Self-Configuring Network Traffic Generation

Joel Sommers
University of Wisconsin–Madison
jsommers@cs.wisc.edu

Paul Barford
University of Wisconsin–Madison
pb@cs.wisc.edu

ABSTRACT

The ability to generate repeatable, realistic network traffic is critical in both simulation and testbed environments. Traffic generation capabilities to date have been limited to either simple sequenced packet streams typically aimed at throughput testing, or to application-specific tools focused on, for example, recreating representative HTTP requests. In this paper we describe Harpoon, a new application-independent tool for generating representative packet traffic at the *IP flow level*. Harpoon generates TCP and UDP packet flows that have the same byte, packet, temporal and spatial characteristics as measured at routers in live environments. Harpoon is distinguished from other tools that generate statistically representative traffic in that it can *self-configure* by automatically extracting parameters from standard Netflow logs or packet traces. We provide details on Harpoon's architecture and implementation, and validate its capabilities in controlled laboratory experiments using configurations derived from flow and packet traces gathered in live environments. We then demonstrate Harpoon's capabilities in a router benchmarking experiment that compares Harpoon with commonly used throughput test methods. Our results show that the router subsystem load generated by Harpoon is significantly different, suggesting that this kind of test can provide important insights into how routers might behave under actual operating conditions.

Categories and Subject Descriptors: C.2.6 [Computer-Communication Networks]: Internetworking—*Routers*; C.4 [Performance of Systems]: Measurement techniques, Modeling techniques, Performance attributes

General Terms: Measurement, Performance

Keywords: Traffic Generation, Network Flows

1. INTRODUCTION

The network research community has a persistent need to evaluate new algorithms, systems and protocols using tools that (1) create a range of test conditions similar to those experienced in live deployment and (2) ensure reproducible results [15, 30]. Having appropriate tools for generating scalable, tunable and representative network traffic is therefore of fundamental importance. Such tools are critical in laboratory test environments (such as [10, 11]) where they can be used to evaluate behavior and performance of new systems and real networking hardware. They are also critical in emulation environments (such as [48, 49]) and simulation environments (such as [9, 12]) where representative background traffic is needed. Without the capability to consistently create realistic network test conditions, new systems run the risk of unpredictable behavior and unacceptable performance when deployed in live environments.

Current best practices for traffic generation have focused on either simple packet streams or recreation of a single application-specific behavior. Packet streaming methods such as those used in tools like `iperf` [6] consist of sequences of packets separated by a constant interval. These methods form the basis for standard router performance tests such as those recommended in RFC 2544 [21] and RFC 2889 [40]. Another example is an infinite FTP source which is commonly used for traffic generation in simulations. While these approaches provide some insight into network system capabilities, they lack nearly all of the richness and diversity of packet streams observed in the live Internet [32, 34, 44].

A number of successful application-specific workload generators have been developed including [13, 18, 37]. These tools typically focus on generating application-level request sequences that result in network traffic that has the same statistical properties as live traffic from the modeled application. While they are useful for evaluating the behavior and performance of host systems (such as servers and caches), these tools can be cumbersome for use in router or switch tests and obviously only recreate one kind of application traffic, not the diversity of traffic observed in the Internet. These observations motivate the need for a tool capable of generating a range of network traffic such as might be observed either at the edges or core of the Internet.

The contribution of this paper is the description and evaluation of a new network traffic generator capable of recreating IP traffic flows (where flow is defined as a series of packets between a given IP/port pair using a specific transport protocol - see Section 3 for more detail) representative of those observed at routers in the Internet. We are aware of no other tools that target representative traffic generation at the IP flow level. Our approach is to abstract flow-level traffic generation into a series of application-independent file transfers that use either TCP or UDP for transport. Our model also includes both temporal (diurnal effects as-

IMC'04, October 25–27, 2004, Taormina, Sicily, Italy.

sociated with traffic volume) and spatial (vis-à-vis IP address space coverage) components. The resulting constructive model can be manually parameterized or can be self-parameterized using Netflow [2] or packet trace data and components of the traffic generation tool itself. The self-parameterizing capability simplifies use and further distinguishes our system from other tools that attempt to generate statistically representative traffic.

We realized this model in a tool we call *Harpoon* that can be used in a router testbed environment or emulation testbed environment to generate scalable, representative network traffic. Harpoon has two components: *client threads* that make file transfer requests and *server threads* that transfer the requested files using either TCP or UDP. The result is byte/packet/flow traffic at the first hop router (from the perspective of the server) that is qualitatively the same as the traffic used to produce input parameters.

We evaluated Harpoon's capabilities in two ways. First, we conducted experiments in a controlled laboratory environment using two different data sets to show that we can qualitatively recreate the same traffic using Harpoon. We used a one week data set of Netflow records collected at a border router of the University of Wisconsin-Madison and a series of packet traces (from which we constructed flow records using a standard tool) collected at the border of the University of Auckland. These experiments demonstrate Harpoon's capability to generate representative flow level traffic. Next, we used the same laboratory environment to conduct a throughput evaluation of a Cisco 6509 switch/router following the testing protocol described in RFC 2889. We show that while throughputs achieved using standard packet streaming methods compared with Harpoon are similar, the loads placed on router subsystems are substantially different. The implication of this result is that a tool like Harpoon can augment standard test suites to give both router designers and network operators insight into system behavior under actual operating conditions.

This paper is organized as follows. After discussing related work in §2, we describe the design requirements and architecture of our flow-level workload model in §3. We present the details of Harpoon's implementation in §4 and the results of the validation tests in §5. The results from our throughput evaluation with Harpoon are given in §6. We conclude and outline future work in §7.

2. RELATED WORK

There is a large literature on the general topics of Internet traffic characterization and modeling. Some of the most successful models of traffic focus on the correlation structure (self-similarity) that appears over large time scales in both local area [38] and wide area traffic [44]. Crovella and Bestavros extend this work by showing evidence that self-similarity in Web traffic arises from multiplexing ON/OFF sources (as proposed in [50]) with heavy-tailed file sizes transferred [26]. This observation forms the basis of our constructive model for generation of IP flows.

Several flow-level network traffic models have been proposed including [17, 22, 35]. These models have been used to study fairness, response times, queue lengths and loss probabilities under different assumptions and using a variety of mathematical techniques. Our work differs from these in that we focus on building a flow-level model based on combining empirical distributions of characteristics that can be measured at a router in a live network. Our model can be realized in a tool for generating representative packet traffic in a live (or emulated) network environment or for creating traffic in a simulation environment.

An approach similar to ours has been used in a number of application-specific workload generators. In [18], the authors describe the SURGE Web workload generator which was built by combining distributional properties of Web use. A similar approach was used in developing Web Polygraph [13] and GISMO [37] which are used to generate scalable, representative Web cache and streaming workloads respectively. Harpoon differs from these tools in that it is application independent, uses empirical distributions, generates both UDP and TCP traffic and includes both spatial and temporal characteristics. It should also be noted that there are several commercial products that target application-specific workload generation including [8].

A model closely related to ours was introduced by Feldmann *et al.* in [28]. That model generalizes the SURGE model for Web traffic for the purpose of general background traffic generation in the ns-2 simulator. A similar model based on connection-rate superposition was developed by Cleveland *et al.* in [25]. Our model is more general in that it has no Web-specific parameters, includes the capability to transfer files via UDP and incorporates temporal and spatial request characteristics. Uhlig presents a flow level traffic model in [47] that is also similar to ours. That model, however, uses three parameters (compared with the eight that we use) and does not rely on an underlying transport protocol to transfer individual files. Our work is also distinct from each of these models in that we create and evaluate a tool (Harpoon) that can be used for traffic generation on real systems and that can be automatically parameterized from flow records, thus requiring no additional modeling steps.

In contrast to workload generators based on models designed to synthesize distributional characteristics are tools that replay traffic based on raw packet traces. `tcpreplay` and `flowreplay` [46], for example, take a very low-level approach by attempting to generate packets (while optionally rewriting IP addresses) that mirror the specific timings recorded in a packet trace. Monkey-See Monkey-Do [23] takes a slightly higher-level approach by extracting certain features, such as estimated bottleneck bandwidth, delayed ACK behavior, and receiver window from raw packet traces in order to replay HTTP transactions in a test environment while emulating original client protocol and network conditions. The approach taken in Harpoon differs significantly from these traffic generators. First, Harpoon is designed to generate representative background traffic with scalability and flexibility as fundamental objectives. It is unlikely that tools mimicking network behavior at such a low level can scale to high-speed links. Second, Harpoon uses *source-level* traffic descriptions that do not make assumptions about the transport layer, rather than *packet-level* descriptions based on prior network state embedded in low-level timings [31]. Generating packet traffic based on network-induced timings is problematic, especially with closed-loop protocols such as TCP, where transient, context-dependent network conditions can substantially alter individual packet timings.

3. ARCHITECTURE

The design objectives of Harpoon are (1) to scalably generate application-independent network traffic at the IP flow

level, and (2) to be easily parameterized to create traffic that is statistically identical to traffic measured at a given vantage point in the Internet. Figure 1 depicts a high-level process flow of these objectives. We start with the basic definition of an IP flow to create a constructive model for network traffic generation which we describe below.

