On Interactive Internet Traffic Replay

Seung-Sun Hong[1] and S. Felix Wu

University of California, Davis CA 95616, USA,
{hongs,wu}@cs.ucdavis.edu

Abstract. In this paper, we introduce an interactive Internet traffic replay tool, TCPopera. TCPopera tries to accomplish two primary goals: (1) replaying TCP connections in the stateful manner, and (2) supporting traffic models for trace manipulation. To achieve these goals, TCPopera emulates the TCP protocol stack and replays trace interactively in terms of the TCP connection-level parameters and the IP flow-level parameters. Due to the stateful TCP connection replay feature of TCPopera, it ensures no ghost packet generation which is a critical feature for the test environments where the accuracy of protocol semantics are of fundamental importance. In our validation tests, we showed that TCPopera successfully reproduces the trace records in terms of a set of traffic parameters. Also we demonstrated how TCPopera can be deployed in the test environments for intrusion detection and prevention systems.

1 Introduction

For the purpose of testing new applications, systems, and protocols, the network research community has a persistent demand for the traffic generation tools that can create a range of test conditions similar to those experienced in live deployment. Having an appropriate tool for generating controllable, scalable, reproducible, and realistic network traffic is of great importance in various test environments including laboratory environments [1, 2], simulation environments [3, 4], and emulation environments [5–9]. When the tools fail to consistently create realistic network traffic conditions, new systems will have the risk of unpredictable behavior or unacceptable performance when deployed in live environments.

There are two different approaches to generate test traffic: the trace-based traffic replaying and the analytic model-based traffic generation. The trace-based traffic replay approach replays a recorded stream of IP packets from the real network to the test network. This approach is easy to implement and mimic activity of a known system, but the replayed traffic might not be representative unless the congestion situation in the test network is the same. Also, because it treats the traffic characteristics of the trace records as a black box, it is difficult to adjust the trace for different test conditions. In contrast, the analytic model-based traffic generation approach starts with mathematical models for various traffic/workload characteristics, and then produces the traffic adhere to the models.

This approach is challenging because it is necessary to identify important traffic characteristics to model and those characteristics must be empirically measured beforehand. Furthermore, it can be difficult to produce a single output that accurately shows all traffic characteristics. However, this approach is very straightforward to tune the parameters for traffic models to adjust the traffic.

Although choosing an appropriate traffic generation method for the test environment depends on its primary goal, there are some test environments where both the realism of the trace contents and the accuracy of protocol semantics are of fundamental importance. For example, the best test traffic for IPS (Intrusion Prevention Systems) test environment is the one capturing the attacks or suspicious behaviors from the real network. Besides, how we can provide this trace record for the test environments without breaking protocol semantics is a challenging issue. For such a test environment, neither the trace-based traffic replay approach nor the analytic model-based traffic generation approach is sufficient to satisfy the test purposes.

In this paper, we introduce an interactive Internet traffic replay tool, TCPopera, that follows a middle road between the trace-based traffic replaying and the analytic model-based traffic generation [1]. The new traffic replay paradigm is based on the idea that the trace records are the combination of packet streams and traffic properties. That is, the traffic properties (expressed in the traffic parameters) decides the timing information of an individual packet. Thus, if we can properly reverse-engineer the trace records, the traffic properties can be separated from the packet streams. Then, the TCPopera replay engine reproduces the trace records according to the new traffic parameters based on analytic traffic models.

The TCPopera architecture is invented to support an interactive traffic replaying at the IP flow level that each flow is reproduced by a POSIX thread [10]. There are several design goals for the TCPopera architecture. First, TCPopera ensures no ghost packet generation. TCPopera emulates the TCP control block for each connection in order to replay TCP connections in the stateful manner. Second, TCPopera supports various traffic models to overcome the inherent drawback of conventional traffic replay tools by implementing the extensible internal library for future traffic models. Third, TCPopera resolves the interconnection dependencies within a single IP flow by employing a simple heuristic based on the assumption that the packet sequences between two hosts reflect the history of activities between two hosts. Next, TCPopera supports the environmental transformation to help replaying trace records in different test environments. Last, TCPopera is designed to support scalability to be deployed in the large-scale test environment. This goal requires the scalable control mechanism based on the (graphic) user interface.

We evaluated the TCPopera's capabilities in two ways. First, we conducted the small test environment using the network emulator (Dummynet [11]) to show the traffic reproductivity of TCPopera in terms of a set of traffic parame-

[1] TCPopera is an interactive version of TCPtransform, which is the property-oriented traffic transformation tool [26]

ters. Second, we evaluated the performance of Snort [13], a public-domain IDS, over various test traffic conditions recreated by TCPopera in order to demonstrated how TCPopera can be deployed in the live test environment for security products. Although we found out the side effects of the current TCPopera implementation, the validation test results showed that the possibility of TCPopera to be used to evaluate new security products under actual operating conditions. For the above tests, we used both the 1999 MIT's Lincoln Lab (LL) IDEVAL dataset [12], because this dataset include the synthetic attack traffic for the purpose of the intrusion detection system evaluation, and the ITRI real traffic trace from Taiwan.

We demonstrated the ability of the current TCPopera implementation throughout the validation tests. We compared the TCPopera traffic to the input trace records in terms of traffic volume and other distributional properties. In the traffic reproduction test, we found that TCPopera successfully reproduced IP flows in terms of traffic parameters without breaking protocol semantics. We also demonstrated how TCPopera can be deployed in the live test environments for evaluating IDS/IPS through the effectiveness test. We observed that Snort generated different results when we changed the test conditions using TCPopera. At least some of these interesting "differences" we discovered, as we will explain later in this paper, are due to the implementation bugs of Snort.

Currently, TCPopera has been used by one commercial security vender almost on a daily basis in developing/debugging their IPS boxes. A typical scenario is that a tester (usually in another city) found a false negative (i.e., not detecting something that should have been detected) of the tested IPS box. The tester is asked to capture the whole traffic trace, and sends the trace to the development team. The development team will then use TCPopera to interactively replay the trace and revise/debug the signatures until the attacks are correctly detected and handled. Similarly, this company is also using TCPopera to revise the signatures to correctly handle false positives found against real traffic. We were told that TCPopera provides a nice framework to easily repeat the tests until the false negatives or positives are fixed.

This paper is organized as follows. After presenting related work in section 2, we describe the issues related to the TCPopera design and implementation to support the interactive traffic replaying in section 3. In section 4, we present the results of out validation tests and analyze them. Then, we conclude our work and present the future direction of the TCPopera development in the section 5.

2 Related work

For the test environments for security products, high-volume traffic/workload generation tools are insufficient to satisfy the test conditions. They can be deployed to provide the background traffic to test the performance under load, but they are not capable of creating the attack traffic. For this reason, the security product testing groups still prefer the trace-based traffic replay approach in order to evaluate the security functions of the products. In this section we present the overview of the open-source traffic replay tools.

