

Thread Assignment in Multicore/Multithreaded Processors: A Statistical Approach

– Appendices –

Petar Radojković, Paul M. Carpenter, Miquel Moretó, Vladimir Čakarević, Javier Verdú, Alex Pajuelo, Francisco J. Cazorla, Mario Nemirovsky and Mateo Valero, *Fellow, IEEE*

Note: For the sake of readability, the appendix includes some statements that have been already discussed in the main document.

APPENDIX A BENCHMARKS

A.1 Benchmark description

This section describes the benchmarks that we used in this study. Since Netra DPS omits standard OS features including dynamic memory allocation and file management, we had to adapt some of the benchmarks to execute in this environment.

The benchmarks used are described next:

(1) **IP Forwarding (IPFwd)** is one of the most representative Layer2/Layer3 network applications. This application decides where to forward a packet for the next hop based on the destination IP address. Depending on the size of the lookup table and destination IP addresses of the packets that are to be processed, IPFwd may exhibit significantly different memory behavior. In order to cover different cases of IPFwd memory behavior, we created two variants of the IPFwd application, both based on the IPFwd application included in the Netra DPS distribution [2]:

(i) The lookup table fits in the L1 data cache (**IPFwd-L1**);

(ii) The lookup table entries are initialized to make IPFwd continuously access the main memory (**IPFwd-Mem** benchmark).

IPFwd-L1 is representative of the best case of IPFwd memory behavior, since it shows high locality in data cache accesses. On the other hand, IPFwd-Mem represents the

worst case of IPFwd memory behavior used in network processing studies, in which there is no cache locality between accesses to the lookup table [26].

(2) **Packet analyzer** is a program that can intercept and log traffic passing over a network or part of a network [12]. Packet analyzers are important tools for network monitoring and management used to troubleshoot network problems, examine security issues, gather and report network statistics, detect suspect content, and filter it from the network traffic [1][27][34].

The packet analyzer used in the experiments captures each packet that passes through the Network Interface Unit (NIU), decodes the packet, and analyzes its content according to the appropriate RFC specifications [21]. The packet analyzer can display information from packet header fields at Layer 2, Layer 3, and Layer 4, and about the packet payload. A user can decide to log all traffic that passes through the NIU, or to define filters based on many criteria. In the experiments presented in this paper, we used the packet analyzer to log the MAC source and destination address, time-to-live field, Layer 3 protocol, source and destination IP address, and source and destination port number of all packets passing through the processor's NIU.

(3) **Aho-Corasick** is a string matching algorithm. String matching is the basic technique to analyze network traffic at the application layer [19]. In networking, string matching algorithms search for a set of keywords (strings) in the payload of the network packets. Aho-Corasick is an efficient algorithm that locates all occurrences of any keyword in the given set within a string of text (packet payload in case of packet processing). The algorithm constructs a finite state machine (finite automaton) for keyword pattern matching and uses it to process the text in a single pass [4]. The Aho-Corasick string matching algorithm has linear performance, and it is suitable for searching for a large set of keywords concurrently. The Aho-Corasick algorithm is used in state-of-the-art network intrusion detection systems such as Snort [23].

In the experiments presented in this paper, we used the Aho-Corasick algorithm to search the packet payloads for the keywords in the Snort Denial-of-Service set of intrusion detection rules (version 2.9).

- P. Radojković, P. M. Carpenter, M. Moretó, and V. Čakarević are with Barcelona Supercomputing Center (BSC), Barcelona, Spain. email: {petar.radojkovic, paul.carpenter, miquel.moreto, vladimir.cakarevic}@bsc.es
- J. Verdú and A. Pajuelo are with UPC, Barcelona, Spain. email: {jverdú, mpajuelo}@ac.upc.edu
- F. Cazorla is Scientific Researcher in the Spanish National Research Council (IIIA-CSIC) and with BSC, Barcelona, Spain. email: francisco.cazorla@bsc.es
- M. Nemirovsky is ICREA Research Professor and with BSC, Barcelona, Spain. email: mario.nemirovsky@bsc.es
- M. Valero is with UPC and BSC, Barcelona, Spain. email: mateo@ac.upc.edu

(4) **Stateful packet processing** is an important component of state-of-the-art network monitoring tools [24][31] and intrusion prevention and detection systems [27]. Unlike stateless applications, which process packets independently (examples include the IPFwd, Packet analyzer, and Aho-Corasick benchmarks above), stateful packet processing keeps information from the processing of previous packets.

The packets that belong to the same *flow*, i.e. that have the same *flow-keys*,¹ share common information called the *flow-record* [14]. The flow-record of a given flow identifies whether the flow is open (the connection is established), safe, malicious, etc. The information about the active flows is stored in a hash table, which is indexed based on the flow-keys. The common main components of stateful packet processing are: (1) Determine the flow-keys of a packet; (2) Use the hash function on the flow-keys to find the corresponding hash table entry; (3) Access the hash table: lock, read, and update the flow-record of an already-existing flow, or create a flow-record for a new flow.

