*29th Annual International Symposium on Fault-Tolerant Computing, 15-18 June 1999, Madison, Wisconson*

# Testing Protocol Implementation Robustness

John Linwood Griffin
*Laboratory for Computer Systems*
*Carnegie Mellon University*
*Pittsburgh, Pennsylvania*
*griffin2@ece.cmu.edu*

## Abstract

We describe preliminary work toward a new robustness testing tool, PIRANHA, that exercises boundary and exceptional conditions of network protocol implementations. This automated tool will provide system developers and maintainers the ability to repeatably identify and eliminate robustness failures in protocol subsystems.

## 1. Introduction

Network protocols are at the heart of our global communications infrastructure. Protocols provide guaranteed services such as in-order byte delivery and error detection to network applications. A service failure at any level of the protocol stack could corrupt data, impact system security, or worse. It is therefore essential that protocol implementations be robust against failure, especially when processing invalid or extraordinary information from the network--whether from software defects, hardware failure, or intentional attacks. Unfortunately, the increasing complexity of protocols leads us to suspect implementations may not be as robust as we would like.

### 1.1. What is protocol robustness?

Although *robustness* is widely cited throughout computer systems literature, we found no commonly accepted use or definition of the term. For authority, we consult two sources: the IEEE to provide a general definition of robustness and the "Host Requirements" Internet Standard to apply the concept of robustness to Internet protocols. The IEEE defines robustness in [1] as "the degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions." From this definition, our focus is on the presence of *invalid* and *unspecified inputs* at the protocol interface. Specifically, we are interested in invalid packet header information, as discussed by the Internet Engineering Task Force in [2]:

> [C]onsider a protocol specification that contains an enumeration of values for a particular header field -- e.g., a type field, a port number, or an error code; this enumeration must be assumed to be incomplete. Thus, if a protocol specification defines four possible error codes, the software must not break when a fifth code shows up. An undefined code... must not cause an error.

We extend this emphasis on handling invalid headers to exercise possible robustness deficiencies in a protocol implementation.

### 1.2. Exceptional network information

Exceptional values in packet headers are a certainty on most networks. Invalid headers originate from several sources, including in-transit packet corruption, buggy hosts, and malicious network users.

**1.2.1. In-transit packet corruption.** Paxson [3] measures the Internet data packet corruption rate at 0.02%, or one corrupted packet per 5,000 transmissions. Because of the limited 16-bit checksum available in the Transmission Control Protocol (*TCP*) header, Paxson postulates that about one in every 300 million TCP packets are accepted *with corruption*. Single-bit or multiple-bit errors could occur anywhere in network control or data packets.

**1.2.2. Buggy hosts and malicious users.** It is exceedingly difficult to produce an error-free network application or protocol implementation. In addition, many modern systems allow user-level applications to transmit arbitrary packets onto the network. As a result, developers must guard against malicious programmers and nonconformant protocol implementations, even when lower-level protocols guarantee error-free data transmission. (We note the availability and wide distribution of malicious user-level protocol routines in forums such as [4].)

Consider a user-level TCP implementation that sets the TCP *urgent pointer* beyond the last byte of the current TCP segment (for background, see [5]). If the receiving protocol mishandles this pointer, a serious failure could occur--for example, a receiver that attempts to reference data located beyond the packet buffer may experience a segmentation violation and a failure of the protocol processing module.

### 1.3. Protocol complexity

Network protocols are complex, and their implementations are more so. Systems must conform to rules governing states and state transitions, flow and congestion control, fragmentation and reassembly, and many others. Moreover, the set of requirements placed on implementations grows with time, as evidenced by the steady stream of books and Internet Requests For Comments (RFC's) discussing protocol issues and behavior (see, for example, [6] and [7]).

With this increasing complexity comes a trade-off for the system developer: robustness versus development effort and code efficiency. Constrained development schedules and performance issues are key concerns for many developers. Programmers often do not have the time or resources to exhaustively identify and test all possible critical conditions. As a result, an engineer modifying a system for changed requirements or performance optimizations may unintentionally leave a protocol implementation vulnerable to failure when that protocol encounters invalid packets.

