INTEGRATION OF A NETWORK AWARE TRAFFIC GENERATION DEVICE INTO A COMPUTER NETWORK EMULATION PLATFORM

Suné von Solms [a,b], Schalk W. Peach [a]
[a] Council for Scientific and Industrial Research, Pretoria, South Africa
[b] School for Electronic, Electric and Computer Engineering, North West University, Potchefstroom, South Africa
svsolms@csir.co.za
speach@csir.co.za

Abstract: Flexible, open source network emulation tools can provide network researchers with significant benefits regarding network behaviour and performance. The evaluation of these networks can benefit greatly from the integration of realistic, network aware traffic into the network emulation platform. Traffic generators are often systems that replay captured traffic packet-by-packet or generate traffic according to a specified model or preconfigured sequence. Many of these traffic generators can be easily integrated into emulation platforms, but are not responsive to network behaviour and performance as it does not adapt to current network conditions and will blindly transmit data with minimal understanding of the network and its protocols (Botta, Dainotti & Pescapè, 2010). Network aware traffic generation systems communicate to the network using various protocols before traffic generation starts, enabling the generation of realistic, network aware traffic. In this paper we describe the integration of the Common Open Research Emulator (CORE), a network emulation platform, with a network aware traffic generation system, namely BreakingPoint. The integration is possible with the implementation of CORE in FreeBSD, using the Netgraph system. This solution allows real-time bidirectional communication between the virtualised nodes in the emulation platform and the external traffic generation system. This successful integration enables researchers to assess and analyse networks with the existence of realistic, network aware traffic.

Keywords: Network Emulation, Traffic Generation, Integration

1. Introduction

A difficulty researchers face when studying networks is that hardware experiments have high reliability, but are expensive to acquire, maintain and difficult to expand. In addition, the setup of all the hardware devices for every network test to be run is time consuming. On the other hand, simulation environments offer a repeatable, controllable environment that can easily be modified, but lacks realism through the use of simplified and approximated models (Wei & Mirkovic, 2006).

The natural trade-off between these two extremes is network emulation platforms. Network emulation platforms offer an acceptable amount of realism and hardware requirements, as well as multiple virtual network devices that can easily be reconfigured expanded. Therefore network emulation is an attractive and cost-effective tool for researchers interested in the behaviour and performance of networks. All virtual nodes on the network emulator can run the processes that is to run on the real hardware, as the virtual device duplicates the exact functions of the external real hardware system.

A good network emulation platform provides a range of functionalities, including the ability to create virtual replicas of servers, workstations and other network hardware to model existing or planned computer networks. The virtual replicas of network nodes enable users to create, install and run real life applications on the nodes. The applications can be accessed and used by the users, where real-life traffic generated from these applications flows through the network emulation. However, the traffic generated by the installed applications is not the only traffic present in a realistic network. Traffic is constantly generated by network appliances, background applications or by the control plane (Botta, Dainotti & Pescapè, 2010). Also included in the network is data received from outside the network, which can be traffic that originated remotely and is destined for one of the nodes in the network, misaddressed traffic, malware etc. (Douglass & Goldstein, 2004). The combination of all these functionalities ensures that the network emulator provides a realistic representation of the real-life network.

One problem continuously faced by researchers, however, is the realistic generation of network traffic. There exist a wide range of traffic generators, many providing non-accurate traffic patterns that can lead to inaccurate test results (Botta, Dainotti & Pescapè, 2010), (Papadopoulos, 2012). The effects of non-realistic traffic generation in a network emulation can include (Douglass & Goldstein, 2004):

- experiments that may not repeatable,
- comparisons between tests that may be infeasible,
- test results that may be inaccurate.

These traffic generators may contain a wide range of features, like the replication of complex traffic models and packet captures, but are often just systems with limited awareness that will blindly transmit data in a preconfigured sequence into the network with minimal understanding of the network and its protocols. Not communicating with the network before traffic generation starts leads to a lack in network awareness. The generation of realistic network traffic that is aware of what is happening in the network is essential for the realistic simulation of a network (Venkatesh & Vahdat, 2008). Therefore, the use of an accurate traffic generator is imperative in a network emulation platform so that virtual networks can be as close to real life as possible.

