Design issues in building a scalable network simulation/emulation middleware

Zrinka Puljiz Miljenko Mikuc
University of Zagreb
Faculty of Electrical Engineering and Computing
Department of telecommunications
Email: zrinka.puljiz@fer.hr miljenko.mikuc@fer.hr

***Abstract*— Constant inventions in the field of distributed systems raise up the demands on network simulators. Resources needed for simulation of thousands to millions of nodes can be provided only by making simulators distributed. In the design process of distributed systems scalability and transparency present the key requirements. Some of the existing models for distributed systems can be used to model both of this needs, for example peer-to-peer architecture can model scalability, while middleware can model the transparency. By identifying key management issues in distributed simulation and by borrowing these concepts from distributed systems, we describe a new simulation layer that can be used by a variety of simulators.**

I. Introduction

Current trends in distributed applications lead to an increasingly scalable system with more complex structure. This brings up the problem of valid simulation. Peer-to-peer systems like [1] to be validly simulated require much more resources than a standard client-server application. As the resources available on only one machine are not enough, we seek the solution in distributed simulation. However, the chosen distributed simulator may not scale enough, or the researcher may not have enough machines that he could dedicate to running the simulation.

The available resources for network simulation increase proportionally with the number of machines in the best case scenario [2]. As resources owned by just one researcher may not be enough, joining the private machines of more researchers can increase the resources available in a more efficient manner. This was recognized by some of the existing large-scale simulators like Planetlab [3] and Emulab [4] where the users of the distributed system contribute to the whole system by dedicating machines and later on they can use the available resources for their experiment.

Further on, the scalability of the distributed system, as well as distributed network simulator, depends on the system architecture. Centralized solutions have a bottleneck. Decentralized solutions on the other hand are more loosely coupled, having more management issues, but at the same time they are easily extendable and much more scalable and in long run they tend to be self-organizing.

All network simulators face some common problems when they are distributed. These common problems include resource management, users management, experiment management and traffic management. The lessons learned from the distributed system evolution [5] can be applied to this specific case. Some of the existing scalable architectures such as peer-to-peer architecture can be used in the design of future network simulators. In addition, network simulators can benefit from a middleware layer that simplifies the view of the distributed system. The middleware layer can be used to simplify the development of distributed version of only one emulator, but it is fully used when more emulators can reside on top of it.

This paper is mainly focused on two areas. We discuss the problem of scalability of network simulators and as a solution we propose a decentralized distributed system. We also identify the common management issues in distributed network simulation and propose a solution in the form of middleware that would alow a rapid development of new network simulators.

This paper is organized as follows: In Section II we describe network simulators and classify them. Following, in Section III we compare and contrast centralized distributed systems to decentralized ones. Next, we describe middleware as an additional layer of abstraction in Section IV. Our idea of middleware solution to the common distributed network simulator tasks is presented in Section V. Finally, there is a conclusion in Section VI.

II. Network Simulation

Network simulation is a broad term and has more than one definition. We use the term of network simulation as a system that is capable of representing real world communication networks in terms of connectivity (topology) as well as communication (networking protocols and/or traffic).

Topology is a network model that consists of nodes and the description of how the nodes are connected. Depending on the type of the network simulator there can be only one type of nodes or several types of nodes, specialized for certain purposes. The connectivity between nodes is commonly described with links that connect one pair of nodes, which also have some specific properties.

Most network simulators offer specific network protocols as well as complete protocol suits to support the traffic generation and capturing. For instance, TCP/IP protocol suite can be found in most of the existing network simulators since it is the most widespread protocol suit in use today.

