mbps sample consequently referenced as *high quality* video. Both of them have a total length of 79 seconds. These videos are requested simultaneously by N clients in each experiment run. In order to avoid resource consumption from video displaying, video frames are dropped by the client. Therefore, we can increase the number of clients and so the traffic to stress the emulator. Finally, every experiment is repeated 3 times and the duration of each one is 120 seconds.

5.2 Single-machine

Single-machine environment supports all applications: emulator, clients and servers in the same machine. This is a Dell PowerEdge 860 with an Intel Xeon Processor Dual Core 2.40 Ghz, 1 GB of DDR2 RAM and 2 Gigabit Ethernet network interfaces. The operating system is Ubuntu Server 8.04 (32 bits). The process for these experiments is the following. First, the emulator is launched and the virtual tap interfaces created. Then, client processes are attached to their corresponding interface, one tap each. On the tap that represents the server machine in the topology; one server process is created for each client. We use one server process for each client, instead of one for all, to avoid bottlenecks produced by the server. Then, clients and servers connect through the emulated network and, hopefully, stream the video. After 120 seconds, the emulation is finalized and all processes killed. The full process is monitored using network sniffers (tcpdump) and resource analyzers (sar, pidstat). Figure 5 represents the connection of real applications through the emulator.

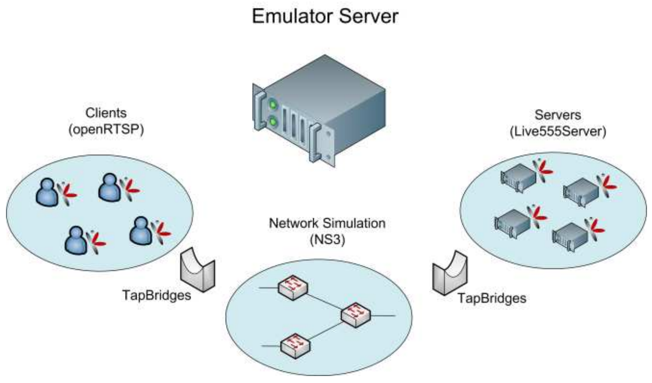

Figure 5. Single-machine testbed

5.3 Distributed ns-3

This environment spreads client and server applications in different machines. For that purpose, we use our *distributed clients extension* to connect the emulator with the server and client processes running in different computers. Ns-3 is hosted in the same computer used for single-machine emulations. Clients run over Pentium III computers with 512 MB of DIMM RAM and an Ethernet network interface. Their operating system is Ubuntu Server 8.04 (32bits). A/V servers are also in a different computer. This computer has the same characteristics as the Dell PowerEdge used to run ns-3. Figure 6a shows how the environment is configured physically. Computers for clients, servers and emulator are connected through a 10/100 network. Figure 6b illustrates the configuration from the point of view of the virtual network. Because resource consumption of the clients is not very high, one machine can host several processes. On the one hand, we introduce more clients with the same amount of computers. On the other hand, clients in the same computer share the same real network interface. For that reason, we have modeled the network interface of each computer as a switch. In conclusion, the topology emulated by ns-3 now is just a switch, because second level switches are the interfaces of the client hosting computers.

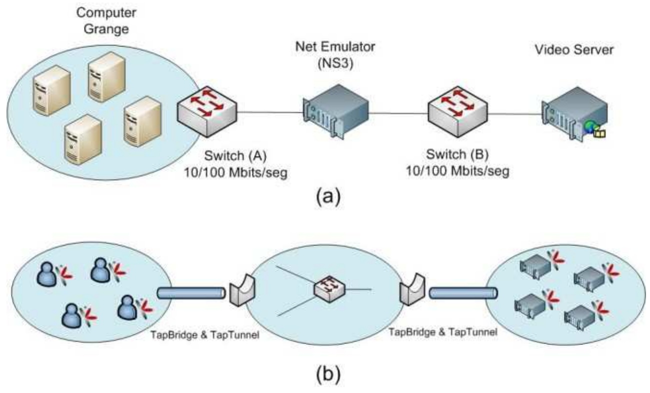

Figure 6. Distributed real (a) and emulated (b) configuration

Experiments on the distributed environment should be as similar as possible to the single-machine and real case. Therefore, machines have been synchronized to follow a similar starting sequence, although there are some differences. First, applications in different computers communicate with the emulator through a *taptunnel* process. In addition, resource monitors must be placed in different places. Nevertheless, results from both environments should be comparable.