The stateful packet processing benchmark that we used in the experiments is comprised of these three components. The stateful benchmark uses the same hash function as that implemented in the *nProbe* network monitor [14][24]. The hash table contains 2^{16} entries, which is a size previously used for testing network monitoring tools [14], and is sufficient to store the records of active flows of fully-utilized 10Gb link [30].

A.2 Benchmark implementation

As shown in Fig. 1, each benchmark is divided into three threads, Receiving (R), Processing (P), and Transmitting (T), forming a software pipeline. This is a common approach for implementing network applications [3][35]. The workload consists of several benchmark instances running concurrently, each of which has the following three threads:

- The receiving thread (R) reads packets from the Network Interface Unit (NIU) associated with the receiver's 10Gb network link, and writes pointers to packets into the R→P memory queue.
- The processing thread (P) reads the packeted pointers from the memory queue, processes the packets, and writes the packet pointers to the P→T memory queue. Packet processing is different for each benchmark; e.g. the P threads in the *IPFwd-L1* and *IPFwd-Mem* benchmarks read the destination IP address, call the hash function, and access the lookup table; the P thread in the *Aho-Corasick* benchmark searches for the keywords in the packet payload.
- Finally, the transmitting thread (T) reads the packet pointers from the P→T memory queue, and sends the packets to the network through the NIU associated to the 10Gb network link.

1. The flow-keys are typically the source and destination IP address, the source and destination port, and protocol.

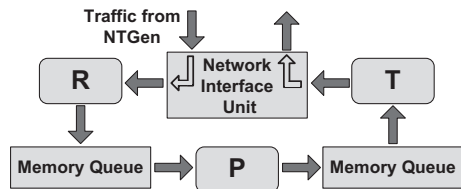


Fig. 1. The schematic view of the benchmarks

APPENDIX B PEAK OVER THRESHOLD (POT) METHOD

B.1 Theoretical background

We estimate the performance of the best thread assignment using the Peak Over Threshold (POT) statistical method. The POT method analyzes the distribution of the observations that exceed a given (high) threshold. We will illustrate the theoretical foundation for the POT method using the cumulative distribution function (CDF). For example, assume that F is the CDF of a random variable X . The POT method can be used to estimate the cumulative distribution function F_u of values of x above a certain threshold u . The function F_u is called the *conditional excess distribution function*, and it is defined as:

$$F_u(y) = P(X - u \leq y \mid X > u), \quad 0 \leq y \leq x_F - u,$$

where X is the observed random variable, u is the given threshold, $y = x - u$ are the exceedances over the threshold, and $x_F \leq \infty$ is the right endpoint of the cumulative distribution function F . Figure 2 shows a CDF of a random variable X (upper chart) and the corresponding conditional excess distribution function $F_u(y)$ (lower chart).

The POT method is based on the *Pickands-Balkema-de Haan* theorem [6][25]:

Theorem 1: For a large class of underlying distribution functions F , the conditional excess distribution function $F_u(y)$, for u large, is well approximated by *Generalized Pareto Distribution* $G_{\xi,\sigma}(y)$ where

$$G_{\xi,\sigma}(y) = \begin{cases} 1 - (1 + \frac{\xi}{\sigma}y)^{-1/\xi} & \text{for } \xi \neq 0 \\ 1 - e^{-y/\sigma} & \text{for } \xi = 0 \end{cases}$$

for $y \in [0, (x_F - u)]$ if $\xi \geq 0$ and $y \in [0, -\frac{\sigma}{\xi}]$ if $\xi < 0$.

This means that the F_u of numerous distributions that present real-life problems can be approximated with Generalized Pareto Distribution (GPD). For each particular problem, the decision as to whether GPD can be used to model the problem is made based on how well the sample of observations can be fitted to GPD. We describe the goodness of fit of observations to GPD in Appendix B.3, in Steps 2, 3, and 4. GPD is defined with two parameters: shape parameter ξ and scaling parameter σ . An important characteristic of GPD used in this study is that for $\xi < 0$ the upper bound of the observed value (in our study, the performance of the best thread assignment) can be computed as $u - \frac{\sigma}{\xi}$, where σ and ξ are the GPD parameters and u is the selected threshold [16][22].

