Statistical Models for Gnutella Signaling Traffic

Dragos Ilie [*], Adrian Popescu

School of Engineering, Department of Telecommunication Systems, Blekinge Institute of Technology (BTH), 371 39 Karlskrona, Sweden

Abstract

The paper is focused on signaling traffic between Gnutella peers that implement the latest Gnutella protocol specifications (v0.6). In particular, we provide analytically tractable statistical models at session level, message level and IP datagram level for traffic crossing a Gnutella ultrapeer at Blekinge Institute of Technology (BTH) in Karlskrona, Sweden. To the best of our knowledge this is the first work that provides Gnutella v0.6 statistical models at this level of detail. These models can be implemented straightforward in network simulators such as `ns2` and `OmNet++`.

The results show that incoming requests to open a session follow a Poisson distribution. Incoming Gnutella messages across all established sessions can be described by a compound Poisson distribution. Mixture distribution models for message transfer rates include a heavy-tailed component.

Key words: Gnutella, signaling traffic, statistical models, traffic measurements

1. Introduction

During the past five years peer-to-peer (P2P) file-sharing has gradually replaced the Web as the Internet “killer” application. This phenomenon is largely due to the wide availability of communication bandwidth, i.e., link capacity at the edges of the network where end-user network equipment is located.

P2P literature tends to define P2P networking as a fuzzy relationship among interconnected nodes that alternate between the roles of client and server. The exact characteristics defining a P2P network seem quite elusive at first, since authors tend to focus on the characteristics relevant to their work. Some attempts to settle this situation can be found in [1–3]. In general these definitions establish that a P2P network architecture implies a distributed network with decentralized control and dynamic membership, in which participants share resources, e.g., storage space, processing power bandwidth, in order to achieve some common goal.

In the light of the growing volume of P2P traffic on the Internet it becomes increasingly important to gain a better understanding of the characteristics associated with this type of traffic. Such understanding would be invaluable in the design of analytical and simulation models to control P2P traffic or in the ability to offer value-added services on top of P2P networks.

Our work is focused on signaling traffic in the Gnutella network. In particular, we provide analytically tractable statistical models at session level, message level and IP datagram level for traffic crossing a Gnutella ultrapeer.

The rest of the paper is organized as follows. Section 2 gives an overview of related work. The Gnutella protocol is described in Section 3. The P2P measurement infrastructure developed at BTH is presented in Section 4. In Section 5 we describe our modeling methodology, which was used to construct

[*] Corresponding author.
Email addresses: `dragos.ilie@bth.se` (Dragos Ilie), `adrian.popescu@bth.se` (Adrian Popescu).

statistical models. The actual statistical models are presented in Section 6. Conclusions and an outline for future work are the subject of Section 7.

2. Related work

Perhaps the oldest and most cited paper is [4], which looks into the social aspects of the Gnutella network. The authors instrumented a Gnutella client to log protocol events. The main contribution of the paper was to show evidence for the occurrence of free-riding in the Gnutella network.

In [5,6] the authors created *crawlers* for Napster and Gnutella networks. A crawler is a special purpose software agent, which discovers and records the network topology through an automated, iterative process. The authors used information from the crawlers to measure properties of individual peers, e.g., bandwidth and latency. The data from their measurements indicated that both Gnutella and Napster exhibit highly heterogeneous properties, e.g., in terms of connectivity, speed, shared data. Another important finding, which corroborates the conclusions in [4], is that users are typically unwilling to cooperate with each other, few of them acting as servers and the remaining majority acting as clients.

A different approach was taken in [7]. The authors performed non-intrusive flow measurements at a large ISP. Their goal was to analyze FastTrack (a protocol used by Kazaa and Grokster), Gnutella and DirectConnect networks. Flows belonging to any of these networks were identified by well-known port numbers. The major findings in the paper are that all three networks showed increases in the traffic volume across consecutive months, skewed distributions for traffic volume, connectivity and average bandwidth, few hosts with a long uptime, and uniformity in terms of number of P2P nodes from individual network address prefixes.

Measurements from a 1 Gbit/s link in the France Telecom IP backbone network revealed that almost 80 % of traffic on the link in question was produced by P2P applications [8]. Further, the authors showed that flows can be partitioned into “mice” — short flows, mostly due to signaling, and “elephants” — long flows due to data transfers.

The P2P traffic identification methods in [9,8] assume that applications use well-known ports. This assumption rarely holds nowadays, when P2P applications use dynamic ports in order to camouflage themselves. Karagiannis et al. [10–12] use increasingly better heuristics to detect P2P traffic. Their measurement results showed that, if anything, P2P traffic was not declining in volume. Further, they showed that P2P traffic is using predominantly dynamic ports.

With the exception of [10–12] the related work considered here focuses on the first version of the Gnutella protocol, i.e., Gnutella v0.4 [13]. In the last 2 years, Gnutella developers have implemented new functionality in Gnutellla v0.6 [14], which have dramatically changed the traffic characteristics. Our results are based on traffic generated by peers that implement the latest protocol specifications. Furthermore, we detect and decode Gnutella traffic irrespective of the port numbers used. To the best of our knowledge this is the first work that provides Gnutella statistical models at this level of detail. These models should be straightforward to implement in network simulators such as `ns2` and `OmNet++`.

3. The Gnutella protocol

Gnutella is a heavily decentralized P2P system. Nodes[1] can share any type of resources, although the currently available specification covers only computer files [14].

The overlay network uses an unstructured topology with a two-level hierarchy: ultrapeers (UPs) and leaf nodes (LNs). UPs are faster nodes in the sense that they are connected to high-capacity links and have a large amount of processing power available. LNs maintain a single connection to their UP. A UP maintains 10-100 connections, one for each LN and 1-10 connections to other UPs [15]. The UPs perform signaling on behalf of the LNs, thus shielding them from large volumes of signaling traffic. A UP does not necessarily have LNs, in which case it works standalone.

The activities of Gnutella peers can be divided into two main categories: signaling and user data transfer (further referred to as data transfer). Signaling activities are concerned with peer discovery, overlay topology maintenance, content search and other management functions. Data transfer occurs when a peer has localized during content search one or more files of interest.

According to the Gnutella Development Forum (GDF) mailing list, the Gnutella community has re-

[1] A Gnutella node is also called a servent, which is a combination of the words server and client.

cently adopted what is called support for high outdegree [16]. This implies that UPs maintain at least 32 connections to other UPs and 100–300 connections to different lead nodes. LNs are recommended to maintain approximately 4 connections to UPs. The numbers may slightly differ between different Gnutella vendors. The claim is that high outdegree support allows a peer to connect to the majority of Gnutella peers in 4 hops or less [17].

3.1. *Bootstrap*

A Gnutella node that attempts to join the overlay for the first time must bootstrap itself into the overlay. This implies finding and connecting to one or several peers that are already part of the overlay. A list of active servents can be obtained from a Gnutella Web Cache (GWC) [18] server. A GWC server is essentially an HTTP server maintaining a list of active peers with associated listening sockets. A listening socket is the IP address and port number that can be used to connect to the corresponding servent. UPs update the list continuously, ensuring that new peers can always join the overlay.

Once the node joins the overlay, additional peers can be found through the exchange of PING and PONG messages. The servent saves peer addresses in a local host cache in order to avoid connecting to a GWC server upon restart. The local host cache is used also if the servent supports the UDP Host Cache (UHC) protocol. The protocol works as a distributed bootstrap system, transforming UHC-enabled servents into GWC-like servers [19] and offloading the actual GWC servers.

3.2. *Peer connections*

Peer signaling occurs over TCP connections. Once a TCP connection has been setup, the peers at each end of the TCP connection perform a three-way Gnutella handshake. The Gnutella handshake allows the negotiation of a common set of capabilities to be used during the session. The type of capabilities negotiated are UP - LN relationship, support for high outdegree, traffic compression, etc.

If the handshake fails the TCP connections is teared down. Otherwise, the client and the server start exchanging binary Gnutella messages over the existing TCP connection. The connection lasts until one of the peers decides to terminate the session. At that point the node ending the connection can optionally send a BYE message to notify its peer of its departure. The TCP connection will then be closed.