We aim to assist in the generation of realistic network traffic in emulation platforms by integrating the BreakingPoint traffic generator (IXIA, 2012) into the Common Open Research Emulator (CORE) emulation platform (Ahrenholz, et al., 2008), (The Boeing Company, 2012).

CORE is an emulation platform that focuses on the accurate duplication of network devices through the construction of virtual machines. Protocols and applications intended for the real devices can be run on the virtual machines (nodes) without modification and these live networks can be connected to real-life external networks and devices. CORE is an open system, which enables us to seamlessly integrate generated traffic with the network emulation.

The BreakingPoint system focuses on the generation of network aware traffic. This system does not only generate traffic based on statistical data, but creates simulated users that generate the requested traffic as a real user would communicate to a server or other users in the network. The users simulated in BreakingPoint communicate to the network using various protocols before traffic generation starts, enabling the generation of realistic, network aware traffic.

The integration of these two systems would greatly enhance the accuracy of network testing as CORE users would be able to create realistic network traffic in a network emulation. This includes background and application traffic for the emulated nodes even if the application is not installed and running on the node. This will create a comprehensive network traffic profile that models real-life network scenarios.

The rest of the paper is organised in the following order: The next section considers related work. Section 3 describes the benefits of traffic generator integration. Section 4 describes the two integration methods that was considered while Section 5 concludes this paper.

2. Related Work

2.1 CORE

CORE has its roots in the Integrated Multiprotocol Network Emulator/Simulator (IMUNES) system (Marko & Miljenko, 2004). CORE is built in a modular fashion which mainly consists of a GUI for the building of network topologies, a services layer which instantiates the virtual machines, as well as an application programming interface (API) that specifies how the different software components should interact. This modular architecture allows CORE to work with FreeBSD jails, discussed in Section 2.2, through the use of VIMAGE (Marko & Miljenko, 2004), (Zec, 2003).

The CORE system for FreeBSD was developed to use the existing operating system's virtualisation techniques to construct virtual networks. This container-based emulation system provides the high-fidelity emulation for layer 3 (network layer) and above. Real user and system applications as well as network protocols can run on the virtualised machines, or network nodes, without alteration. These processes are isolated in its own container with a virtual network stack and shared hardware resources. Links between the machines are created virtually, so that the virtual nodes and links form a complete emulated network. In addition, real network interfaces may be added to the virtual node, which enables the connection of the node to a live external network or device.

2.2 FreeBSD Netgraph subsystem

FreeBSD jails provide virtual environments for running programs in isolation. Each jail has its own network stack and network variables, like interfaces and protocol states. A jail along with the network stack form a lightweight virtual machine capable of running user applications (The Boeing Company, 2012).

Netgraph, a graph-based networking subsystem available in FreeBSD, allows the arbitrary linking of different networking components. The Netgraph subsystem enables the arrangement of nodes into arbitral graphs. Nodes have "hooks" which are used to connect two nodes together, thus forming a graph or virtual network. These connections can be seen as virtual links in the network over which nodes communicate (The Boeing Company, 2012).

Recent additions to the FreeBSD 10.x kernel (Voras, 2013), (Admin-GvE, 2013) such as VPS1 (Ohrhallinger, 2010), bhyve (Dexter, 2013), (Grehan, 2013) and NetMap (Rizzo, 2012), (Rizzo, Guiseppe & Maffione, 2013) present new opportunities for enhancing node characteristics of container based emulator systems. These are the features which we are to use in order to integrate realistic network traffic. Through the use of FreeBSD jails and the Netgraph subsystem, CORE can provide applications running in real time on an emulated network where the hardware requirement is relatively small while a large number of virtual nodes can be instantiated (Ahrenholz, et al., 2008).

2.3 BreakingPoint Traffic Generator

The BreakingPoint system by Ixia (IXIA, 2012) is a network simulation device, or cyber tomography machine (CTM), which allows for detailed simulation of various aspects of network traffic, including users browsing, emailing, texting, talking, and spreading malware. This includes an extensive library of user applications, including Facebook and Youtube, which is updated and maintained on a regular basis. A Markov chain approach is used to increase the level of realism in user simulation and dynamic recreation of real world traffic flow in computer networks.