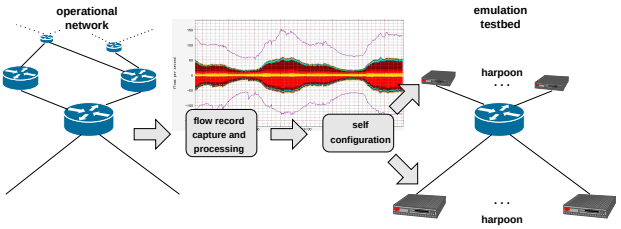


Figure 1: Flow records are collected at a given vantage point in an operational network using standard software like `flow-tools`. Key aspects of the live flows are extracted during a self-configuration step. These parameters are used to generate traffic in a testbed that statistically matches the temporal (diurnal) volume characteristics as well as the spatial (source and destination IP address frequency) characteristics of the live flows.

An IP flow as defined in [24] is a unidirectional series of IP packets of a given protocol traveling between a source and a destination IP/port pair within a certain period of time. The final condition of this statement is ambiguous, so we tie our definition to the tools we use to gather and analyze network flow data: `flow-tools` [36] and FlowScan [45]. Netflow data includes source and destination AS/IP/port pairs, packet and byte counts, flow start and end times, protocol information, and a bitwise `OR` of TCP flags for all packets of a flow, in addition to other fields. This data is exported either on timer deadlines or when certain events occur (e.g., a TCP FIN or RST, or a cache becomes full), whichever comes first. While this pragmatically resolves ambiguity in the definition of a flow, specific expiration-related timing behaviors can vary [7]. We discuss how this variation can affect parameterization in Section 4. An example of how a single transaction, such as an FTP transfer, can appear in flow records is shown in Figure 2. The transaction is represented as multiple data flows between the two hosts. Each direction of the control and data connections is reported, resulting in four flow records. TCP flags, packet and byte counts accumulate over the duration of each connection.

From this operational definition of a flow, Harpoon's architecture begins with the notion of unicast file transfers using either TCP or UDP. Harpoon does not address the packet level dynamics of TCP file transfers. Rather, it relies on the version(s) of TCP running on end hosts to transfer the requested file. Modeling UDP traffic is complicated by the fact that packet emission behaviors are largely application-specific. At present, Harpoon contains three models of UDP packet transfer: a simple parameterized constant packet rate, a fixed-interval periodic ping-pong, and an exponentially distributed ping-pong. The first source type is similar to some audio and video streams, while the latter two types are intended to mimic the standard Network Time Protocol (NTP) and Domain Name Service (DNS), respectively. UDP traffic in today's Internet is likely to be made up of a wider variety of application level traffic (including voice, SQL worms, etc.) whose behavior is not captured in our three source types. Development of a model with a more diverse set of UDP traffic sources is left for future work.

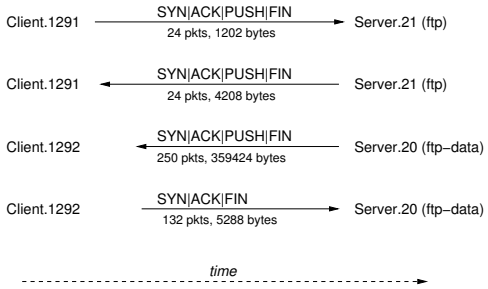


Figure 2: Flow level decomposition of a simple FTP transaction. Each line is logged as a separate entry by NetFlow.

The Harpoon flow model is a two level architecture and is depicted in Figure 3. We refer to the lower level of the Harpoon model as the *connection level*. It is made up of two components that have measurable distributional properties. The first component is the *size* of the file transferred, and the second component is the time interval between consecutive file transfer requests, the *inter-connection time*. Harpoon makes requests for files with sizes drawn from an empirical distribution $P_{FileSize}$. Connection initiations are separated by time intervals drawn from an empirical distribution $P_{InterConnection}$.

The upper level of the Harpoon model is referred to as the *session level*. Harpoon sessions are divided into either TCP or UDP types which then conduct data transfers using that protocol during the time that they are active. The session level has two components: the number of *active sessions* and the *IP spatial distribution*. By modulating the number of sessions that are active at any point in time, Harpoon can match the byte, packet, and flow volumes from the original data and realize the temporal (diurnal) traffic volumes that are a common characteristic of the Internet [43]. The average number of sessions of each type (TCP/UDP) that are active at any point in a day is derived from a flow data time series for consecutive non-overlapping intervals of length $IntervalDuration$ seconds to create an empirical model for $P_{ActiveSessions}$. Scalability is naturally achieved by dividing the number of active sessions across any number of hosts comprising the testbed. For each session, Harpoon picks source and destination addresses from ranges of available addresses to make a series of file transfer requests. The address selection is made preferentially using weights drawn from empirical distributions $P_{IPRange_{src}}$ and $P_{IPRange_{dest}}$. A series of file transfer requests then takes place between the source and destination for $IntervalDuration$ seconds. When Harpoon is started, it begins with the average number of sessions in the first interval and proceeds through consecutive intervals for the duration of the test.

In summary, the Harpoon model is made up of a combination of five distributional models for TCP sessions: file size, inter-connection time, source and destination IP ranges, and number of active sessions. These parameters are summarized in Table 1. There are three distributional models for UDP sessions: constant bit-rate, periodic ping-pong, and

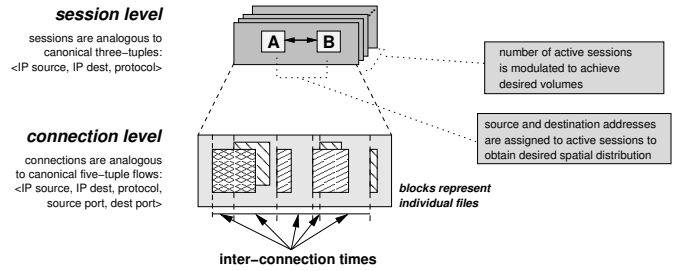


Figure 3: Harpoon's flow-based two-level hierarchical traffic model. Sessions are comprised of a series of connections separated by durations drawn from the inter-connection time distribution. Source and destination IP address selection (A and B in the figure) is weighted to match the frequency distribution of the original flow data. The number of active sessions determines the overall average load offered by Harpoon. A heavy-tailed empirical file size distribution and an ON/OFF transfer model generate self-similar packet-level behavior.

exponential ping-pong. Each of these distributions can be specified manually or extracted from packet traces or Netflow data collected at a live router. These models enable the workload generated by Harpoon to be application independent or to be tuned to a specific application. The models are combined in a constructive manner to create a series of file transfer requests that results in representative flow-level network traffic.

4. IMPLEMENTATION

A key feature of Harpoon is that it is self-configuring. Netflow logs or packet traces are used for parameterization without any intermediate modeling step, obviating the need for Harpoon users to become experts in distribution and parametric estimation. In this section, we first discuss the relevant implementation issues and limitations in transforming flow records into a suitable configuration for Harpoon. We follow with a description of the implementation of the traffic generation component of Harpoon.

4.1 Self-Configuration

The self-configuration phase of Harpoon takes flow records as input and generates the necessary parameters for traffic generation. The key parameters are distributional estimates of (1) file sizes, (2) inter-connection times, (3) source and destination IP addresses, (4) and the number of active sessions. We divide the input flow records into a series of intervals of equal duration to generate the number of active sessions in order to match average byte, packet, and flow volumes of the original data over each interval. We also discuss below how the interval duration, a configurable parameter, is set. For each parameter, we use the empirical distribution derived from the original flow or packet data and do not attempt fitting to a known distribution (although Harpoon could be trivially enhanced to generate variates from known distributions).

File Sizes Flow records contain packet and byte counts, so a first approximation of file sizes (flow payload sizes) can be extracted by $ByteCount - PacketCount * 40$ (assuming no IP or TCP options). To this calculation we make the following refinements. First, due to flow record timeouts, a single flow may be split into multiple flow records. To counter this effect, we perform "flow surgery" [27] on the raw flow records, coalescing records where addresses, ports, protocol, timestamps and TCP flags indicate that the records refer to the same flow. Second, we only perform the calculation if there are start and end markers present in the TCP flags, *i.e.*, a SYN flag and a RST or FIN flag. This check ensures that we do not underestimate application payloads because of missing the beginning or end of a flow. Third, we discard flows that appear to be "ACK flows" or flows that are very small (*e.g.*, the request direction for an HTTP transfer).

There are two practical complications to calculating file sizes from flow records. First, some routers do not record TCP flags in flow records. In this implementation of Harpoon, we assume that these flags are available. Also, it is common practice on high-bandwidth links to perform packet sampling (*e.g.*, by recording every Nth packet or recording a packet with probability $\frac{1}{N}$) to reduce the storage and processing requirements of maintaining flow records. Duffield *et al.* [27] describe methods for recovering the distribution of flow sizes from sampled flow records. Application of these methods to the self-configuration step of Harpoon and relaxing assumptions regarding presence of TCP flags are areas for future work.

Inter-connection Times To extract the inter-connection time distribution, we again make use of TCP flags in the flow records. For each source and destination IP address pair encountered, we create an ordered list of start times for flows that contain a SYN flag. The collection of differences between consecutive SYNs for each address pair constitutes the inter-connection time empirical distribution. In practice, we impose a bound on the maximum inter-connection time (*e.g.*, 60 seconds). We discuss implications of this bound below.

