Geist

Internet Traffic Generator for Server Architecture Evaluation

Krishna Kant,
Vijay Tewari,
Ravishankar Iyer

Intel Corporation

Overview

- Motivation
- Other Traffic Generators
- Features of Geist
- Architecture of Geist
- Performance Issues
- Uses
- Limitations & Future Work

Motivation

- Need to generate representative HTTP traffic.
- Typical usage: Detailed study of Web and E-commerce servers.
- Interested primarily in aggregate traffic as seen by the server.
- Lab environment:
  - Need to emulate user behavior from aggregate traffic perspective.
  - High speed LAN connection => Need to emulate network impact on the traffic.

Other Tools: WAS

- Web Application Stress (WAS) Tool from Microsoft
- Strengths
  - Secure requests to server.
  - Can use multiple client boxes.
  - Collects data from all clients and can collect Perfmon data from the server under test.
  - Good UI
- Weaknesses
  - Emulates users directly => Cannot generate high traffic rates.
  - Access characteristics must be set individually.
  - Cannot specify arrival time distribution and hence cannot be used to generate “Bursty” traffic.
  - Cannot make secure requests via a proxy.
  - Subject to implicit feedback from the server based on response time of the server.
  - No access to source code.

Other Tools: SURGE

- **SURGE: Scalable URL reference generator.**
- **Strengths**
  - Generates references matching
    - Server File size distribution.
    - Request Size distribution.
    - Relative file popularity.
    - Embedded File references.
- **Weaknesses**
  - Not based on aggregate traffic as seen by the server.
  - Cannot generate traffic for overload studies.
  - No capability of generating secure traffic.

Traffic Generator Features

- **Flexible arrival process characteristics**
  - Marginal distribution (Inter arrival times).
  - Includes user behavior (self-similarity).
  - Includes network effects (multifractality).
  - Can generate non-stationary & overload situations.
- **Request and Response features:**
  - Request size distribution.
  - Popularity distribution of Files/ Directories.
- **Generates aggregated traffic:**
  - Not subject to implicit feedback from the server.
  - Scalable to high traffic levels.

Traffic Generator Features

- **Can generate e-commerce traffic**
  - Transactional classification.
  - Dependencies between successive user transactions.
- **Other features:**
  - Fine grained logging of response times and HTTP response codes.
  - Secure and Non Secure Traffic, direct and via Proxy servers
  - Custom HTTP Directives in the request.
  - Automatic distribution of traffic among a number of clients.

Overall Architecture of Traffic Generator

- **Two step process.**
  - Trace generation.
    - Generates request arrival times and other parameters.
    - Based on statistical characterization of traffic.
  - Request generation.
    - Distributes trace files among clients.
    - Generates actual requests and handles responses.

- **Why a two step process ?**
  - Trace generation can be quite expensive
  - Waiting time of request generators must be minimized.
  - Gives more flexibility:
    - Request generator could be driven from an actual trace.
    - Trace generator run once and output used repeatedly.

Traffic Trace Generator

- **Traffic arrival process issues:**
  - Need to control marginal distribution of arrival counting process.
  - Impact of user behavior on arrival process
    - Heavy-tailed active and inactive periods
    - => Burstiness at large time scales (I.e., asymptotically self-similar traffic)
  - Impact of network on the traffic (segmentation, reassembly, TCP retries, network delays, embedded requests, etc.)
    - Traffic burstiness at medium time scales (e.g., RTT level).
    - Can be captured using multifractal paradigm (e.g., a cascade construction).
  - Non-stationarity in traffic, if relevant.
    - Assume that correlational properties are stationary.
    - Represent nonstationarity using a level shift process.