### 1.4. Our work

In light of such hazards, the Internet Engineering Task Force emphasizes the Internet Robustness Principle by directing systems to be "liberal" in the amount of information and misinformation they can accept and handle [2]. The authors note that protocol implementations "should be written to deal with every conceivable error," as probability dictates that packets *will* arrive with invalid combinations of errors and attributes.

Unfortunately, few tools are available to aid developers in determining a particular implementation's robustness. Most protocol testing tools focus on measuring performance or behavioral and functional correctness, not robustness (see, for example, the TCP testing tools identified in [8]). Developers require a tool to ensure that structural decisions, optimizations, and functional changes do not have an adverse effect on system robustness.

We are developing *PIRANHA*--an automated Protocol Implementation Robustness And Network Hardness Analysis tool--for testing remote protocol implementations for robustness failures. By establishing connections with remote systems and probing those connections with extraordinary packets during protocol steady-state and state-transition periods, PIRANHA will identify and reproduce robustness failures. Furthermore, the automated nature of PIRANHA will allow developers and system administrators to test any developmental or deployed system. Although our current work specifically targets TCP, we expect to find our methodology is extensible to general protocol testing.

## 2. Protocol robustness testing

In this section we identify related approaches to protocol testing and discuss our integration of the Ballista testing methodology into PIRANHA. We conclude with comments on the implementation of our prototype tool.

### 2.1. Related work

Much of the recent work in protocol implementation testing centers around the TCP/IP protocol suite. As a representative example of these, we note three alternative approaches to protocol testing.

The *active probing* approach applies black-box testing to TCP implementations. By using specially designed probe procedures to control the packet stream to a remote system and analyzing packet traces from the remote connection, Comer and Lin identify implementation flaws and protocol violations on remote systems [9]. Similarly, the *ORCHESTRA* tool employs script-driven software fault injection to test dependability and timing properties of protocols. ORCHESTRA places a fault injection layer directly in the protocol stack to introduce new test packets and perform filtering and manipulation on intercepted packets [10]. Both approaches observe behavioral characteristics such as time-out and keep-alive behavior, zero window probing, and message reordering and buffering response.

Alternatively, Paxson's *tcpanaly* tool takes a passive approach to behavioral analysis: studying packet traces of snooped connections to compare an implementation's behavior to that of other analyzed implementations [11]. This approach allows tcpanaly to observe behavioral variance in congestion window and congestion avoidance threshold maintenance, fast retransmission and fast recovery behavior, reaction to response delays, and other protocol characteristics.

PIRANHA compliments these approaches. For example, the active probing and ORCHESTRA tools manipulate high-level packet events to observe a system's behavioral response (*e.g.*, delaying and dropping packets, or inserting zero-window probe packets) whereas PIRANHA performs low-level packet manipulation to observe robustness failures (*e.g.*, modifying the "header length" field and updating the header checksum accordingly). PIRANHA also actively creates test packets for any purpose, while other tools have only a limited ability to dispatch invalid packets. We believe PIRANHA represents an important and neglected approach to testing protocol implementations.

### 2.2. The Ballista methodology

The Ballista project [12,13] provides proven testing techniques for isolating robustness failures in software modules. Through the systematic approach of applying

combinations of valid and invalid parameters at a module's interface and observing the module's high-level response, Ballista tools repeatably and deterministically identify robustness failure modes of a module under test. We extend the Ballista approach to network protocol testing: by viewing protocol header fields as analogous to module interface parameters, we can identify a set of valid and invalid test values and probe a remote protocol implementation with these values to determine the robustness of its response. As with existing Ballista tools, this will afford deterministic results (experiments will be controlled and repeatable), portability across multiple platforms (all implementations of a protocol use standard packet formats), extensibility to other protocols (*e.g.*, IP, UDP, or ICMP), and scalability with the number of parameters under test.