Source and Destination IP Addresses To emulate the spatial characteristics present in flow records captured in a live environment, we first extract the empirical frequency distributions of source and destination IP addresses. We map the resulting rank-frequency distributions onto source and destination address pools (from the perspective of a Harpoon client) used in the testbed. For example, if the source address pool for a testbed host is specified as a class C network, we map the frequency distribution of the top 254[1] addresses from the live flows to Harpoon configuration parameters.

Number of Active Sessions One way to calculate the number of sessions that should be active over a series of intervals is to start with the observation that each source and destination IP address pair (the analog of a session in Harpoon) contributes to the overall load during one or more intervals. For each host pair, we find the earliest flow start time and latest end time and "spread" a value proportional to the lifetime of that session over the corresponding intervals.

While the above technique appears to be the most direct way of calculating the number of active sessions, it fails because the end timestamp of flow records frequently does not reflect the precise time of the final packet of a flow. The inaccurate timestamps extend flow durations and therefore cause the number of sessions that should be active over a series of time intervals to be overestimated. This inflation

[1]256 addresses in a full class C minus host address (.0) minus broadcast address (.255) equals 254 usable addresses.

Table 1: Summary of Harpoon configuration parameters for TCP sources.

Parameter	Description
$P_{FileSize}$	Empirical distribution of file sizes transferred.
$P_{InterConnection}$	Empirical distribution of time between consecutive TCP connections initiated by an IP source-destination pair.
$P_{IPRange_{src}}$ and $P_{IPRange_{dest}}$	Ranges of IP addresses with preferential weights set to match the empirical frequency distributions from the original data.
$P_{ActiveSessions}$	The distribution of the average number of sessions (IP source-destination pairs) active during consecutive intervals of the measured data. By modulating this distribution, Harpoon can match the temporal byte, packet and flow volumes from the original data.
$IntervalDuration$	Time granularity over which Harpoon matches average byte, packet and flow volumes.

Table 2: Summary statistics of differences (in milliseconds) between Netflow timestamps from a Cisco 6509 and flow records generated from a DAG 3.5 packet trace.

timestamp	mean	median	standard deviation
flow begin	19	0	8
flow end	454	461	254

could cause the byte, packet and flow volumes generated by Harpoon to exceed the original volumes. The inaccuracy in the end timestamp may be caused by delays from waiting until a flow export packet or cache fills, or by lingering out-of-order packets[2] [7]. In contrast, flow start timestamps appear to accurately reflect the first packet of a flow (allowing us to use them in calculating the inter-connection times).

To quantify the timing inaccuracies introduced by Netflow, we generated traffic through a Cisco 6509, capturing Netflow records from the router and simultaneously taking a packet trace using a high precision DAG 3.5 capture card[3] [4]. Table 2 shows the sample mean, median, and standard deviation for begin and end timestamp differences in milliseconds. While most differences in the begin timestamp are zero (note the median) and otherwise quite small, differences in the end timestamp are, on average, significantly larger. These end timestamp differences, when compounded over several thousand flows per minute (see Figure 8(c), for example) cause our initial algorithm to fail. On the other hand, using flow records constructed from raw packet traces with accurate timestamps leads to a good match. Results for the Auckland trace, described in Section 5, were generated in this way.

The revised approach we take for tuning the number of active sessions is outlined in the pseudocode shown in Figure 4. We first make the assumption that, in the overwhelming majority of cases, a file request made during interval I_j is also completed during interval I_j. This assumption is reasonable based on relatively large values for $IntervalDuration$, such as 300 or 600 seconds, and a reasonable bound on round-trip times. We determine how many sessions are required to generate the required byte volume over the duration of an interval by mimicking the action of Harpoon sessions as they alternate between waiting for an amount of time drawn from the $P_{InterConnection}$ distribution and requesting files whose sizes are drawn from the $P_{FileSize}$ distribution[4]. We successively increment the number of sessions, imitating the action of each session and noting when we surpass the requisite byte volume for the given interval.

In practice, we run the algorithm for each interval N times, using the mean number of sessions required to generate the necessary volume. As shown in the validation experiments of Harpoon, tuning the number of active sessions to the required byte volume using the technique we outline here leads to an accurate match with the original byte, packet, and flow volumes over relatively coarse intervals.

```
BytesGenerated = 0                          # Total bytes generated by mimicked
                                            # sessions.
SessionsRequired = 0                        # Sessions required to create the
                                            # intended volume.

while BytesGenerated < IntendedByteVolume:  # We want to generate at least as many
                                            # bytes that were originally sent.

    SessionsRequired += 1                   # One more session is required...

    ElapsedTime = 0                         # ElapsedTime holds amount of time the
                                            # current session has been active
                                            # during this interval.

    while ElapsedTime < IntervalDuration:   # This loop mimics the action
                                            # of a single session.

        ElapsedTime += InterConnectionTimes.next()  # Get the next inter-connection
                                            # time from the empirical distribution.

        BytesGenerated += FileSizes.next()  # Get the next file size from
                                            # the empirical distribution (and
                                            # assume the file is transferred
                                            # in the current interval).
    end while
end while

# The number of sessions needed to generate required byte volume is
# now stored in the variable SessionsRequired.
```

Figure 4: Pseudocode for algorithm used to determine the number of sessions that should be active over a given interval to produce a certain byte volume. Intended byte volume, $IntervalDuration$, $P_{FileSize}$ and $P_{InterConnection}$ distributions are given as inputs to the algorithm.

Interval Duration The value $IntervalDuration$ is the time granularity over which we match byte, packet, and flow volumes between the originally measured flow data and Harpoon. The duration can be selected somewhat arbitrarily, but there are two practical constraints. First, since we choose a source and destination address for each active session at the start of each interval, there is a tradeoff between

[2]There are also inaccuracies that are related to the coarse-grained nature of flow records. Since we subtract contributions of headers (including connection setup and teardown packets and associated latency) to file sizes, we would really like the start and end timestamps to reflect the first and final payload packets of a flow.

[3]The DAG capture point was situated one hop after the 6509. Differences in timestamps due to propagation and forwarding delays were assumed to be negligible.

[4]Note that the connection initiation schedule defined by the $P_{InterConnection}$ distribution is made independent of network feedback.

the length of an interval and how well the spatial distribution can be approximated over the course of a full test. With longer intervals, there is less opportunity for sampling each spatial distribution. With shorter intervals, there is an increased internal overhead from managing sessions and timers. In our experience, intervals such as five minutes work well. Five minutes also happens to be a common interval over which flow records are aggregated (as implemented by the widely used `flow-tools` [36]). The second practical consideration is that $IntervalDuration$ should be at least as long as the maximum inter-connection time. One reason is that a session may randomly get an inter-connection time that causes it to be idle for the entire interval. Another reason is that at the end of an interval, there may be some sessions that are waiting for an inter-connection time to expire before initiating a connection – we do not prematurely halt sessions at the end of an interval to ensure sampling the longest inter-connection times. The sessions that are waiting for the next connection time at the end of an interval are technically active but not engaged in file transfers and count against the target number of active sessions for the new interval. The effect of these temporarily frozen sessions is that Harpoon may not generate the intended volumes for every interval after the first. Setting $IntervalDuration$ to a large enough value along with setting an appropriate cap on the maximum inter-connection time when processing the original flow data resolves this potential problem.

The value $IntervalDuration$ is used during parameterization for determining the $P_{ActiveSessions}$ distribution and is also used to control traffic generation. The values used for each phase need not be the same. This decoupling enables time compression (or expansion) relative to the original data during traffic generation. Harpoon can either be set to match the aggregate volume of the original intervals or to match the bitrate. For example, if N TCP sessions are measured from flow logs in interval I_j and M TCP sessions are measured in interval I_{j+1}, Harpoon could be configured to initiate $N + M$ sessions over a test interval I_k to realize similar aggregate volume. To match the original bitrate, the traffic generation $IntervalDuration$ is simply set to a smaller value than the duration used for parameterization. However, there is a potential pitfall when matching bitrates by using a shorter traffic generation $IntervalDuration$. If the interval is too short, there may be insufficient sampling of the $P_{InterConnection}$ and $P_{FileSize}$ distributions with the configured number of active sessions. The result is that the byte and packet volumes generated over the interval may vary far from the expected value.

While it is true that our use of rather coarse intervals for matching original volumes tends to ignore issues of packet arrivals over short (sub-RTT) time scales, this is intentional. Additional testbed parameters could have been required with a Harpoon setup (such as a distribution of round-trip times, link capacities, MTUs, etc.) to match parameters derived from a live trace, but at this point we do not specify such configurations. Creating the necessary volumes over longer time scales to produce self-similarity and diurnal patterns in a way that real application traffic is generated is the intent and domain of Harpoon. We believe it is preferable to allow burstiness over short time scales to arise endogenously from TCP closed-loop control and network state, rather than to exogenously attempt to force such state (*e.g.*, by introducing a random packet dropping element along a path to effect a certain loss rate). We demonstrate limitations of Harpoon with respect to short time scales in Section 5.3.

4.2 Traffic Generation

Harpoon is implemented as a client-server application. Figure 5 illustrates the conceptual aspects of the implementation. A Harpoon client process consists of threads that generate file requests. The hierarchical session/connection model is realized by individual threads, and the distribution for active sessions maps to a number of threads actively making file requests. In each thread, connections are initiated according to the inter-connection time distribution. The duration of any file transfer is dictated by the dynamics of the transport protocol and the underlying testbed configuration. Inter-connection times, on the other hand, are followed independently of the transport layer so an active session may be multiplexing multiple concurrent connections. A Harpoon server process consists of threads that service Harpoon client requests. The server controls the sizes of files transferred according to the input distribution.