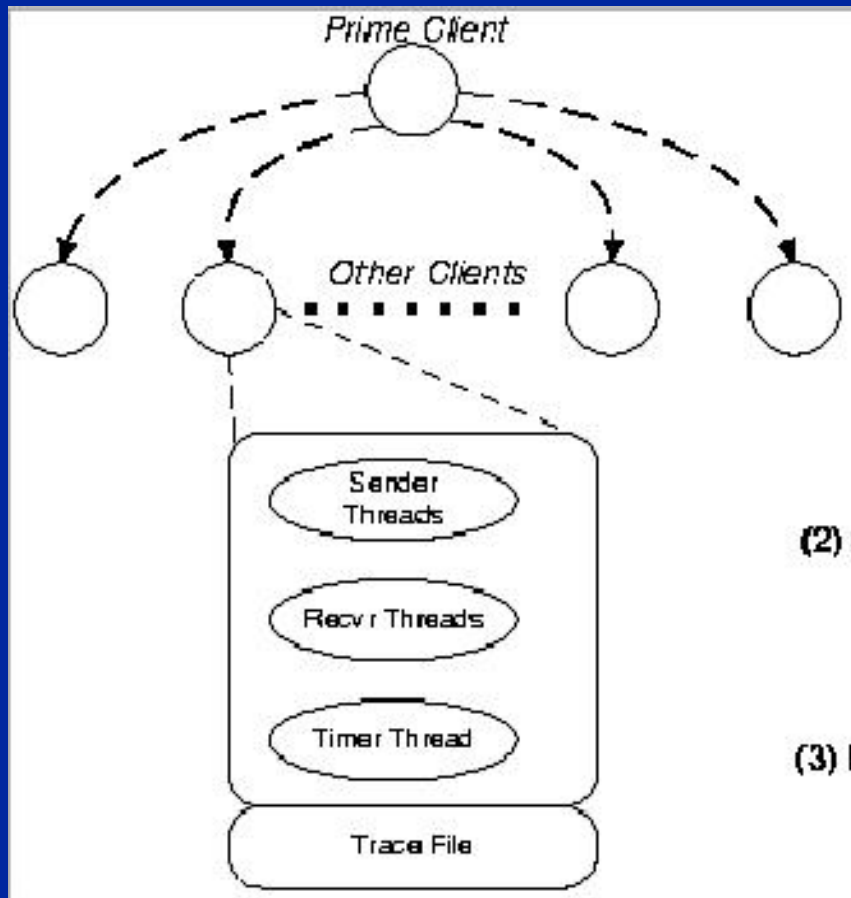
Traffic Trace Generator

- **Request characteristics**
  - Request size distribution
  - Request type (get, post, etc.): requested script & its parameters.
  - Frequency and locality of accesses to the web pages.
  - Other characteristics:
    - Percentage of secure traffic.
    - Cache control and other pragmas.
  - Multiple requests/connection & pipelining of requests (not done currently).

Arrival Process Generation

- **Use M/G/\infty traffic, I.e.**
  - A slotted-time virtual delay server.
  - Poisson input and heavy-tailed service time.
  - #arrivals/slot taken as #customers in the queue at the end of each time slot.
- **Multifractal behavior introduced by a limited semi-random cascade:**
  - Collect arrivals over N consecutive slots.
  - Redistribute this mass over left & right haves using a symmetric RV with mean 0.5.
  - Continue this until interval size is 1 slot.