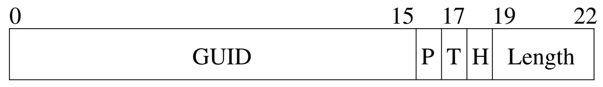


Fig. 1. The Gnutella header

If the capability set used by the peers includes stream compression [20], then all data on the TCP connection is compressed, with the exception of the initial Gnutella handshake. The type of compression algorithm can be selected during the handshake, but the currently supported algorithm is *deflate*, which is implemented in `zlib` [21].

3.3. *Gnutella message headers*

Each Gnutella message starts with a generic header that contains the fields shown in Figure 1 (the numbers in the figure denote bytes):

– Message ID using a Globally Unique Identifier (GUID) to uniquely identify messages on the Gnutella network [22].
– Payload type code, denoted by P in Figure 1, that identifies the type of Gnutella message. The currently supported messages are: PING, PONG, BYE, QRP, VEND, STDVEND, PUSH, QUERY, QUERY_HIT and HSEP.
– Time-To-Live (TTL), to limit the signaling radius and its adverse impact on the network. Messages with TTL > 15 are dropped[2]. This field is denoted by T in Figure 1.
– Hop count to inform receiving peers how far the message has traveled, denoted by H in Figure 1.
– Payload length in bytes to describe the length of the message, not including the header. The payload length indicates where in the byte stream the next Gnutella generic message header can be found.

The generic Gnutella header is followed by the actual message, which may have its own headers. Also, the message may contain vendor extensions. Vendor extensions are used when a specific type of servent wants to implement experimental functionality not covered by the standard specifications.

[2] Nodes that support high outdegree drop messages with TTL > 4.

3.4. *Peer discovery*

Each successfully connected servent sends PING messages periodically to its neighbors. The receiver of a PING message decrements the TTL in the Gnutella header. If the TTL is greater than zero the node increments the hop counter in the message header and then forwards the message to all its directly connected peers, with the exception of the one from where the message came. PING messages do not carry any user data (not even the sender's listening socket). This means that the payload length field in the Gnutella header is set to zero.

PONG messages are sent only in response to PING messages. More than one PONG message can be sent in response to one PING. The PONG message travels in the reverse direction on the path used by the corresponding PING message. Each PONG message contains detailed information about *one* active Gnutella peer. It also contains the same GUID as the PING message that triggered it.

UPs use the same scheme, however they do *not* forward PINGs and PONGs to and from the LNs attached to them.

Gnutella peers are required to implement some form of flow control in an effort to prevent PING-PONG traffic generated by malfunctioning servents from swamping the network. A simple flow control mechanism is specified in [23].

The BYE message is an *optional* message used when a peer wants to inform its neighbors that it will close the signaling connection. The message is sent only to hosts that have indicated during handshake that they support BYE messages.

3.5. *Resource discovery*

Gnutella peers use QUERY messages to search for files. The message payload consists of a text string, information about the minimum speed, i.e., upload rate of servents that should respond to this message, and in some cases additional extensions that are not within the scope of this work. The most important part of the query is the text string, which is used to match files on the nodes receiving the message.

Gnutella v0.6 sends QUERY messages through a form of selective forwarding called *dynamic query* [16]. A dynamic query first probes how popular the targeted content is. This is done by using a low TTL value in the QUERY message that is sent to a small set of directly connected peers. A large number of replies indicate popular content, whereas a low number of replies imply rare content. For rare content, the QUERY TTL value and the number of directly connected peers receiving the message are gradually increased. This procedure is repeated until enough results are received or until an upper bound on the number of QUERY receivers is reached. This form of resource discovery requires all LNs to rely on UPs for their queries, i.e., LNs do not perform dynamic queries.

If a peer that has received the QUERY message is able to serve the resource, it responds with a QUERY_HIT message. The GUID for the QUERY_HIT message must be the same as the one in the QUERY message that triggered the response. The QUERY_HIT message lists all file names that match the text string from the QUERY message, their size in bytes and some other information [14]. In addition, the QUERY_HIT messages contain the listening socket to be used by the message receiver when it wants to download the matched files. The Gnutella specification discourages the use of messages with size greater than 4 KB. Consequently, several QUERY_HIT messages may be issued by the same servent in response to a single QUERY message.

3.6. *Query Routing Protocol (QRP)*

QRP was introduced in order to mitigate the adverse effects of flooding used by the Gnutella file queries [24]. QRP is based on a modified version of Bloom filters [25]. The idea is to break a query into individual keywords and have a hash function applied to each keyword. Given a keyword, the hash function returns an index to an element in a finite discrete vector. Each entry in the vector is the minimum distance, i.e., number of hops to a peer holding a resource that matches the keyword in the query. Queries are forwarded *only* to leaf nodes that have resources that match *all* the keywords. This substantially limits the bandwidth used by queries. Peers run the hash algorithm over the resources they share and exchange the routing tables, i.e., hop vectors at regular intervals.

LNs send route table updates only to UPs, i.e., not to any other LNs. UPs propagate these tables only to directly connected UPs [26].

3.7. *Content distribution*

Data exchange takes place over a direct HTTP connection initiated by the receiver of a QUERY_HIT message. Both HTTP 1.0 and HTTP 1.1 are supported but use of HTTP 1.1 is strongly recommended [14].

PUSH messages can be used when the file owner is protected by a firewall that does not allow incoming TCP connections or if the host is behind a Network Address Translator (NAT) device. In that specific case, the file requester opens a listening socket and puts information about the socket in a PUSH message. The PUSH message is sent across the signaling path to the file owner who, upon message receival, is able to open a TCP connection to the file requester. At that point the HTTP transfer can be performed.

The PUSH message does not help if both peers are protected by firewalls or NAT devices that block incoming TCP connections.

3.8. *Horizon Size Estimation Protocol (HSEP)*

The Horizon Size Estimation Protocol (HSEP) [27] is used to obtain estimates on the number of reachable resources i.e., nodes, shared files and shared kilobytes of data. Hosts that support HSEP announce this as part of the capability set exchange during the Gnutella handshake. If the hosts on each side of a connection support HSEP, they start exchanging HSEP message approximately every 30 seconds. The HSEP message consists of `n_max` triples. Each triple describes the number of nodes, files and kilobytes of data estimated at the corresponding number of hops from the node sending the message. The `n_max` values is the maximum number of hops supported by the protocol. The recommended value is 10 hops [27].

The horizon size estimation can be used to quantify the quality of a connection, e.g., the higher the number of reachable resources, the higher the quality of the connection.

4. Measurement infrastructure

Network traffic measurements can be generally divided into active and passive measurements. The main difference between the two is that in active measurements specific patterns of traffic are injected into the network and analyzed when they exit the network. Changes on the injected traffic pattern are used to draw inferences about various properties of the network. In the case of passive measurements, traffic flows seen at specific nodes are observed or recorded, without sending any additional traffic in the network. In general, when the focus is on the characteristics of traffic crossing a single network element, the passive method is more appropriate [28]. This was our choice as well, since we were interested only in the traffic crossing the BTH ultrapeer.

There are two main approaches to perform passive application layer measurements: application logging or link-layer packet capture with application flow reassembly [29]. A large advantage of link-layer packet capture is that it allows for traffic analysis at any layer in the TCP/IP stack. This enables a more accurate view of how the application affects the network and vice-versa. Another advantage is that packet timestamping is performed in the kernel and not in user space as is the case of application logging [30]. This means that packet timestamps are less affected by, for example, process preemption due to scheduling in the OS and queuing and scheduling in the TCP/IP stack. Link-layer packet capture with application flow reassembly is the approach used here.

A measurement infrastructure dedicated to P2P measurement has been developed at BTH [31]. It consists of peer nodes and protocol decoding software. `Tcpdump` [32] and `tcptrace` [33] are used for traffic recording and protocol decoding. Although the infrastructure is currently geared towards P2P protocols, it can be easily extended to measure other protocols running over TCP. The measurement infrastructure has been successfully used for Gnutella [34,29] and BitTorrent measurements [35,36].