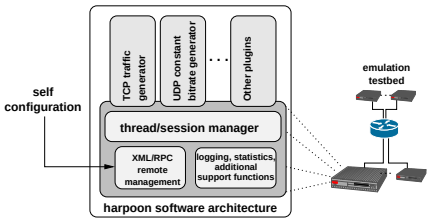


Figure 5: Harpoon software architecture. A core session manager controls dynamically loadable traffic generator plugin modules and through an XML/RPC interface indirectly handles remote requests to stop or start plugins, load new configurations, or retrieve plugin statistics.

While TCP file transfers are controlled by protocol dynamics, UDP dictates no control over packet emissions. Currently, Harpoon can send UDP datagrams at roughly constant bit rates (configurable, but by default 10kbps), at fixed intervals, and at exponentially distributed intervals. Parameters used for these traffic sources in our tests are summarized in Table 3. Unlike the constant bit rate source type, the latter two UDP source types have no notion of multi-packet files or inter-connection times; all "files" consist of a single datagram whose size depends on file size distributions for each source type. As such, fixed interval and exponentially distributed interval sources do not require any control messages to coordinate when packets are sent. The same is not true for the constant bit rate sources, where we must ensure that file requests take place according to the file size and inter-connection request distributions. Each client maintains a TCP control connection with the target server over the duration of an active UDP session. File requests are made over this channel and UDP data is then sent by the server. The client sends the port number of a locally bound UDP socket in order for the server to set the proper destination of UDP traffic. The client additionally sends a datagram size and rate for the server to use when sending data, and a unique request identifier. Once the server finishes sending the requested file, it sends a completion indication to the client for the original request identifier.

In addition to the distributional input parameters, each

Table 3: Summary of Harpoon configuration parameters for UDP sources used in validation tests.

Source Type	Key Parameter
Constant Packet Rate	Rate = 10kbps
Fixed-interval Periodic Ping-Pong	Interval = 64 seconds
Exponentially Distributed Ping-Pong	λ = 30 milliseconds

Harpoon client is given a range of local IPv4 addresses to use as a source address pool, and a range of IPv4 addresses and associated ports of target Harpoon servers. Address ranges are specified as CIDR prefixes to Harpoon. The source addresses may be bound to physical interfaces on the client host or to address aliases assigned to an interface. When starting a new user, a thread picks a source address and destination address from this pool. The address pools are constructed in such a way that random selection of addresses generates the required spatial frequency distribution.

Harpoon is designed in a modular fashion to enable relatively easy addition of new traffic models. Traffic generation modules for TCP, UDP constant bit-rate stream, and the fixed and exponential interval UDP datagram sources are implemented as plugin modules that are dynamically loaded and controlled by a core Harpoon thread manager. Harpoon reads XML parameter files produced by the self-configuration components, loading and starting the necessary plugin modules.

Management of resources and tools within large-scale testbed environments can be challenging. To counter this problem we implemented an HTTP server in Harpoon, along with an XML parser to enable remote management via XML-RPC [14]. Users can remotely reset, stop, and gather statistics from running Harpoon processes. Traffic module object files can be disseminated to remote Harpoon processes and configured by distributing XML parameter files from a central location.

Currently, a single Harpoon process can produce a few hundreds of megabits per second of network traffic using reasonably provisioned commodity workstations. Performance is largely dependent on the nature of the inter-connection time distribution (because of timer management) and the number of active sessions. The memory footprint of Harpoon (client or server) is relatively small, normally less than 10MB with modestly sized input distributions. The code is written in C++ and currently compiles and runs under FreeBSD, Linux, MacOS X, and Solaris. Porting to new platforms is limited only by support for POSIX threads, capability for dynamic loading of objects (*e.g.*, `dlopen()` and friends), the C++ standard template library, and the eXpat XML library[5].

5. VALIDATION

In this section we validate the ability of Harpoon to generate traffic that qualitatively exhibits the same statistical signatures as the distributional models used as input.

We used two different data sets to self-parameterize Harpoon in our validation tests. Our first data set consisted of one week of Netflow records collected between July 31, 2002 and August 6, 2002. The second data set was a series of packet traces from the University of Auckland, taken on June 11 and 12, 2001, from which we constructed flow records. We modified the `crl_flow` tool in the CoralReef software suite [3] to produce wire-format Netflow version 5 records from the packet traces. We refer to these data sets as "Wisconsin" and "Auckland" below, and they are summarized in Table 4.

Table 4: Summary of Data Sets Used in Validation Experiments

Data Set	Total Flows	TCP Flows
Wisconsin 31 July–6 August 2002	423,836,790	241,484,962
Auckland 11–12 June 2001	12,912,462	12,144,434

5.1 Tests

Our validation test environment consisted of two workstations running the FreeBSD 5.1 operating system. Each machine had 2 GHz Intel Pentium 4 processors, 1 Gigabyte of RAM, and used kernel defaults for TCP/IP parameters, which meant that the NewReno congestion control algorithm, time stamping, and window scaling TCP options were enabled, and the default receive window size was 64kB. The default Ethernet MTU of 1500 bytes was used. Each machine was dual-homed, with a separate Intel Pro/1000 interface used solely for experiments. The machines were each connected to a Cisco 6509 router via 1 Gbps Ethernet.

One FreeBSD machine was used as Harpoon data server and two were used as Harpoon clients for generating requests and receiving the resulting data packets. We monitored each host during our tests to ensure the systems did not exhaust all available CPU or memory resources.

The client machines were configured to generate requests using an IPv4 class C address space (2^8 addresses). Likewise, the server machine was configured to handle requests to an IPv4 class C address space. In each case, the address creation was accomplished by creating aliases of the loopback interface, and creating static routes at the Harpoon clients, server and intermediate switch. Aliases of the loopback adapter were created to avoid ARP overhead with the switch.

5.2 Results

Figures 6 and 9 examine Harpoon's capability to generate the desired temporal characteristics for the Wisconsin and Auckland data sets, respectively. Figure 6 compares the bitrate over a period of 8 hours from the original Wisconsin data with Harpoon. Figure 9 compares the bitrate over a period of two days of the original Auckland data with the bitrates produced by Harpoon when emulating the same time period. The level of traffic is determined by the input distribution specifying the number of active users ($P_{ActiveSessions}$). In each test, the Harpoon traffic exhibits burstiness due to the distribution of file sizes, inter-connection times, and the closed-loop nature of TCP. Nonetheless, over the duration of each test the hourly pattern emerges due to control over the number of active sessions at the Harpoon client process. Some variability is introduced because Harpoon does not immediately truncate an active session at each emulation interval. Instead, it lets the final connection finish naturally, thus avoiding abrupt shifts between each interval.

Figure 7 compares the inter-connection time, file size, and destination frequency empirical distributions derived from the Wisconsin data set with the distributions generated by Harpoon. For inter-connection times, shown in Figure 7(a), there is a good overall match except at the shortest

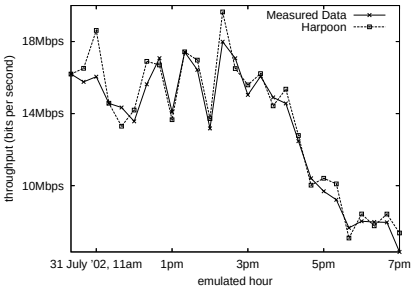


Figure 6: Emulation of temporal volume characteristics for Wisconsin data.

inter-connection times. The visible "bumps" for the Harpoon tests at these inter-connection times are an effect of a coarse-grained operating system scheduler interacting with the shortest durations of this distribution. FreeBSD (as well as Linux), by default, uses time slices of 10 milliseconds and the steps in Figure 7(a) are at multiples of this time slice. In tests with operating systems that have different scheduling mechanisms (*e.g.*, MacOS X), these particular artifacts do not appear. Results for the Auckland data set are qualitatively similar.

Figure 7(b) compares file sizes extracted from the Wisconsin data set with the file sizes transferred by Harpoon. There is a close qualitative match, and results for the Auckland data set are similar.

Figure 7(c) plots frequency vs. rank on a log-log scale for destination addresses for the Wisconsin data set. We observe a close match between the original data and the Harpoon data. Results for source addresses are qualitatively similar, as are results for the Auckland data set.

Our final validation test was to compare the traffic volumes generated by Harpoon against the original flow traces used to generate parameters for each data set. In Figure 8 we compare the distributions of packets, bytes, and flows per measurement interval to those derived from the Wisconsin data set. We make the same comparisons for the Auckland data set in Figure 10. As shown in these plots, Harpoon accurately approximates the number of bytes and flows per interval. For each data set, there are more packets sent in the original trace than from Harpoon. The reason is that our testbed is homogeneous with respect to link-layer maximum transmission unit sizes, resulting in average packet sizes that are larger than the original trace. Figure 11(a) shows a time series of the mean packet size over 60 second intervals of a one hour segment (12pm-1pm, 11 June 2001) of the Auckland data set and a similar time series for Harpoon. Harpoon packet sizes average almost twice as large as the original trace. As shown in Figures 8(b) and 10(b), when we scale the Harpoon volumes by the ratio of average packet sizes between Harpoon and the measured data, we observe a close match.

