A Multi-core Supported Intrusion Detection System

Tian Daxin[1,2], Xiang Yang[2]
1. School of Computer Science and Technology, Tianjin University, 300072, China
2. School of Management and Information Systems, Central Queensland University, Queensland 4702, Australia
tiandaxin@gmail.com, y.xiang@cqu.edu.au

Abstract

Integrated multi-core processors with on-chip application acceleration have established themselves as the most efficient method of powering next-generation networking platforms. New research has been conducted for addressing the issues of multi-core supported network and system security. This paper put forward an asymmetrical multiprocessing architecture multi-core supported anomaly intrusion detection system. The key idea is to use an independent core to run the intrusion detection system to monitor the host system. The detection method is based on the Hebb rule and uses libpcap to grab the network transmission packages. In the experiments, we use VMware which is configured to run the Ubuntu to simulate the IDS core. The results show that when the intrusion threshold is 0.3-0.5 the system performs best.

1. Introduction

The move by major microprocessor vendors toward processors containing multiple homogeneous processor cores is arguably the most important trend in contemporary computer architectures, almost all major processor vendors have already offered or released roadmaps for their own multi-core designs [1-4]. Network processors are typically characterized as distributed, multi-processor, multi-threaded architectures designed for hiding memory latencies in order to scale up to very high data processing rates. For example, the architecture of Intel IXP2850 is motivated by the need to provide a building block for 10-Gbps packet processing applications. Integrated multi-core processors with on-chip application acceleration have established themselves as the most efficient method of powering next-generation networking platforms. For example, the Cavium Networks' OCTEON family of Multi-Core MIPS64 processors offers industry-leading performance, scalability, low-power, and advanced hardware acceleration for intelligent networking applications ranging from 100Mbps to full-duplex 10Gbps. Recently, new researches have been conducted for addressing the issues of multi-core supported network and system security.

Paper [5] proposes a new intrusion survivable computing and self-healing scheme based on emerging chip multiprocessor (CMP) or multi-core platforms. Through the BIOS settings, some processor cores (or monitor cores) can be configured to run at a higher security or privileged level than the remaining cores (or protected cores). These higher privileged cores can monitor, reset, or recover the protected cores.

Paper [6] presents an integrated framework utilizing multi-core processors to detect intrusions and recover from infected states. The processor cores are divided as resurrectors and resurrectees and memory space is also insulated. Resurrectees cannot access resurrectors' memory but resurrectors can access all the memory space. Fine grain internal state logging for low privileged cores, resurrectees, is employed. Resurrectors dynamically check the states of resurrectees. If any suspicious intrusions are detected, a logged state will be recovered.

Paper [7] presents a unique multi-core security architecture based on Extensible Firmware Interface (EFI) [8]. This architecture combines secure EFI environment with insecure OS so that it supports secure and reliable bootstrap, hardware partition, encryption service, as well as real-time security monitoring and inspection. With this architecture, secure EFI environment provides users with a management console to authenticate, monitor and audit insecure OS. This architecture also has a unique capability to protect authentication rules and secure information such as encrypted data even if the security ability of an OS is compromised.

Paper [9] proposes a secure Chip-Multiprocessor architecture (SecCMP) to handle security related problems such as key protection and core authentication in multi-core systems. It employs a distributed secret sharing approach [10] to protect and distribute critical secrets shared by all processor cores. Threshold secret sharing scheme is employed to protect critical keys because secret sharing is a distributed security scheme that matches the nature of multi-core systems. A critical secret is divided and distributed among multiple cores instead of keeping a single copy that is sensitive to exposure. The SecCMP can not only enhance the security and fault tolerance in key protection but also support core authentication.

978-0-7695-3354-4/08 $25.00 © 2008 IEEE
DOI 10.1109/NPC.2008.19

In this paper an anomaly network intrusion detection system based on multi-core is presented. The main benefits of this system include:

Real-time: An independent core monitors the host system's network transmission by checking the passing packets. If an intruder is trying to intrude the system through network, the intrusion detection system can find it in real time and alarm immediately.

Intelligent: The main task of intrusion detection system is to distinguish abnormal behavior from normal behavior. It can study the normal behavior of host system through a Hebb neural network. Thus the intrusion detection system does not need expert knowledge and it can find unknown or novel intrusions.