The BTH measurement nodes run the Gentoo Linux 1.4 operating system, with kernel version 2.6.5. Each node is equipped with an Intel Celeron 2.4 GHz processor, 1 GB RAM, 120 GB hard drive, and 10/100 Mbit/s Ethernet network interface. The network interface is connected to a 100 Mbit/s switch in the lab at the Department of Telecommunication Systems, which is further connected through a router to the GigaSUNET backbone as shown in Figure 2.

Figure 3 shows the measurement process flow, which consists of six stages. The data enters each stage sequentially, from top to bottom.

Each measurement node has `tcpdump` 3.8.3 installed on it. When the node is running measurements, `tcpdump` is started before the Gnutella servent in order to avoid missing any connections. `Tcpdump` collects Ethernet frames from the switch

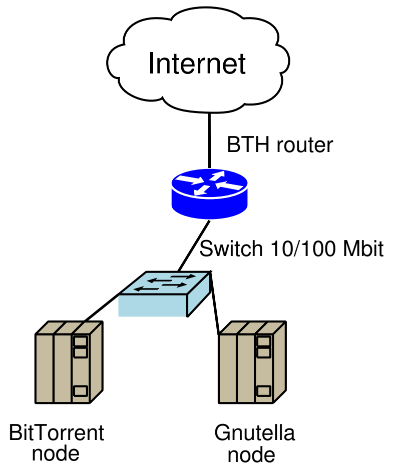


Fig. 2. Measurement network infrastructures

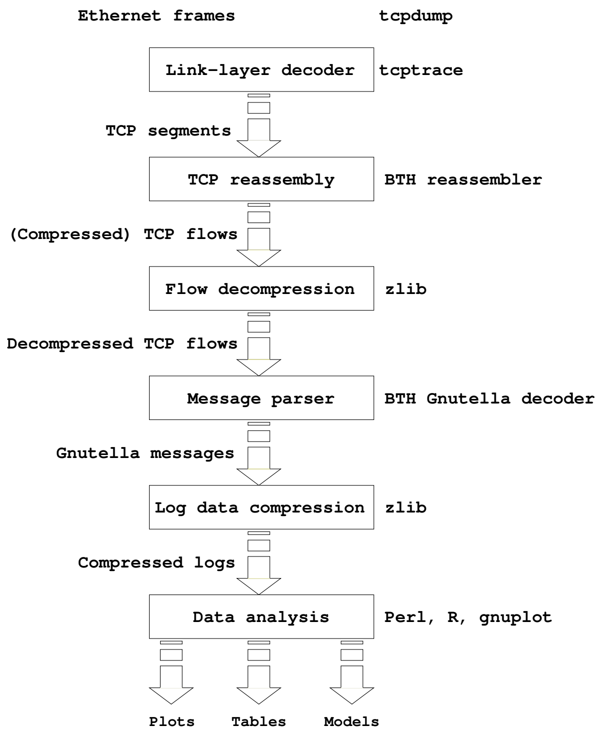


Fig. 3. Measurement process

port where the ultrapeer node is connected. The collected data is saved in PCAP format [32]. Since P2P applications tend to use dynamic ports, all traffic reaching the switch port must be collected. In addition, Ethernet frames cannot be truncated since we need the entire payload in order to decode the signaling traffic.

During the first stage of the measurement process, we use `tcptrace` to extract TCP segments from the Ethernet frames.

The TCP segments are then sent to the next stage, whose task is to reassemble them to a flow of ordered bytes. The TCP reassembly module developed at BTH [29] builds on the TCP engine available in `tcptrace` and is similar to the one used by the FreeBSD TCP/IP stack as described in [37]. The reassembly engine is capable of handling out-of-order segments as well as forward and backward overlapping between segments.

When a new Gnutella connection is found, the application reassembly module first waits for the handshake phase to begin. If the handshake fails, the connection is marked invalid and it is eventually discarded by the memory manager.

If the handshake is successful, the application reassembly module scans the capability lists sent by the nodes involved in the TCP connection. If the nodes have agreed to compress the data, the connection is marked as compressed. Further segments received from the TCP reassembly module for this connection are first sent to the decompresser, before being appended to previous data that has not been consumed yet.

The decompresser uses `zlib`'s *inflate()* function to decompress the data available in the new segment [21]. Upon successful decompression the decompressed data is appended to the data buffer.

Immediately after the handshake phase, the application reassembly module attempts to find the Gnutella message header of the first message. Using the payload length field, it is able to discover the beginning of the following message. This is the only way to discover message boundaries in the Gnutella protocol and thus track application state changes [29]. Based on the message type field in the message header, the corresponding decoding function is called, which outputs a message record to the log file. The message records follow a specific format required by the post-processing stage [29].

Since the logs can grow quite large, they can be processed through an optional stage of data compression. The compression is achieved by using the on-the-fly `deflate` compression offered by `zlib`. Additional data reduction can be achieved if the user is willing to sacrifice some detail by aggregating data over time.

The data analysis module interprets the (optionally compressed) log data and it is able to demultiplex it based on different types of constraints: message type, IP address, port number, etc. The data output format of this stage is suitable for input to numerical computation software such as MATLAB and standard UNIX text processing software such as `sed`, `awk` and `perl`.

5. Methodology for statistical modeling

The measurement infrastructure described in the previous section was used to collect Gnutella traffic crossing the BTH ultrapeer. By decoding the recorded traffic data, flows were recreated at several layers in the TCP/IP stack. The flows consist of discrete protocol data units: IP datagrams at the network layer, TCP segments at the transport layer, and finally, Gnutella messages at the application layer. The Gnutella messages are logically grouped in peer sessions. The time when the protocol data units reached the link layer was recorded together with their size. For peer sessions, the session duration was recorded as well. Due to the complexity of the protocol we have used a statistical approach [36,29] to describe the quantities of interest, which is similar to the methodology introduced by Paxson in [38].

5.1. *Model construction*

Each quantity of interest is modeled by a random variable X that changes its value whenever a new protocol data unit (or session) is considered. The actual values taken by X are denoted by the small letter x. The random variable X is assumed to have a theoretical cumulative distribution function (cdf) $F_X(x;\theta)$, where θ is a set of one or more parameters that control the distribution function, e.g., $\theta = \{\mu, \sigma\}$ in the case of the normal distribution.

For each quantity of interest, the set of values extracted from the recorded traffic is considered to be a *random sample* from the population of the random variable X. The elements of the random sample are denoted by $X_1, X_2, \ldots, X_n$ and the actual recorded values (*data sample*) by $x_1, x_2, \ldots, x_n$. The index n is the number of available values from the measurement.

The first step in our modeling methodology is to identify a distribution family $F_X(x;\theta)$. This is done through an Exploratory Data Analysis (EDA) approach that combines graphs of the data, e.g., histograms and distribution plots, and summary statistics e.g., mean, median and standard deviation [39,40].

We then proceed to obtain point estimates for the parameters θ of the distribution family. This is done by using the Maximum Likelihood (ML) method [41,42]. At this stage a member of the distribution family has been selected and the model is complete.

5.2. *Fitness assessment*

After a probability distribution has been fitted to the data as described in the previous section, the next step in our methodology is to estimate the quality of the fit. A variety of goodness-of-fit tests e.g., the χ^2, Kolmogorov-Smirnov and Anderson-Darling tests, can be used for this purpose. Their common denominator is the test of the null hypothesis:

H_0 : The random sample $X_1 \ldots X_n$ is drawn from the distribution $\hat{F}(x, \hat{\theta})$

Unfortunately, these tests tend to erroneously reject the null hypothesis when the number of samples is large (Type 1 error) [43–45]. Therefore, a different approach is used where the hypothesis test is avoided.

A goodness-of-fit measure called error-percentage measure ($E_\%$) was introduced in [35] and used later in [36,46]. The method is based on the Probability Integral Transform (PIT) [44,45]. The PIT method works as follows. Given a *continuous* random variable R with cdf $F_X(x)$, then

$$F_X(R) = P[X \leq R] = Y \tag{1}$$

with $P[Y \leq y] \overset{d}{=} U[0,1]$, where $U[0,1]$ denotes the uniform distribution between zero and one[3]. The algorithm to compute $E_\%$ is shown below.