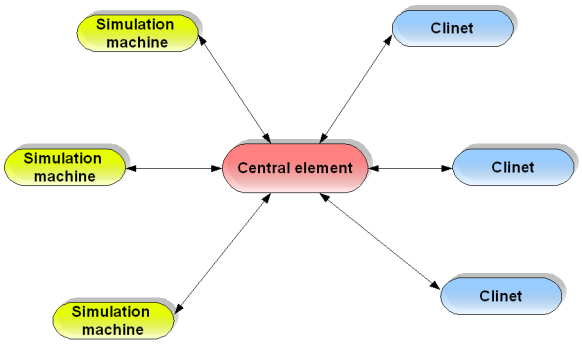

Fig. 1. Centralized distributed network simulator

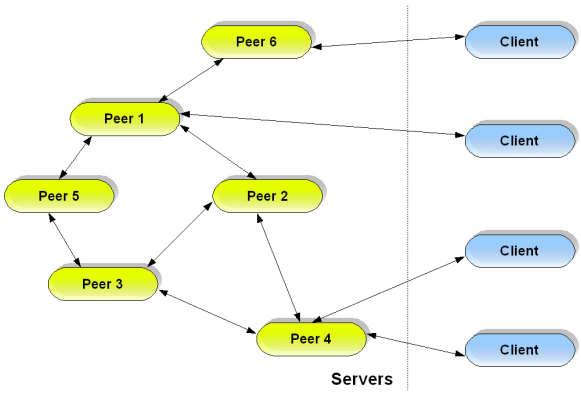

Fig. 2. Peer-to-peer servers architecture in decentralized network simulator

Further on, we distinguish between two kinds of network simulators, off-line simulators, or just simulators, and emulators. The key difference is in the notion of the time. Emulators use real time for processing while simulators use logical time.

Simulators use time stamps to provide correct sequence of events disregarding the actual time that needs to pass between two different events, while emulators use absolute or wall-clock time for executing the events in correct order and they do the processing in real-time. The simulation executed in off-line simulators can be either faster or slower than in real-time systems. In this category we can find network simulators such as ns-2 [6], PADS [7] and GloMoSim [8].

Emulators process events in real time and are also called real-time simulators. They are designed to highly resemble real systems, so they need more resources than off-line simulators. Usually they provide connectivity with real systems and can use unchanged applications in the simulation. In this category we can find emulators such as EMPOWER [9], Emulab [4], ModelNet [10] and Planetlab [3].

When comparing network simulators with so called testbeds, i.e. real networks used solely for experimentation in closed conditions, the benefits form the simulation are prevailing. These benefits are time and cost effective because they include easier experiment manipulation and data acquisition as well as less space and less equipment required.

As the network architecture evolves during the time, requirements are made of the network simulators to be more and more scalable, so even large networks can be simulated [2]. Furthermore, there is emergent focus on various kinds of distributed system models such as peer-to-peer systems or publish-subscribe models that are more complex and resource consuming.

III. Centralized vs. Decentralized

When developing a new distributed system of any kind, the first question we ask ourselves is how will we manage the system. There are two main solutions - systems can be centralized or decentralized. We analyze the existing network simulators, and based on the architecture used, we place them into a centralized or decentralized category. But we must be aware that some of the systems like ModelNet [10] or PADS [7] require a manual setup of the distribution model and do not fall in any of the following categories.

A. Centralized architecture in network simulators

Centralized architecture has a central element that presents the bottleneck of the system. However, centralized architectures are more widely used since they offer easier management and more controllable systems. The synchronization between elements is straight-forward, since there is one arbiter that has the view of the whole system and he makes all the decisions about the system. Centralized architecture is presented in figure 1.

Netbed [11] is an integrated environment for simulation and emulation. Real network elements are intermingled with the simulated ones, each in charge of a different network portion. In the core of Netbed is a cluster system allowing time-shared and space-shared experimentations. Netbed architecture [11] contains a central element called masterhost that takes care of the web interface as well as database and SNMP management. Netbed evolved into Emulab [4], still relying on the centralized architecture.

NTCUns simulator [12] is capable of simulating wired and wireless networks. It has an open architecture and is easily extendable. The architecture of NTCUns has a dispatcher as a centralized element. This dispatcher is the critical element of the system, and must remain alive all time to manage the running simulations. All the communication between users and simulation machines goes through the dispatcher.

B. Decentralized architectures in network simulation

Most of today's attention in the design on distributed systems is focused in the field of decentralized architectures. The scalability problem with centralized architectures puts the limit on the size of the system that can still function with benefits. In centralized architectures the central element always presents the bottleneck of the system. Decentralized systems try to work a way around this limit and have a scalability that largely extends the scalability of centralized architectures.

In [5] we find the description of a decentralized network emulator based on IMUNES [13]. This system provides an integrated environment for TCP/IP networks. The described

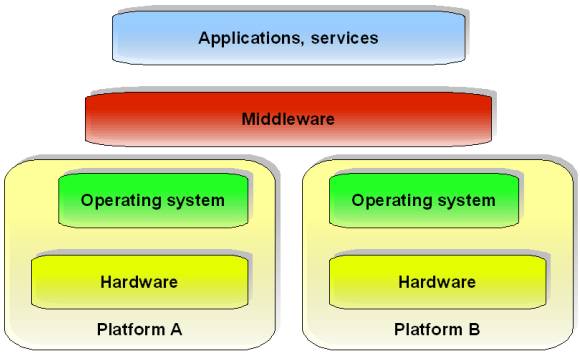

Fig. 3. Middleware

system, although it has no central element, is still not scalable enough because the architecture requires all the simulation machines to keep the information about all the experiments.