TCPreplay [14], originally developed to provide more precise testing methodology for the research area of network intrusion detection, is a tool designed to replay the trace files at arbitrary speeds. The recent versions of TCPreplay have added the multiple interface support for testing in-line devices. TCPreplay provides a variety of features for replaying traffic for both passive sniffer devices as well as in-line devices such as routers, firewalls, and IPS. IP addresses can be rewritten or randomized, MAC addresses can be rewritten, transmission speeds can be adjusted, the truncated packets can be repaired, and the packets are selectively sent or dropped. Because the main purpose of TCPreplay is to send the capture traffic back to the test network, the exact opposite of TCPdump [15], it cannot connect to services running on a device. To overcome this problem, the developers of TCPreplay developed the Flowreplay program. Flowreplay [14] can connect via TCP or UDP to server and sends/receives data based on a pcap capture file [16]. It provides more testing methodologies for testing environments, however, the major limitation of Flowreplay is that it is only capable or replaying the client side of a pcap against a real service on the target host.

TCPivo [17] is a high-performance replay engine that accurately reproduces traffic from a variety of existing trace collection tools. The design goal of TCPivo is to have a cost-effective tool that easily runs on pre-existing systems such as x86-based systems. To achieve this goal, TCPivo considered the following issues. First, TCPivo uses the on-the-fly prefetching of a packet from a trace file to minimize the latency of I/O operations. Using `mmap()` and `madvise()` functions, TCPivo implemented a double buffered approach that one buffer for prefetching and the other for being actively accessed. Second, TCPivo uses `usleep()` with real-time priority set to improve the accuracy. Next, TCPivo used the null-padded payload by getting rid of reading the payload from the file system in order to speed-up the packet transmission loop. Last, TCPivo used a real-time scheduling priorities for Linux. Alternatively, making the kernel preemptible or reducing the longest non-preemptible path can be other choices for real-time processing scheduling.

Monkey is a tool to replay an emulated workload identical to the site's normal operating conditions [18]. Monkey infers delays caused by the client, the protocol, the server, and the network in each captured flow and replays each flow according to them. Monkey has two major components: *Monkey See*, a tool for TCP tracing, *Monkey Do*, a tool for TCP replaying. Monkey See captures TCP packet traces at a packet sniffer adjacent to the Web server being traced and performs offline trace analysis to extract observable link delay, packet losses, bottleneck bandwidth, packet MTUs, and HTTP event timing. Monkey Do consists of three emulators. The client emulator replays client HTTP requests in sequence by creating user-level sockets for each connection. The server emulator presents the HTTP behavior of a Google server interacting with a client. Last, The network emulator recreates the network conditions identical to the one at the time the trace was captured.

Tomahawk is a tool for testing the performance and in-line blocking capabilities of IPS devices [19]. It is run on a machine with three network interface cards (NIC): one for management and two for testing. Two test NICs are typically connected through a switch, crossover, or NIPS. Tomahawk divides a packet trace into two parts: The client packets, generated by the client, and the server packets, generated by the server [2]. When Tomahawk replays the packet, the server packets are transmitted on eth1 and the client packets are transmitted on eth0 as default. If a packet is lost, the sender retries after a timeout period. If progress is not made after a specified number of retransmissions, the session is aborted. When the replay is finished, Tomahawk reports whether the replay completed or timed out. For the IPS testing, if the connection containing the attacks are timed out, it implies that IPS blocked the attack successfully. Besides, Tomahawk can test the performance of IPS such as the replaying multiple copies of the same trace in parallel, replaying multiple packet traces simultaneously, and Repeatability testing ensuing the IPS is deterministic. However, Tomahawk has some inherent limitations. First, it is impossible to have the generated traffic pass through routers because it can only operate across a layer 2 network. Second, it cannot handle traces containing badly fragmented traffic, and multiple sessions in the same pcap can sometimes confuse it, Third, there is mishandling of TCP RST packet sent by an IPS.

The most significant difference of TCPopera from aforementioned traffic replay tools is that TCPopera is designed for interactive trace replaying by supporting the stateful TCP connection. Both TCPreplay and TCPivo work fine with the passive sniffer devices, but they have the problem in testing in-line devices such as routers, firewalls, and IPS. Although TCPreplay have recently added multiple interface support that similar to the functionality of Tomahawk, its functionality is limited to split the traffic into different NICs. Comparing to TCPreplay, Tomahawk uses the clever method to control the replay of TCP connections, but its inherent drawbacks prevent it from deploying in the real test environment such as DETER. Flowreplay and Monkey differ from other replay tools in that they eventually emulates the TCP connections in the trace records. However, they also have the limitation in that Flowreplay are only capable of emulating the client side of the connection and Monkey only emulates the HTTP traffic.

3 TCPopera

3.1 Design Goals

TCPopera is an advanced traffic replay tool that interactively replays the trace records in the live test environments. With respect to the live traffic replaying, there are several requirements TCPopera must take into account in its design. The following list describes these requirements.

[2] The first time an IP address is seen in a file, it is assigned to the client if it is seen in the IP source address field, or assigned to the server if it is in the destination address field.

– **No ghost packet generation**: The reason why the traffic replay tools generates ghost packets is because they are not capable of replaying TCP connections in the stateful manner. Since ghost packets break the protocol semantics, they degrade the accuracy of the testing results. For this reason, no ghost packet generation is one of fundamental requirements in the TCP-opera design.

– **Traffic models support**: One drawback of existing traffic replay tools is that is is difficult to adjust traffic for various test conditions. To overcome this drawback, TCPopera should support various traffic models to adjust trace records in terms of traffic parameters. To achieve this goal, TCPopera should have an appropriate reverse-engineering tool to extract the traffic properties from the input traces.

– **Inter-connection dependency**: Identifying dependencies among TCP connections from the trace records is a complicated task because it requires not only a wide understanding of TCP applications, but also a huge amount of computation. To make it worse, if the connection dependencies exists across IP flows, e.g. stepping stone connections, it is challenging to identify inter-connection dependencies without inspecting the packet payload. TCPopera should support inter-connection dependencies at the reasonable cost of computation.

– **Scalability and extensibility**: No traffic replay tool has been used in the large-scale emulation environment such as DETER because they are originally designed to be used in small-scale test environment. Scalability is a critical requirements for TCPopera to be deployed in the large-scale emulation environments. Extensibility is another important requirement from the aspect of software engineering. New traffic models and various protocol implementations should be added without major changes in the TCPopera design.

– **Environment transformation**: One of challenges in the traffic replaying is to map the original network configurations to those of the target network because this task requires more than a simple address remapping currently used in most of traffic replay tools. TCPopera must know the routing information of test environments to correctly deliver the replayed packets to the its destination.