Accurate: It uses packet capture library libpcap to grab packets from the network card directly. Libpcap provides implementation independent access to the underlying packet capture facility provided by the operating system.

The rest of this paper is organized as follows. In Section 2 we present the architecture of multi-core intrusion detection system and the detection method based on Hebb neural network. Detailed study of performance and simulation experimental results are presented in Sections 3, and conclusions are made in Section 4.

2. The design of multi-core supported IDS

2.1. The mutli-core IDS architecture

There are two types of software structures on a multi-core processor: Asymmetrical Multiprocessing (AMP) and Symmetrical Multiprocessing (SMP), illustrated in Fig.1 and Fig.2. AMP is a function-distribution software architecture that assigns fixed roles to each core. On the other hand, SMP is a load-distribution software architecture that determines the roles of CPU cores dynamically. We use the AMP since this approach enables security method to be independently executed on one core.

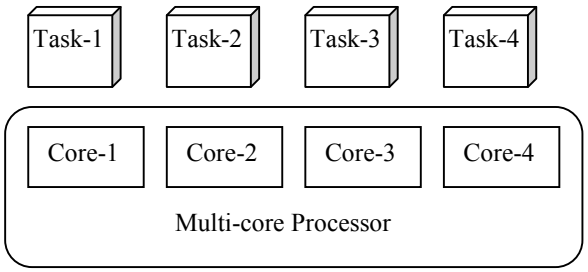

Fig.1 AMP structure

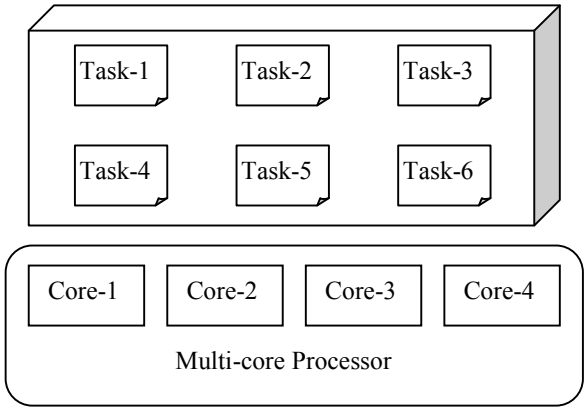

Fig.2 SMP structure

To detect intrusions, we employ the system architecture shown in Fig.3. The IDS core implements the intrusion detection capability, and the other cores run the application system. The IDS core can also run a different operating system from the rest.

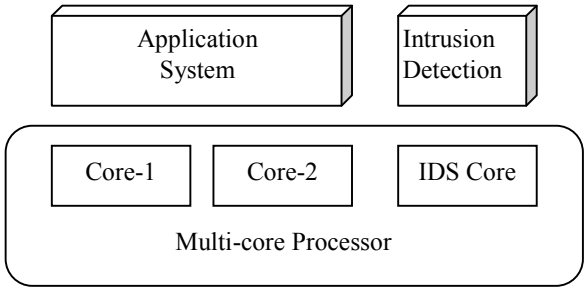

Fig.3 Multi-core IDS structure

2.2. Detection method based on Hebb rule

Intrusion detection techniques can be classified into two categories: misuse detection and anomaly detection. Misuse detection looks for signatures of known attacks. Any matched activity is considered an attack. Examples of misuse-detection techniques include the STAT [11] and IDIOT [12]. They use patterns of known attacks or weak spots of the system to match and identify known intrusions. Misuse detection can detect known attacks effectively, though it usually cannot accommodate unknown attacks. Anomaly detection models a user's behaviors, and any significant deviation from the normal behaviors is considered the result of an attack [13,14]. Anomaly detection techniques can be effective against unknown or novel attacks since no a prior knowledge about specific intrusions is required. However, anomaly detection systems tend to generate more false alarms than misuse detection systems.

2.2.1 Hebb rule neural network

Artificial neural networks have been applied in many fields. A neural network contains no domain knowledge in the beginning, but it can be trained to make decisions by mapping exemplar pairs of input data into exemplar output vectors, and adjusting its weights so that it maps each input exemplar vector into the corresponding output exemplar vector approximately. Well-trained neural networks represent a knowledge base in which knowledge is distributed in the form of weighted interconnections where a learning algorithm is used to modify the knowledge base from a set of given representative cases.