BreakingPoint defines traffic flow patterns through the use of Clients and Servers. BreakingPoint simulates clients communicating to servers using the applications and protocols specified by the researcher (IXIA, 2012). In this manner, the clients generate connections to the specified servers to establish live connections for communication. Before traffic is generated and transmitted, the clients communicate to the servers via the emulated network nodes to establish the current network conditions, thus enabling the generation of network aware traffic that responds and adapts to the changing network conditions.

The BreakingPoint device generates network aware traffic that can be specified and customised specifically for the intended purpose of the user. Individual traffic flows can be specified in terms of source and destination IP ranges, host definitions, protocol type, action parameters etc. A BreakingPoint system with a single 1GB blade can generate up to 5 million simultaneous sessions (application traffic sessions) at a rate of 500 000 sessions per second. The variety and volume of traffic that can be generated make the BreakingPoint traffic generator able to provide realistic, network aware traffic which can enhance a network emulator to provide real-world network and traffic conditions (IXIA, 2012).

3. Advantages in Integration

Although the CORE emulation platform provides virtual nodes that enable the running of real applications and real-time user interaction, it is of utmost importance to evaluate the virtual network under realistic networking conditions. One aspect relating to realistic networking conditions is that of realistic network traffic (Venkatesh & Vahdat, 2008).

The CORE emulation platform has a built in traffic generator, but the traffic flows are limited to generalised traffic models which includes:

- 10, 100, 512 kbps POISSON
- 3 second fixed BURSTS every 15 seconds
- 0-5 second random BURSTS every 5 seconds
- JITTER 10 kbps 0.05-0.15 seconds
- cloned packet capture (PCAP) files

Even though the CORE traffic generator offers a range of features, including the replication of packet captures, it is only generated to put a load on the network, not to look like realistic network traffic.

Many research has been done on the impact of non-realistic and simplified traffic models on network application and protocol behaviour (Botta, Dainotti & Pescapè, 2010), (Papadopoulos, 2012), (Venkatesh & Vahdat, 2008). Venkatesh & Vahdat (2008) proves how realistic background traffic can significantly improve the accuracy in test results as applications are sensitive to burstiness of background traffic. Due to these generalised traffic patterns available, it was thought viable and very useful to integrate BreakingPoint with CORE.

The FreeBSD jails used in CORE make the software configurable so that realistic network aware traffic can be routed to each virtualised node from BreakingPoint. The integration of network aware traffic by BreakingPoint in CORE offers the users of CORE more options. Users can now virtualise a variety of networks where background as well as application traffic specific to the network scenario can be created. In addition to the reconfigurable architecture of CORE, the CORE GUI can provide researchers a comprehensive overview of the complete network, individual nodes as well as the traffic flowing over the network.

CORE enables comprehensive traffic flow analysis and visualisation through the use of throughput and display widgets as well as packet capturing via tshark (Wireshark, 2013) on all node interfaces. The flow of realistic generated traffic as well as the traffic from the applications running on the nodes can be monitored live using the CORE GUI. CORE allows run-time interaction with the network from the virtualised nodes and external devices, which can influence application performance and behaviour. Due to the fact that BreakingPoint generates application aware traffic, it is aware of the behaviour of the network and the nodes it is communicating with. Thus, a change in the network would influence the traffic flow to that node as it would in real-life.

4. Integration and Performance

As CORE and BreakingPoint are two different systems running on separate hardware, the interaction needs to be managed and coordinated. This interaction can be managed in various different ways, each with their own advantages and disadvantages. Subsequently, the two different integration methods are discussed.

4.1 Packet Routing

When a CORE emulation is instantiated, each virtualised node in the emulation is able to communicate with external devices or networks in real-time. To enable communication with external devices, emulated nodes can be linked to physical networked interfaces on the host machine.

Traffic generated by BreakingPoint can be routed through an emulation by connecting a physical network interface to an emulated router. As this is network aware traffic, the traffic can be routed throughout the network and back to the BreakingPoint system through the same or another emulated router connected to BreakingPoint. A diagram explaining the traffic flow is shown in Figure 1.