5.4 Discussion

In this section, we compare results obtained with the single-machine (s-m) and the distributed (dist.) approaches with an increasing number of a/v clients in the scenario. Both video qualities examined, low and high, are considered. First, we identify common patterns detected on the experiments. For that purpose, we analyze a representative run. Then, we present an overview of the full set of experiments. We observe interesting effects as a result of the increase of the number of clients. For that purpose, we have chosen three main metrics. First, we analyze CPU consumption of both the ns-3 process and the whole machine emulator. In addition, network traffic generated by the server is measured. Finally, the service performance is evaluated using the jitter of RTP packets at one random client.

First of all, we have found two instances of undesirable behavior of ns-3 during our experiments. First, when the number of clients increases, it is more likely that a scenario will not run. Random errors start to occur with more than 30 clients. They become more frequent with the increase in number of clients until it is almost impossible to obtain valid runs. The maximum number is around 60. The explanation of this may be the management of resources carried out by the emulator. One thread is created for each TapBridge, which may be overloading the system. However, the errors are obtained while building the scenario, without introducing traffic. At that time, ns-3 activity should be very little. Thus, experiments are breaking down before needing any resources. Figure 7 shows a typical error extracted from an ns-3 scenario debug. Another interesting effect is that although ns-3 creates many threads, during our evaluation all of them use one of the two CPUs available in the computer. This is a clear impediment to achieving optimal performance. These factors prevent us from finding clear boundaries on the ns-3 performance, which could be solved with a more deterministic behavior. To get a better view of these pitfalls, other execution environments should be studied, for example, modifying available resources, ns-3 compilation or operating system configuration. Moreover, ns-3 is a novel product and improvements can be expected. Indeed, the ns-3 community is aware of the issue related to multithreading implementation and there is a public project addressing it [10].

```
Program received signal SIGSEGV, Segmentation
fault.
[Switching to Thread 1216416080 (LWP 31319)]
0x00002aae1ffd2491                            in
std::_Rb_tree_rebalance_for_erase ()
   from /usr/lib/libstdc++.so.6
```

Figure 7. Sample error shown by ns-3

In order to get a view of the experiment's behavior, we analyze a single run of a scenario with 20 clients requesting the high quality video (1 Mbps). Figure 8 shows the CPU utilization of the ns-3 process and Figure 9 shows the throughput of the video server, i.e. sent traffic. A clear relationship between CPU consumption and traffic managed is deduced. Furthermore, CPU utilization is lower in the distributed setup. This pattern has been found in all the other runs examined and will be discussed later on. Despite the difference in mean, evolution of the CPU is very similar and related to video encoding. However, there is a peak of traffic observed at the beginning of the experiment, around second 8. This is not an expected value in a real service and it is often present on single-machine realizations. The reason may be that the CPU utilization of ns-3 is close to 70% and that affects the ability of the server to send packets. Server process can not get CPU time to send its packets, so they are accumulated. Then, it has to do all the work at once. In conclusion, above a certain level of CPU, service metrics may not be fully reliable.