Multiple server architecture is known in the literature [14] as the hybrid model between client-server systems and peer-to-peer systems. One of the possible realizations of this architecture is when servers are connected in peer-to-peer manner. Peer-to-peer architecture of servers presented in Figure 2 shows one of the possible decentralized architectures. There is no central element, all of the servers are equal among themselves and there are ways of adding new peers or removing the existent ones without the need for a global lock of the system. The systems like chord [1] or pastry [15] present some of the implementations of peer-to-peer architectures. In those systems algorithms were developed for adding new and locating the existing information.

Decentralized systems offer a lot of benefits mainly in the area of building scalable and cost effective systems. However, they are also much more complicated to manage and in practice they are harder to design. Once a good decentralized system is designed, we want to reuse it as often as we can.

IV. Middleware

Middleware is a software layer used in distributed systems that masks the underlying differences between system parts making them transparent to the upper layer programmers. The abstraction level implemented in middleware allows the programmer to interact with the distributed system using tools like remote procedure call or remote event notification. These tools simplify management of the distributed system providing a uniform view of a heterogeneous system. The position of middleware in the context of layers is presented in figure 3.

Remote procedure call (RPC) is one of the oldest distributed system concepts. RPC function is to provide a transparency for the system by masking the fact that procedures called are implemented remotely. On the side of the client (calls the remote procedure) there must be a part implemented to recognize that the procedure is implemented remotely and on the part of the server there must be a way to get the arguments right from the client. By implementing support for RPC, we can use different programming languages on the client and the server, as long as the conversion takes place when passing the arguments as well as results.

Emergence of event-driven programming caused the development of new distributed event-notification systems. One of those is the publish-subscribe system. The publish-subscribe system provides the transparency by masking the fact that the event that is triggering an action has happened remotely. There is a side that publishes new information and all of the machines subscribe to get certain type of information. The publish-subscribe structure also allows constant inflow of the information into the system, and the propagating of new information to the interested sides.

A specific type of middleware created to transparently provide access to shared resources on a very large scale is called grid. A grid is a collection of loosely coupled machines that cooperate together without fully trusting each other.

V. Distributed network simulation middleware

Based on the experience gathered in the development of a distributed network simulator [5], we were able to identify key management problems in distributed simulation/emulation. We could deal with these problems one by one, providing an innovative solution to each one one of them, or having a system that does this annoying task for us. Having a middleware level of abstraction that would take care of all the questions concerning similar management problems, would speed up the process of distributing a network simulator. This middleware could also provide a common platform through which different network simulators could be able to communicate.

The management problems that we can identify through many different network simulators are:

- Resource management
- User management
- Experiment management
- Traffic management

Resource management provides for scalable and transparent usage of resources. Scalability means that new machines are easily added to the system and that addition of new machines will result in more resources available. Transparent usage of resources means that all of the resources of the machines in the system will be available in a uniform way. For meeting the scalability machines form a decentralized architecture. The architecture used takes care of addition of new machines that present new resources to the distributed system, as well as removal of machines from the system and failure management if one of the machines fails. To achieve the transparency we propose a naming scheme used for addressing the resources, i.e. machines, that is independent of their physical location. These issues are addressed in similar fashion in peer-to-peer systems, so we propose usage of peer-to-peer systems for resource management.

User management takes care of all the users that are available. This includes the tasks of authorization, authentication and propagation of rights of existing users. The authentication mechanism can be by login and password or better still with a key authentication. Propagation of rights starts as an event

triggered by the user that notifies the system that there is a user that needs more resources. If there is a machine willing to accept this user, the user rights are propagated to that machine and user gains access to more resources. Furthermore, the user management also takes care of new users, using the same kind of events to gather the resources for new user. The user information is kept in more than one location, more precisely it is kept on each machine the user has access to. After a user account is closed, another notification process takes place to remove all the user information and processes from the system. This event driven behavior resembles the publish-subscribe systems, and by using this existing model we can provide this functionality more easily.