Overview of Traffic Generation

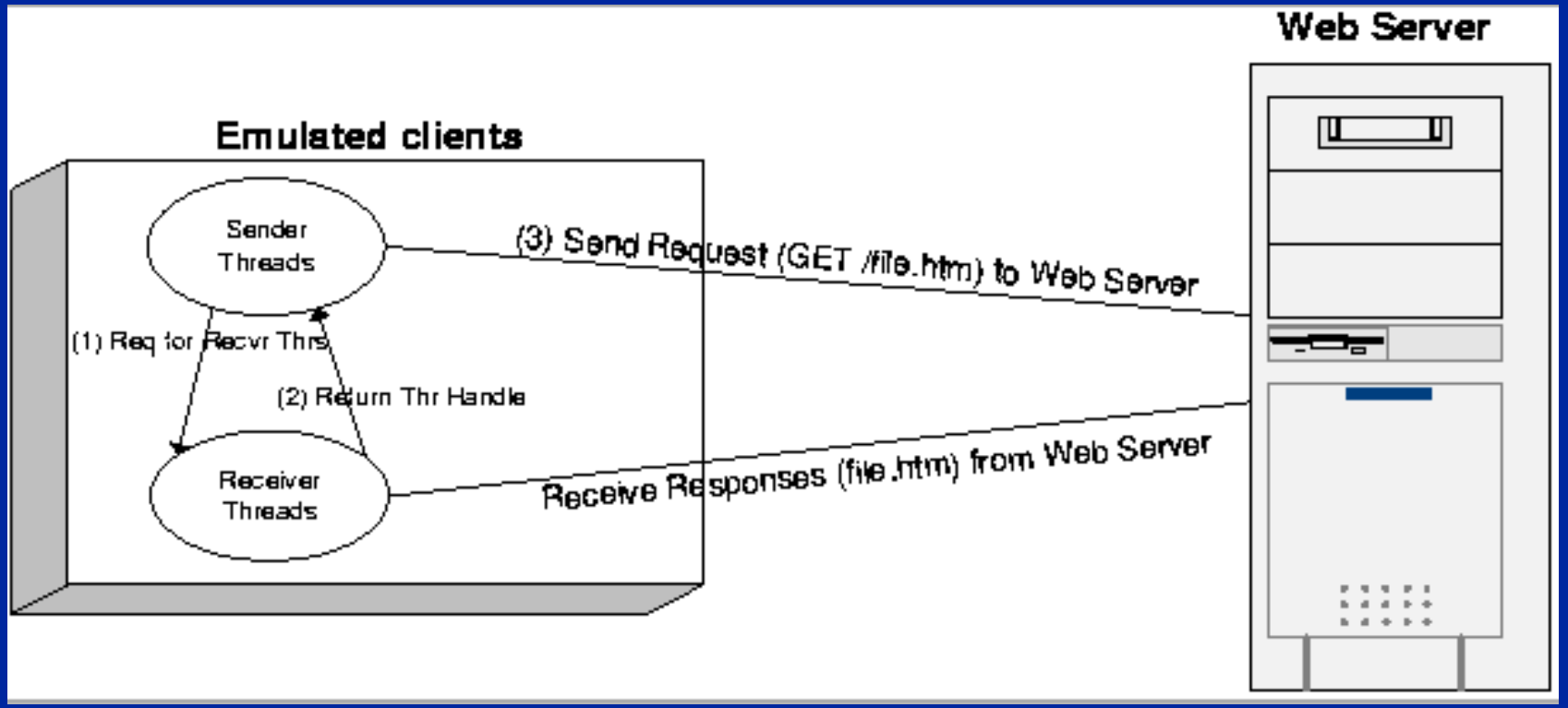


(1) Broadcast Config Information

(2) Generate Requests based on Trace File and Config Information Received

(3) Process Received Responses and Store Statistics and Logs

Request Generation Process



Request Generator Architecture

- Windows Based console application developed in "C".
- Configuration parameters specified in a file.
  - IP address and port number of Web Server to send requests to.
  - IP address and port number of proxy server.
  - Warm time (No logging).
  - Run time of the test.
  - Number of sender and receiver threads.
- Traffic Generator runs on multiple boxes
  - One "prime" client that controls all others.
  - Prime client uses UDP to distribute configuration information to other clients.
- Timer thread controls warm time and run time.

Send/Receive Operations

- **De - linked “send” and “receive” to avoid implicit feedback .**
- **Sender threads loop through**
  - Remains in an efficient sleep state till it is time to make a request as specified in the trace file (Memory mapped to avoid disk I/O bottleneck).
  - Once awoken the Sender thread looks for Receiver threads that are currently available to handle responses and marks them as used.
  - Makes a connection to the server and sends an HTTP request.
  - Hands over the socket to the receiver thread to receive the response.
- **Receiver Threads loop through**
  - Remain in an efficient sleep state till awoken by a sender.
  - Receive the HTTP response from the server and log appropriate information.
  - Returns to blissful state.

Secure Request Implementation

- Secure requests use SSL
  - Open-source version OpenSSL (http://www.openssl.org).
  - May not be as efficient as proprietary implementations.
- Basic procedure
  - Sender thread establishes TCP connection with the server on SSL port (generally 443).
  - Proceeds with the SSL handshake.
  - Encrypts and sends the HTTP request using the secret key and the agreed algorithm (e.g., RC4).
  - Receiver threads receives and decrypts the HTTP response.