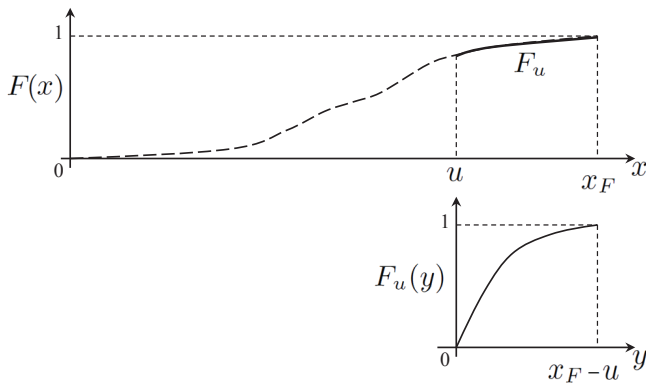


Fig. 2. Cumulative distribution function $F(x)$ and corresponding conditional excess distribution function $F_u(y)$

B.2 POT requirements

EVT has two main prerequisites, which should be validated before applying the theory to a particular real-life problem [17]. The first prerequisite is that the sample under study is comprised of independent and identically distributed (i.i.d.) observations. The second prerequisite is that the probability distribution of the statistics under study should be continuous.

B.2.1 Independent and identically distributed

Random variables X and Y are independent if their joint cumulative distribution function is the product of their (separate) cumulative distribution functions: $F_{X,Y}(x,y) = F_X(x)F_Y(y)$. Intuitively, this means that random variables are independent if knowing the value of one of them gives no new information about the values of the others. They are identically distributed if they each have the same probability distribution. Note that it is not necessary that the observations are uniformly distributed. Uniform distribution means that each valid thread assignment is selected with the same probability. The i.i.d. requirement can be assured in three ways: (1) Proper design of the sampling method, (2) Proper execution of the experiments, (3) Using statistical tests that confirm or reject the i.i.d. property of the observations in a sample.

(1) Proper design of the sampling method: The sampling method should be designed to select independent observations that all have the same marginal distribution. This means that the thread assignments have to be taken from a single population using sampling with replacement [29].

The sampling method used in this study is described next. For example, assume that a workload of T threads is to be assigned on a hardware contexts of the processor with integers from 1 to V . To generate a random thread assignment, we randomly select, for each thread in the workload, an integer from this interval. The number represents the hardware context to which the thread is mapped. After mapping all threads, we check if the generated thread assignment is valid. An assignment is not valid if two or more threads are mapped to the same hardware context. If

this is the case, we discard the invalid assignment and try again. This process is repeated until the sample contains the required number of thread assignments. As assignments in the sample are independent and they are sampled from a single population using sampling with replacement, the generated sample is comprised of i.i.d. thread assignments.

(2) Proper execution of the experiments: Each random thread assignment is executed on the target platform, in order to measure its performance. It is important that the performance of the corresponding thread assignment is not affected by previous assignments, which can cause differences in the the initial system state, such as the contents of the caches and TLBs. We measured the total execution time for three million network packets per benchmark instance, and observed that the network packets at the beginning of the run had the same execution time as those later in the run, showing that the results are not affected by the initial system state.

(3) Statistical tests that confirm or reject the i.i.d. property of the samples: Different statistical tests can be used to confirm or reject the i.i.d. property of the observations in a sample. In order to validate that selected thread assignments in a given sample are mutually independent we apply the *Wald–Wolfowitz* test [8]. In order to validate whether the observations in the sample are identically distributed, we used a two-sample *Kolmogorov–Smirnov* test [15].

Wald–Wolfowitz test: The *Wald–Wolfowitz test* or *runs test* examines whether the observations in the sample are mutually independent [8][13]. The test comprises two main steps. First, the performance of thread assignments (non-negative real numbers) have to be converted into binary values. We converted the performance of a given thread assignment to ‘0’ if its value was below the median performance (in processed network packets per second) in the sample, and converted it to ‘1’ otherwise. This way, the sequence of non-negative real numbers was converted into a sequence of 0s and 1s, e.g. 000110000. In the second step, the test analyzes the sub-sequences of consecutive identical values (0s or 1s), which are referred to as *runs*. For example, the sequence 000110000 is composed of three runs: 000, 11, and 0000. The Wald–Wolfowitz test validates that the observations in the sample under study are mutually independent if the lengths of the runs follow a Gaussian distribution [8]. The mutual independence hypothesis was tested at the 0.05 significance level. All the samples used in the study passed the test.

Kolmogorov–Smirnov test: In order to validate that the thread assignments in a given sample are identically distributed, we used a two-sample *Kolmogorov–Smirnov test* [13][15]. The test compares the ECDFs of two data sets and, based on the maximum distance between them, it confirms or rejects the hypothesis that the data sets correspond to the same distribution. This test has three steps. First, we generated a uniformly distributed random sample of 5,000 thread assignments and observed the performance of each assignment. The order thread assignments in the sample followed the order in which the assignments

were generated. Second, in each experiment, we observed two randomly-selected segments of m consecutive values from the original sample. Finally, we used a two-sample *Kolmogorov–Smirnov* test to check whether the randomly selected segments of the sample have the same probability distribution. If the thread assignment performances are indeed identically distributed, then all segments of consecutive values in the sample have the same distribution. For each sample used in the study, we performed the test for segments of $m = 100, 500$ and $1,000$ observations. All the samples used in the study passed the test at the 0.05 significance level.