Experiment management presents another important issue, since we have to track the experiments and to keep the information about them available through a simple, but efficient mechanism. Experiments need to have a unique identifier so that the information can be kept without interference between different experiments. Each experiment can belong to only one user. Creation of a new experiment is another event, and upon the creation of a new experiment all of the machines that provide access to the owner of the experiment should be notified. This information about new experiment is kept on all the machines that are available to the user that created this experiment. New experiment name is bound to the machine that he is communicating with instead of user that is creating the experiment; this is done in order to circumvent the racing conditions in case of a simultaneous creation of more than one experiment. Here again we can see that the publish-subscribe model can be used for experiment management as well.

In traffic management it is important to separate each traffic flow, one from another. This is required since there might be more than one experiment using the same addresses. We are modeling the system to prevent that kind of situations. For traffic management we must use some traffic separators. Usage of tunneling (either TCP or UDP) introduces additional delay in the system but at the same time it separates the simulated/emulated traffic from the real Internet traffic as well as separating one traffic flow from another. The format of the traffic is simulator specific. Using tunnels removes the need for global synchronization, and there are also no racing conditions to be met.

By identifying all of these management issues and putting them together in forming a unique middleware, we are creating a platform on which any kind of network simulator can be easily distributed and benefit from the services described here. The architecture of the proposed system is presented in Figure 4.

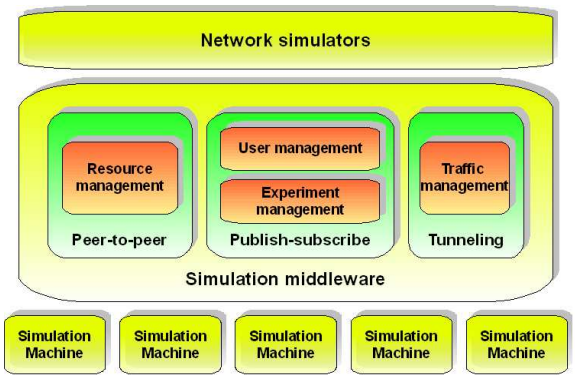

Fig. 4. Architecture of middleware for network simulation

VI. Conclusion

In this paper we describe the functions of resource, user, experiment and traffic management in distributed network simulation. We compare and contrast centralized architecture to the decentralized architecture of distributed systems. Based on the better scalability performance of decentralized architecture we describe the idea of creating a middleware system that would implement the common functionalities of network simulators and create a support for rapid development of distributed network simulators. The proposed middleware realization is based on peer-to-peer architecture and a distributed event notification functionality, or the publish-subscribe model.

To conclude, we present a way of creating new distributed network simulators in a time-effective as well as cost-effective manner.

References

[1] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proceedings of the ACM SIGCOMM '01 Conference*, San Diego, California, August 2001.

[2] R. M. Fujimoto, K. Perumalla, A. Park, H. Wu, M. H. Ammar, and G. F. Riley, "Large-scale network simulation: How big? how fast?" *mascots*, vol. 00, p. 116, 2003.

[3] M. Beck, T. Moore, and J. S. Plank, "An end-to-end approach to globally scalable programmable networking," in *FDNA '03: Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture*. New York, NY, USA: ACM Press, 2003, pp. 328–339.

[4] S. Guruprasad, R. Ricci, and J. Lepreau, "Integrated network experimentation using simulation and emulation," in *TRIDENTCOM '05: Proceedings of the First International Conference on Testbeds and Research Infrastructures for the DEvelopment of NeTworks and COMmunities (TRIDENTCOM'05)*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 204–212.

[5] Z. Puljiz and M. Mikuc, "Distributed network emulator based on imunes," *SoftCOM 2006*, 2006.

[6] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, and H. Yu, "Advances in network simulation," *Computer*, vol. 33, no. 5, pp. 59–67, 2000.

[7] S. Lee, J. Leaney, T. O'Neill, and M. Hunter, "Performance benchmark of a parallel and distributed network simulator," in *PADS '05: Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 101–108.

[8] L. Bajaj, M. Takai, R. Ahuja, R. Bagrodia, and M. Gerla, "Glomosim: A scalable network simulation environment," 1999.

[9] L. Ni and P. Zheng, "Empower: A network emulator for wireline and wireless networks," 2003.

[10] K. Yocum, K. Walsh, A. Vahdat, P. Mahadevan, D. Kostic, J. Chase, and D. Becker, "Scalability and accuracy in a large-scale network emulator," *SIGCOMM Comput. Commun. Rev.*, vol. 32, no. 3, pp. 28–28, 2002.

[11] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An integrated experimental environment for distributed systems and networks," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 255–270, 2002.