5.3 Limitations

Two limitations to Harpoon's traffic model are:

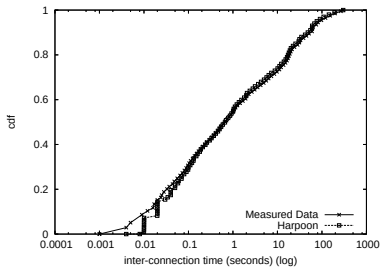
1. it is designed to match byte, packet, and flow volumes over relatively coarse intervals (*e.g.*, 300 seconds) and may not match over shorter intervals;
2. since packet-level dynamics are not specified, traffic produced by Harpoon may not match other metrics of interest, such as scaling characteristics, queue length distribution for the first-hop router, packet loss process, and flow durations.

Packet-level dynamics created by Harpoon arise from the file size distribution, the inter-connection time distribution, TCP implementations on end hosts, and testbed parameters such as round-trip time distribution, link capacities, and MTU sizes. Round-trip time is a key parameter because it affects flow durations for TCP sources. As a consequence, the nature of the ON/OFF process and therefore the correlation structure of packet arrivals over both short and long time scales are affected. It was shown in [29] that dynamics over short time scales differ between LAN and WAN environments, and in [28] that short time scale effects arise due to the TCP feedback loop and network environment variability characteristic of WANs. Effects across both short and long time scales are of interest in testbed traffic generation because performance measurements can differ substantially between LAN and WAN settings (*e.g.*, see [19]).

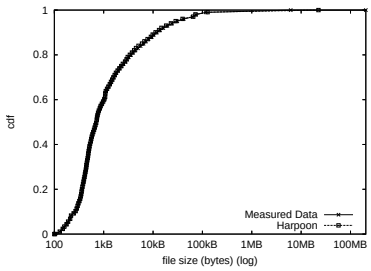
In Figure 11(b) we compare time series of bytes transferred over 1 second intervals between a segment of the Auckland trace and Harpoon using an $IntervalDuration$ of 300 seconds. There is no propagation delay emulated in the testbed, so the round-trip time for Harpoon sources is on the order of 1 millisecond. There is greater variability evident in the testbed configuration, in part, from a tight TCP feedback loop resulting in higher throughput and shorter flow durations. For the Auckland trace, dominant round-trip times of around 200 milliseconds (roughly the RTT to North America) lead to longer flow durations, a greater degree in flow concurrency, and less observable variability in utilization.

A standard method of characterizing behavior of traffic dynamics over a range of time scales is the log-scale diagram described in [16]. The log-scale diagram was developed as a means for identifying self-similar scaling in network traffic and for estimating the Hurst parameter. Evidence for self-similar scaling is associated with a range of scales where there exists a particular linear relationship between scale and the log of a normalized measure of energy (variability). A different scaling regime has been shown to exist in measured WAN traffic and is separated from the linear self-similar region by a pronounced dip in energy at the time scale associated with dominant round-trip times.

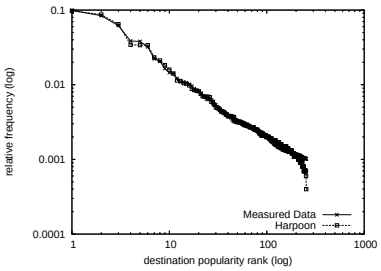
In Figure 11(c) we compare log-scale diagrams based on wavelet decompositions of the time series of byte volumes over 500 microsecond intervals for a one hour segment of the Auckland data set (12pm-1pm, 11 June 2001) and for Harpoon with two different values of $IntervalDuration$ (60 or 300 seconds) and two emulated round-trip times (0 or 200 milliseconds). (We do not plot confidence intervals for clarity.) We first note the difference between Harpoon configurations using different round-trip times. Comparing Harpoon using an RTT of 200 milliseconds with the original trace, there is a common range of time scales indicative of self-similar scaling (scales 9-14) and a clear dip at the 256 millisecond time scale (scale 9) because of dominant round-trip times in the original trace, and the singular round-trip time for Harpoon. For the Harpoon configurations with no emulated propagation delay, there is more energy across almost all time scales and nothing in common with the original trace. Finally, we note that over sub-RTT time scales, Harpoon does not match the Auckland trace for any configu-



(a) Inter-connection time distribution.

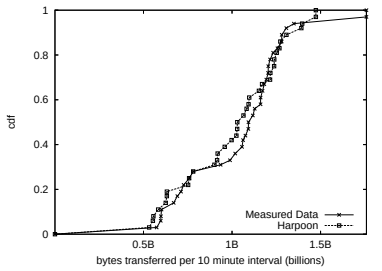


(b) File size distribution.

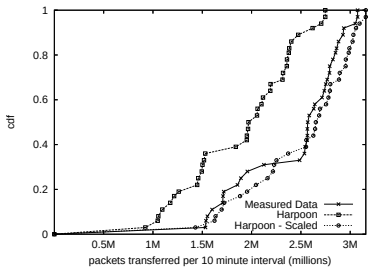


(c) Destination IP address frequency distribution.

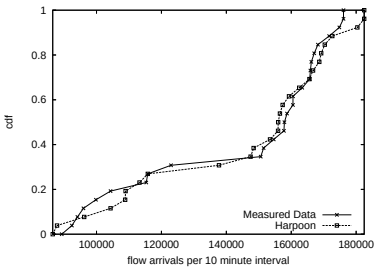
Figure 7: Comparison of empirical distributions extracted from Wisconsin data with distributions produced during Harpoon emulation. Results shown for one day of Wisconsin data (31 July 2002). Results for other days and for the Auckland data are qualitatively similar.



(a) Byte volumes.



(b) Packet volumes. Harpoon data also shown scaled by ratio of average packet size between Harpoon and measured data.



(c) Flow volumes.

Figure 8: Comparison of byte, packet, and flow volumes extracted from of Wisconsin data with volumes produced during Harpoon emulation. Results shown for one day of Wisconsin data (31 July 2002). Results for other days are qualitatively similar.

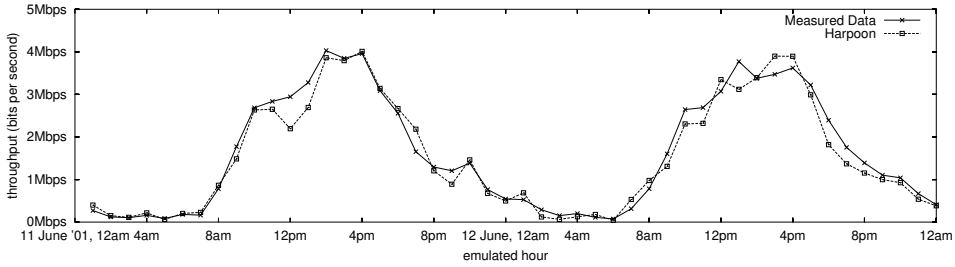
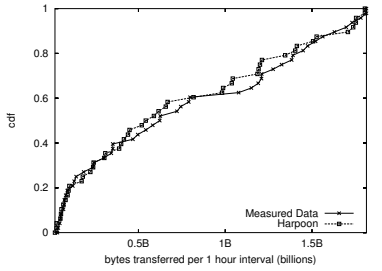
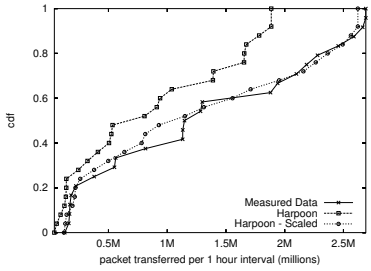


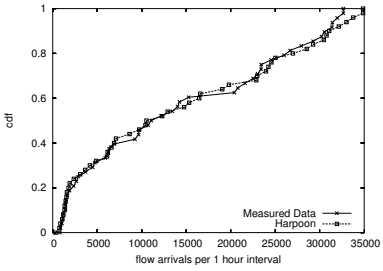
Figure 9: Emulation of temporal volume characteristics for Auckland data.



(a) Byte volumes.



(b) Packet volumes. Harpoon data also shown scaled by ratio of average packet size between Harpoon and measured data.



(c) Flow volumes.

Figure 10: Comparison of byte, packet, and flow volumes extracted from original Auckland data with volumes produced during Harpoon emulation.

ration. Since our testbed lacks diversity in round-trip times (as well as MTUs and link capacities), we do not expect a match over these time scales. Even though Harpoon accurately approximates the original volumes over 300 second intervals for both round-trip time values (not shown here), there are vast differences between the original correlation structure and that produced by Harpoon.

Comparing the two configurations of $IntervalDuration$, there is slightly less overall energy when using 60 second intervals because the byte volumes produced by Harpoon are less than the original volumes. The reason is that the interval is too small compared with the default maximum inter-connection time of 60 seconds: a session may randomly get an inter-connection time that causes it to be idle for the length of the entire interval. The end effect is that Harpoon produces less traffic than intended (see also Section 4).

These differences have important implications not only for Harpoon, but in configuring any laboratory testbed and in the interpretation of results. First, it is clear that queuing dynamics will be different depending (at least) on configured round-trip times, resulting in different delay, jitter, and loss characteristics than might have been measured in a live environment. Second, such differences will very likely affect transport and application performance, requiring careful interpretation of measurements, and characterization of their applicability outside the laboratory. While we feel that Harpoon represents a step forward in testbed traffic generation, the goal of predicting performance in a live environment based on laboratory experiments remains somewhat out of reach, since specifying the ingredients necessary to produce realistic packet dynamics over all time scales in both simulation and laboratory testbeds is, in general, an open problem [42].