In this paper the learning algorithm is based on the Hebbian Postulate "When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased." [15]

The learning rule for a single neuron can be derived from an energy function defined as

$$E(\mathbf{w}) = -\psi(\mathbf{w}^T\mathbf{x}) + \frac{\alpha}{2}\|\mathbf{w}\|_2^2 \tag{1}$$

where $\mathbf{w}$ is the synaptic weight vector (including a bias or threshold), $\mathbf{x}$ is the input to the neuron, $\psi(\ )$ is a differentiable function, and $\alpha \geq 0$ is the forgetting factor. Also,

$$y = \frac{\mathrm{d}\psi(v)}{dv} = f(v) \tag{2}$$

is the output of the neuron, where $v = \mathbf{w}^T\mathbf{x}$ is the activity level of the neuron. Taking the steepest descent approach to derive the continuous-time learning rule

$$\frac{\mathrm{d}\mathbf{w}}{\mathrm{d}t} = -\mu\nabla_{\mathbf{w}}E(\mathbf{w}) \tag{3}$$

where $\mu > 0$ is the learning rate parameter, we see that the gradient of the energy function in (1) must be computed with respect to the synaptic weight vector, that is, $\nabla_{\mathbf{w}}E(\mathbf{w}) = \partial E(\mathbf{w})/\partial\mathbf{w}$. The gradient of (1) is

$$\nabla_{\mathbf{w}}E(\mathbf{w}) = -f(v)\frac{\partial v}{\partial\mathbf{w}} + \alpha\mathbf{w} = -y\mathbf{x} + \alpha\mathbf{w} \tag{4}$$

Therefore, by using the result in (4) along with (3), the continuous-time learning rule for a single neuron is

$$\frac{\mathrm{d}\mathbf{w}}{\mathrm{d}t} = \mu[y\mathbf{x} - \alpha\mathbf{w}] \tag{5}$$

the discrete-time learning rule (in vector form) is

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \mu[y(t+1)\mathbf{x}(t+1) - \alpha\mathbf{w}(t)] \tag{6}$$

In this paper, the inverse distance kernel function is used in Hebb learning. The dissimilarity measure function is Minkowski metric:

$$d_p(\mathbf{x},\mathbf{y}) = \left(\sum_{i=1}^{l} w_i |x_i - y_i|^p\right)^{1/p} \tag{7}$$

where x_i, y_i are the i th coordinates of $\mathbf{x}$ and $\mathbf{y}$, $i = 1,\cdots,l$, and $w_i \geq 0$ is the i th weight coefficient.

If set forgetting factor $\alpha = 1$, When the neuron is active $y = 1$, then the learning rule of ith neuron is

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + \mu * \gamma_i[\mathbf{x}(t+1) - \mathbf{w}_i(t)] \tag{8}$$

where,

$$\gamma_i = \begin{cases} 1, & \text{winner}, i = j \\ -K(d_i), & \text{others}, i = 1,\cdots,m \text{ and } i \neq j \end{cases} \tag{9}$$

and $K(d_i)$ is the inverse distance kernel,

$$K(d_i) = \frac{1}{1 + d^p} \tag{10}$$

If the winner's dissimilarity measure $d < \vartheta$ (ϑ is the threshold of dissimilarity), then update the synaptic weight by learning rule (8), else add a new neuron and set the synaptic weight $\mathbf{w} = \mathbf{x}$.

The main learning process is:

Step0: Initialize learning rate parameter μ, the threshold of dissimilarity ϑ;

Step1: Get the first input $\mathbf{x}$ and set $\mathbf{w}_0 = \mathbf{x}$ as the initial weight;

Step2: If the training is not over, randomly takes a feature vector $\mathbf{x}$ from the feature sample set $\mathbf{X}$ and computes the dissimilarity measure between $\mathbf{x}$ and each synaptic weight as (7);

Step3: Decide the winner neuron j and tests tolerance: If ($d_j >= \vartheta$) add a new neuron and sets synaptic weight $\mathbf{w} = \mathbf{x}$, goto Step2, else continue;

Step4: Compute γ_i by using the result of inverse distance $K(d_i)$;

Step5: Update the synaptic weight as (8) , goto Setp2.

