Policy-based traffic generation for IP-based networks

Falko Dressler
Autonomic Networking Group, Dept. of Computer Science 7
University of Erlangen-Nuremberg
Erlangen, Germany
dressler@informatik.uni-erlangen.de

***Abstract*—Many testbeds and research projects rely on the generation of artificial network traffic. Such solutions are used for protocol or architecture verification, performance tests, and demonstrations. Even though a huge amount of traffic generators have been developed in the past, many of them still suffer from similar problems, e.g. they are either designed for throughput tests or protocol tests. In this paper, we present npag, a new generation traffic generator for IP-based networks. It features a policy-based configuration and a modular design making it feasible to employ it in many application scenarios ahead of simple throughput tests. For example, it allows the generation of packets with arbitrary header and payload information and to measure detailed quality of service parameters.**

Keywords—Network Monitoring, Traffic Analysis, Statistical Evaluation, Network Security

I. INTRODUCTION

The analysis and test of mechanisms in communication networks is essential to develop and verify new protocols, or for performance analysis and quality of service measurements. Especially, if a testbed or experiments in real networks is not feasible, such as in case of denial of service attacks or other security concerns, mostly artificial network traffic is used that is generated by packet and traffic generators. The requirement for such approaches is, for example, discussed in [1]. Many tools are available today, e.g. the measurement tools ttcp and Iperf, the packet generator netcat, or the flow generator harpoon [3]. As shown in section III, all these tools are limited in the one or other direction. In this paper, we present npag (Network Packet Generator) and its distributed control system paco (Packet Coordinator). These tools have been developed having two application scenarios in mind: network security tests and protocol engineering. Special focus was laid on the distributed and coordinated functioning and the flexible configuration. Therefore, we developed a policy-based description system to describe packet information as well as traffic patterns. Both tools were made available as open source.

II. HISTORY PROJECT

The aim of the HISTORY (High Speed Network Monitoring and Analysis) project is to build an architecture, methods, and tools for distributed analysis of network traffic [2]. In cooperation between the Autonomic Networking Group (University of Erlangen, Germany) and the Computer Networks and Internet group (University of Tübingen, Germany), we work on new methods for high-speed network monitoring, which build a basis for network security architectures including intrusion detection and traceback mechanisms. The network monitoring environment makes it possible to collect information about network traffic and its behavior in distributed network environments capable to operate on high-speed network links. Additionally, we ensure the interoperability of our tools by contributing to the standardization process in these areas in the IETF working groups IPFIX, PSAMP, and NSIS. In addition to the presented traffic generation environment, the main objective is to develop methods for handling high amounts of statistics and packet data even with cheap low-end components. Visualization techniques and anonymization methods round off the big picture of a visionary environment for all challenges in network monitoring and analysis. Developed tools are made available under an open source license. The applicability was already verified by employing the monitoring equipment in research projects focusing on efficient intrusion detection or accounting.

III. RELATED WORK

There are many traffic generators available. A summary of selected tools is provided in the following. The basic idea of this overview is to show needed dimensions of configurability and application.

Toolkit	**Packet definition**	**Traffic model**	**Parallel operation**	**Packet types**
ttcp	No	Through-put	No	TCP, UDP
NetPerf	No	Through-put	No	TCP, UDP
NetSpec	Limited	Variable	Yes	TCP, UDP
Iperf	No	Though-put	Yes	TCP, UDP
DBS	Limited	Variable	Yes	TCP, UDP
Harpoon	No	Variable	Yes	TCP, UDP
npag	Yes	Variable	Yes	TCP,UDP, ICMP, user-defined

IV. NPAG – NETWORK PACKET GENERATOR

A. Objective

The primary objective for developing "yet another" traffic generator was to overcome the disadvantages or missing functionality of other tools. The following requirements were considered during the development and distinguish npag from other proposals:

- Variable traffic rates including bursty behavior
- Long and short term measurements
- Support for different types of packets (IP, ICMP, …) including user-defined packets, e.g. for protocol tests
- Quality of service measurements (loss, delay, …)
- Support for parallel data streams
- Flexible configuration scheme

B. Design

In a distributed manner, npag is intended to be used to generate traffic at multiple locations in the network in order to simulate observable network traffic (see Figure 1). For example, DDoS (distributed denial-of-service) attacks can be formulated and executed.

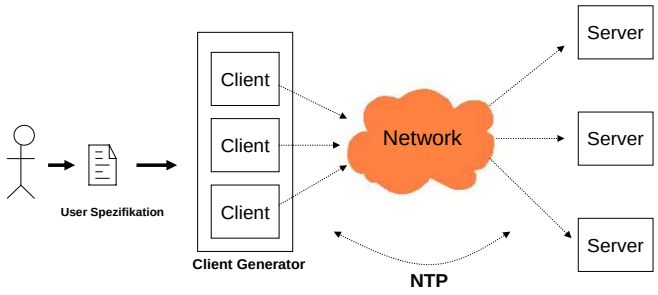

Figure 1. System architecture

The design of npag follows the following abstractions and paradigms. Basically, a client-server model is presumed that allows considering bi-directional protocol actions such as complete TCP connections. If there is no response from a client desired, the client abstracts the final destination for packets and packet flows. Multiple flows are supported by concurrently running threads that are controlled by a central coordinator. The measurement of quality of service parameters is provided by automatically included timestamps and sequence numbers into each packet (for particular measures, such as one-way delay, we consider previously synchronized clocks). The main objective for npag was to make it controllable by policy-based configuration schemes. This was established using a policy-language for traffic description that is described in the following. The architecture of npag is shown in Figure 1.