6. COMPARISON WITH PACKET-ORIENTED TRAFFIC GENERATORS

The first and most obvious application for Harpoon is in scalable background traffic generation for testbed environments such as Emulab [49] and WAIL [11]. In this section we investigate another application of Harpoon in a router benchmarking environment. We compare router performance using workloads generated by Harpoon with workloads generated by a standard packet level traffic generator[5]. Our motivation for comparing Harpoon with a standard traffic generation system is to both demonstrate the differences between the two approaches, and to consider how the standard tools might be tuned to exercise routers more comprehensively.

6.1 Environment and Methodology

For the hardware environment used in this series of experiments we used the two Harpoon source machines and one Harpoon sink machine configured in the same way as our validation tests described above. Each host was connected to a Cisco 6509 router[6] [1] via 1 Gbps Ethernet links. Each link was connected to separate physical line cards on the 6509, forcing traffic to cross the internal switching fabric of the router. During tests, the 6509 was connected only to the traffic generation machines and to our management network, over which we polled specific MIB variables via SNMP. No other traffic traversed the router. The system we used to generate the standard packet traffic was a Spirent AX/4000. The AX/4000 generated traffic over two 1 Gbps Ethernet links and the traffic returned to the AX/4000 over another 1 Gbps link. While our tests contrast Harpoon with the AX/4000, the model of traffic generation in the AX/4000 is representative of many tools that generate packets with constant spacing. We therefore use the term "constant spacing packet generator" in the discussion and graphs of results.

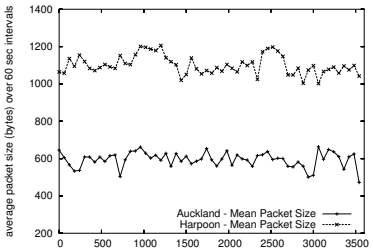
Our experiments were based on a series of tests described at `www.lightreading.com` [41] and on benchmarking terminology and methods described in RFCs 2285 [39] and 2889 [40]. These tests were designed to evaluate packet forwarding performance of Internet backbone routers under a variety of conditions, including the use of minimum-sized packets, and forwarding tables comparable in size to those found in the Internet backbone. The LightReading tests also investigated many other performance capabilities of routers, such as BGP table capacity, packet filtering, and forwarding performance under unstable path conditions. Our tests focused on packet forwarding performance and the impact of workloads on router subsystems.

Our basic metric for comparison was packet forwarding rate. As RFC 2285 points out, it is problematic to report forwarding rates in the *absence* of packet losses as the earlier RFC 1242 [20] had defined. Obviously, packet loss rates depend on the offered load and router or switch implementation artifacts. Our challenge was to determine how to match offered loads between two fundamentally different packet generation methods. Instead of attempting to compare forwarding rates across a broad range of offered loads, we chose two different regimes, a "low" load and a "high" load. With each offered load, we polled the 6509 every 20 seconds using SNMP to obtain the forwarding rate, packet loss counts and other system utilization variables available via SNMP on each interface. SNMP variables are updated internally every 10 seconds in the 6509. Our polling period of 20 seconds was chosen to minimize additional load placed on the 6509 while ensuring each SNMP request would obtain updated counters. The low and high offered load regimes were roughly tuned to 60% ($\approx$ 600Mbps) of egress bandwidth and 90% ($\approx$ 900Mbps) of egress bandwidth, respectively[7]. For Harpoon, we used the $P_{FileSize}$ and $P_{InterConnection}$ distributions extracted from the Wisconsin data and used the algorithm of Figure 4 to determine how many session threads should be active to generate the intended load levels on the 6509. We then calculated the average offered load from SNMP measurements of the ingress interfaces and used this average to configure the packet-level generator. Because of this tuning approach, including other application-level generators in this evaluation would have required imprecise trial-and-error tests to tune them to the same level. Thus, we chose only to compare Harpoon with the constant load generator. For all tests described in this section, Harpoon was initialized with the same random number generator seed to ensure

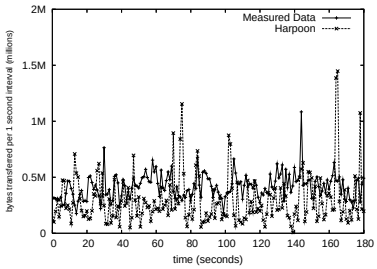
[5]Standard packet generation tool range from free software tools such as `iperf` [6] to large-scale, feature-rich hardware platforms such as the Spirent AX/4000 [8].

[6]The Cisco 6500 series of devices can be configured either to run Cisco CatOS, Cisco IOS, or a hybrid of the two. The choice depends on anticipated traffic, customer requirements, and modules installed. We configured our 6509 to run native IOS, version 12.1(13)E6.

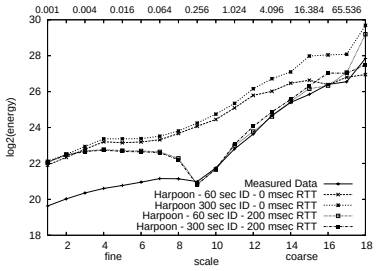
[7]With a deterministic packet generation tool like the AX/4000, tuning the offered load is straightforward. With Harpoon, the load is tuned by adjusting the number of client and server threads.



(a) Average packet sizes for one hour segment of Auckland data and equivalent Harpoon trace. For the Auckland data, the overall sample mean and standard deviation are 610 and 629, respectively. For Harpoon, the overall sample mean and standard deviation are 1106 and 615, respectively.



(b) Time series of bytes transferred for a 300 second segment of Auckland trace and for Harpoon (no RTT configured in testbed).



(c) Log-scale diagrams for one hour segment of Auckland data set (12pm-1pm, 11 June 2001) and Harpoon with two different $IntervalDuration$ (ID) values and two different round-trip times.

Figure 11: Limitations of Harpoon related to packet volumes generated, and to matching original volumes over a range of time scales.

identical sequences in input distributions.

We compared forwarding rates in a series of tests run for 10 minutes each using the two different offered loads while separately changing traffic source burstiness, traffic source packet size, and forwarding table size. Note that changes in traffic source burstiness and packet size are only applicable to the constant spacing generator; Harpoon (intentionally) does not dictate these parameters[8]. Packet-level burstiness in Harpoon arises from the distributional parameters configured in Harpoon and the closed-loop dynamics of TCP.

For the constant spacing generator, we used burst sizes of 1 (uniformly spaced packets), 232, 465, 697, and 930. The burst size is defined by RFC 2285 [39] as a sequence of frames emitted with the minimum legal inter-frame gap, and a maximum recommended burst size is given as 930. The range of burst sizes we used covers $\frac{1}{4}$, $\frac{1}{2}$, and $\frac{3}{4}$ of the maximum burst size.

The packet sizes (number of bytes excluding link-layer framing) used in the constant spacing generator were 40, 1500, and a trimodal distribution configured so that 61% of the generated packets were 40 bytes, 17% were 576 bytes, and 22% were 1500 bytes. The empirical basis for this mix of packet sizes roughly approximates the values used in the LightReading tests, and comes from measurements taken at Merit Networks Inc. between 28 August, 2000 and 13 September, 2000 [33].

The sizes of forwarding tables used were 32, 1,024, 32,768, or 65,536 entries. The two traffic sources (two server workstations in Harpoon configuration, two separate GE cards in the AX/4000) were each configured with a single IP address. The traffic sink (one client workstation in Harpoon, a single GE card in the AX/4000) was configured to use an entire IPv4 class B address range (2^{16} addresses). With the AX/4000, this is simply a parameter in configuring the generated packets. With Harpoon, we created 2^{16} interface aliases. This address space was reached by installing static routes on the traffic source workstations and the forwarding tables on the 6509. We aliased the loopback adapter rather than the physical Ethernet adapter to avoid ARP overhead between the client host and the 6509, and to force a destination address lookup in the forwarding tables of the 6509.

[8]It is possible to change the MTU at each interface (physical or alias) used by Harpoon, indirectly allowing Harpoon to generate different packet sizes. We did not modify the default Ethernet MTU of 1500 bytes on the FreeBSD hosts running Harpoon.

6.2 Results

We begin by showing a qualitative comparison of the time series of bit forwarding rates for Harpoon and the constant spacing generator shown in Figure 12(a). In this test, Harpoon was configured to generate roughly 600Mbps of traffic over a 30 minute test period. The constant spacing generator was configured to match the average offered load of Harpoon and emitted 1500 byte packets with uniform spacing. Harpoon maintains a high level of offered load in the presence of variable file sizes transferred and inter-connection times while still generating burstiness in packet traffic. As expected, the constant spacing generator presents a virtually unwavering load to the router. In further tests, we explore how these fundamental differences in packet emission processes affect the performance of the router.

We examine the effects of burst sizes of 1, 465, 697, and 930 on forwarding rate in the results shown in Figure 12(b). The vertical bars on each data point indicate the range of one standard deviation above and below the average. The router was configured with a forwarding table size of 2^{15} and the high regime of offered load was used for these tests. Harpoon results for a forwarding table size of 2^{15} and high offered load are shown for comparison. The drop in bit forwarding rate for the constant spacing generator at a burst size of 697 accompanies a corresponding increase in the number of packets dropped at the egress line card of the router. The drop in forwarding rate between a burst size of 465 and a burst size of 930 is roughly 180 Mbps. This increase in stress placed on the router does not come with an increase in forwarding rate variability, as noted by the low variation in the height of vertical bars drawn for the constant spacing generator tests.