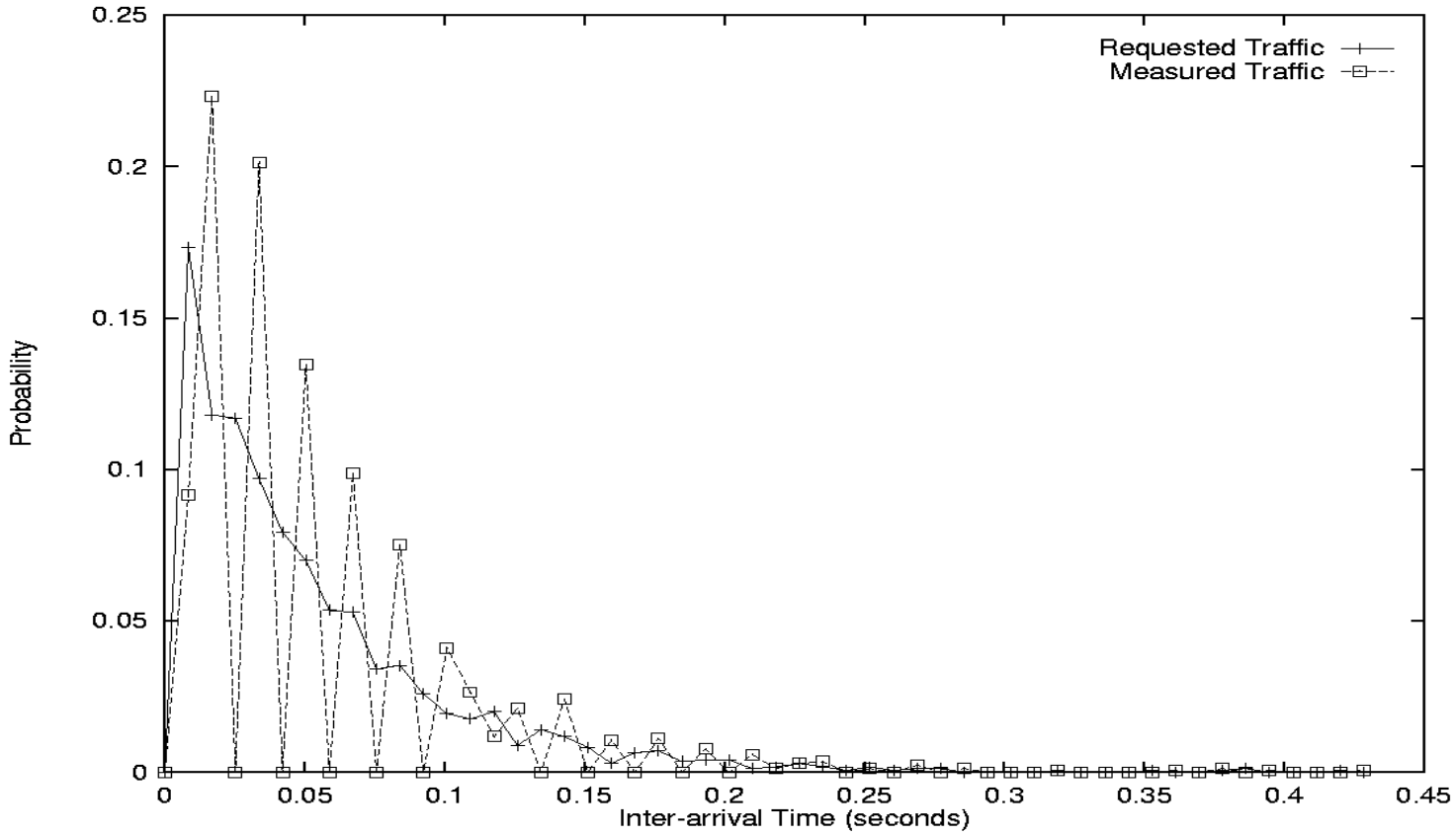
Secure requests via a Proxy

- Secure Traffic via a Proxy
  - Client establishes a TCP connection with the proxy server and sends the following command
    - "CONNECT www.servername.com:443 HTTP/1.0"
  - Proxy server establishes TCP connection with the Web Server and sends the following to the client
    - HTTP/1.0 200 Connection established
  - Client now starts SSL handshake. (Note that the proxy knows nothing of the content).
  - Client exchanges data with the server as before with proxy acting as a transparent transfer agent.