2.2.2 Network behavior

In this paper, we form the network behavior vector based on the character of network software structuring technique. In the TCP/IP Reference Model, the Internet layer defines an official packet format and protocol called IP (Internet Protocol); the layer above the Internet layer is transport layer, two end-to-end protocols have been defined here. The first one, TCP (Transmission Control Protocol) is a reliable connection-oriented protocol that allows a byte stream originating on one machine to be

delivered without error on any other machine in the Internet. The second protocol in this layer, UDP (User Datagram Protocol), is an unreliable, connectionless protocol. We use the heads of packets to present the network behavior.

```
typedef struct _EthernetBehavior
{
/* destination ethernet address */
u_int8_t ethernet_dest[12];
/* source ehternet address */
u_int8_t ethernet_sour[12];
/* packet type ID field */
u_int16_t ethernet_type;
}EthernetBehavior;
```

The *ethernet_type* shows the nested structure of protocol headers. It may be an IP, ARP, or some other protocols. For instance, an IP header can be defined as:

```
typedef struct _IPBehavior
{
/* the header length */
unsigned int header_len;
/* version of the protocol */
unsigned int version;
/* type of service */
u_int8_t tos;
/* total length of datagram */
u_short total_len;
/* identification */
u_short identification;
/* flags and fragment offset */
u_int8_t flag_off;
/* the limit of packet lifetimes */
u_int8_t time_live;
/* TCP or UDP */
u_int8_t protocol;
/* header checksum */
u_int8_t checksum;
/* source address */
struct in_addr source_addr;
/* destination address */
struct in_addr destination_addr;
} IPBehavior;
```

The variable *protocol* tells what type of protocol will be used in the upper layer, it can be TCP, UDP, ICMP, etc. For example the definition of TCP is:

```
typedef struct _TCPBehavior
{
/* source port */
u_int16_t sour_port;
/* destination port */
u_int16_t dest_port;
/* sequence number */
tcp_seq seq_num;
/* acknowledgement number */
tcp_seq ack_num;
/* flags */
u_int16_t flag;
/* window size */
u_int16_t win_size;
/* header checksum */
u_int16_t check_sum;
/* urgent pointer */
u_int16_t urg_pointer;
}TCPBehavior;
```

During the period of training, we use the behavior variables of the normal packets to form the behavior matrix. In the detection period, propose $\mathbf{W}$ is the weight matrix of the stable neural network, $\mathbf{w}_j$ is column j of $\mathbf{W}$. The IDS core grabs the packet transmitting in the application system and the packet is translated into behavior variable $\mathbf{p}_i$. If $d(\mathbf{p}_i, \mathbf{w}_j) > t$, where t is the threshold value of intrusion, IDS will alarm and display the detailed information of the intruder.

3. Experiments

In the experiments, we use VMware to simulate the IDS core. The VMware is configured to run the Ubuntu 7.10 Linux OS, and the intrusion detection system is implemented and run on Ubuntu. The underlying host system that VMware is running on, is a 2.26GHz Intel Pentium processor machine with 512MB RAM. The IDS core monitors and detects the host system's network behavior using libpcap library which provides a flexible filtering framework. The main functions we used are:

pcap_open_live() is used to obtain a packet capture descriptor to look at packets on the network.

pcap_lookupnet() is used to determine the network number and mask associated with the network device.

pcap_lookupdev() returns a pointer to a network device suitable for use with pcap_open_live() and pcap_lookupnet().

pcap_loop() is used to collect and process packets.

The network IDS needs to grab packets from network card, but if the CPU is slow or network speed is high, the IDS may dropping some packets thus may lose some attacking information, which will increase false alarm rate. We first do the experiment to evaluate the method's real-time performance. The TG2.0 [16] is used to generate high volumes of traffic on the testing network. The TG's client is set to open a UDP socket to send packets to the TG server waiting at 192.168.10.1 and port 4322, with packet data length being 576 and packet number is 2000. On the multi-core system, the odd number packets are processed by one core and the even number packets are processed by another core. The dropping rate under different interpacket transmission time (0.02 secs equals 1/0.02=50 packets/sec) is shown Fig.4. From the result we can find that multi-core can with low dropping rate than one processor system.