Figure 12(c) shows the effects of different packet sizes on forwarding performance. The constant spacing generator was configured to emit packet sizes of 40, 1500, and a trimodal distribution as described above. As with the burst tests above, the router was configured with a forwarding table size of 2^{15} and the high regime of offered load was used for these tests. Harpoon results for a forwarding table

size of 2^{15} and high offered load are plotted for comparison. With minimum sized IPv4 packets, the forwarding rate of the test router under the same offered load drops by roughly 140Mbps while exhibiting very low variability. Packet loss was measured for the 40 byte case, but there was no packet loss measured for the 1500 byte or trimodal case.

Figure 13 shows the affect of different forwarding table sizes on forwarding rates. The results from the low offered load test are shown in Figure 13(a) and results from high offered load test are shown in Figure 13(b). The obvious features in the figures are the near-perfect matching of average bit forwarding rates between Harpoon and the constant packet spacing generator at low loads, and the poor match for the test using a high offered load. Recall that the offered load used in the constant packet spacing tool was configured using the average offered load generated by Harpoon. For the high load test shown in Figure 13(b), the difference in forwarding rate comes as a direct result of packet loss that occurred during the Harpoon tests, but not during the tests using the constant spacing generator. Though the same average load is offered to the router by each traffic generator, the forwarding rate is clearly lower for the Harpoon test. Another important feature of these graphs is the trend in variance for the Harpoon tests. As the forwarding table size increases, so does the variance in bit forwarding rate. For the constant packet spacing generator, variance remains very low for all tests. Since Harpoon was configured so that each test used the same sequence of input parameters, we conclude that the burstiness in the traffic presented by Harpoon results in distinctly different and potentially highly variable forwarding performance at the router.

Figure 14(a) shows time series of packet loss for selected tests described above. All tests shown used a 2^{15}-entry forwarding table at the router and high offered load. It is clear from Figure 14(a) that even when the constant spacing generator emits bursts of packets, the loss rate measured at the router remains basically constant. The same is true for the constant spacing generator test with 40 byte packets. This observation is not surprising considering the packet-level generator operates in an open-loop, making no adjustment in offered load in response to packet loss. Because of the closed-loop nature of TCP, Harpoon is likely to generate some packet loss with *almost any* configured offered load, and it is likely that these losses will be correlated and bursty [51]. Finally, we note that all packet losses measured during our tests occurred as output buffer drops on the line card connected to the Harpoon client machine or the module on the packet-level generator acting as the data sink.

The effect of four test configurations on switch fabric utilization is shown in Figure 14(b)[9]. Surprisingly, the results for the constant spacing generator using uniformly spaced 1500 byte packets exhibits a wider range of fabric utilization levels than for the same generator using uniformly spaced 40 byte packets. The fluctuation in utilization measured during the 1500 byte test is likely due to limited precision of the utilization calculation internal to the 6509 (it is simply an integer). The highest level of switch fabric utilization is measured during the constant packet spacing generator test using 40 byte packets. It is interesting to note that despite sending much larger packets on average and despite a much lower packet loss rate (see Figure 14(a)), utilization during the Harpoon test comes within two percent of the highest measured fabric utilization. Fabric utilization during the Harpoon test also exhibits much greater variability than any of the constant spacing generator tests.

While not shown here, we ran experiments using the constant spacing generator and a limited cross-product of parameters explored above (burst size, packet size, forwarding table size, and offered load). As one might expect, loss rates when sending 40 byte packets in bursts are greater than when sending uniformly spaced 40 byte packets. The key observation from all these tests is that variability in bit forwarding rate, loss rate, and switch fabric utilization remains very low for the constant spacing generator.

6.3 Implications

It is clear that precisely controlled traffic streams are useful for Internet RFC conformance testing and for subjecting network systems to extreme conditions along certain dimensions. However, our experiments demonstrate that a workload based on measured characteristics of real Internet traffic generates a fundamentally different and more variable load on routers. Our results suggest ranges of behaviors that can be expected for given average loads. These ranges could be used to tune constant bit rate streams to explore an appropriate operational space. Finally, the subsystem load variability imposed by Harpoon should provide insights to system designers on the stresses that these systems might be subjected to under real operating conditions. This could inform the allocation of resources in future system designs.

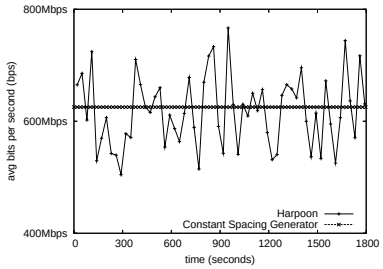
7. CONCLUSIONS AND FUTURE WORK

Harpoon is a new tool for generating representative IP traffic based on eight distributional characteristics of TCP and UDP flows. Parameters for these distributions can be automatically extracted from NetFlow data collected from a live router. These characteristics enable Harpoon to generate statistically representative traffic workloads that are independent of any specific application. We are not aware of any other workload generation tool with this capability. We implemented Harpoon as a client-server application that can be used in testbed environments. We parameterized Harpoon using data collected from a NetFlow trace and from a set of packet traces and verified in controlled laboratory tests that Harpoon generates traffic that is qualitatively the same as the input data.

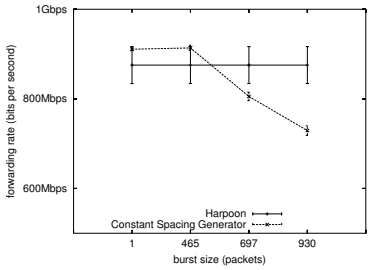
We demonstrated Harpoon's utility beyond simple background traffic generation through a series of throughput tests conducted on a Cisco 6509 router. We compared and contrasted the workload generated by Harpoon with the constant bit rate workloads recommended for standard tests. We found that Harpoon generated similar results for overall throughput, but that the stresses placed on router subsystems by Harpoon during these tests were significantly different. These results suggest that (in addition to the background traffic generation) Harpoon could be useful as a tool for providing network hardware designers and network operators insight into how systems might behave under realistic traffic conditions.

An area for future work is to extend our parameterization tools and model to accommodate sampled flow records and the absence of TCP flags. We also intend to augment the

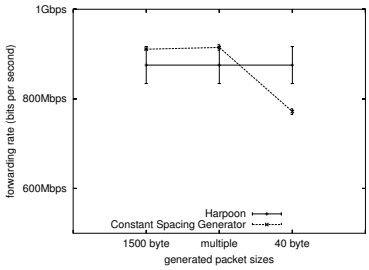
[9]Switch fabric utilization on the 6509 is reported for both input and output channels to each module installed in the chassis. We show results for the output channel on the module where traffic is multiplexed on the path to the Harpoon client machine or packet-level generator data sink module.



(a) Packet forwarding rate time series for Harpoon and the constant spacing packet generator.

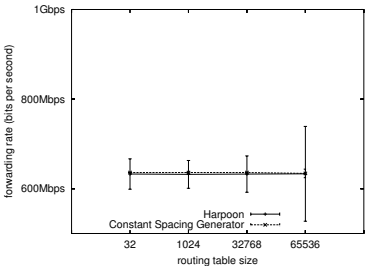


(b) Average forwarding rates using different burst sizes for the constant spacing traffic generator. Configuration is 2^{15} entries in forwarding table and high offered load. Harpoon results for 2^{15} forwarding table entries and high offered load shown for comparison. Vertical bars span one standard deviation above and below the mean.

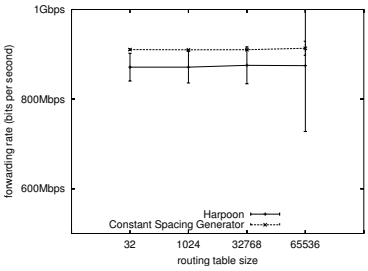


(c) Average forwarding rates using different packet sizes for constant spacing packet generator. Configuration is 2^{15} entries in forwarding table and high offered load. Harpoon results for 2^{15} forwarding table entries and high offered load shown for comparison. Vertical bars span one standard deviation above and below the mean.

Figure 12: Qualitative contrast of forwarding rates between Harpoon and the constant spacing traffic generator, and comparisons of forwarding rates between Harpoon and the constant spacing traffic generator using different burst lengths and packet sizes.



(a) Low Offered Load



(b) High Offered Load

Figure 13: Average forwarding rates for Harpoon and the constant spacing traffic generator for different router forwarding table sizes. Vertical bars span one standard deviation above and below the mean.

UDP traffic model to enable a broader set of UDP traffic characteristics. Finally, Harpoon assumes all sources are well behaved, which is far from the case in the Internet. We intend to pursue creation of traffic anomaly models that can be incorporated into the Harpoon framework.

8. ACKNOWLEDGMENTS

Thanks to Dave Plonka at the University of Wisconsin for helpful discussions regarding Netflow, and to Spirent Communications for use of the AX/4000 system. We also thank the anonymous reviewers and our shepherd, Anja Feldmann, for constructive criticism.

This material is based upon work supported by the National Science Foundation under Grant No. 0335234 and by support from Cisco Systems. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or Cisco Systems.

9. REFERENCES

[1] Catalyst 6500 series switches. http://www.cisco.com/univercd/cc/td/doc/product/lan/cat6000/index.htm. Accessed August 2004.