Algorithm 1 Calculate Error Percentage

```
Fit a distribution $\hat{F}_X(x;\hat{\theta})$ to the random sample $X_1, X_2, \ldots, X_n$
Obtain the order statistics $X_{(1)}, X_{(2)}, \ldots, X_{(n)}$
Transform the random sample with PIT: $\hat{U}_i = \hat{F}_X(X_{(i)};\hat{\theta}), \quad i = 1, \ldots, n$
$E_\% = 100 \frac{\sum_{i=1}^{n} \left| U_i - \hat{U}_i \right|}{n\, E_{max}}$, where $U_i = \frac{i}{n} \overset{d}{=} U[0,1]$
return $E_\%$
```

If the distribution $\hat{F}$ is a perfect fit, then the PIT transforms the random sample to a uniform

[3] The symbol $\overset{d}{=}$ denotes equality in distribution.

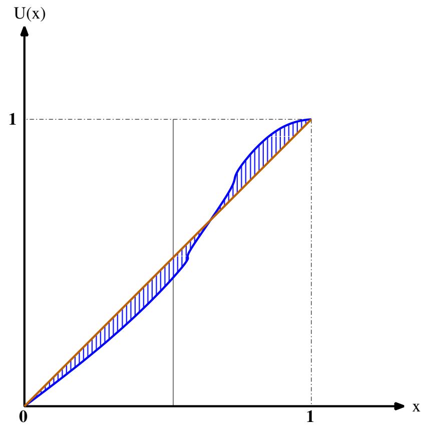


Fig. 4. Error in random sample transformed with the PIT

distribution, $U[0,1]$. However, since perfect fittings rarely occur in reality, the transformed distribution, $\hat{U}$, only approximates the uniform distribution. The discrepancies between U and $\hat{U}$ are computed and their average is normalized to the highest possible error E_{max} for the distribution $U[0,1]$ [36] where,

$$\begin{aligned} E_{max} &= \int_0^1 \sup\{U(x), 1-U(x)\}\ \mathrm{d}x \\ &= \int_0^{1/2} [1-U(x)]\ \mathrm{d}x + \int_{1/2}^1 U(x)\ \mathrm{d}x = \frac{3}{4} \end{aligned} \tag{2}$$

Figure 4 and Figure 5 provide additional visual clues on how the $E_{\%}$-method works. Figure 4 shows a hypothetical edf for a random sample transformed with the PIT, i.e., the diagonal straight line. The blue shaded area represents the error (discrepancy) when the edf is compared to the ideal $U[0,1]$ distribution. The size of that area is the $E_{\%}$ score. The size of the shaded area in Figure 5 is the maximum error E_{max} that can occur when the PIT is applied to a random sample. This is the value that is used to normalize the $E_{\%}$ score.

$E_{\%}$ is expressed in the form of a percentage. The criteria used here to accept a candidate distribution is that $E_{\%} < 6$. We call this value the *accepted error percentage*. The accepted error percentage was decided experimentally by observing that most distributions that provide a visually acceptable fit in both body and tail have $E_{\%} < 6$. Table 1 presents a mapping between various $E_{\%}$ ranges and qualitative statements about the fit.

The main disadvantage of the $E_{\%}$-method is that it cannot be used with discrete distributions. The reason is that when the PIT method is applied to a

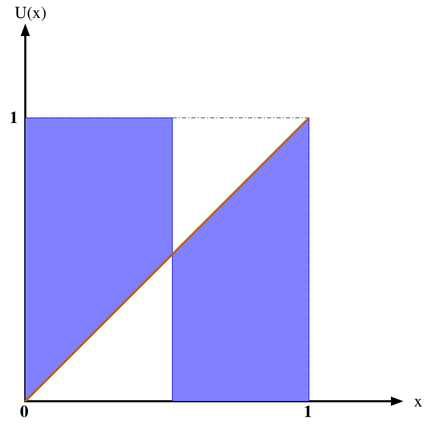


Fig. 5. Maximum error for a PIT transformation

Table 1
Quality-of-fit mapping

Range	$0 \leq E_{\%} < 2$	$2 \leq E_{\%} < 4$	$4 \leq E_{\%} < 6$	$E_{\%} \geq 6$
Quality	Excellent	Good	Acceptable	Unacceptable

discontinuous distribution, the transformed variable is not uniformly distributed [47].

5.3. *Mixture distributions*

Sometimes a single cdf cannot accurately describe the distribution of the random variables of interest, i.e., the $E_{\%}$-method yields an unacceptable score. A more accurate model may be constructed by using a mixture of two distributions or more. In the case of a mixture of two distributions one component of the mixture accounts for the main body of the empirical distribution and a different one describes the behavior in the tail. In the case of more than two components, each cdf accounts for specific modality found in the data. The crux of the problem becomes to find a way to combine the two distributions in a meaningful way. The method used here is based on finite mixture distributions as described in [48].

A mixture distribution $F_X(x)$ with n components has the following distribution function:

$$F_X(x) = \sum_{i=1}^{n} \pi_i\, G_i(x) \qquad \pi_1+\pi_2+\cdots+\pi_n = 1 \tag{3}$$

where $G_i(x)$ is the ith distribution in the mixture and each π_i is a constant called *mixing weight*. The mixing weight π_i is selected such that $0 < \pi_i < 1$ and it decides how much each component is allowed to influence the distribution $F_X(x)$.

The first step in building a mixture distribution with two components is to identify a distribution family $G_1(\cdot)$ that matches the body of the data, preferably the tail as well. This is done using the EDA approach, as explained in Section 5.1. The parameters of the distribution are then estimated, yielding a specific distribution function $\hat{G}_1(x;\hat{\theta}_1)$. $\hat{G}_1(x;\hat{\theta}_1)$ is then visually compared to the true distribution to asses the fit in the tail. If the fit appears good, then the goodness-of-fit measure $E_{\%}$ is computed as explained in Section 5.2. Otherwise, it is necessary to find the *cutoff point* x_c and the corresponding *cutoff quantile* q_c, where $\hat{G}_1$ diverges from the true distribution. The probability mass between q_c and 1 is used to identify the distribution family $\hat{G}_2(\cdot)$ that matches the tail. The parameters of the new distribution must be estimated as well, yielding $\hat{\theta}_2$. Then, a finite mixture distribution is assembled:

$$\hat{F}(x;\hat{\theta}) = \pi\hat{G}_1(x;\hat{\theta}_1) + (1-\pi)\hat{G}_2(x;\hat{\theta}_2) \qquad (4)$$

where $\pi = q_c$. Since the single distributions, $\hat{G}_1$ and $\hat{G}_2$, are now combined in a finite mixture, the parameters $\hat{\theta}_1$ and $\hat{\theta}_2$ must be recomputed [4]. Their original values may be used as a starting point. An optimal value for π must be computed as well. The parameter set $\hat{\theta}$ in $F(x;\hat{\theta})$ is the set containing the parameters for both distributions and π, i.e., $\hat{\theta} = \{\hat{\theta}_1, \hat{\theta}_2, \pi\}$. Numerical methods for computing the set of optimal parameters θ are presented in Section 5.5.

It is often the case that a mixture distribution (in particular one with only two components) still cannot describe the data accurately enough. This may be further improved by increasing the number of components in the mixture distribution at the expense of an increase in the number of parameters. However, a different approach was used here.

Typically, the major discrepancies between the estimated distribution and the true one appear either in the body or in the tail. If, for example, the discrepancies appear in the tail, one can attempt to improve the model accuracy by adjusting the values of the distribution parameters. However, our experience was that this is likely to decrease the accuracy of the model in the body. Similarly, attempts to increase the accuracy in the body may lead to (higher) discrepancies in the tail. Thus, a trade-off is required, accuracy in the body versus accuracy in the tail [29]. Accordingly, a decision must be taken on which part of the distribution (body or tail) is more important to model accurately.

[4] Recall that G_1 was estimated using the entire probability mass.

For example, in the case of transfer rates the tail of the distribution models high rates of traffic (bursts) that occur rarely. On the other hand, the body of the distribution models the "average" size of transfer rates. For message/packet size the body accounts for small packets and the tail for large ones. In the case of interarrival and interdeparture times the body accounts for "dense" traffic and the tail for "sparse" traffic. In our models, when a trade-off was required, we have favored to model accurately bursty, dense traffic with large packets/messages.