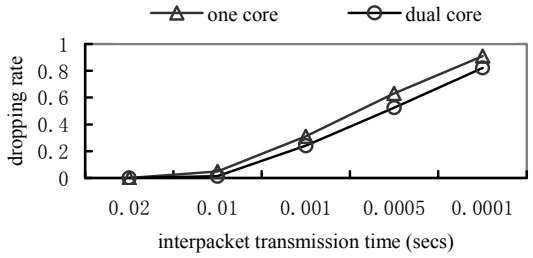

Fig.4 Dropping rate

Multi-core can increase the processing speed of the whole system, so it will provide the IDS more calculation time, thus can reduce the dropping packet number and will decrease false rate. To evaluate the IDS core's performance, a series of experiments are conducted. IDS errors consist of false positive errors and false negative errors. The false positive errors occur because IDS misinterprets normal packets as an attack, the false negative errors occur because an attacker is misclassified as a normal user. The recording format of the test results is shown in Table 1. False alarm is partitioned into False Positive (FP, normal is detected as intrusion) and False Negative (FN, intrusion is not detected). True detect is also partitioned into True Positive (TP, intrusion is detected rightly) and True Negative (TN, normal is detected rightly).

Table1. Recording format of test result

		Detection Results				
		Normal	**Intrusion -1**	**Intrusion -2**	**---**	**Intrusion-n**
Actual Behaviors	**Normal**	TN00	FP01	FP02	---	FP0n
	Intrusion -1	FN10	TP11	FP12	---	FP1n
	Intrusion -2	FN20	FP21	TP22	---	FP2n
	⋮	⋮	⋮	⋮	⋮	⋮
	Intrusion -n	FNn0	FPn1	FPn2	---	TPnn

Definition 1. The right detection rate of ith behavior $TR = T_{ii} \Big/ \sum_{j=1}^{n} R_{ij}$, where T_{ii} is the value lies in table1's ith row and ith column.

Definition 2. The right prediction rate of ith behavior $PR = T_{ii} \Big/ \sum_{j=1}^{n} R_{ji}$, where T_{ii} is the value lies in table1's ith row and ith column; R_{ji} is the value lies in table1's jth row and ith column.

Definition 3. Detection rate (DR) is computed as the ratio between the number of correctly detected intrusions and the total number of intrusions. If regard Table1's record as an $(n+1)\times(n+1)$ metric $\mathbf{R}$, then $DR = \sum_{i=1}^{n}\sum_{j=1}^{n} R_{ij} \Big/ \sum_{i=0}^{n}\sum_{j=0}^{n} R_{ij}$.

Definition 4. False positive rate (FPR) is computed as the ratio between the number of normal behaviors that are incorrectly classifies as intrusions and the total number of normal connections, according to the Table1's record, $FPR = \sum_{i=1}^{n} FP0i \Big/ \sum_{i=1}^{n} FP0i + TN00$.

The anomaly intrusion detection method put forward in this paper cannot classify what kind of intrusion is happening, if the behavior value is bigger than the intrusion threshold value we will believe that an intrusion is happening. So there is only one column and one raw of intrusion record. We first train the neural network with different normal features, then use the stable neural network to monitor the host system's network transmission where some abnormal behaviors may are happening under the same environment. The experiments are conducted to analyze the effects of varying the value of intrusion threshold to system errors. The tests results are graphically represented in Fig.5.

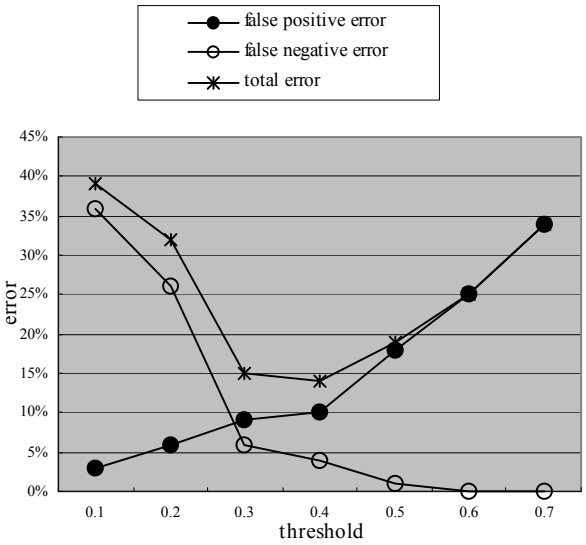

Fig.5 Test results

We can find that the performance of the method is sensitive according to intrusion threshold. As the threshold value increases, false positive errors increase while false negative errors decrease. Since a false negative error is more important in IDS, we need to concentrate on the decrease of false negative errors according to the change of the threshold value. The optimal threshold value is 0.3-0.5.