Figure 10 represents the average CPU utilization of the ns-3 process against the number of clients in the experiment. This average is calculated using the consumption along three runs of the same setup. Several comments can be noted. First, there seems to be an almost linear relationship between CPU consumption and the number of clients. This could be expected, because each new client increases the traffic that ns-3 must manage. Second, there is a significant reduction of ns-3 CPU utilization in the distributed setup, although the traffic managed for each scenario is almost the same, as shown in Figure 11. The reason behind this seems to be ns-3 management of tap interfaces. In single-machine scenarios, ns-3 takes care of the taps, which seems a high load for the process. In other words, it is more efficient when the traffic has to be eventually sent to a real network. Ns-3 is just linked to the virtual interfaces and not owning them, which is less resource consuming. For that reason, the maximum number of clients achieved by the distributed setup is slightly higher. In a single-machine, the mentioned errors were more frequent and it was not possible to obtain valid runs. In addition, we can observe a saturation of the server throughput when reaching 50 clients with both video qualities. Once again, this effect is not expected in a real service, which should increase the traffic proportionally. Although CPU is not overloaded in this case, this effect may be caused by thread context switching management. There are too many threads in the process, so the system expends more time switching threads than executing them.

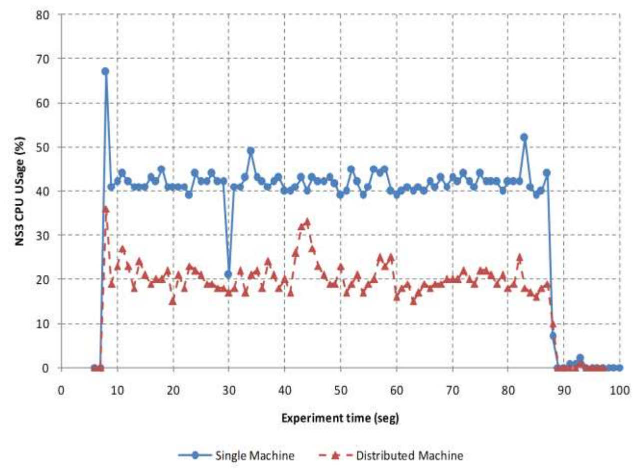

Figure 8. ns-3 CPU utilization: HQ video with 20 clients

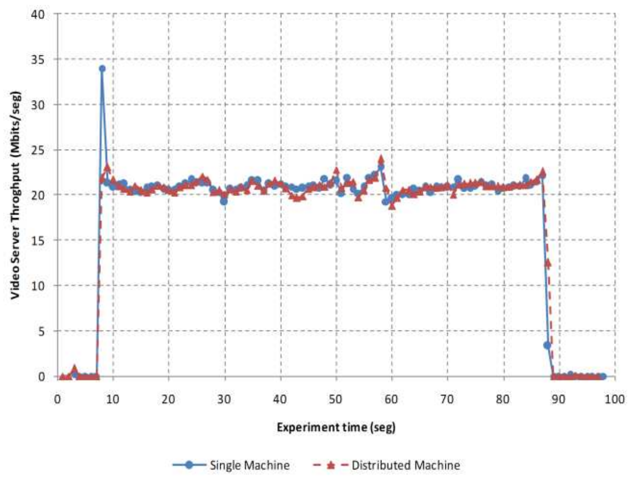

Figure 9. Server throughput: HQ video with 20 clients

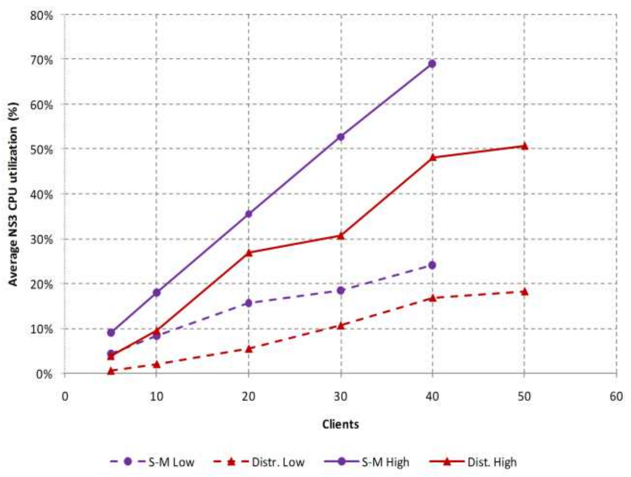

Figure 10. Average CPU utilizations of ns-3 process

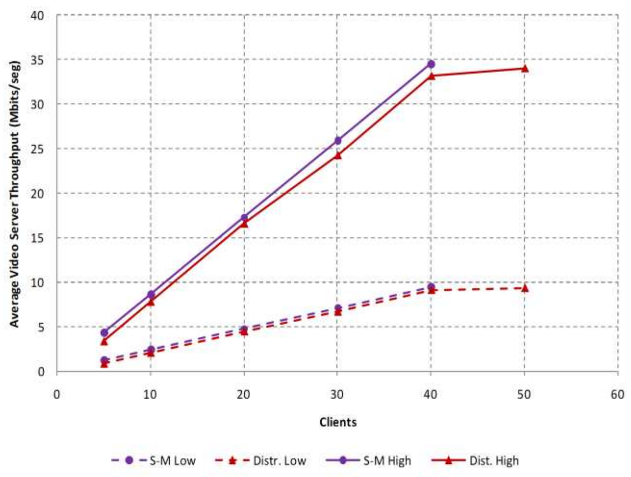

Figure 11. Average video server throughput

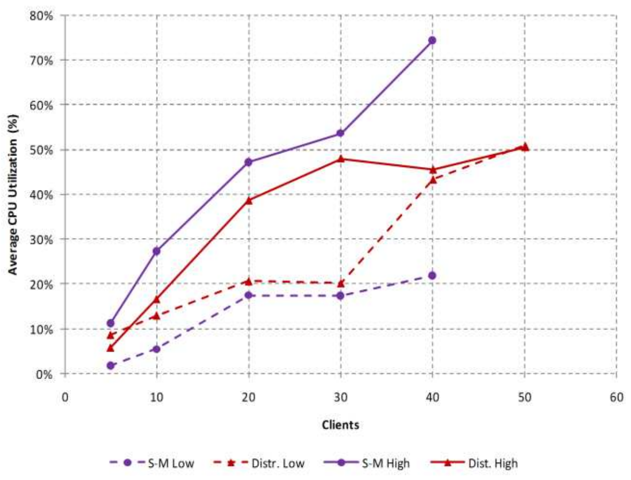

Figure 12. Average CPU utilizations of the computer

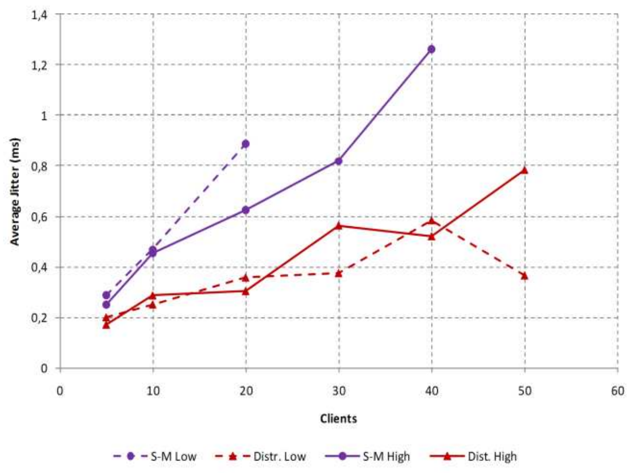

Figure 13. Average packets jitter in one a/v client

Figure 12 shows the CPU utilization in the whole machine. It is important information, because it reflects the consumption of a/v applications in the single-machine and the *taptunnel* processes in the distributed case. Selected a/v client and server are lightweight. In addition, video is not displayed or stored in the client, but dropped. On the contrary, *taptunnel* forwards a significant amount of traffic, which requires a significant CPU load. For the low quality video, the consumption of *taptunnels* keeps the total CPU utilization of the distributed case above the single-machine case. However, for the high quality video, the big difference in the consumption of ns-3 pays for the expected consumption of the *taptunnels*. Thus, if the amount of traffic in the network is kept low, a single-machine emulation may be better than a distributed emulation. Nevertheless, the whole situation would be reversed, if emulated applications were not lightweight. The emulation of heavy applications would show a bigger outperformance of the distributed model, at least in terms of total CPU consumption.