5.4. *Methodology review*

The goal of this section is to present a formal process for the modeling methodology discussed in the previous sections. The process assumes that the variable of interest has been measured (sampled) n times. The values $x_1, x_2, \ldots, x_n$ resulting from the n measurements are assumed to be the result of a random sample $X_1, X_2, \ldots, X_n$. The complete process for building the statistical models is presented in Algorithm 2.

Step 15–16 in Algorithm 2 may be confusing since no criteria has been provided on how to decide to either select a different quantile or to start over. In our work, the quantile was changed in increments of 0.05 to either sides of the original value. If that did not result in any improvement, the decision was to start over.

5.5. *Numerical software and methods*

The process presented here was implemented by using the statistical software package `R` [49]. `R` is an interpreted computer language with syntax similar to `S` and `S-PLUS`. The software package contains in addition to the language, a run-time environment with graphics, a debugger and a large library of functions.

As mentioned in Section 5.3, the $E_{\%}$-method relies on numerical optimization for finding a minimum. ML-estimation requires also numerical optimization in many cases where no closed form ML estimators exist. In our work we have used the `R`-functions `optimize()` and `optim()` for numerical optimization.

The function `optimize()` performs optimization in one dimension. The underlaying algorithm is a combination of golden section search and successive parabolic interpolation [49,50].

Algorithm 2 Methodology for statistical modeling

```
1: Use EDA visual tools, i.e., histogram, edf and cedf plots, to explore the data. The summary statistics provide hints about range, skewness and spread
2: Select a distribution family $G_1$, which appears to provide a good fit
3: Estimate the unknown parameters $\theta_1$ using ML-estimation to obtain a candidate distribution $\hat{G}_{1X}(x, \hat{\theta}_1)$
4: Compare the plots of $\hat{g}_{1X}(x; \hat{\theta}_1)$, $\hat{G}_{1X}(x; \hat{\theta}_1)$, and $\overline{\hat{G}_{1X}}(x; \hat{\theta}_1)$ to the histogram, edf, and cedf plots obtained in Step 1
5: if high visual discrepancy then
6:     Go back to Step 1
7: end if
8: Compute $E_{\%}$ for $\hat{G}_{1X}(x; \hat{\theta}_1)$ using $x_1, x_2, \ldots, x_n$.
9: if $E_{\%} < 6$ then
10:    return $E_{\%}$ and $\hat{G}_{1X}(x, \hat{\theta}_1)$
11: end if
12: Identify the cutoff quantile $q_c$
13: Fit a distribution $G_2(\cdot)$ to the probability mass $(1 - q_c)$ as outlined in Step 1–8
14: if $E_{\%} > 6$ then
15:    Either go back to Step 12 and select a different quantile $q_c$ or,
16:    Go back to Step 1. This is equivalent to starting over. Try using a different distribution family $G_1(\cdot)$
17: end if
18: Assemble the mixture distribution $F(\cdot) = \pi\, G_1(\cdot) + (1 - \pi)\, G_2(\cdot)$
19: Estimate the unknown parameters $\theta = \{\theta_1, \theta_2, \pi\}$ using $E_{\%}$ method. Use the estimated values from previous steps as initial values
20: if $E_{\%} < 6$ then
21:    return $E_{\%}$ and $F_X(x; \theta)$
22: else
23:    Go back to step 1
24: end if
```

General purpose multi-dimensional optimization is performed by the `optim()` function. The function has support for several optimization algorithms. In our methodology the default algorithm, Nelder-Mead [50–52], is used primarily. The algorithm does not require any derivatives, being quite stable although not extremely efficient in terms of number of iterations.

When the Nelder-Mead algorithm fails to converge to a solution, the L-BFGS-B [53] algorithm is used instead. This algorithm requires a lower and an upper bound for each variable. The thumb rule used to provide the bounds is to allow variables with initial values $m_0 \geq 1$ a range of $0.2\, m_0$ between the upper and lower bound. For variables with initial values $m_0 < 1$ the range between bounds was $0.1\, m_0$. This thumb rule was designed empirically and is by no means optimal in any way. In fact, the bounds needed often additional adjustment to obtain convergence.

6. Statistical models

In order to keep the mathematical formulas brief we use the following conventions. Cdfs are denoted by capital letters and pdfs by lower case letters, as shown in Table 2. The parameters are as follows: μ and σ are the mean and standard deviation, while α, β and κ are the shape, scale and location parameters. For the uniform distribution, a and b are the lower and upper boundary, respectively, of the range of x-values for which the distribution is valid. In particular, the parameter a is equivalent to a location parameter, while $(b - a)$ is equivalent to a scale parameter [44]. All logarithmic empirical distribution plots (edfs) use $\log_{10}$-transformations for both axes.

Table 2
Model notation

Uniform	$u_X(x; a, b)$	$U_X(x; a, b)$
Poisson	$\mathrm{po}_X(x; \mu)$	$\mathrm{PO}_X(x; \mu)$
Exponential	$\exp_X(x; \mu)$	$\mathrm{EXP}_X(x; \mu)$
Normal	$\mathrm{n}_X(x; \mu, \sigma)$	$\mathrm{N}_X(x; \mu, \sigma)$
Log-normal	$\mathrm{ln}_X(x; \alpha, \beta)$	$\mathrm{LN}_X(x; \alpha, \beta)$
Generalized Pareto	$\mathrm{gpa}_X(x; \alpha, \kappa, \beta)$	$\mathrm{GPA}_X(x; \alpha, \kappa, \beta)$

The generalized Pareto distribution [54] and the corresponding density function are defined as

$$F_X(x; \alpha, \kappa, \beta) = 1 - \left[1 + \frac{\alpha(x - \kappa)}{\beta}\right]^{-\frac{1}{\alpha}} \tag{5}$$

$$f_X(x; \alpha, \kappa, \beta) = \frac{1}{\alpha}\left[1 + \frac{\alpha(x - \kappa)}{\beta}\right]^{-\frac{1}{\alpha} - 1} \tag{6}$$

where $\alpha \neq 0$ is the shape parameter, $\kappa \leq x$ is the location parameter, and $\beta > 0$ is the scale parameter.

6.1. *Ultrapeer settings and packet-trace statistics*

The results reported here were obtained from an 11-days long link-layer packet trace collected from the BTH network with the methods described in Section 4. The `gtk-gnutella` servent at BTH was configured to run as ultrapeer and to maintain 32–40 connections to other ultrapeers and approximately 100 connections to leaf nodes. The number of connections is a vendor preconfigured value, which is close to the suggested values [15,16]. Although `gtk-gnutella` is capable of operation over UDP, this functionality was turned off. Consequently, the ultrapeer used only TCP for its traffic. No other applications, with the exception of an SSH daemon, were running on the ultrapeer for the duration of the measurements. One SSH connection was used to remotely check on the status of the measurements and the amount of free disk space. The SSH connection was idle for most of the time. The firewall was turned off during the measurements.

The total amount of PCAP data collected with `tcpdump` is approximately 33 GB. The PCAP data generated approximately 45 GB log files. The recorded traffic contains 234 million IP datagrams. The log files show 604 thousand Gnutella sessions that were used to exchange 267 million Gnutella messages. A total of 423 thousand sessions (70%) were unable to perform a successful Gnutella handshake. The main reasons for the unsuccessful handshakes are filled-up connection queues [5] and refusal to accept uncompressed connections. The remaining sessions consist of $181,805$ sessions where both peers used compression, 22 where one of the peers used compression and 10 uncompressed sessions.

6.2. *Session characteristics*

A Gnutella session is defined to be the set of Gnutella messages exchanged over a TCP connection between two directly connected peers that have successfully completed the Gnutella handshake. The session lasts until the TCP connection is closed by either FIN or RST TCP segments.

To describe the Gnutella handshake we have created three pseudo-message types: CLI_HSK, SER_HSK, and FIN_HSK. The CLI_HSK message is the first part of the handshake and it is sent by the peer that opened the TCP connection, i.e., the client. The SER_HSK message is the reply from the peer that received the CLI_HSK, i.e., the server. The FIN_HSK message, which is sent by the client, is the final part of the handshake.

[5] Code 409: "Vendor would exceed 60% of our slots".