4. Conclusions

Recently, multi-core processors have been targeted for use in a wide variety of networking and security equipment, including routers, switches, unified threat management (UTM) appliances, content-aware switches, application-aware gateways, triple-play gateways, WLAN and 3G/4G access and aggregation devices. This paper presents a multi-core supported anomaly intrusion detection system to provide highly reliable network services and systems. It uses an AMP structure based multi-core system to achieve real-time intrusion detection. The IDS runs on a core and detects the host system's network behavior with libpcap library. The anomaly detection method based on the Hebb rule neural network can distinguish abnormal behavior from normal behavior. In the experiment, we simulate the IDS core using VMware. The VMware is configured to run the Ubuntu 7.10, and the intrusion detection system is implemented and run on Ubuntu. The experimental results show that when the value of intrusion threshold is 0.3 to 0.5 the performance is optimal.

Acknowledgement

This work is supported by the "Personal Computer Bodyguard: Using Multi-core to Support Security-Related Applications", Central Queensland University, Research Development and Incentives Program.

References

[1] Gelsinger P.P., Gargini P.A., Parker G.H., Yu A.Y.C. Microprocessors circa 2000, IEEE Spectrum, Volume 26, Issue 10, 1989, pp.43-47.

[2] AMD Multi-core White Paper, http:// www.sun.com/emrkt/innercircle/newsletter/0505multicore_wp.pdf.

[3] Intel Multi-Core Processors, http:// www.intel.com/technology/architecture/downloads/quad-core-06.pdf.

[4] Tendler J.M., Dodson J.S., Jr.Fields J.S., Le H., Sinharoy B. POWER4 system microarchitecture. IBM Journal of Research and Development, 46(1), 2002, pp.5-25.

[5] Shi W.D., Lee H.H.S., Gu G.F., Falk L., Mudge T.N., Ghosh M. An intrusion-tolerant and self-recoverable network service system using a security enhanced chip multiprocessor. In Proceedings of the Second International Conference on Autonomic Computing, 2005, pp.263-273.

[6] Shi W.D., Lee, H.H.S., Falk L., Ghosh M. An integrated framework for dependable and revivable architectures using multicore processors. In Proceedings of the 33rd International Symposium on Computer Architecture, 2006, pp.102-113.

[7] Zhang X.Z., Xie Y., Lai X.J., Zhang S.S., Deng Z.J. A multi-core security architecture based on EFI. OTM Confederated International Conferences CoopIS, DOA, ODBASE, GADA, and IS 2007, pp.1675-1687.

[8] Intel. Extensible Firmware Interface Specification Version 1.10 (December 2002), http://developer.intel.com/technology/efi/

[9] Li Y., Lu P. SecCMP: a secure chip-multiprocessor architecture. In Proceedings of the 1st Workshop on Architectural and System Support for Improving Software Dependability, 2006, pp.72-76.

[10] Pedersen T.P. A threshold cryptosystem without a trusted party. In Proceedings of Advances in Cryptology-EUROCRYPT, 1991, pp.522-526.

[11] Ilgun K, Kemmerer R.A, and Porras P.A. State transition analysis: A rule-based intrusion detection approach. IEEE Transactions on Software Engneering, 21(3), 1995, pp.181-199.

[12] Kumar S, Spafford E.H. A software architecture to support misuse intrusion detection. In Proceedings of the 18th National Conference on Information Security. 1995, pp.194-204.

[13] Denning D.E. An Intrusion-Detection Model. IEEE Transaction on Software Engineering, Volume 13, Issue 2, 1987, pp.222-232.

[14] Lee W.K., Stolfo S.J. A framework for constructing features and models for intrusion detection systems. ACM Transactions on Information and System Security, Volume 3, Issue 4, 2000, pp.227-261.

[15] Hebb, Donald O. The Organization of Behavior. New York: Wiley, 1949.

[16] McKenney P.E., Lee D.Y., Denny B.A. Traffic generator software release notes http://www.postel.org/services.html