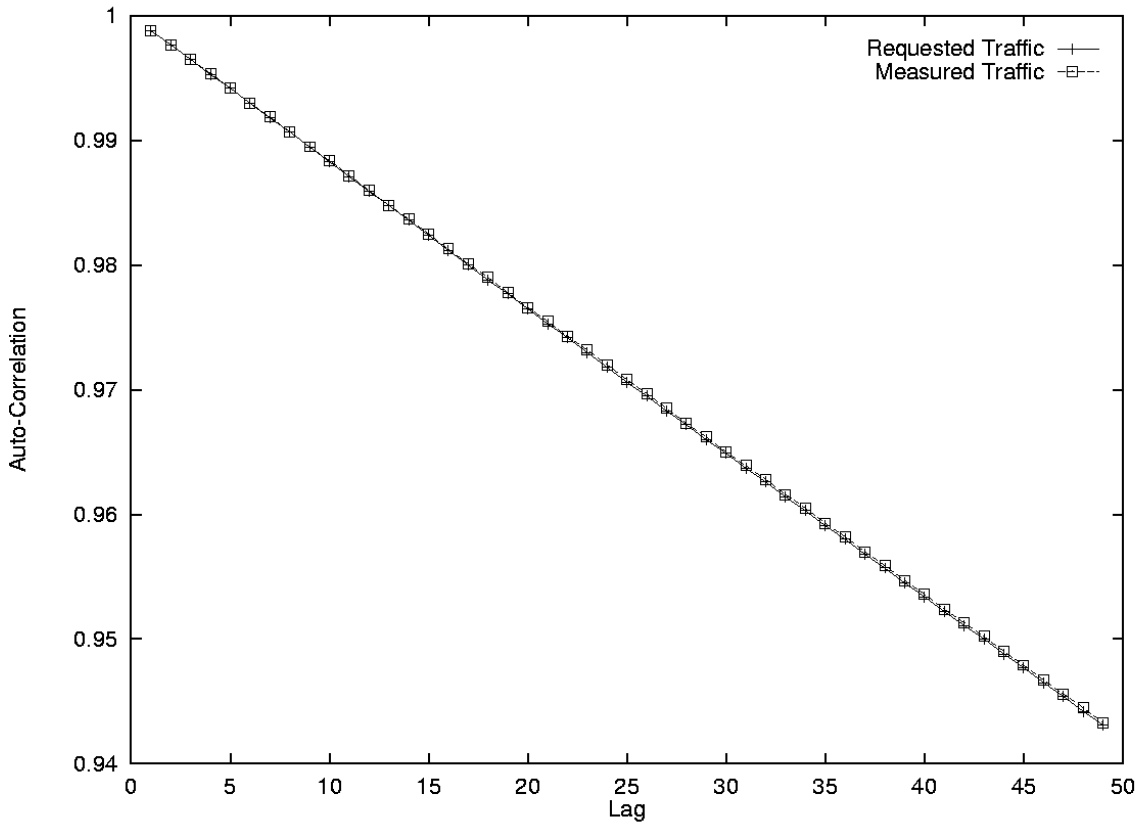
Performance Issues

- Parameters for optimal performance and accuracy in request generation
  - Number of sender threads : number of receiver threads.
  - Number of requests per attempt.
  - Number of Clients.
- Time Slippage: Timing difference between actual request times and those in the trace file.
  - Slippage depends upon
    - Number of threads
    - Traffic rate per client.
    - Traffic burstiness
    - Platform (processor and memory).