Wald–Wolfowitz and Kolmogorov–Smirnov used in the study were developed on the top of on Matlab[®] R2007a [10] functions *runstest()* and *kstest2()*, respectively.

B.2.2 Continuity assumption

In our study, we tested the continuity assumption, by plotting the performance of the observed thread assignments. We saw that these values covered a wide range, with a visibly smooth behavior, meaning that although the distribution is in fact discrete, the difference from continuous behavior is negligible. If the observed performance has large discontinuities, sophisticated techniques can be used to minimize any systematic errors caused by violation of the continuity assumption [17].

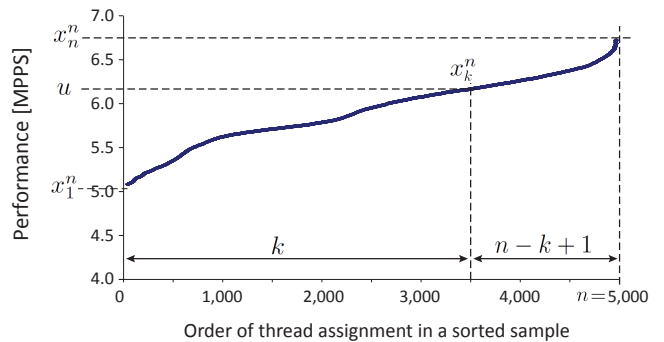
B.3 Application of POT method to the thread assignment problem. Detailed analysis.

We use the POT method to estimate the optimal system performance for a given workload (i.e. the performance of the best thread assignment) based on the measured performance of the random assignments in the sample. In *Pickands-Balkema-de Haan* theorem (Theorem 1), the definition of $G_{\xi, \sigma}(y)$ for parameter $\xi = 0$ can only be used to model problems with an infinite upper bound [16][22]. As in this study we use GPD to estimate application performance when running in a real computer system, the upper bound of the observed value is finite and the estimated values of the parameter ξ always satisfy $\hat{\xi} < 0$. Therefore, for the sake of simplicity of the presented mathematical formulas, in the rest of the paper we do not present $G_{\xi, \sigma}(y)$ formulas for parameter $\xi = 0$.

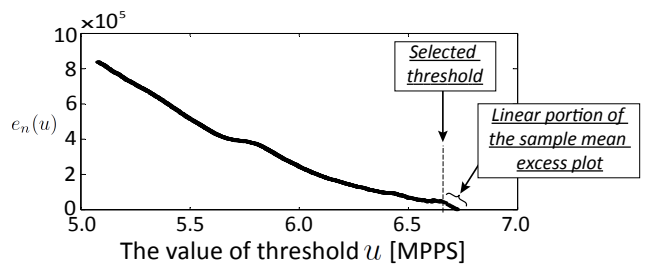
The application of the POT method involves the following four steps:

Step 1: *Generate the sample of random thread assignments.* In this step, we also execute the thread assignments on the hardware platform under study, and measure the performance of each assignment. Once that we collect the performance measurements, we verify the i.i.d. properties of the sample and the continuity assumption (see Appendix B.2).

Step 2: *Select the threshold u .* The selection of the threshold u is an important step in POT analysis. Gilli and K ellezi [16][22] propose using the *sample mean excess plot*, a graphical method for threshold selection. This



(a) Ordered sample of thread assignments



(b) Sample mean excess plot

Fig. 3. Selection of the threshold u

method first sorts all thread assignments in a sample in non-decreasing performance order: $x_1^n \leq x_2^n \leq \dots \leq x_n^n$. Figure 3(a) shows the sorted performance of 5,000 random thread assignments for a workload comprised of 24 threads of the *IPFwd-LI* application. Then, the possible threshold u takes the values from x_1^n to x_n^n ($x_1^n \leq u \leq x_n^n$) and for each value we compute the sample mean excess function $e_n(u)$:

$$e_n(u) = \frac{\sum_{i=k}^n (x_i^n - u)}{n - k + 1}, \text{ where } k = \min\{i \mid x_i^n > u\}.$$

In this formula, the factor $n - k + 1$ is the number of observations that exceed the threshold. Finally, the sample mean excess plot is defined by the points $(u, e_n(u))$ for $x_1^n \leq u \leq x_n^n$. Figure 3(b) shows the example of the sample mean excess plot for 24 threads of *IPFwd-LI* application.

As we are interested in the upper performance bound estimation, the estimated parameter ξ of GPD has to be negative ($\hat{\xi} < 0$). One of the characteristics of the GPD with parameter $\xi < 0$ is that it has linear mean excess function plot. In order to have a good fit of the conditional distribution function F_u to GPD, the threshold should be selected so that the observations that exceed the threshold have a roughly linear sample mean excess plot. As an example, for the data presented in Figure 3(b), the threshold should be selected to be around 6.6 MPPS. Sample mean excess plot is also a very good method to test whether GPD can be used to model a particular set of observations. If the right portion of the mean excess plot for the sample of measured thread assignments performance is not (roughly) linear, that particular problem cannot be modeled using GPD. For all experiments presented in the study, we carefully observed the shape of the right tail of the sample mean excess plot in order to determine whether

a given sample of thread assignments could be modeled using GPD. In all experiments, the form of sample mean excess plots strongly suggested that the observations follow a Generalized Pareto Distribution.