Table 3
Incoming session statistics

Type	Max	Min	Mean	Median	Stddev
Duration (s)	767553	0.03	517.30	0.86	6780.99
Messages	7561532	4	585.18	11	22580.99
Bytes	535336627	780	53059	1356	2034418

Table 4
Outgoing session statistics

Type	Max	Min	Mean	Median	Stddev
Duration (s)	470422	0.12	3949.86	2459.10	11170.80
Messages	2644660	6	23145.15	15716.50	58627.75
Bytes	182279191	1574	2173564	1457360	4458468

Table 5
Incoming and outgoing session statistics

Type	Max	Min	Mean	Median	Stddev
Duration (s)	767553	0.03	651.98	0.87	7036.85
Messages	7561532	4	1470.34	11	25375.64
Bytes	535336627	780	136258	1357	2219411

The session duration is computed as the time duration between the instant when the CLI_HSK message is recorded (at link layer) until the recorded time for the last Gnutella message on the same TCP connection.

An incoming session is defined as being a session for which the CLI_HSK message was *received* by the ultrapeer at BTH. Outgoing sessions are sessions for which the CLI_HSK message was *sent* by the ultrapeer at BTH. Tables 3 and 4 show duration (in seconds), number of exchanged messages and bytes for incoming and outgoing sessions, respectively. Table 5 shows the same statistics when no distinction is made between incoming and outgoing sessions.

A Gnutella session is considered valid (in the sense that it is used to compute session statistics) if the Gnutella handshake was successfully completed and at least one Gnutella message was transfered between the two hosts participating in the session. Our data contains 173,711 valid incoming sessions and 7094 valid outgoing sessions.

The tables show that outgoing sessions transfer about 40 times more data than incoming sessions. Furthermore, by comparing the mean and median values for messages and bytes it can be observed that a few sessions transfer the majority of data. This can be partly accounted for by the hierarchy inherent in Gnutella: UPs are bound to transfer more data than their LNs. In addition, most incoming sessions have very short duration (< 1 second), which can be observed by comparing the mean and median duration values for incoming sessions. This translates in little data being exchanged.

These observations confirm earlier results already reported in [5,9,8]. Following the taxonomy used in [8,55], we observe that, although we analyze only signaling traffic without considering data transfers, the sessions can be divided into "mice" i.e., sessions carrying small amounts of data, and "elephants" i.e., sessions responsible for large volumes of traffic.

The same type of heterogeneity appears when we consider session duration. We observe both "dragonflies", which are very short sessions and "tortoises", which are sessions with very long duration.

6.2.1. *Session interarrival and interdeparture times*

The statistics and models for session interarrival and interdeparture times are shown in Table 6 and Table 7. It is observed that interarrival times can be modeled by the lognormal distribution, which is subexponential. In contrast, session interdeparture times require a mixture distribution with a heavy-tailed component, i.e., the Pareto distribution, to provide an acceptable fit.

A possible explanation for the appearance of the the heavy-tailed component is given by the connection cap described in Section 6.1. When a Gnutella peer reaches the preset number of connections it does not attempt to establish more connections until existing connections are terminated. This leads to large session interdeparture times that have a non-negligible probability of occurrence.

The *absence* of the heavy-tailed component from the session interarrival times distribution can be explained as follows. We have noticed that many of the short duration (< 1 second) incoming sessions presented in Section 6.2 transfer one BYE message and are then terminated. This behavior cannot be traced to any of the Gnutella specifications. We assume that the behavior is due to the `gtk-gnutella` implementation, but further study is required to confirm. It appears that `gtk-gnutella` discovers that the connection cap has been reached after the handshake has completed. It then sends the BYE message to terminate the connection, when normally this connection should have been aborted during handshake. Nonetheless, since these sessions are considered valid according to our criteria, the session interarrival times are shorter and we can model them without introducing a heavy-tailed component.

Table 6
Session interarrival and interdeparture times statistics (s)

DIR	Max	Min	Mean	Median	Stddev
IN	1119.01	4.05e-6	5.47	2.20	20.38
OUT	5192.62	0.20e-3	133.99	71.78	210.34

Table 7
Models for session interarrival and interdeparture times (s)

DIR	Model	$E_{\%}$
IN	$\mathrm{LN}_X(x; 0.71, 1.08)$	3.0%
OUT	$0.77\,\mathrm{EXP}_X(x; 0.01) + 0.23\,\mathrm{GPA}_X(x; 0.7, 0, 132.9)$	3.3%

An interesting characteristic was observed when all session interarrival times were considered, that is even those for invalid sessions. This is equivalent to interarrival times for incoming requests to open a session, i.e., incoming CLI_HSK messages. It turns out that the set of all interarrival times is exponentially distributed with parameter $\lambda = 0.58$, as shown in Figure 6 and Table 8. The session arrival rate was analyzed to verify that this is not a measurement error. It is well-known that exponentially distributed interarrival times imply a Poisson arrival rate [56]. As it can be observed in Figure 7, a Poisson distribution $\mathrm{PO}_X(x; 0.58)$ fits well, at least visually. Unfortunately, no $E_{\%}$ measure can be provided since the method does not work with discrete distributions. However, the edf of the data should leave little doubt that the data is indeed Poisson distributed. The edf is plotted without log-scaled axes, since most of the data, 99.9% of the probability mass, is clustered around the values $0, 1, \ldots, 4$.

Table 8
Gnutella (valid and invalid) session interarrival times

Statistic	Model	$E_{\%}$
Interarrival times (s)	$\mathrm{EXP}_X(x; 0.58)$	1.7%
Rate (session/second)	$\mathrm{PO}_X(x; 0.58)$	N/A

The same relation does not hold for outgoing traffic, which is well modeled by a mixture distribution $0.88\,\mathrm{LN}_X(x; -2.32, 1.41) + 0.12\,\mathrm{EXP}_X(x; 0.008)$ with 1.1% error.

The appearance of the Poisson distribution can be explained by the mixture of arriving CLI_HSK message from different sources. If one assumes that these

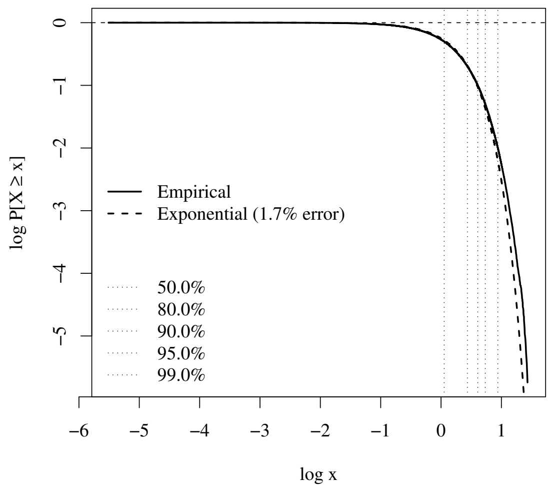


Fig. 6. Interarrival times (s)

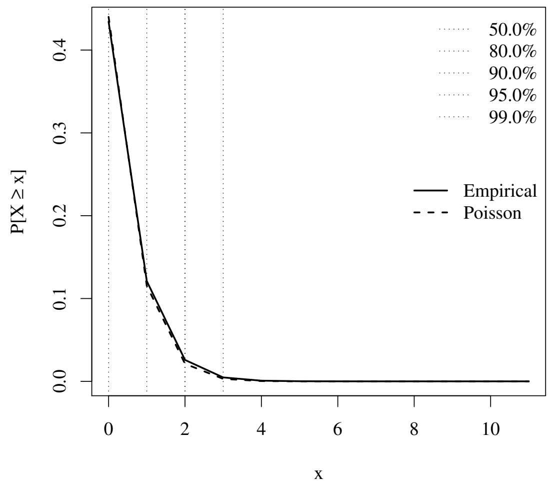


Fig. 7. Incoming session rate (sessions/s)

arrivals are generated by a number of point processes, then the superposition of point processes converges, under some mild assumptions, to a Poisson distribution as the number of sources increases [57–59].

This hypothesis does however not apply to outgoing CLI_HSK due to the connection cap. Once `gtk-gnutella` reaches the preset amount of connections it does not attempt to establish new ones.