C. Policy-based configuration

The traffic as observed in real networks such as the Internet shows a sophisticated behavior. Bursty behavior, long-range dependencies, and other abstractions are used to model such behavior. The traffic behavior of artificially generated traffic should be configurable similar. Obviously, the data rate must be adaptable as well as packet size distribution. The following example depicts the description of traffic behavior for npag:

```
1   traffic {
2       burst { # 1st burst
3           packets = 100;
4           delay = 0.1;
5           repeat = 4;
6       }
7       bust { # 2nd burst
8           packets = 200;
9           delay = 0.2;
10          repeat = 2;
11      }
12  }
```

Two bursts are configured (line 2 and 7). The first one is intended to send 100 packets in a burst, wait for 0.1 seconds, and repeat this pattern 4 times. Similarly, the second one sends 200 packets each 0.2 seconds. Both bursts represent the same statistics over a long period of time (400 packets in 0.4 seconds) but a different behavior regarding the bursty behavior.

The following example shows a complete stream definition. In this example, a TCP connection is established between two systems defined by their IP addresses and TCP port numbers. Then, 100 packets of size 100 are exchanged.

```
1   stream {
2       tcp { # TCP protocol information
3           sport = 5000;
4           dport = 5001;
5       }
6       ip {  # IP protocol information
7           src = 192.168.178.2;
8           dst = 192.168.178.1;
9       }
10      traffic { # traffic behavior
11          burst {
12              packets = 100;
13              size = 100;
14          }
15      }
16  }
```

The envisioned application range includes protocol tests as well. Therefore, it is necessary to support the composition of variable user-defined packets even if they are not protocol conform. Examples include IP address spoofing, address misuses (invalid addresses, source equal to destination address), invalid flags, or invalid set other header fields. The following example outlines the use of npag for this purpose:

```
1   stream {
2       ip {
3           src = 192.168.178.24;
4           dst = 192.168.178.24;
5           version = 3; # IP version 3
6           df = 1; # modify IP
7           mf = 1; # header fields
```

```
8                       ttl = 255;
9               }
10              tcp {
11                      sport = 5000;
12                      dport = 5001;
13                      seq = 5;        # TCP header
14                      acknum = 12345; # fields
15                      fin = 1;        # and flags
16                      syn = 1;
17                      ack = 1;
18              }
19              traffic {
20                      burst {
21                              packets = 100;
22                              size = 100;
23                      }
24              }
25      }
```

Npag was implemented in C, whereas the further extensibility was especially focused on. Therefore, the system design follows a modular structure and a flexible API to integrate new functionality. In general, npag runs on every UNIX-based operating system but some special libraries to support user-defined packet headers are limited to Linux.

D. Outlook

Additional functionality can easily be includes. For example, an extension for dynamic packet generation, i.e. the creation of packets based on random variables to fill particular header fields, is an important functionality to test network security solutions. We think of an additional function that replaces static assignments in the packet definition by ranges or pure statistical methods.

V. PACO – PACKET COORDINATOR

Paco was developed to provide a centralized coordination of distributed packet generators. Basically, its development was driven by the need for more convenient control and management.

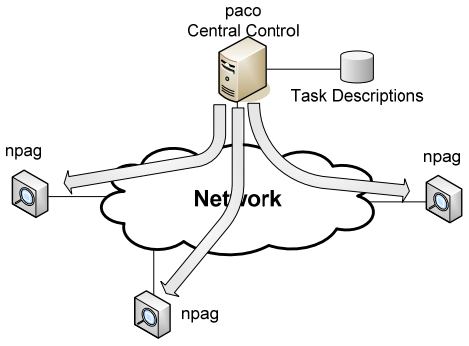

Figure 2. Architecture of paco

The main functionality of paco is the management of npag jobs. The JAVA-based paco allows adding standard jobs like UDP, TCP, or ICMP floods. Additionally, individual jobs can be configured using the npag's policy-language. The maintenance of the centrally stored jobs is independent from the state of the probes. Regularly, the jobs are transmitted to the probes for subsequent execution. Therefore, the probes can operate independently and only a limited connectivity to the management unit is necessary. The basic architecture is shown in Figure 2. The design of paco was driven by these requirements: distributed operation, distinction between a central management unit (MU) and decentralized probes, and support for a graphical user interface (a screenshot of paco is shown in Figure 3). The following tasks have to be executed:

MU	Probe
Probe communication • Availability • Req. for time synchr. • Submission of jobs	MU communication • State maintenance • Time synchr. • Job reception
Job management • Creation • Removal • Inspection • Termination	Job management • Execution • Termination

Figure 3. Screenshot of paco's main screen

Future extensions will include the management of packet receivers, the collection of measurement results, and a secured connection between the management unit and the Probes.

ACKNOWLEDGEMENTS

This work is part of the HISTORY project. Especially, I wish to thank my students Christian Bannes and Rodrigo Nebel for doing most implementations.

REFERENCES

[1] P. Barford and M. E. Crovella, "Generating representative workloads for network and server performance evaluation," Proceedings of ACM SIGMETRICS, Madison, WI, June 1998, pp. 151-160.

[2] F. Dressler and G. Carle, "HISTORY - High Speed Network Monitoring and Analysis," Proceedings of 24th IEEE Conference on Computer Communications (IEEE INFOCOM 2005), Miami, FL, USA, March 2005.

[3] J. Sommers, H. Kim, and P. Barford, "Harpoon: A Flow-Level Traffic Generator for Router and Network Tests," ACM SIGMETRICS 2004, New York, NY, USA, Abstract and Poster, June 2004.