### 2.3. Implementation status

In order for PIRANHA to effectively control the composition of test packets and position the exact protocol state of a system under test, our testing tool must have extremely fine-grain control over its local networking activity. To achieve this, we are implementing PIRANHA under the Linux 2.2 operating system with *xio* user-level networking support. (The *xio* extensible I/O library is introduced in [14].) *Xio* allows network applications such as PIRANHA the option of manipulating packet composition and dispatch without worrying about low-level protocol implementation details.

We are addressing several open issues with our approach to robustness testing. For example, what is the correct definition of *success* or *failure* when probing protocols with valid and invalid packets? We have rejected the idea of testing the behavioral response of a protocol--*e.g.*, checking whether a sender correctly ceases transmissions when the receive window size reaches zero. Such an approach is neither portable among different protocols nor scalable with the number of parameters tested. As an alternative, we hope robustness failures map into classes similar to those of the CRASH severity scale (described in [13]). For example, we may discover a failure mode when an implementation sends a protocol reset packet or an application reset signal when processing a PIRANHA test packet. Another failure mode might involve a remote system ceasing all network activity in response to a single invalid packet. Further research is needed before we expect to understand how to taxonomize network robustness failures.

## Acknowledgments

This research is sponsored by the Pennsylvania Infrastructure Technology Alliance, a joint program of Carnegie Mellon and Lehigh University funded under the Commonwealth of Pennsylvania's Department of Community and Economic Development Contract No. 98-050-0018. The author thanks Greg Ganger and Laurel Fan for their work on the prototype PIRANHA tool, Garth Goodson for his continued developmental work on the *xio* user-level networking library, and Phil Koopman for providing scalability and determinism to our testing approach.

## References

[1] *IEEE Standard Glossary of Software Engineering Terminology*, IEEE Std 610.12-1990, Dec. 1990.

[2] R. Braden, ed., "Requirements for Internet Hosts -- Communication Layers", RFC 1122, Oct. 1989, pp. 12-13.

[3] V. Paxson, "End-to-End Internet Packet Dynamics", *Computer Communication Review*, 27(4), ACM SIGCOMM, Cannes, France, Sept. 1997, p. 142.

[4] Miff, "Homemade TCP Packets", *2600: The Hacker Quarterly*, 15(3), Fall 1998, pp. 6-9.

[5] G.R. Wright and W.R. Stevens, *TCP/IP Illustrated, Volume 2: The Implementation*, Addison-Wesley, Reading, Massachusetts, 1995, pp. 878-879.

[6] W. Stevens, *TCP/IP Illustrated, Volume 1: The Protocols*, Addison-Wesley, Reading, Massachusetts, 1994.

[7] M. Allman, V. Paxson, W. Stevens, "TCP Congestion Control", RFC 2581, April 1999.

[8] S. Parker and C. Schmechel, "Some Testing Tools for TCP Implementors", RFC 2398, Aug. 1998.

[9] D. Comer and J. Lin, "Probing TCP Implementations", *USENIX Summer 1994 Technical Conf.*, Boston, USA, June 1994.

[10] S. Dawson et al., "Testing of Fault-Tolerant and Real-Time Distributed Systems via Protocol Fault Injection", *26th Intl. Symp. on Fault-Tolerant Computing*, Sendai, Japan, 1996.

[11] V. Paxson, "Automated Packet Trace Analysis of TCP Implementations", *Computer Communication Review*, 27(4), ACM SIGCOMM, Cannes, France, Sept. 1997.

[12] N.P. Kropp, P.J. Koopman, and D.P. Siewiorek, "Automated Robustness Testing of Off-the-Shelf Software Components", *28th Intl. Symp. on Fault Tolerant Computing*, Munich, Germany, June 1998.

[13] P. Koopman and J. DeVale, "Comparing the Robustness of POSIX Operating Systems", *29th Intl. Symp. on Fault-Tolerant Computing*, Madison, USA, June 1999.

[14] F. Kaashoek et al., "Application Performance and Flexibility on Exokernel Systems", *Proc. 16th Symp. on Operating Systems Principles*, ACM SIGOPS, Saint-Malo, France, Oct. 1997, pp. 52-65.