3.2 TCPopera Architecture

TCPopera is property-oriented traffic replay tool that follows a middle road between the trace-based traffic replay approach and the analytic model-based traffic generation approach. TCPopera separates the traffic properties from IP packet streams in trace records, and recreates new IP packet streams that statistically equivalent to the input traffic models. Each TCPopera node represents

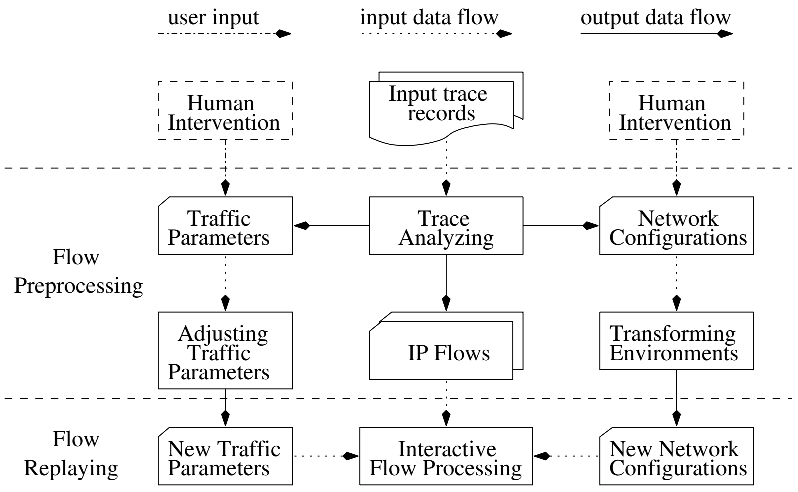


Fig. 1. The flow processing model of TCPopera

a set of hosts/networks and interacts with its peer TCPopera nodes. Figure 1 depicts the flow model of each TCPopera node.

TCPopera performs two phases of trace processing for an interactive trace replaying: flow preprocessing and flow replaying. During the flow preprocessing phase, a TCPopera node reads input trace records and extracts information including traffic parameters and network configurations. The TCPopera users can add or modify the traffic parameters and network configuration for a new test environment. When the flow preprocessing phase is completed, the TCPopera node first synchronizes the timing information with its peers, and then begins the interactive traffic replaying in terms of traffic parameters configured at the previous phase.

Figure 2 shows the important components of the TCPopera architecture. TCPopera users can edit the configuration files to configure test conditions. Then, the Flow Preprocessing module reads these configuration files and adjust traffic parameters and network configurations. Also, it extracts replayable IP flows from the trace records. The IP flows prepared by the Flow Preprocessing module is replayed by the IP Flow Processing module. Each IP flow is reproduced by a POSIX thread [10] in terms of traffic parameters. The TCPopera control module provides the out-of-band communication channels among TCPopera nodes to synchronize the replaying information and timing. The IP Flow Processing module keep track of the state of TCP connections using the TCP functions library and the TCPopera timer provides the timing information needed to emulate the TCP control block. In addition, the Packet Capturing module helps reading inbound packets from the link, and the Packet Injection module helps reconstructing and injecting packet on the wire.

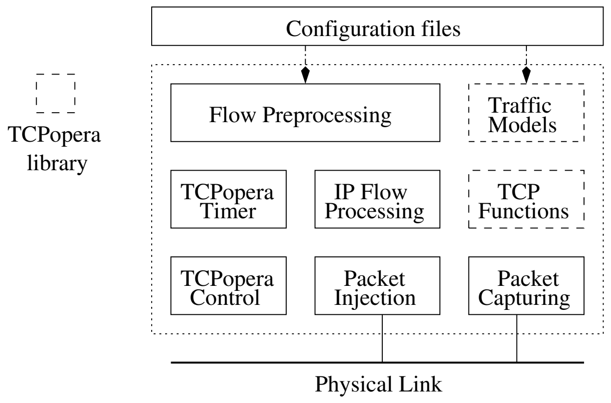


Fig. 2. The TCPopera Architecture

3.3 Implementations

Flow Preprocessing The Flow Preprocessing module is responsible for preparing the IP flows and network configurations for the flow replaying phase. It extracts the IP flows from the trace records according to the host/network representation list that can be configured by the TCPopera user or the self-configuration option. During the IP flow extraction, the Flow Preprocessing module also collects traffic parameters for each connections including round-trip time (RTT), transmission rate, packet loss rate, TCP receiver buffer size, path MTU, and other parameters for the initiation of the TCP control block. This module is also responsible for environment transformation including address remapping, and the defaultrouter setting. The IP addresses and MAC address should be rewritten based on them in order for the replayed packets to be accurately routed to its destination in the test environment.

IP Flow Processing The IP Flow Processing module is the key component of the interactive traffic replay feature of TCPopera. It creates a POSIX thread for each IP flow, and the creation time of each thread is strictly based on the empirical values observed in the input trace files [3]. The key feature of this module is the stateful replaying of TCP connections. The IP Flow Processing module emulates the TCP control block for each TCP connection using the TCP functions library. Because of this stateful replaying approach, TCPopera can guarantee no ghost packet generation. Another important feature of the IP Flow Processing module is how to preserve the inter-connection dependencies. the current implementation of TCPopera only supports the inter-connection dependencies within a single IP flow, and this feature will be extended to support them across IP flows in the following versions of TCPopera. To implement this feature, TCPopera

[3] The shared resources are strictly synchronized among IP flow threads using the thread synchronization mechanism

used the simple heuristic that the IP Flow Processing module strictly preserves the packet sequence of trace records within a single IP flow. This heuristic is based on the assumption that the trace records of a single IP flow captures the communication history between two hosts. For the inter-flow dependencies, the TCPopera only supports the inter-flow time that empirically measured.

TCPopera Control The TCPopera Control module is responsible for synchronizing the time and information among TCPopera nodes. This module provides an out-of-band communication channel to exchange the control message among them. First, TCPopera nodes synchronize their host/network representation list in order to find out their peer nodes are active [4]. Based on this synchronization result, each TCPopera node sorts out the replayable IP flows. Second, TCPopera nodes synchronize the timing of the replaying phase by transmitting the control packets via the out-of-band communication channel. In the current TCPopera implementation, one of the TCPopera nodes plays a synchronization server and controls the whole synchronization procedure.

Packet Injection/Capturing The Packet Injection/Capturing modules are the components to make it possible the live traffic replaying. Any outgoing packet from the TCPopera node is passed to the Packet Injection module to be written on the wire. If there is any modification in the packet, the checksum value is recalculated. The Packet Injection module is implemented using the libnet library, a high-level API to construct and inject network packets [20]. The libnet library helps construction new packets and modifying existing packets by providing simplified interfaces. Likewise, any incoming packet destined to the virtual addresses of a TCPopera node is captured by the Packet Capturing module and passed to the flow processor. This module is implemented using one of most widely used packet capturing utilities, pcap [21]. Since each TCPopera node can have multiple virtual network addresses, the pcap process should set the filtering rules to only capture packets destined to its virtual addresses.

TCP Functions The TCP functions library provides TCP functionalities needed to emulate the TCP control block for each TCP connection. This library includes most of TCP features related to TCP timers, timeout & retransmission, fast retransmit & fast recovery, flow & congestion control, and RTT measurement. The current implementation of the TCP functions library is heavily based on the TCP implementation of BSD4.4-Lite release, described in [22]. The following list shows the implementation details about the TCP functions library.