RTP packets jitter is a significant metric in the analysis of a/v services. For that reason, we have analyzed it in order to measure the influence that the emulation environment has on the service performance. Given a fixed number of clients, Figure 13 shows the average jitter in the multimedia sessions for one of them. This client is selected randomly at the beginning, but is always the same in the following experiments (e.g. client number 3). The first noticeable effect is that there are no results from some single machine video setups, which indicates that the client did not receive packets. Due to the overload suffered by the emulation machine, the a/v client was not able to take part in the multimedia session successfully. It is also worth pointing out that jitter is always lower in the distributed environment, independently of the video quality used or the number of clients. This would not be expected beforehand, because of the external network introduced. The distribution of applications into different machines connected by a network should introduce an extra, although small, delay. The jitter could also be affected, so worse results could be expected for the distributed case. However, the low CPU utilization in the distributed environment compensates the effect of the external network. In addition, jitter increases with the number of clients in both situations. Although more experiments are necessary to establish the exact progression, the general tendency indicates bigger increments in the single-machine setup. In conclusion, the higher the resource consumption of the machine hosting ns-3 is, the less accuracy is achieved in the results. Finally, provided that jitter in a real deployment of this service is expected to be very low, we can asseverate that results from the distributed environment are more accurate.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented an in-depth study of emulation alternatives with ns-3. Emulation of services has been identified as an interesting option for evaluation with other possibilities than test-beds or simulations. For example, real applications can be used so saving investment on network equipment as with test-beds. However, the configuration of an emulation environment is not a trivial question, especially if we aim to evaluate services with many users. For that purpose, we have analyzed the possibilities and limitations of the recently released simulator/emulator ns-3. To go beyond single-machine emulations, we have developed a *distributed clients extension,* which allows the distribution of real applications in different computers. This different setup presents certain advantages, such as less resource consumption by ns-3, but also some pitfalls, such as the introduction of processes external to emulation, *taptunnels*, or synchronization difficulties. Both alternatives have been studied by increasing the number of a/v clients streaming a video from a server while keeping a fixed emulated network, simple enough so as to avoid unexpected behaviors caused by larger emulated topologies. Furthermore, two video qualities were employed to see the possible implications.

As a general conclusion extracted from our evaluation, a distributed environment optimizes resource consumption. High resource consumption implies unexpected behavior of the emulator and less accuracy in the results. Thus, a distributed environment is a feasible possibility to emulate larger services and still maintain results accuracy. Although we have scaled the clients up to 50, it was not possible to go above this number. This was mainly due to resource consumption of ns-3. It is not clear whether this is a limitation by ns-3 itself or by the hardware platform. Future tests over a more powerful hardware are being programmed to examine more deeply this constraint. From the usage of two different video qualities, we can also comment that there is a maximum of 50 nodes. Thus, the limiting factor is not directly related with the amount of traffic, but with the number of nodes. This highlights an inefficient management of the TapBridge objects. For the emulation of large-scale services, a solution for these limitations must be proposed.

This paper is a first step in the state-of-the-art of ns-3 emulations. Although some interesting contributions have been exposed, more could be achieved with some future work. In order to obtain a complete view, we see three main lines: service, network and platform. First, other services could be evaluated, as their behavior may cause different performance of ns-3 and discover new constraints of the extension. Furthermore, more complex network topologies, which include larger number of emulated nodes and other technologies, could be analyzed. Our current and future work on this subject includes the distribution of ns-3 processes into different machines. Finally, the influence of the hardware used for the emulation could be better defined testing other architectures. Not only different machines for the emulator or the hosts, but also other network structures or technologies to connect them, for example, gigabit Ethernet.