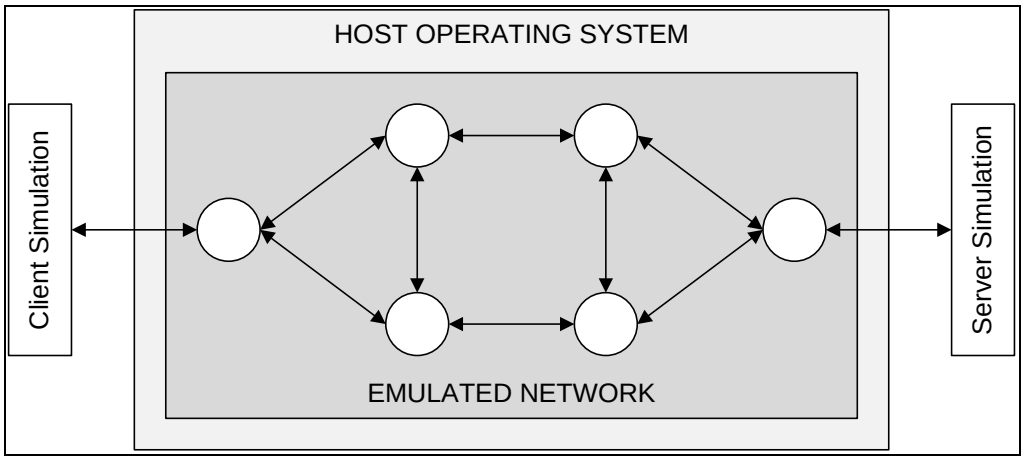

Figure 1: Integration through Packet Routing

As seen in Figure 1, the BreakingPoint device (Client Simulation side) is connected to the emulated network via a physical connection to the host machine. That physical network interface is linked to an emulated router, presented as the circle on the left in the diagram. The simulated clients on BreakingPoint communicate to the connected router, establishing the network configuration so that network aware traffic can be generated. It can be seen that the router forms part of the emulated network running on the host operating system. As shown by the bidirectional arrows, the traffic generated over the network can be routed back to the BreakingPoint system (Client Simulation side or Server Simulation side).

The disadvantage associated with this configuration is that BreakingPoint can simulate application and background traffic from thousands of users, though all traffic will appear to enter the emulated network from a single point, namely the router attached to the BreakingPoint device. Thus, using this packet routing based integration shown in Figure 1, end-user devices in the emulated network will not appear to generate the externally generated traffic, as it would appear to originate from a single router. To increase the fidelity of the emulation, an outside observer must not be able to distinguish between the origin of traffic generated by BreakingPoint and traffic generated by an emulated node.

4.2 Packet Forwarding

Utilising the capabilities of the BreakingPoint device to set the address ranges of the simulated clients and servers for traffic generation, combined with the capabilities of CORE to set individual IP addresses of emulated nodes, corresponding network configurations in both systems can be created.

To make it appear as though packets generated by the BreakingPoint device originate from a specific emulated node on the edge of the network, each individual packet must be forwarded to the edge node with the corresponding source IP address and then forwarded to the rest of the network. The BreakingPoint device expects all traffic generated from the client side to reach the server side, and vice versa. All traffic must therefore be sent back to the BreakingPoint device when reaching the node with the destination IP address.

Figure 2 illustrates the configuration of the emulation of client and server traffic as well as emulation of networked devices. The configuration consists of three layers, namely the externally generated traffic, components operating from within the host operating system and the emulated network. As can be seen in Figure 2, bidirectional communication is required between all components in the configuration, illustrated by the bidirectional arrows.