The linear sample mean excess plot is not the only constraints that should be considered when selecting the threshold. If the threshold is too low, the estimated parameters of GPD may be biased to the median values of the cumulative distribution function instead of to the maximum values. In order to avoid this bias, when selecting a threshold we have to ensure that the number of observations that exceed the selected threshold is not higher than 5% of the thread assignments in the whole sample. This is a commonly used limit in studies that use POT analysis [16][22].

Step 3: Fit the GPD function to the observations that exceed the threshold and estimate parameters ξ and σ . Once the threshold u is selected, the observations over the threshold can be fitted to GPD, and the parameters of the distribution can be estimated. Different methods can be used to estimate the parameters of GPD from a sample of observations [9][18][20][28]. In our study, we use the estimation based on the *likelihood* function.

The likelihood is a statistical method that estimates distribution parameters based on a set of observations [5]. For the sake of simplicity, we assume that observations from x_k^n to x_n^n in the sorted sample presented in Figure 3(a) exceed the threshold. We rename the exceedances $y_{i-k+1} = x_i^n - u$ for $k \leq i \leq n$ and use the set of elements $\{y_1, y_2, \dots, y_m\}$ to estimate the parameters of GPD. The number of elements in the set, $m = n - k + 1$, is the number of exceedances over the threshold. The GPD is defined with parameters ξ and σ . The likelihood that a set of observations $\{y_1, y_2, \dots, y_m\}$ is the outcome of a GPD with parameters $\xi = \xi_0$ and $\sigma = \sigma_0$ is equal to the probability that GPD with parameters ξ_0 and σ_0 has the outcome $\{y_1, y_2, \dots, y_m\}$.

We use the likelihood function to compute the probability that different values of GPD parameters have for a given set of observations $\{y_1, y_2, \dots, y_m\}$. As the logarithm is a monotonically increasing function, the logarithm of a positive function achieves the maximum value at the same point as the function itself. This means that instead of finding the maximum of a likelihood function, we can determine the maximum of the logarithm of the likelihood function - the *log-likelihood* function. In statistics, log-likelihood is frequently used instead of the likelihood function because it simplifies the computation. The estimation of parameters ξ and σ of $G_{\xi, \sigma}(y)$ involves the following three steps:

(i) Determine the corresponding probability density function as a partial derivate of $G_{\xi, \sigma}(y)$ with respect to y :

$$g_{\xi, \sigma}(y) = \frac{\partial G_{\xi, \sigma}(y)}{\partial y} = \frac{1}{\sigma} \left(1 + \frac{\xi}{\sigma} y\right)^{-\frac{1}{\xi} - 1}$$

(ii) Find the logarithm of $g_{\xi, \sigma}(y)$:

$$\log(g_{\xi, \sigma}(y)) = -\log \sigma - \left(\frac{1}{\xi} + 1\right) \log\left(1 + \frac{\xi}{\sigma} y\right)$$

(iii) Compute the log-likelihood function $L(\xi, \sigma|y)$ for the GPD as the logarithm of the joint density of the observations $\{y_1, y_2, \dots, y_m\}$:

$$L(\xi, \sigma|y) = \sum_{i=1}^m \log g_{\xi, \sigma}(y_i)$$

$$L(\xi, \sigma|y) = -m \log \sigma - \left(\frac{1}{\xi} + 1\right) \sum_{i=1}^m \log\left(1 + \frac{\xi}{\sigma} y_i\right)$$

We compute estimated values of parameters $\hat{\xi}$ and $\hat{\sigma}$, to maximize the value of the log-likelihood function $L(\xi, \sigma|y)$ for observations $\{y_1, y_2, \dots, y_m\}$:

$$L(\hat{\xi}, \hat{\sigma}|y) = \max_{\xi, \sigma} (L(\xi, \sigma|y))$$

$$L(\hat{\xi}, \hat{\sigma}|y) = \max_{\xi, \sigma} \left(-m \log \sigma - \left(\frac{1}{\xi} + 1\right) \sum_{i=1}^m \log\left(1 + \frac{\xi}{\sigma} y_i\right)\right)$$

In order to determine the parameters $\hat{\xi}$ and $\hat{\sigma}$, we find the minimum of the negative log-likelihood function, $\min_{\xi, \sigma} (-L(\xi, \sigma|y))$, using the procedure *fminsearch()* included in Matlab[®] R2007a [10]. The values $\hat{\xi}$ and $\hat{\sigma}$ are called the point estimate of the parameters ξ and σ .