– **TCP timers**: TCP maintains seven TCP timers for each connections. TCPopera implemented two TCP timer functions: one is called every 200ms (the fast timer) and the other is called every 500ms (the slow timer). While the delayed ACK timer is implemented using the fast timer, other six timers

[4] To replay an IP flow, two TCPopera nodes are needed to represent both endpoints.

are implemented using the slow timer. Based on the TCP implementation in [22], we implemented the six timers excluding the delayed ACK timers using four timer counters that decrement the number of clock ticks whenever the slow timer expires.

– **Timeout & retransmission**: Fundamental to TCP's timeout and retransmission is the measurement of RTT experienced on a given connection because the retransmission timer has values that depend on the measure RTT for the connection. The retransmission timer is updated by measuring RTT for data segments and keeping track of smoothed RTT estimator and smoothed mean deviation estimator [23, 24]. If there is any outstanding TCP data segment unacknowledged when the retransmission timer expires, TCPopera retransmits the data segment.

– **Fast retransmit & fast recovery**: In TCP, it is assumed that three or more duplicate ACKs in a row is a strong indication of a packet loss. The TCP sender then retransmits the missing segment without waiting for a retransmission timer expires. Next, the congesting avoidance, but not slow start is performed. This is called *fast retransmit* and *fast recovery.* TCPopera implements these two TCP features according to the modified TCP congestion avoidance algorithms proposed in 1990 [25].

– **Flow & congestion control**: Congestion avoidance is the flow control imposed by the sender, while the advertised window is the flow control imposed by the receiver. The former is based on the sender's assessment of perceived network congestion, and the latter is related to the amount of available buffer space at the receiver for the connection. TCPopera supports slow start and congestion avoidance that are independent algorithms with different objectives. Congestion avoidance and slow start require that two variables for each connection: a congestion window (*cwnd*) and a slow start threshold size (*ssthresh*). When the congestion is indicated by a timeout or the reception of duplicate ACKs, both variables are adjusted.

– **RTT measurement**: Since RTT measurement is fundamental to TCP's timeout and retransmission, the accuracy of the RTT measurement is important. As the most Berkeley-driven TCP implementation, TCPopera measures only one RTT value per connection at any time. The timing is done by incrementing a counter every time according to the slow timer (500ms tick). TCPopera calculates the retransmission timeout (RTO) by measuring RTT of data segments and keeping track of the smoothed RTT estimator and a smoothed mean deviation estimator[23]. Besides the retransmission timer, the persist timer also depends on the measured RTT values.

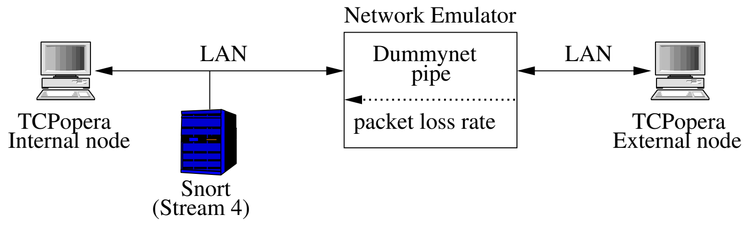


Fig. 3. The validation test environment

4 Validation

In this section, we validate the ability of TCPopera to reproduce traffic that statistically similar to the input trace and the effectiveness of TCPopera traffic to evaluate the security products. For the validation tests, we used 1999 MIT's IDEVAL dataset (IDEVAL99). Specifically, we used the first 12 hours of traffic collected from the inside network of the testbed at the first day of the first testing weeks (03/29/1999) [5]. In addition, we have been using a real traffic dataset contributed by ITRI (Industrial Technology Research Institute, Taiwan).

4.1 Test Environment

In our validation test environment, two TCPopera nodes are used as shown in Figure 3. The internal TCPopera node represents the home network appeared in the input trace and the external TCPopera node represents all external hosts in the input trace. Both TCPopera nodes runs on the machine with 2.0 GHz Intel Pentium 4 processor with 768MB RAM installed. The Internal TCPopera nodes runs on Redhat 8.0 (with 2.4.18 kernel) and the external one runs on Redhat 9.0 (with 2.4.20 kernel). Two TCPopera nodes are directly connected to each interface of the dual-homed FreeBSD 5.0 Firewall (`ipfw`), running on 455MHz Pentium II Celeron processor with 256MB RAM installed, with the Dummynet support to emulate network conditions. During the test, we used Snort 2.3, a public-domain IDS as the target security system to evaluate. The Snort 2.3 ruleset was used and the stream4 analysis is enabled to test its stateful operations.

4.2 Results

Reproductivity test First, we validate the traffic reproductivity of TCPopera. We reused the TCP connection-level parameters to emulate the TCP control block for each connection. Second, we applied 1% packet loss for the inbound traffic at the network emulator to test how TCPopera reacts against one of

[5] The reason we used IDEVAL99 in our test is because why this dataset contains attack traffic synthetically generated to test IDS.

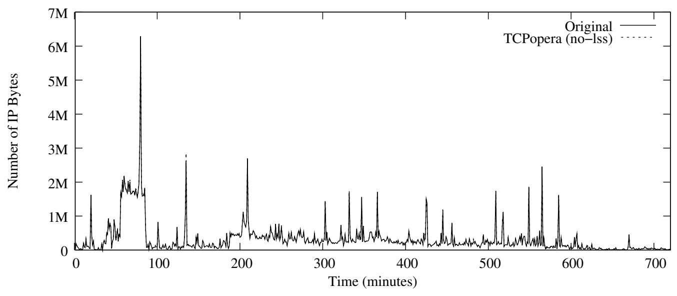
common network feedback. Table 4.2 shows the simple comparison of traffic volume and TCP connections between the input trace and the replayed traffic by TCPopera. Hereafter, we refer them to *Input trace*, *TCPopera (no-loss)*, and *TCPopera (1%-loss)*.

Table 1. Comparison of traffic volume and TCP connections between the input trace and the replayed traffic.

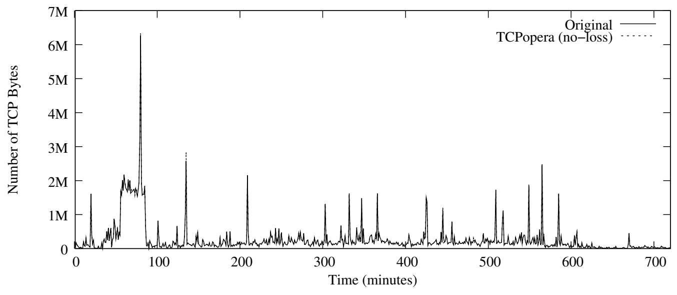
Category	Input trace	TCPopera no loss	TCPopera 1 % loss
IP Packets	1,502,584	1,552,882	1,531,388
IP Bytes	234,434,486	234,991,187	232,145,926
TCP Packets	1,225,905	1,276,195	1,254,762
TCP Bytes	194,927,209	195,483,762	192,647,088
UDP Packets	276,286	276,294	276,234
UDP Bytes	39,474,602	39,475,286	39,466,797
ICMP Packets	393	393	392
ICMP Bytes	32,675	32,139	32,041
TCP connections replayed	18,138	18,138	18,043
TCP connections completed	14,974	14,971	14,796

According to the TCP packets categories, both TCPopera traffic send more TCP packets than the Input trace. The increase of TCP packets in both TCP-opera traffic is caused by the TCP control packets such as the delayed ACKs during the TCP control block emulation. When we compare the increase of the TCP packets to that of TCP bytes, we can figure out more TCP packets are the control packets (e.g. pure ACKs). This behavior of TCPopera is observed more clearly in the long-lived interactive TCP connections such as telnet, ssh applications. For the TCPopera (1%-loss) traffic, the TCP bytes are less than that in the Input trace while the number of TCP packets replayed is greater than that of the Input trace. This difference is caused by the dropping of many short-lived connections. That is, the connection drop from the packet loss during 3-way handshaking caused the connection drop, and the data packets in the dropped connection has never been replayed.