Table 9
Session size and duration models

Statistic	Model	$E_{\%}$
Session size (bytes)	$0.69\,\mathrm{N}_X(x; 1356, 5.9)+$ $0.31\,\mathrm{LN}_X(x; 9.0, 3.17)$	4.7%
Session duration (s)	$0.57\,\mathrm{N}_X(x; 0.85, 0.07)+$ $0.33\,\mathrm{LN}_X(x; 0.37, 0.96)+$ $0.10\,\mathrm{U}_X(x; 18.45, 2460)$	2.3%
Session duration, upper 5% (s)	$\mathrm{GPA}_X(1.1, 1800, 1870.4$	2.4%

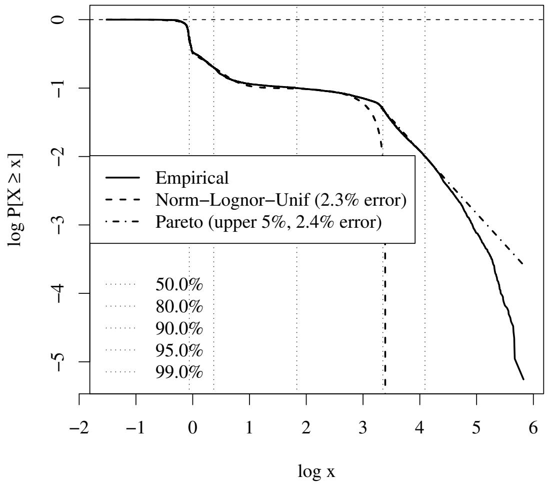


Fig. 8. Gnutella session duration (s)

6.2.2. *Session size and duration*

The session size and duration models are reported in Table 9 and Figure 8. It is observed that the session duration statistic has a very complex ccdf, which cannot be modeled with only two distributions. This is the only model reported here that uses a mixture of three distributions. Alternatively, the upper 5% of the tail can be modeled with a Pareto distribution.

Most of the observed session sizes (64.8%) lie in the range 1300–1400 bytes, 9.6% are smaller than 1300 bytes and 25.6% are larger than 1400 bytes.

6.3. *Message characteristics*

In this section message statistics are reported for each Gnutella message type. The message type UNKNOWN denotes messages with a valid Gnutella header, but with unrecognized message type. These messages are either experimental or corrupted. The message type ALL is used for statistics computed over all messages, irrespective of type. Only models for the aggregated message streams, i.e., message type ALL, are presented. The tables for message statistics do not contain any information on the number of samples used to compute the statistics for each message type. However, this information is fully available in [29]. We have observed 114.7 million incoming messages and 152 million outgoing messages.

Table 10 shows interarrival times for messages received by the BTH ultrapeer and Table 11 shows interdeparture times for messages sent by the BTH ultrapeer. Although the PCAP timestamps have microsecond resolution [30], the times presented here have only 100 μs precision. This is due to memory limitations in the postprocessing software.

Summing over the number of samples for each message type does not add up to the value shown in the number of samples for message type ALL. The reason is the analysis software which ignores messages that generate negative interarrival/interdeparture times. Negative times appear because the application flow reassembly handles several (typically more than a hundred) connections at the same time. On each connection the timestamp for arriving packets is monotonically increasing. However, the interarrival/interdeparture statistics presented here are computed across all connections. To ensure monotonically increasing timestamps even in this case, new messages from arbitrary connections are stored in a buffer, where they are sorted by timestamp. The size of the buffer is limited to 500,000 entries due to memory management issues. We have observed on average 280 messages per second [29]. This means that the buffer can store about 30 minutes of average traffic and much less during traffic bursts. If there are delayed messages due to TCP retransmissions or other events, they reach the buffer too late and are discarded.

The large interarrival and interdeparture times in handshake messages (CLI_HSK, SER_HSK, FIN_HSK) observed in Table 10 and Table 11 occur because once a servent reaches the preset amount of connections, it no longer accepts or attempts to open new connections until one or more of the existing connections is closed. This behavior also explains the large interarrival and interdeparture times for BYE messages.

Table 10
Message interarrival time statistics (s)

Type	Max	Min	Mean	Median	Stddev
CLI_HSK	28.4591	0.0001	1.7246	1.1256	1.8644
SER_HSK	5185.0490	0.0001	19.6294	0.2090	92.1849
FIN_HSK	1118.9920	0.0001	5.3165	2.1942	19.4800
PING	13.5871	0.0001	0.2762	0.1931	0.2726
PONG	2.2624	0.0001	0.1404	0.0979	0.1383
QUERY	1.4514	0.0001	0.0343	0.0240	0.0340
QUERY_HIT	19.2778	0.0001	0.1842	0.0976	0.2661
QRP	50.0632	0.0001	2.0475	1.0534	2.8707
HSEP	1780.4420	0.0003	6.1560	4.3834	8.4758
PUSH	40.1396	0.0001	0.0677	0.0405	0.1157
BYE	1119.5930	0.0001	5.9160	2.3591	22.3494
VENDOR	30.8037	0.0001	0.4346	0.2207	0.5993
UNKNOWN	51576.8600	3.0680	2075.3190	6.9379	9298.3600
ALL	9.8299	0.0001	0.02436	0.0169	0.0243

Table 11
Message interdeparture time statistics (s)

Type	Max	Min	Mean	Median	Stddev
CLI_HSK	5189.2340	0.0002	17.9655	0.1273	88.8506
SER_HSK	28.4595	0.0003	1.7298	1.1287	1.8712
FIN_HSK	5185.5150	0.0006	28.4784	0.3305	110.2372
PING	20.5910	0.0001	1.3773	0.5077	2.1342
PONG	2.7215	0.0001	0.1573	0.1012	0.1682
QUERY	12.1151	0.0001	0.0295	0.0003	0.0541
QUERY_HIT	19.2818	0.0001	0.2188	0.1285	0.2885
QRP	603.3599	0.0001	2.6350	0.0004	19.8572
HSEP	358.3067	0.0001	2.5020	1.4089	5.8293
PUSH	76.5303	0.0001	0.0429	0.0003	0.1713
BYE	3849.4550	0.0001	134.8121	77.2090	187.7784
VENDOR	64.6689	0.0001	1.8253	1.1124	2.4838
UNKNOWN	N/A	N/A	N/A	N/A	N/A
ALL	1.5450	0.0001	0.0178	0.0003	0.0353

It is interesting to see that interarrival times are exponentially distributed as shown in Table 12 and Figure 9. Analysis of the arrival process reveals that this is is not a pure Poisson process, but rather a compound Poisson process [60,61,56] since simultaneous message arrivals do occur. To understand why this happens, recall that before the messages can be extracted from the TCP flows, these flows pass

Table 12
Models for message interarrival and interdeparture times (s)

DIR	Message	Model	$E_{\%}$
IN	ALL	$\mathrm{EXP}_X(x; 40.96)$	0.16%
OUT (upper 26.1%)	ALL	$0.261\,\mathrm{EXP}_X(x; 20.23)$ (see Table 13 for the body)	3.8%

Table 13
Probability mass points for message interdeparture times (s)

Interdeparture times	0.0001	0.0002	0.0003	0.0004	0.0005
Probability	0.024	0.515	0.155	0.033	0.012

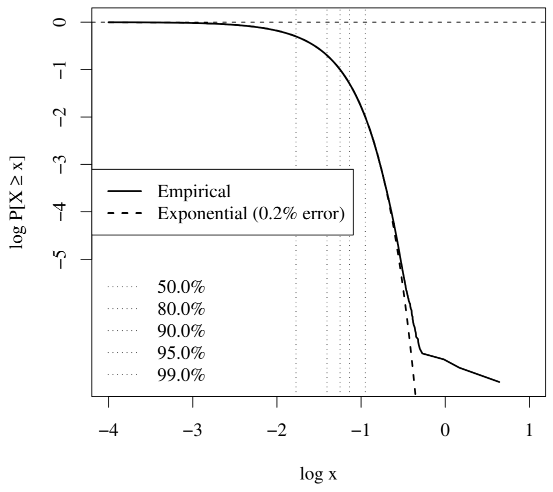


Fig. 9. Message interarrival times (s)