7. ACKNOWLEDGMENTS

This work has been funded by the Spanish National Research Program (FUTURMEDIA Project TSI2007-60474).

8. REFERENCES

[1] Farooq, J. and Turletti T. 2009. Wimax Module for NS-3. ACM SIMUTools, 2nd Conference on Simulation Tools and Techniques.

[2] Göoktürk E. 2005. Emulating ad hoc networks: Differences from simulations and emulation specific problems. New Trends in Computer Networks Conference.

[3] Grau A., Maier S., Herrmann K., and Rothermel K. 2008. Time Jails: A Hybrid Approach to Scalable Network Emulation pads. 22nd Workshop on Principles of Advanced and Distributed Simulation.

[4] Henderson T. R., Lacage M., and Riley G. F. 2008. Network Simulations with the ns-3 Simulator. Demo paper at ACM

SIGCOMM. Association for Computing Machinery's Special Interest Group.

[5] Hubert B., Graf T., Maxwell G., Van Mook R., Van Oosterhout M., Schroeder P.B., Spaans J., and Larroy P. 2003. Linux Advanced Routing && Traffic Control How-to.

[6] Kristiansen S., and Plagemann T. 2009. ns-2 Distributed Clients Emulation: Accuracy and Scalability. SIMUTools 2009, 2nd Conference on Simulation Tools and Techniques

[7] Mahrenholz D., and Ivanov S. 2004. Real-time network emulation with ns-2. Distributed Simulation and Real-Time Applications, Eighth IEEE International Symposium on, pages 29–36.

[8] Mahrenholz D., and Ivanov S. 2006. Adjusting the ns-2 Emulation Mode to a Live Network. Kommunikation in Verteilten Systemen (KiVS).

[9] Mahrenholz D., and Ivanov S. 2004. How-to: Wireless Network Emulation using NS2 and Distributed Applications. University of Magdeburg, Germany.

[10] Multithreaded implementation for multicore . http://www.nsnam.org/wiki/index.php/Current_Development #Multi-threaded_simulation_implementation_for_multicore Last Visited January 21, 2010.

[11] Muthukumar S.C., Li X., Liu C., Kopenay J. B., Oprea M., and Loo B.T. 2009. Declarative Toolkit for Rapid Network Protocol Simulation and Experimentation. SIGCOMM, Association for Computing Machinery's Special Interest Group.

[12] Muthukumar S.C., Li X., Liu C., Kopenay J. B., Oprea M., Correa R., Loo B.T., Basu P. 2009. RapidMesh: Declarative Toolkit for Rapid Experimentation of Wireless Mesh Networks. 4th ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization (WiNTECH 2009), in conjunction with ACM MobiCom.

[13] NS-3. http://www.nsnam.org/, Last Visited January 21, 2010.

[14] PuZar M., Plagemann T. 2005. NEMAN: A Network Emulator for Mobile Ad-Hoc Networks. 8th Int. Conf. on Telecommunications.

[15] Riley G. F., Jaafar T.M., Fujimoto R.M., and Ammar M. H. 2004. Space-Parallel Network Simulations using Ghosts. Proceedings of the 18th workshop on Parallel and Distributed Simulation, IEEE.

[16] Sendtoself patch. http://www.ssi.bg/~ja/#loop. Last Visited January 21, 2010.

[17] Video traces: http://trace.eas.asu.edu/yuv/index.html. Last Visited January 21, 2010.

[18] Wingärtner W., Schmidt F., Heer T., and Wehrle K. 2008. Synchronized Network Emulation: Matching prototypes with complex simulations. Workshop session: The first workshop on hot topics in metrics.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8630
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8630