For the further analysis, we plotted the traffic volume over the 1 minute interval. First, we compared the TCP/IP traffic volume of TCPopera (no-loss) to the Input trace in Figure 4. According to the traffic volume graphs in both TCP and IP bytes, we observed that TCPopera successfully reproduced the traffic similar to the Input trace. Also, we plotted the comparison between the Input trace and the TCPopera (1%-loss) traffic in Figure 5. Unlike the comparison result in Figure 4, Figure 5 showed the significant difference at the second hour of the replaying. We believe this is effect of the packet drop at the network emulator. To verify this claim, we carefully investigated the input trace and learned the large amount of short-lived HTTP connections has been replayed during the



(a) IP Bytes



(b) TCP Bytes

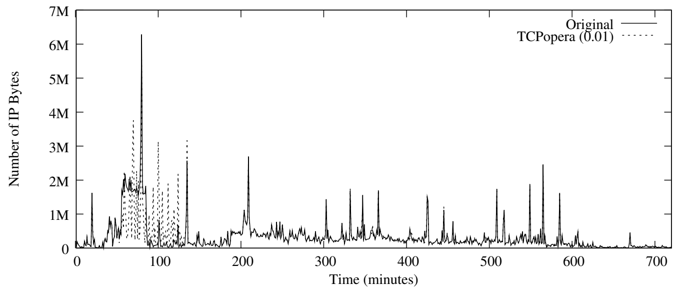
Fig. 4. Comparison of traffic volume between the Input trace and TCPopera (no-loss)

second hour of the Input trace [6]. Packet losses at the network emulator changed the short-lived connections to live longer to complete the connection, and further the connection drop caused by the SYN packet loss delayed the replaying time of following connections in the same IP flow. This is because why the TCPopera node waits for the connection-establishment timer (75 seconds) expires before dropping connection [7]. This TCPopera behavior changes the traffic pattern after the packet loss event and the amount of changes grows when the the packet loss happens where the density of short-lived connections is high.

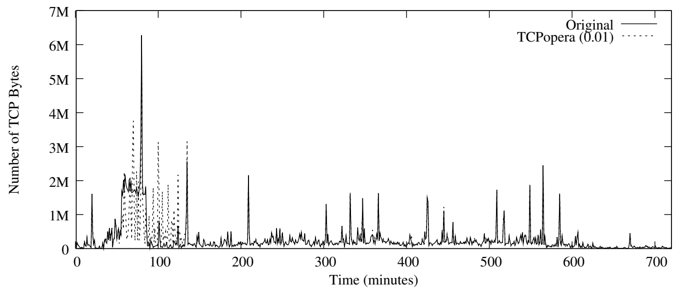
The effect of the packet loss event to TCPopera is more clear when we compare the two distributional properties, inter-connection time and the session duration, as shown in Figure 6. Both the inter-connection time and the session duration showed the similar distributional characteristics in that the number of

[6] About 30% of connections in the Input trace has been replayed during the second hour.

[7] Because of the heuristic for inter-connection dependencies support within a single IP flow, the following connections after the packet loss is until TCPopera figures out how to handle the current connection.



(a) IP Bytes

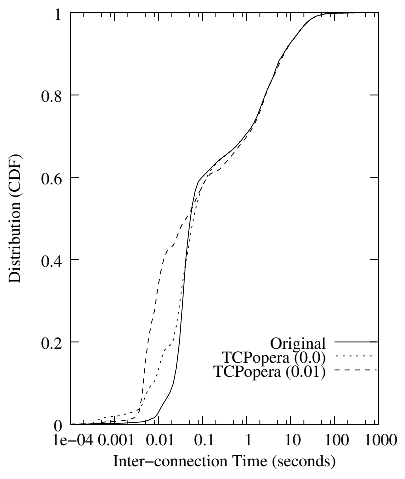


(b) TCP Bytes

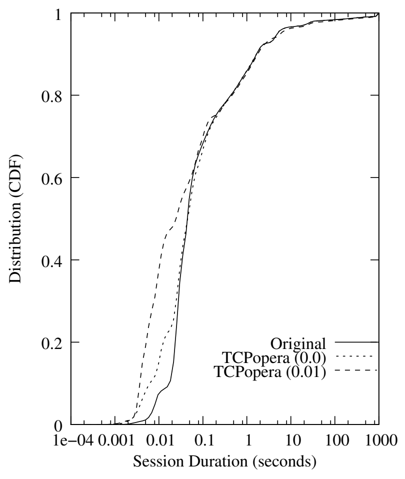
Fig. 5. Comparison of traffic volume between the Input trace and TCPopera (1%-loss)

samples less than 0.1 second increases in both distributional graphs. When there is a packet loss or a TCP connection drop by a SYN packet loss, the following connections within the same IP flow are replayed faster than the Input trace in order to keep up with the original transmission speed.

Figure 7 shows the example of how the Input trace has been changed by the packet loss handling of TCPopera. For instance, let us assume that TCPopera is about to replay 5 connections within a single IP flow. Each connection lived for 0.4 second seconds with 0.1 second inter-connection time, so the total replaying time planned is 2.4 seconds. Now, let us suppose that the first TCP connection (C1) experienced the packet loss event and its session duration is extended to 1.4 seconds to retransmit the packet. Then, there are 4 more connections to be replayed within 1 seconds, and the inter-connection time and session duration of the following 4 connections should be adjusted to fit in the 1 second period. This TCPopera behavior has changed the traffic pattern in the second hour of the Input trace where large amount of short-lived connections are located. This change in the traffic pattern is more serious if the packet loss event causes the



(a) Inter-connection distribution



(b) Session duration distribution

Fig. 6. Comparison of two distributional properties: Inter-connection time & Session duration (log-scale)

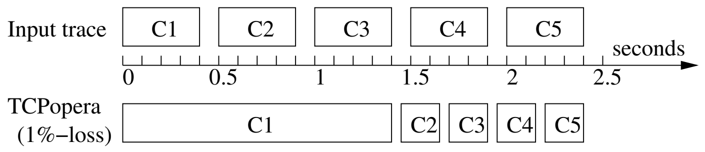


Fig. 7. Example of the changes in the session duration and inter-connection time after TCPopera handles the packet loss event.

connection drop by the SYN packet loss because TCPopera should wait until the connection-establishment timer expires which is a typically set to 75 seconds.