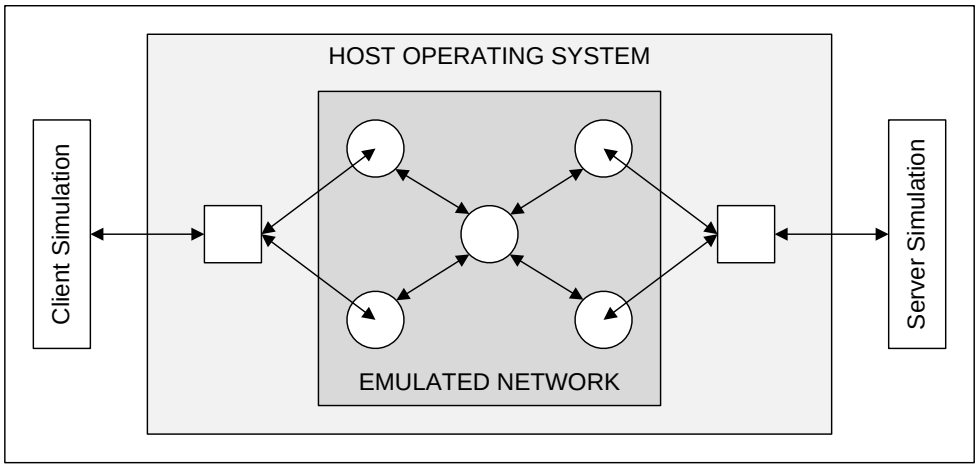

Figure 2: Integration through Packet Forwarding

Packets generated by the external traffic generator (BreakingPoint) is directed to physical network interface cards of the host hardware platform. A modified NetGraph bridge (ng_bridge*) node (represented by the square in Figure 2 and 3) receives all packets and forwards the packets to nodes within the emulated network (represented by circles in Figure 2 and 3). The network interface of each emulated node is a modified NetGraph Tee (ng_tee*) node, shown in Figure 3. This configuration enables all traffic received from the BreakingPoint device and all traffic generated by applications running on the node itself to be modified and forwarded into the emulated network. All network traffic coming into the node can then be modified and sent

back to the BreakingPoint device, through the modified ng_bridge* node. Figure 3 shows the internal architecture of the developed mechanisms.

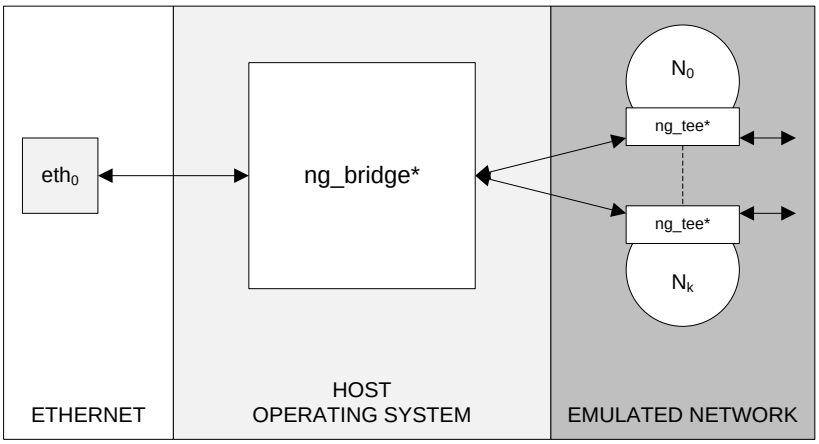

Figure 3: Packet Forwarding Operation

4.3 Packet Forwarding Operation

With both the BreakingPoint device and the CORE emulation configured to emulate the same IP ranges for both clients and servers, mechanisms to receive, modify and redistribute packets are required.

4.3.1 IP Address Resolution and Redistribution Configuration

When emulating nodes within a dynamic host address configuration environment, the integration of external traffic cannot be achieved until all emulated nodes have received an IP address. When a node receives a DHCP address from a DHCP server within the emulated network, the address assigned to the node is extracted by the modified ng_tee* node and a message containing the configuration is sent to the modified ng_bridge* node. The modified ng_bridge* updates a look-up table that associates source IP address configuration with the ng_hook* that is attached to the forward hook (see Figure 5) of the node configured with the associated IP address. Any incoming packet (generated by BreakingPoint) will be sent to the node corresponding to the IP source address of the packet by doing a lookup for the ng_hook* associated to the source IP address of the packet.