Another important method that can be used to understand whether a given sample of observations can be modeled with a Generalized Pareto Distribution is a *quantile plot* [7][22]. In a quantile plot, the sample quantiles x_i^n are plotted against the quantiles of a target distribution $F^{-1}(q_i)$ for $i = 1, \dots, n$. If the sample data originates from the family of distributions F , the plot is close to a straight line. For all experiments presented in the study, we plotted the quantiles of the samples of observations against the quantiles of GPD with estimated parameters $\hat{\xi}$ and $\hat{\sigma}$. In all experiments, the form of quantile plots strongly suggested that the observations follow a Generalized Pareto Distribution.

Step 4: Estimate the optimal system performance, i.e. the upper performance bound of all thread assignments.

The upper bound of the observed value can be determined only for $\hat{\xi} < 0$ which is satisfied for all data sets that are presented in this paper. The point estimate of the Upper Performance Bound (UPB) is computed as $\widehat{\text{UPB}} = u - \hat{\sigma}/\hat{\xi}$. In order to indicate the confidence of the estimate, we compute the confidence intervals of the estimated $\widehat{\text{UPB}}$. UPB confidence interval is computed using *likelihood ratio test* [5] which consists of the following four steps:

(i) Define GPD as a function of ξ and UPB:

$$G_{\xi, \text{UPB}}(y) = 1 - \left(1 - \frac{1}{\text{UPB} - u} y\right)^{-1/\xi}$$

(ii) Determine the corresponding probability density function:

$$g_{\xi, \text{UPB}}(y) = \frac{\partial G_{\xi, \text{UPB}}(y)}{\partial y} = -\frac{1}{\xi(\text{UPB} - u)} \left(1 - \frac{1}{\text{UPB} - u} y\right)^{-\frac{1}{\xi} - 1}$$

(iii) Compute the joint log-likelihood function for observations $\{y_1, \dots, y_m\}$:

$$L(\xi, \text{UPB}|y) = \sum_{i=1}^m \log g_{\xi, \text{UPB}}(y_i)$$

$$L(\xi, \text{UPB}|y) = -n \log(-\xi(\text{UPB} - u)) - \left(1 + \frac{1}{\xi}\right) \sum_{i=1}^n \log\left(1 - \frac{1}{\text{UPB} - u} y_i\right)$$

(iv) Find the UPB confidence interval. We determine the confidence interval for UPB using likelihood ratio

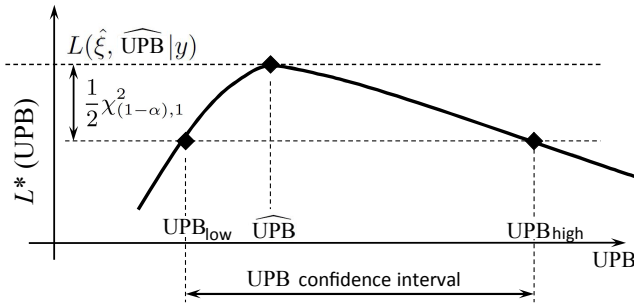


Fig. 4. UPB confidence interval

test [5] and Wilks's theorem [11][32][33]. The maximum log-likelihood function is determined as:

$$L(\hat{\xi}, \widehat{\text{UPB}}|y) = \max_{\xi, \text{UPB}} (L(\xi, \text{UPB}))$$

The function $L(\hat{\xi}, \widehat{\text{UPB}}|y)$ has two parameters that are free to vary (ξ and UPB), hence it has two degrees of freedom, $df_1 = 2$. As UPB is our parameter of interest, the profile log-likelihood function is defined as:

$$L^*(\text{UPB}) = \max_{\xi} L(\xi, \text{UPB})$$

The function $L^*(\text{UPB})$ has one parameter that is free to vary i.e. one degree of freedom, $df_2 = 1$. Wilks's theorem applied to the problem that we are addressing claims that, for large number of exceedances over the threshold, distribution of $2(L(\hat{\xi}, \widehat{\text{UPB}}) - L^*(\text{UPB}))$ converges to a χ^2 distribution with $df_1 - df_2$ degrees of freedom. Therefore, the confidence interval of UPB includes all values of UPB that satisfy the following condition Equation 1:

$$L(\hat{\xi}, \widehat{\text{UPB}}) - L^*(\text{UPB}) < \frac{1}{2} \chi_{(1-\alpha),1}^2 \quad (1)$$

$\chi_{(1-\alpha),1}^2$ is the $(1 - \alpha)$ -level quantile of the χ^2 distribution with one degree of freedom ($df_1 - df_2 = 1$). α is the confidence level for which we compute UPB confidence intervals. We illustrate the computation of the UPB confidence interval in Figure 4. The figure plots $L^*(\text{UPB})$ for different values of UPB. For $\text{UPB} = \widehat{\text{UPB}}$, L^* reaches its maximum. The confidence interval of UPB includes all values of UPB that satisfy the condition $L^*(\text{UPB}) > L(\hat{\xi}, \widehat{\text{UPB}}) - \frac{1}{2} \chi_{(1-\alpha),1}^2$ which corresponds to the Equation 1. We computed the UPB confidence interval using an iterative method based on the `fminsearch()` function included in Matlab[®] R2007a.