Effectiveness test To test the effectiveness of TCPopera traffic in evaluating security products, we test Snort 2.3, with the latest detection ruleset (2.3) including the stream4 analysis. We first ran Snort over the Input trace, and then we employed Snort into our test environments as shown Figure 3 while TCPopera replays the traces. For the test, we used two dataset, one is the IDEVAL99 dataset and the other is ITRI dataset. Due to the space limitation, we only provide the analysis results of the ITRI dataset in Table 4.2. The ITRI dataset contains 20-minute real traffic capturing TCP connections between the internal host (140.96.114.97) and about 2000 external hosts. We added the test results of the IDEVAL99 dataset in the appendix.

Table 2. The test results on the ITRI dataset over various test conditions. All Snort rules and stream 4 analysis are enabled during the test.

Signature	Classification	Number of alerts			
		Input	TCPopera		
		trace	no loss	1% loss	3% loss
ICMP Destination/Port Unreachable	misc-activity	5	5	5	5
P2P eDonkey Transfer	policy-violation	3	3	3	3
ICMP Destination Unreachable Fragmentation needed but DF bit is set	misc-activity	1	1	1	1
ICMP Destination/Host Unreachable	misc-activity	2	2	2	2
(stream4) Possible retransmission detection	unclassified	38	212	200	181
(stream4) WINDOW violation detection	unclassified	488	3	1	4
Total		537	226	212	196

Interestingly, Snort only showed difference in both stream4 signatures [8]. So, we performed the deep-inspection on both stream4 signatures to identify the reasons that caused the difference in the test results. First, *(stream4) possible retransmission detection* signature generates an alert when it observes the retransmission of packet that has been already acknowledged. This signature is originally designed to capture potential packet replaying attack. As shown in Table 4.2, while Snort issued 38 alerts for the input trace, it produced about 5-6 times more alerts for other TCPopera traffic. Based on the careful inspection of alerts, we found out that the difference between the input trace and the TCPopera traffic was from the TCPopera's delayed ACK functionality. TCPopera invokes the delayed ACK function every 200ms and transmits the delayed ACK if there are data segments has not been acknowledged. However, this feature will cause a confusion to Snort if either the receiver of data segments holds the acknowledgment, or data segments/acknowledgments have been lost during transmission. Figure 8 shows the example of how TCPopera converts the original trace into the new one with delayed ACK feature.

Another interesting observation for the *(stream4) Possible retransmission detection* signature is the number of alerts is decreasing while we increase the packet loss rate at the network emulator. This is the expected result because we believe that the connection drop rate by the failure of connection establishment is higher with the higher packet loss rate. According to our observation, TCPopera replayed 3849 TCP connections and completed 1762 connections for TCPopera (no-loss) traffic. For both TCPopera (1%-loss) and TCPopera (3%-loss) traffic, TCPopera replayed 3836 and 3812 connections, and completed 1726 and 1668, respectively [9]. The reason of the decrease in the number of replayed TCP con-

[8] For all non-stream4 signatures, Snort triggered alerts from the same packets.

[9] The number of completed connections includes the gracefully closed connections (using FIN) and the ungracefully closed connections (using RST) after the completion

```
01:20:49.403876 IP 24.7.116.14.4662 > 140.96.114.97.1134: P 376:431(55) ack 324 win 65212
01:20:49.405044 IP 140.96.114.97.1134 > 24.7.116.14.4662: P 324:326(2) ack 431 win 65105
01:20:50.723002 IP 140.96.114.97.1134 > 24.7.116.14.4662: P 324:364(40) ack 431 win 65105
01:20:50.933149 IP 24.7.116.14.4662 > 140.96.114.97.1134: P 431:449(18) ack 364 win 65172
```

(a) Input trace: the acknowledgment of two data segments from 140.96.114.97 has been piggy-backed to the last packet.

```
17:24:28.866305 IP 24.7.116.14.4662 > 140.96.114.97.1134: P 376:431(55) ack 324 win 65212
17:24:29.389348 IP 140.96.114.97.1134 > 24.7.116.14.4662: P 324:326(2) ack 431 win 65105
17:24:29.789172 IP 24.7.116.14.4662 > 140.96.114.97.1134: . ack 326 win 65212
17:24:30.711409 IP 140.96.114.97.1134 > 24.7.116.14.4662: P 324:364(40) ack 431 win 65105
17:24:30.733341 IP 24.7.116.14.4662 > 140.96.114.97.1134: . ack 364 win 65212
```

(b) TCPopera (no-loss): TCPopera sends the delayed ACKs for both data segments from 140.96.114.97.

Fig. 8. TCPdump output to compare the difference between the input trace and the TCPopera-replayed traffic with respect to the TCPopera's delayed ACK function.

nections is because the replayed trace is captured from the inner interface of our firewall in the testbed.

The next stream4 signature is *WINDOW violation detection*, which is originally created to detect the suspicious behavior to write the data the outside of the window. In fact, this behavior was often witnesses in the TCP implementation of the Microsoft Windows Operating Systems. The stream4 reassembler of Snort issues an alert for this signature if the following condition is true.

$$(\text{sequence no.} - \text{last acked no.}) + \text{data length} > \text{receiver's window size}$$

As observed in Table 4.2, there is a big difference between the input trace and the TCPopera traffic. After the deep-inspection on the alerts, we found out that there are only 18 legitimate alerts in the Input trace and others are false positives. These false positives are caused by the incorrect initialization for the incomplete TCP connection. We will explain more details related to this matter with the following example [10]. Figure 9(a) shows an example of TCPdump outputs that caused many false positives in the input trace.

The problem of Snort in processing the TCP connection in Figure 9(a) is that a variable used for the condition checking is not properly initialized. When Snort reads the last packet in Figure 9(a), it executes the following program segment in *spp_stream4.c*, which mistakenly changes the listener(220.141.33.182)'s state to the ESTABLISHED and make Snort think the 3-way handshaking is finally

of the connection establishment. TCPopera also outputs the details about the closing results of each TCP connections.