4.3.2 Packet Forwarding

The unmodified NetGraph ng_tee module has four hooks, left, right, left2right and right2left (FreeBSD Man Pages, 2004). The unmodified ng_tee node will send all data received on left to the right and left2right hooks, whilst all data received on right will be sent to left and right2left. An illustration of the unmodified NetGraph ng_tee module can be seen in Figure 4.

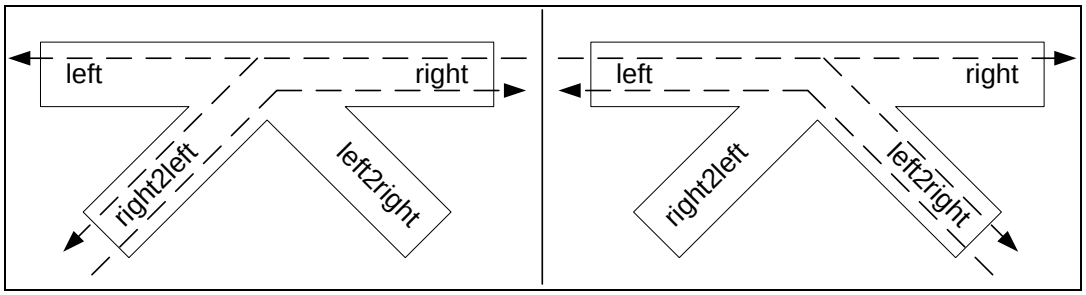

Figure 4: Unmodified ng_tee Operation

For the integration of the traffic generated by BreakingPoint into the emulated nodes, the ng_tee was modified. The modified version of the ng_tee node has three hooks, inner, outer and forward. The inner and outer hooks connect the emulated node to the next emulated node in the network. The operation of the modified ng_tee* node is similar to the operation of the standard ng_tee node. All data received on inner (the emulated node itself) is sent out the outer hook (onto the next emulated node), but not to the forward hook. All data received by the outer hook is forwarded to the inner and the forward hooks. Any packet received by

the forward hook is sent to the outer hook exclusively. Figure 5 illustrated the operation of the modified ng_tee* node.

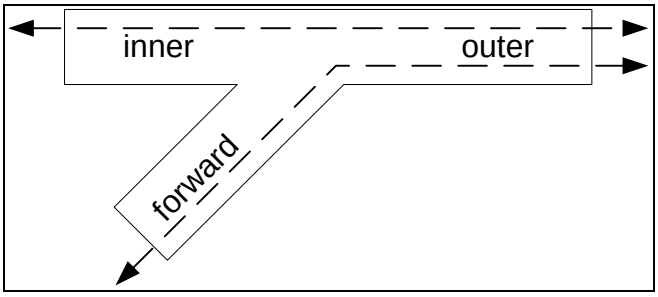

Figure 5: Modified ng_tee* Operation

The forward hook receives all data sent to it by the modified ng_bridge* node, which is the data sent by the BreakingPoint device. The packets generated by the BreakingPoint device will have a MAC address that differs from the MAC address of the emulated node. Packets received by the forward hook cannot be sent on to outer hook raw, as any routing or switching nodes will then receive packets with the same IP address, but conflicting MAC addresses. In order to prevent conflicts, each packet received by the forward hook is modified to have the MAC address of the emulated node. After packet modification, a checksum recalculation is done to ensure that packets are not rejected. Before the packet can be sent to the BreakingPoint device from the outer hook, the packet is modified to have the original MAC address that was generated by the BreakingPoint device, and sent to the forward hook.

4.3.3 Complications in Packet Forwarding

Packet forwarding enables the integration of network traffic generated by an external device. However, due to the unconditional operation of the outer hook, the kernel of the emulated node and the BreakingPoint device will receive unexpected packets. Packets intended for processes running in the emulated node reaches the BreakingPoint device and packets intended for the BreakingPoint device will reach the kernel of the emulated node.

4.4. Discussion

This configuration shows how packets generated by BreakingPoint for a specified client can be routed to its emulated node counterpart seamlessly. With this configuration, two users communicating via video chat, for example, can be generated with the traffic flowing over the network. When an outside observer looks at the network, it would seem that the two specified nodes are generating the traffic, with the correct IPs, protocols etc. This configuration allows network researchers to create networks that mirror realistic network environments.