The code that generates the sample mean excess plots, infers the parameters of the GPD distribution, and estimates the optimal system performance was developed in Matlab[®] R2007a. Figure 5 gives a high-level control-flow diagram showing the main steps in the POT analysis.

REFERENCES

- [1] "TCPDUMP/LIBPCAP public repository," <http://www.tcpdump.org/>.
- [2] *Netra Data Plane Software Suite 2.0 Update 2 Reference Manual*. Sun Microsystems, Inc, 2008.
- [3] *Netra Data Plane Software Suite 2.0 Update 2 User's Guide*. Sun Microsystems, Inc, 2008.
- [4] A. V. Aho and M. J. Corasick, "Efficient String Matching: An Aid to Bibliographic Search," *Commun. ACM*, vol. 18, 1975.
- [5] A. Azzalini, *Statistical Inference Based on the Likelihood*. Chapman and Hall, 1996.
- [6] A. A. Balkema and L. de Haan, "Residual life time at great age," *Annals of Probability*, vol. 2, 1974.
- [7] J. Beirlant et al., *Statistics of Extremes: Theory and Applications*. John Wiley and Sons, Ltd, 2004.
- [8] J. V. Bradley, *Distribution-Free Statistical Tests*. Prentice-Hall, 1968.
- [9] E. Castillo and A. Hadi, "Fitting the Generalized Pareto Distribution to data," *Journal of the American Statistical Association*, vol. 92, 1997.
- [10] S. J. Chapman, *Essentials of MATLAB Programming*. Cengage Learning, 2009.
- [11] H. Chernoff, "On the distribution of the likelihood ratio," *Annals of Mathematical Statistics*, vol. 25, 1954.
- [12] K. J. Connolly, *Law of Internet Security and Privacy*. Aspen Publishers, 2003.
- [13] L. Cucu-Grosjean et al., "Measurement-based probabilistic timing analysis for multi-path programs," in *Proceedings of the 2012 24th Euromicro Conference on Real-Time Systems*, 2012.
- [14] D. Eckhoff, T. Limmer, and F. Dressler, "Hash Tables for Efficient Flow Monitoring: Vulnerabilities and Countermeasures," in *Proceedings of the 34th Conference on Local Computer Networks (LCN)*, 2009.
- [15] W. Feller, *An introduction to Probability Theory and Its Applications*. John Wiley & Sons, Inc., 1971.
- [16] M. Gilli and E. Kellezi, "An application of extreme value theory for measuring financial risk," *Computational Economics*, vol. 27, 2006.
- [17] D. Griffin and A. Burns, "Realism in statistical analysis of worst case execution times," in *10th Intl. Workshop on Worst-Case Execution Time Analysis*, July 2010.
- [18] S. Grimshaw, "Computing the maximum likelihood estimates for the Generalized Pareto Distribution to data," *Technometrics*, vol. 35, 1993.
- [19] F. Guo and T. Chiueh, "Traffic Analysis: From Stateful Firewall to Network Intrusion Detection System," in *RPE Report*, 2004.
- [20] J. R. M. Hosking and J. R. Wallis, "Parameter and quantile estimation for the generalised pareto distribution," *Technometrics*, vol. 29, 1987.
- [21] Internet Engineering Task Force (IETF), "Request for Comments (RFC), <http://www.rfc-editor.org/>."
- [22] E. Kellezi and M. Gilli, "Extreme value theory for tail-related risk measures," International Center for Financial Asset Management and Engineering, FAME Research Paper Series, 2000.
- [23] M. Norton, "Optimizing Pattern Matching for Intrusion Detection, 2004," <http://docs.idsresearch.org/OptimizingPatternMatchingForIDS.pdf>.
- [24] nProbe network monitor, <http://www.ntop.org>.
- [25] J. I. Pickands, "Statistical inference using extreme value order statistics," *Annals of Statistics*, vol. 3, 1975.
- [26] T. Sherwood, G. Varghese, and B. Calder, "A Pipelined Memory Architecture for High Throughput Network Processors," in *Proceedings of the 30th Annual International Symposium on Computer Architecture (ISCA)*, 2003.
- [27] Snort network intrusion prevention and detection system, <http://www.snort.org/>.
- [28] N. Tajvidi, "Design and implementation of statistical computations for Generalized Pareto Distributions," *Technical Report, Chalmers University of Technology*, 1996.
- [29] S. B. Vardeman, *Statistics for Engineering Problem Solving*. PWS publishing company, 1993.
- [30] J. Verdú, *Analysis and Architectural Support for Parallel Stateful Packet Processing, PhD Thesis*. Universitat Politècnica de Catalunya, 2008.
- [31] Vermont (VERsatile MONitoring Toolkit), <http://vermont.berlios.de/>.
- [32] S. S. Wilks, "The large-sample distribution of the likelihood ratio for testing composite hypotheses," *Annals of Mathematical Statistics*, vol. 9, 1938.
- [33] S. S. Wilks, *Mathematical Statistics*. Princeton University, 1943.
- [34] Wireshark network protocol analyzer, <http://www.wireshark.org/>.
- [35] T. Wolf, N. Weng, and C.-H. Tai, "Design considerations for network processor operating systems," in *Proceedings of ACM/IEEE Symposium on Architectures for Networking and Communication Systems (ANCS)*, 2005.