[10] All false positives for the WINDOW violation detection signature was from the same reason as the example in Figure 9(a).

```
01:12:13.811379 IP 140.96.114.97.3269 > 220.141.33.182.4662: S 4166059610:4166059610(0) win 65535 <mss 1460,nop,nop,sackOK>
01:12:13.879016 IP 140.96.114.97.3269 > 220.141.33.182.4662: P 1:102(101) ack 3686742391 win 65535
01:12:14.018670 IP 140.96.114.97.3269 > 220.141.33.182.4662: P 102:142(40) ack 3686742471 win 65455
01:12:14.093459 IP 220.141.33.182.4662 > 140.96.114.97.3269: P 3686742471:3686742513(42) ack 142 win 64659
01:12:14.104423 IP 140.96.114.97.3269 > 220.141.33.182.4662: P 142:164(22) ack 3686742513 win 65413
```

(a) Input trace: The client (140.96.114.97) keep sending packets without receiving any packet from the server (220.141.33.182). The last packet invokes the alert

```
17:15:53.534364 IP 140.96.114.97.3269 > 220.141.33.182.4662: S 4166059610:4166059610(0) win 65535 <mss 1460,nop,nop,sackOK>
17:16:00.250345 IP 140.96.114.97.3269 > 220.141.33.182.4662: S 4166059610:4166059610(0) win 65535 <mss 1460,nop,nop,sackOK>
17:16:27.310699 IP 140.96.114.97.3269 > 220.141.33.182.4662: S 4166059610:4166059610(0) win 65535 <mss 1460,nop,nop,sackOK>
17:17:08.257095 IP 140.96.114.97.3269 > 220.141.33.182.4662: R 4166059611:4166059611(0) win 65535
```

(b) TCPopera (no-loss): The internal TCPopera node retransmits the first SYN packets several times because there was no answer from the server. Then, the connection is reset by the external TCPopera node.

Fig. 9. TCPdump output to compare the difference between the input trace and the TCPopera-replayed traffic with respect to the handling of incomplete TCP connection.

done at this point. Then, it checks the window violation condition on this packet. But, the variable *last_ack* has never been initialized and remains 0 all the time. So, when Snort checks the condition $((4166059752 - 0) + 22 > 64659)$, the last packet violates the condition because the *last_ack* is still 0. In the input trace, there were many instances of this example and they caused 470 false positives for the input trace. In contrast, Snort did not generates this type of false positives for the TCPopera traffic because the TCPopera could not complete the 3-way handshaking as shown in Figure 9(b). The internal TCPopera node retransmitted the first SYN packet until the connection-establishment timer expires and then sent the RST packet [11].

```
switch(listener->state) {
  . . . . . .
  case SYN_RCVD:
       if(p->tcph->th_flags & TH_ACK) {
          listener->state = ESTABLISHED;
          DEBUG_WRAP(DebugMessage(DEBUG_STREAM_STATE,
                     "   %s Transition: ESTABLISHED\n", l););
          retcode |= ACTION_COMPLETE_TWH;
       }
       break;
  . . . . . .
}
```

[11] The RST packet helps TCPopera nodes to remove the incomplete connection from the connection list.

```
17:18:18.947066 IP 140.96.114.97.3756 > 200.82.109.224.http: S 4226095698:4226095698(0) win 65535 <mss 1460,nop,nop,sackOK>
17:18:19.142875 IP 200.82.109.224.http > 140.96.114.97.3756: S 597332127:597332127(0) ack 4226095699 win 8000 <mss 1460>
17:18:19.143128 IP 200.82.109.224.http > 140.96.114.97.3756: R 597332128:597332128(0) win 1
17:18:19.143891 IP 140.96.114.97.3756 > 200.82.109.224.http: . ack 1 win 65535
17:18:19.144149 IP 140.96.114.97.3756 > 200.82.109.224.http: P 1:102(101) ack 1 win 65535
```

Fig. 10. TCPdump output from one of examples of false positives in TCPopera traffic.

Yet another issue in the *(stream4) WINDOW violation detection* signature is related to the RST handling. Basically, the stream4 reassembler of Snort updates the window size for each TCP segment even if it is RST segment by executing the following program segment in *spp_stream4.c*. After processing the RST segment in Figure 10, the window size of the client (`ssn->client.win_size`) is set to 1 because the RST segment is from the server (200.82.109.224) [12]. Later, the last TCP segment is verified against the window violation condition, Snort issues the alert because the condition is true $((4226095699 - 4226095699) + 101 > 1)$.

```
if((direction = GetDirection(ssn, p)) == SERVER_PACKET){
  p->packet_flags |= PKT_FROM_SERVER;
  ssn->client.win_size = ntohs(p->tcph->th_win);
  DEBUG_WRAP(DebugMessage(DEBUG_STREAM,  "server packet: %s\n", flagbuf););
}
else{
  p->packet_flags |= PKT_FROM_CLIENT;
  ssn->server.win_size = ntohs(p->tcph->th_win);
  DEBUG_WRAP(DebugMessage(DEBUG_STREAM,  "client packet: %s\n", flagbuf););
}
```

Based on our analysis of alerts issued by the *WINDOW violation detection* signature, we found two implementation errors in the Snort's stream4 reassembling feature. First, the stream4 reassembler failed to keep track of the connection state when it faces the incomplete connection as shown in Figure 9(a). Second, it has the problem with the RST segment handling when it processes the connection shown in Figure 10. Fixing the type of errors is not simple because they are tightly related to the variables used for various stream4 inspections. We believe that the effectiveness test results successfully demonstrated how TCPopera can be deployed in the testing environments for intrusion detection and prevention systems. Currently, we are planning more in-line devices testing, especially IPS testing, using TCPopera in order to evaluate their blocking capabilities.

5 Conclusion & Future work

TCPopera is a new traffic replay tool for reproducing IP traffic based on various flow-level and connection-level traffic parameters extracted from the input trace

[12] Another strange behavior from Snort is that it does not reset the connection at this point because Snort think this RST packet is invalid.

records. These parameters can be either reused to reproduce traffic or changed to create new traffic. TCPopera sustains the merits of the trace-based traffic replaying that is fast, reproducible, and highly accurate in terms of address mixes, packet loads, and other traffic characteristics. Also, it overcomes the drawback of conventional traffic replay tools, it is hard to adjust traffic for other test conditions from the input traces, by providing various traffic models can be used to tune the trace records during replaying. Unlike conventional traffic replay tools, TCPopera is originally designed to replay traffic on the live test environments. Because TCPopera can replay the trace records without violating protocol semantics, the live test environments where the accuracy of protocol semantic is of fundamental importance for the accuracy of test results are the potential customers of TCPopera.

We demonstrated the ability of the current TCPopera implementation throughout the validation tests. We compared the TCPopera traffic to the input trace records in terms of traffic volume and other distributional properties. In the traffic reproduction test, we found that TCPopera successfully reproduced IP flows in terms of traffic parameters without breaking protocol semantics. We also demonstrated how TCPopera can be deployed in the live test environments for evaluating security products through the effectiveness test. We observed that Snort generated different results when we changed the test conditions using TCPopera and the network emulator and some of them are from the implementation errors.

The TCPopera project consists of multiple development phases and we have completed its first phase whose goal was to implement the core modules for interactive traffic replaying. There are several areas for the next phase of the TCPopera development. The first area is to extend out traffic models including UDP traffic models to improve the accuracy of IP flow reproduction. Supporting inter-connection dependencies across IP flows also has the high priority in the next TCPopera development phase. Furthermore, we will add an "evasive transformation" module into TCPopera such that we can apply different evasive techniques to any Internet traffic traces. We also intend to implement the TCPopera GUI to help the TCPopera configuration (control) and extend TCPopera evaluation tools by integrating them into the TCPopera GUI. As we mentioned earlier, we have one commercial vender using TCPopera almost daily under their development cycle. On the other hand, recently ITRI is using TCPopera to test Netscreen IPS boxes. We are also planning to perform more in-line devices testing including ITRI's Network Processor Units (NPU)-based IPS prototype.