[2] Cisco's IOS Netflow feature. http://www.cisco.com/warp/public/732/netflow. Accessed August 2004.

[3] CoralReef: Passive network traffic monitoring and statistics collection. http://www.caida.org/tools/measurement/coralreef. Accessed August 2004.

[4] Endace measurement systems. http://www.endace.com/. Accessed August 2004.

[5] The eXpat XML parser. http://expat.sourceforge.net. Accessed August 2004.

[6] The iperf TCP/UDP Bandwidth Measurement Tool. http://dast.nlanr.net/Projects/Iperf. Accessed August 2004.

[7] Netflow services solutions guide (Netflow white paper). http://www.cisco.com/univercd/cc/td/doc/cisintwk/-intsolns/netflsol/nfwhite.htm. Accessed August 2004.

[8] Spirent Communications Inc. Adtech AX/4000 broadband test system. http://www.spirentcom.com/analysis/product_line.cfm?pl=1&WS=173&wt=2. Accessed August 2004.

[9] SSFnet network simulator. http://www.ssfnet.org. Accessed August 2004.

[10] The University of New Hampshire Interoperability Laboratory. http://www.iol.unh.edu. Accessed August 2004.

[11] The Wisconsin Advanced Internet Laboratory. http://wail.cs.wisc.edu. Accessed August 2004.

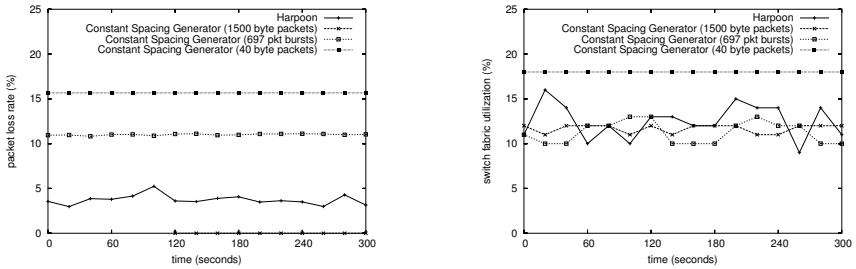
[12] UCB/LBNL/VINT Network Simulator - ns (version 2). http://www.isi.edu/nsnam/ns. Accessed August 2004.

[13] Web polygraph. http://www.web-polygraph.org. Accessed August 2004.

[14] XML-RPC home page. http://www.xmlrpc.org. Accessed August 2004.

[15] Workshop on models, methods and tools for reproducible network research. http://www.acm.org/sigs/-sigcomm/sigcomm2003/workshop/mometools, 2003.

[16] P. Abry and D. Veitch. Wavelet analysis of long range



(a) Packet Loss Rate

(b) Output Channel Fabric Utilization

Figure 14: Packet loss rates and switch fabric utilization using 2^{15} forwarding table entries and high offered load. Results shown for Harpoon, constant spacing generator with uniformly spaced packets of 1500 bytes, bursts of 697 packets of 1500 bytes, and uniformly spaced packets of 40 bytes.

dependent traffic. *IEEE Transactions on Information Theory*, 44(1):2–15, 1998.

[17] C. Barakat, P. Thiran, G. Iannaccone, C. Diot, and P. Owezarski. Modeling Internet backbone traffic at the flow level. *IEEE Transactions on Signal Processing (Special Issue on Networking)*, August 2003.

[18] P. Barford and M. Crovella. Generating representative workloads for network and server performance evaluation. In *Proceedings of ACM SIGMETRICS '98*, pages 151–160, Madison, WI, June 1998.

[19] P. Barford and M. Crovella. A performance evaluation of hyper text transfer protocols. In *Proceedings of ACM SIGMETRICS '99*, Atlanta, GA, May 1999.

[20] S. Bradner. Benchmarking terminology for network interconnect devices. IETF RFC 1242, July 1991.

[21] S. Bradner and J. McQuaid. Benchmarking methodology for network interconnect devices. IETF RFC 2544, March 1999.

[22] T. Bu and D. Towsley. Fixed point approximation for TCP behavior in an AQM network. In *Proceedings of ACM SIGMETRICS '01*, San Diego, CA, June 2001.

[23] Y.-C. Cheng, U. Hölzle, N. Cardwell, S. Savage, and G. M. Voelker. Monkey see, monkey do: A tool for TCP tracing and replaying. In *Proceedings of the USENIX 2004 Conference*, June 2004.

[24] K. Claffy, G. Polyzos, and H.-W. Braun. Internet traffic flow profiling. Technical Report TR-CS93-328, University of California San Diego, November 1989.

[25] W. Cleveland, D. Lin, and D.Sun. IP packet generation: Statistical models for TCP start times based on connection rate superposition. In *Proceedings of ACM SIGMETRICS '00*, Santa Clara, CA, June 2000.

[26] M. Crovella and A. Bestavros. Self-similarity in World Wide Web traffic: Evidence and possible causes. *IEEE/ACM Transactions on Networking*, 5(6):835–846, December 1997.

[27] N. Duffield, C. Lund, and M. Thorup. Estimating flow distributions from sampled flow statistics. In *Proceedings of ACM SIGCOMM '03*, Karlsruhe, Germany, August 2003.

[28] A. Feldmann, A. Gilbert, P. Huang, and W. Willinger. Dynamics of IP traffic: A study of the role of variability and the impact of control. In *Proceedings of ACM SIGCOMM '99*, Boston, MA, August 1999.

[29] A. Feldmann, A. Gilbert, and W. Willinger. Data networks as cascades: Investigating the multifractal nature of Internet WAN traffic. In *Proceedings of ACM SIGCOMM '98*, August 1998.

[30] S. Floyd and E. Kohler. Internet research needs better models. In *Hotnets-I*, Princeton, NJ, October 2002.

[31] S. Floyd and V. Paxson. Difficulties in simulating the Internet. *IEEE/ACM Transactions on Networking*, 9(4), August 2001.

[32] M. Fomenkov, K. Keys, D. Moore, and K. Claffy. Longitudinal study of Internet traffic from 1998-2001: a view from 20 high performance sites. Technical report, Cooperative Association for Internet Data Analysis (CAIDA), 2002.

[33] N. L. for Applied Network Research. http://moat.nlanr.net/Datacube. Accessed August 2004.

[34] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, and C. Diot. Packet-level traffic measurements from the Sprint IP backbone. *IEEE Network*, 2003.

[35] S. Fredj, T. Bonald, A. Proutiere, G. Regnie, and J. Roberts. Statistical bandwidth sharing: A study of congestion at flow level. In *Proceedings of ACM SIGCOMM '01*, San Diego, CA, August 2001.

[36] M. Fullmer and S. Romig. The OSU flow-tools package and Cisco NetFlow logs. In *Proceedings of the USENIX Fourteenth System Administration Conference LISA XIV*, New Orleans, LA, December 2000.

[37] S. Jin and A. Bestavros. GISMO: Generator of Streaming Media Objects and Workloads. *Performance Evaluation Review*, 29(3), 2001.

[38] W. Leland, M. Taqqu, W. Willinger, and D. Wilson. On the self-similar nature of Ethernet traffic (extended version). *IEEE/ACM Transactions on Networking*, pages 2:1–15, 1994.

[39] R. Mandeville. Benchmarking terminology for LAN switching devices. IETF RFC 2285, February 1998.

[40] R. Mandeville and J. Perser. Benchmarking methodology for LAN switching devices. IETF RFC 2889, August 2000.

[41] D. Newman, G. Chagnot, and J. Perser. Internet core router test. http://www.lightreading.com/document.asp?site=testing&doc_id=4009, March 2001. Accessed August 2004.

[42] K. Park and W. Willinger. *Self-Similar Network Traffic and Performance Evaluation*. Wiley Interscience, 2000.

[43] V. Paxson. *Measurements and Analysis of End-to-End Internet Dynamics*. PhD thesis, University of California Berkeley, 1997.

[44] V. Paxson and S. Floyd. Wide-area traffic: The failure of poisson modeling. *IEEE/ACM Transactions on Networking*, 3(3):226–244, June 1995.

[45] D. Plonka. Flowscan: A network traffic flow reporting and visualization tool. In *Proceedings of the USENIX Fourteenth System Administration Conference LISA XIV*, New Orleans, LA, December 2000.

[46] A. Turner. tcpreplay. http://tcpreplay.sourceforge.net/. Accessed August 2004.

[47] S. Uhlig. Simulating interdomain traffic at the flow level. Technical Report Infonet-TR-2001-11, University of Namur, Institut d' Informatique, 2001.

[48] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostic, J. Chase, and D. Becker. Scalability and accuracy in a large-scale network emulator. In *Proceedings of 5th Symposium on Operating Systems Design and Implementation (OSDI)*, Boston, MA, December 2002.

[49] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *Proceedings of 5th Symposium on Operating Systems Design and Implementation (OSDI)*, Boston, MA, December 2002.

[50] W. Willinger, M. Taqqu, R. Sherman, and D. Wilson. Self-similarity through high-variability: Statistical analysis of Ethernet LAN traffic at the source level. *IEEE/ACM Transactions on Networking*, 5(1):71–86, February 1997.

[51] M. Yajnik, S. Moon, J. Kurose, and D. Towsley. Measurement and modeling of temporal dependence in packet loss. In *Proceedings of IEEE INFOCOM '99*, New York, NY, March 1999.