5. Conclusion

In this paper we discussed the integration of the Common Open Network Emulator (CORE) with the BreakingPoint device, which is a traffic generator. A brief overview was given on the two systems and we discussed why CORE is the ideal platform to use for BreakingPoint traffic integration. We motivated why such an integration will be advantageous to network researchers performing network behaviour and performance tests in the CORE emulation environment.

Through the use of the Netgraph system in the FreeBSD platform, we could successfully develop integration mechanisms enabling the flow of network aware, bidirectional traffic from each node in the emulated network. The complete technical setup in order to create Packet Forwarding is described. Future work would include the separation of the packets intended for the kernel and the BreakingPoint device.

This integration of CORE emulation platform and the BreakingPoint device enables researchers to assess and analyse networks with the existence of realistic, network aware traffic.

References

Admin-GvE, 2013. FreeBSD 10's New Technologies and Features. [Online]
Available at: http://www.freebsdnews.net/2013/09/20/freebsd-10s-new-technologies-and-features/
[Accessed 15 02 2014].

Ahrenholz, J., 2010. Comparison of CORE network emulation platforms. MILITARY COMMUNICATIONS CONFERENCE, 2010 - MILCOM 2010, pp. 166-171.

Ahrenholz, J., Danilov, C., Henderson, T. R. & Kim, J. H., 2008. CORE: A real-time network emulator. Military Communications Conference, 2008. MILCOM 2008. IEEE, pp. 1-7.

Botta, A., Dainotti, A. & Pescapè, A., 2010. Do you trust your software-based traffic generator?. IEEE Communications Magazine, 48(9), pp. 158-165.

Dexter, M., 2013. BSD Hypervisor. [Online]
Available at: https://wiki.freebsd.org/action/show/bhyve?action=show&redirect=BHyVe
[Accessed 20 02 2014].

Douglass, P. R. & Goldstein, B. W., 2004. Background Internet Packet Traffic through Home Cable Connections. College of Computer and Information Science.

FreeBSD Man Pages, 2004. FreeBSD Kernel Interfaces Manual. [Online]
Available at: http://www.freebsd.org/cgi/man.cgi?query=ng_tee
[Accessed 20 02 2014].

Grehan, P., 2013. Extending bhyve beyond FreeBD guests. Proceedings of EuroBSDCon.

IXIA, 2012. BreakingPoint 2.0 User Guide, Release 3.0, s.l.: s.n.

Marko, Z. & Miljenko, M., 2004. Operating system support for integrated network emulation in imunes. 1st Workshop on Operating System and Architectural Support for the on demand IT InfraStructure (OASIS).

Ohrhallinger, K. P., 2010. Virtual Private Systems for FreeBSD. Proceedings of EuroBSDCon.

Papadopoulos, G. Z., 2012. Experimental Assessment of Traffic Generators, s.l.: s.n.

Rizzo, L., 2012. Netmap: A Novel Framework for Fast Packet I/O. Proceedings of the 2012 USENIX Annual Technical Conference.

Rizzo, L., Guiseppe, L. & Maffione, V., 2013. Speeding Up Packet I/O in Virtual Machines. ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS).

The Boeing Company, 2012. CORE Manual. s.l.:s.n.

Venkatesh, K. V. & Vahdat, A., 2008. Evaluating Distributed Systems: Does Background Traffic Matter?. Berkeley, CA, USA, USENIX Association, pp. 227-240.

Voras, I., 2013. What's new for FreeBSD 10. [Online]
Available at: https://wiki.freebsd.org/WhatsNew/FreeBSD10
[Accessed 20 02 2014].

Wei, S. & Mirkovic, J., 2006. A realistic simulation of internet-scale events. New York, NY, USA, ACM.

Wireshark, 2013. tshark. [Online]
Available at: http://www.wireshark.org/docs/man-pages/tshark.html
[Accessed 20 02 2014].

Zec, M., 2003. Implementing a Clonable Network Stack in the FreeBSD Kernel. Proceedings of the 2003 USENIX Annual Technical Conference.