References

1. The InterOperability Laboratory (IOL) homepage: http://www.iol.unh.edu. Accessed March 12, 2005.
2. The Wisconsin Advanced Internet Laboratory (WAIL) homepage: http://wail.cs.wisc.edu. Accessed March 12, 2005.
3. The Network Simulator (NS-2) homepage: http://www.isi.edu/nsnam/ns. Accessed March 12, 2005.

4. Scalable Simulation Framework Research Network (SSFNET) homepage: http://www.ssfnet.org. Accessed March 12, 2005.
5. Vahdat, A., Yocum, K., Walsh, K., Mahadevan, P., Kosti, D., Chase, J., Becker, D: Scalability and accuracy in a large-scale network emulator. SIGOPS Oper. Syst. Rev. **36** (2002) 271–284.
6. White, B., Lepreau, J., Stoller, L., Ricci, R., Guruprasad, S., Newbold, M., Hibler, M., Barb, C., Joglekar, A.: An Integrated Experimental Environment for Distributed Systems and Networks. OSDIO2, Boston, MA, (2002) 255–270.
7. Peterson, L., Anderson. T., Culler, A., Roscoe, T.: A blueprint for introducing disruptive technology into the Internet. SIGCOMM Comput. Commun. Rev. **33(1)** (2003) 59–64.
8. Touch, J.: Dynamic Internet overlay deployment and management using the X-Bone. ICNP '00: Proceedings of the 2000 International Conference on Network Protocols (2000) 59–67.
9. Bajcsy, R., Benzel, T., Bishop, M. Braden, B., Brodley, C., Fahmy, S., Floyd, S., Hardaker, W., Joseph, A., Kesidis, G., Levitt, K., Lindell, B., Liu, P., Miller, D., Mundy, R., Neuman, C., Ostrenga, R., Paxson, V., Porras, P., Rosenberg, C., Tygar, J. D., Sastry, S., Sterne, D., Wu, S. F.: Cyber defense technology networking and evaluation. Commun. ACM **47(3)** (2004) 58–61.
10. POSIX Thread tutorial page: http://www.llnl.gov/computing/tutorials/workshops/workshop/pthreads/MAIN.html. Accessed March 13, 2005.
11. Rizzo, L.: Dummynet: a simple approach to the evaluation of network protocols. ACM Computer Communication Review **27(1)** (1997) 31–41.
12. MIT Lincoln Labs. DARPA Intrusion Detection Evaluation.: http://www.ll.mit.edu/IST/ideval/. Accessed March 13, 2005.
13. The Snort homepage: http://www.snort.org/. Accessed March 13, 2005.
14. The TCPREPLAY & FLOWRELAY homepage: http://tcpreplay.sourceforge.net/. Accessed March 14, 2005.
15. The TCPDUMP homepage: http://www.tcpdump.org/. Accessed March 14, 2005.
16. The libpcap project homepage: http://sourceforge.net/projects/libpcap/. Accessed March 14, 2005.
17. Feng, Wu-chang, Goel, A., Bezzaz, A., Feng, Wu-chi, Walpole, J.: TCPivo: a high-performance packet replay engine. MoMeTools '03: Proceedings of the ACM SIGCOMM workshop on Models, methods and tools for reproducible network research (2003) 57–64.
18. Cheng, Y., Hölzle, U., Cardwell, N., Savage, S., Voelker, C. M.: Monkey See, Monkey Do: A Tool for TCP Tracing and Replaying. USENIX Annual Technical Conference, General Track (2004) 87–98.
19. The Tomahawk Test Tool homepage: http://tomahawk.sourceforge.net/. Accessed March 14, 2005.
20. The LIBNET project homepage: http://www.packetfactory.net/libnet/. Accessed March 16, 2005.
21. The libpcap project homepage: http://sourceforge.net/projects/libpcap/. Accessed March 14, 2005.
22. Stevens, W. R., Write, G. R.: TCP/IP illustrated (vol. 2): the implementation. Addison-Wesley Longman Publishing Co., Inc. (1995).
23. Jacobson, V.: Congestion avoidance and control. SIGCOMM Comput. Commun. Rev. **18(4)** (1988) 314–329.

24. Jacobson, V.: Berleley TCP Evolution from 4.3-Tahoe to 4.3-Reno. Proceedings of the Eighteenth Internet Engineering Task Force, University of British Columbia, Vancouver, Canada (1990).
25. Jacobson, V.: Modified TCP Congestion Avoidance Algorithm. end2end-interest mailing list, (1990).
26. Hong, S. S., Wong, F., Wu, S. F., Lilja, B., Yohansson, T. Y, Johnson, H., Nelsson, A.: TCPtransform: Property-oriented TCP traffic transformation. Accepted to be presented in DIMVA'05.

Appendix: The effectiveness test result of IDEVAL99 dataset

Table 3. The detection results from Snort over various test conditions. All Snort rules and stream 4 analysis are enabled during the test.

Signature	Classification	Number of alerts		
		Input trace	TCPopera	
			no loss	1% loss
ICMP Destination Port Unreachable	misc-activity	89	89	89
ICMP PING BSDtype	misc-activity	17	17	17
ICMP PING *NIX	misc-activity	17	17	17
ICMP PING	misc-activity	152	152	152
INFO web bug 0x0 gif attempt	misc-activity	185	185	182
ICMP Echo Reply	misc-activity	152	152	151
INFO TELNET access	not-suspicious	290	289	286
INFO TELNET login incorrect	bad-unknown	47	47	46
POLICY FTP anonymous login attempt	misc-activity	118	118	117
CHAT IRC nink change	policy-violation	7	7	7
CHAT IRC message	policy-violation	281	280	280
ATTACK-RESPONSES Invalid URL	attempted-recon	2	2	2
ATTACK-RESPONSES 403 Forbidden	attempted-recon	5	5	5
SHELLCODE x86 NOOP	shellcode-detect	1	1	1
SCAN FIN	attempted-recon	15	0	0
(stream4) STEALTH-ACTIVITY (FIN scan) detection	attempted-recon	15	0	0
X11 open	unknown	1	1	1
(stream4) Possible retransmission detection	unclassified	2	0	4
(stream4) WINDOW violation detection	unclassified	0	4	6
INFO FTP Bad login	bad-unknown	12	12	11
FTP .rhosts	suspicious-filename-detect	1	1	1
WEB-MISC http directory traversal	attempted-recon	1	1	1
BACKDOOR MISC Solaris 2.5 attempt	attempted-user	1	1	1
ATTACK-RESPONSES id check returned userid	bad-unknown	1	1	1
ATTACK-RESPONSES directory listing	bad-unknown	30	30	30
Total		1442	1412	1408