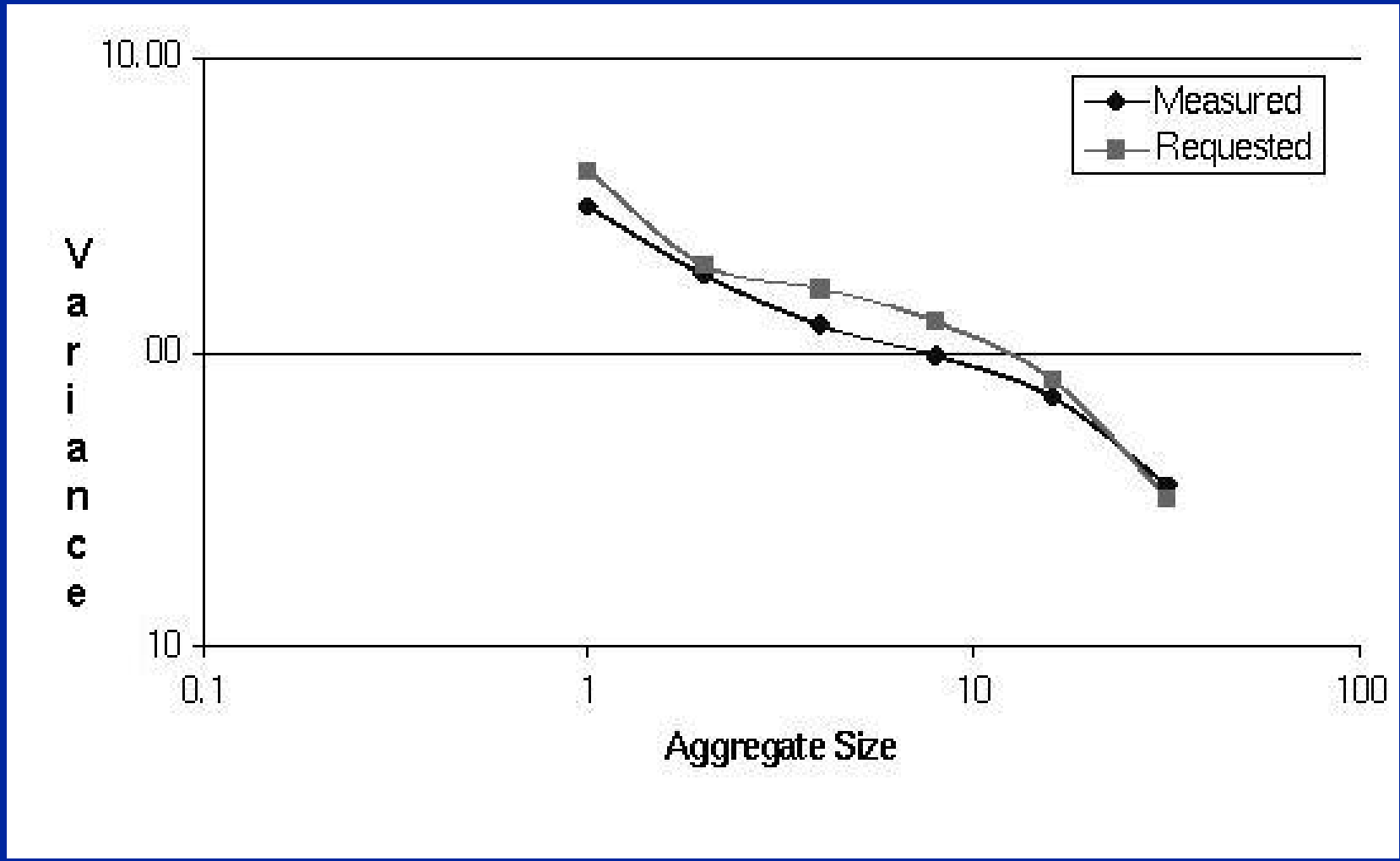
Interarrival time distribution



Autocorrelation Function



Variance Time Plots of Requested and Measured Traffic



Uses

- **Past and current**
  - Used for Overload Control experiments.
  - Being used for SSL deep tracing.
  - Being used for Proxy Server experiments
- **Future**
  - E-Commerce experiments.
  - Firewall Experiments.
  - TCP offloading studies.

Limitations/Future Work.

- **Limitations**
  - Limited to HTTP “GET” requests only.
  - Cannot handle HTTP 1.1 type persistent connections.
  - Does not model user abandonment and retry behavior.
  - Cannot handle Cookies.
  - Not GUI based.
  - Generation of Dynamic “GET” parameters
  - Cannot make Conditional Requests. (If-Modified-Since).
  - Does not follow HTTP redirects.
- **Future Work**
  - Plan on adding the features listed under the current limitations.
  - Release into the Open Source Community.