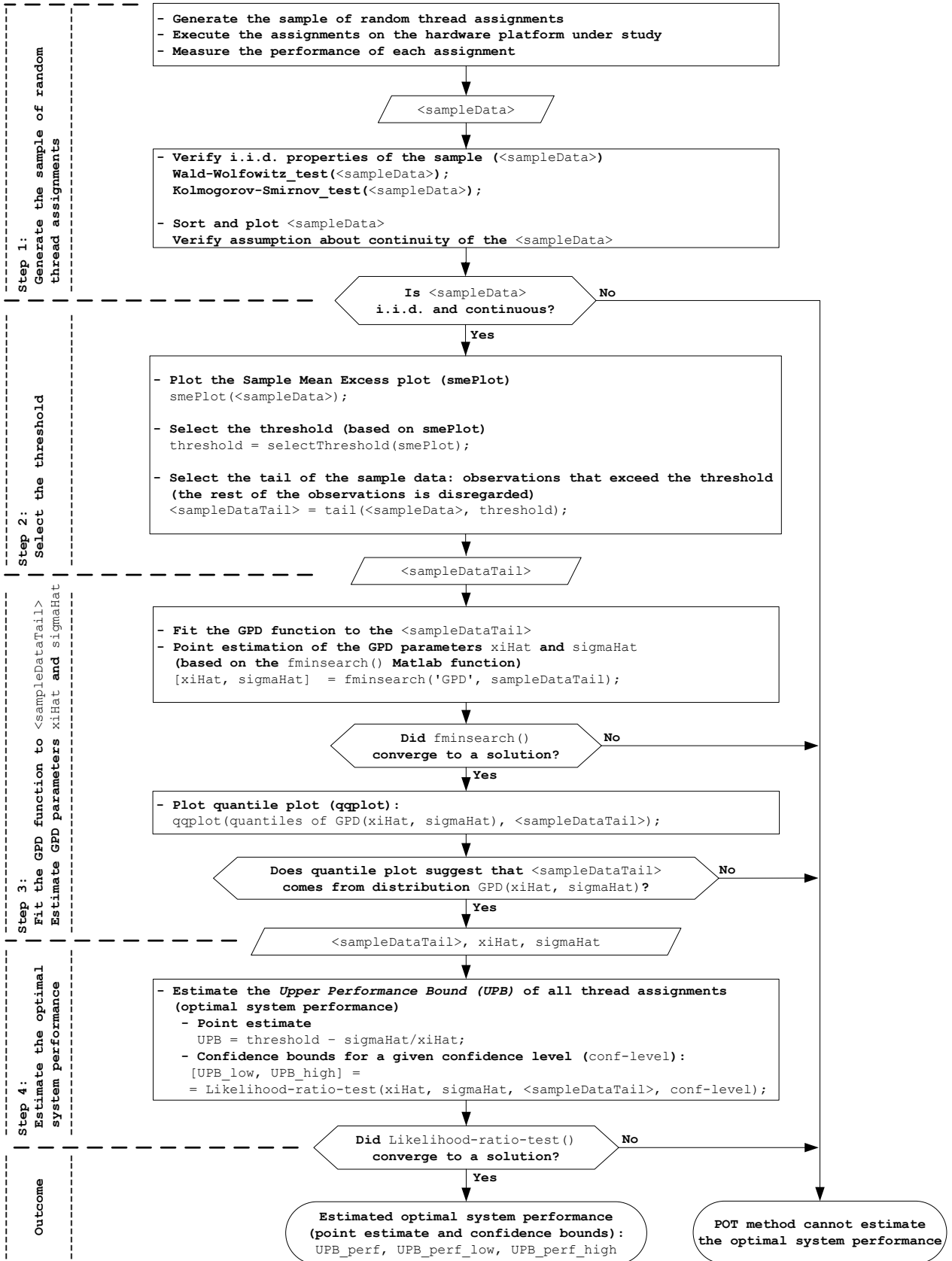


Fig. 5. Overview of the main steps of the POT analysis. `xiHat` and `sigmaHat` in the figure refer to GPD parameters $\hat{\xi}$ and $\hat{\sigma}$, respectively.