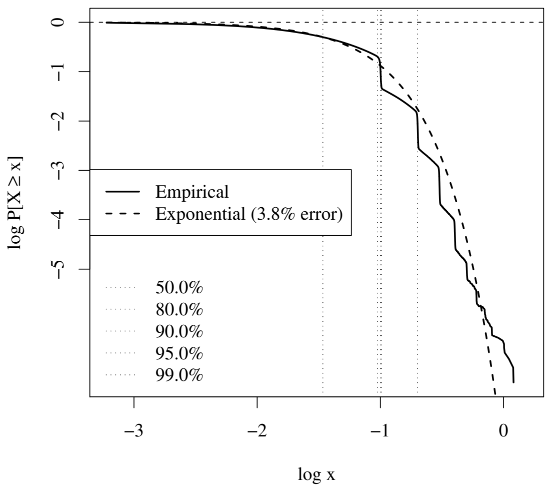


Fig. 10. Message interdeparture times, upper 26.1%

through a decompression layer. Typically, a single TCP segment carries several Gnutella messages. All of them receive the same timestamp, since they traveled in bulk all the way from the source to the destination. Models for the bulk-size distributions are provided in Table 15 and Table 16

The appearance of the Poisson distribution can be explained by arguments similar to those considered in Section 6.2.1.

Message interdeparture times have an interesting distribution. As it can be observed in Table 13, approximately 73.9% of the probability mass is clustered around the values 0.0001–0.0005. The remaining 26.1% of the probability mass can be modeled by an exponential distribution ($\lambda = 20.23$) with 3.8% error.

Table 14 shows the message size statistics for each Gnutella message type. In contrast to the other tables, messages are not classified by direction (incoming or outgoing). The rationale is that the message size is independent of message direction. It can be observed that, on average, QUERY_HIT and QRP messages have the largest size. They are closely followed by handshake messages, where the capability headers account for most of the data. It is interesting to notice that the maximum size of QUERY_HIT messages is 39 KB, which is an order of magnitude

Table 14
Message size statistics (bytes)

Type	Max	Min	Mean	Median	Stddev
CLI_HSK	696	22	336.91	328	65.69
SER_HSK	2835	23	386.83	369	145.69
FIN_HSK	505	23	107.92	76	88.55
PING	34	23	25.48	23	3.88
PONG	464	37	74.96	61	38.68
QUERY	376	26	70.17	55	46.40
QUERY_HIT	39161	58	590.28	358	1223.58
QRP	4124	29	608.60	540	596.70
HSEP	191	47	70.39	71	28.15
PUSH	49	49	49.00	49	0.00
BYE	148	35	40.02	37	15.84
VENDOR	177	31	36.45	33	19.51
UNKNOWN	43	23	23.53	23	3.24
ALL	39161	22	93.45	49	303.26

Table 15
Message size (bytes) and bulk size distribution

DIR	Message	Model	$E_{\%}$
IN/OUT	ALL	$0.81\,\mathrm{LN}_X(x; 3.94, 0.23)+$ $0.19\,\mathrm{LN}_X(x; 5.14, 1.24)$	4.3%
IN/OUT	Bulk size	$0.003\,\mathrm{GPA}_X(x; 0.42, 15, 9.6)$	5.0%

Table 16
Probability mass points for message bulk size

Bulk size (messages)	1	2	3	4	5
Probability	0.586	0.173	0.082	0.049	0.031
Bulk size (messages)	6	7	8	9	10
Probability	0.020	0.012	0.008	0.005	0.003
Bulk size (messages)	11	12	13	14	15
Probability	0.019	0.005	0.002	0.001	0.001

larger than the 4 KB specified in [14].

The model for the message bulk size is reported in Table 15 and Table 16. Bulks of size 1–15 use 99.7% of the probability mass. The remaining 0.3% of the probability mass is modeled with a Pareto distribution.

6.4. *Transfer rate characteristics*

This section reports on transfer rates in bytes/second and in messages/second for each Gnutella message types. All statistics are computed over 950,568 samples. The number of samples is equal to the time duration expressed in seconds (approximately 11 days) for the available measurement data. Models are reported only for aggregate message flows, i.e., type ALL messages. As it can be observed in Table 18 both incoming and outgoing transfer rates are heavy-tailed. In terms of specific message types, QUERY and QUERY_HIT messages dominate incoming and outgoing streams, both in terms of average message rate as well as average byte rates. This was expected since the Gnutella system is used primarily for searching for files.

6.5. *Traffic characteristics at IP layer*

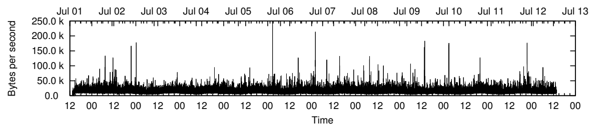
Table 19 provides the summary statistics for the IP byte rates. It is interesting to note that the mean and median IP byte rates are very similar to the corresponding statistics for Gnutella byte rates shown in Table 17. These values alone indicate that the compression of Gnutella messages does not yield

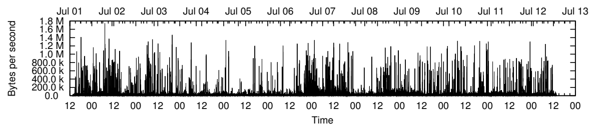
Table 17
Message byte rate (bytes/s) statistics

Type	DIR	Max	Min	Mean	Median	Stddev
CLI_HSK	IN	4126	0	187	0	258
CLI_HSK	OUT	14519	0	27	0	273
SER_HSK	IN	12507	0	31	0	289
SER_HSK	OUT	4001	0	212	0	306
FIN_HSK	IN	982	0	15	0	42
FIN_HSK	OUT	4474	0	9	0	94
PING	IN	1665	0	92	92	50
PING	OUT	503	0	19	0	45
PONG	IN	17043	0	1213	1173	541
PONG	OUT	26050	0	2235	2162	1179
QUERY	IN	24101	0	4441	4317	1426
QUERY	OUT	46424	0	5088	4702	2511
QUERY_HIT	IN	1736791	0	4868	1912	23917
QUERY_HIT	OUT	360235	0	3355	1837	5229
QRP	IN	47340	0	389	0	1408
QRP	OUT	152820	0	353	0	3660
HSEP	IN	940	0	8	0	21
HSEP	OUT	2185	0	32	0	58
PUSH	IN	52332	0	1285	1127	948
PUSH	OUT	200459	0	1964	1568	1829
BYE	IN	1720	0	6	0	16
BYE	OUT	4956	0	1	0	11
VENDOR	IN	210702	0	347	33	2514
VENDOR	OUT	2197	0	44	0	81
UNKNOWN	IN	23	0	0	0	0.1
UNKNOWN	OUT	43	0	0	0	0.1
ALL	IN	1745341	0	12883	10113	24287
ALL	OUT	370825	0	13338	12062	7624

Table 18
Gnutella (ALL) byte rate (bytes/s) modeling results

DIR	Model	$E_{\%}$
IN	$0.76\,\mathrm{LN}_X(x; 9.26, 0.37)+$ $0.23\,\mathrm{GPA}_X(x; 1.06, 0, 4003)$	5.2%
OUT	$0.81\,\mathrm{LN}_X(x; 9.43, 0.39)+$ $0.19\,\mathrm{GPA}_X(x; 0.63, 0, 3704)$	5.3%


(a) Incoming IP byte rate


(b) Incoming Gnutella byte rate (message type ALL)

large gains. However, if one takes into consideration the maximum and standard deviation values it can be observed that the compression removes much of the burstiness from the application layer, leading to smoother traffic patterns. This effect is visible if one compares Figure 11(a) to Figure 11(b).

In Table 19 it can be observed that the incoming and outgoing IP byte rates are quite similar. The statistical models shown in Table 20 are further evidence to that. It is observed that a heavy-tailed component is present in each model.

The upper 52% of the IP datagram size distribution can be modeled by a Pareto distribution defined in Table 21. The ccdf plot for it is shown in Figure 11. The probability mass points for the datagram sizes corresponding to the lower 48% of the distribution are shown in Table 22. It can be observed that 46.7% of the probability mass is accounted for by IP datagrams with size 40 bytes and 52 bytes, respectively. The 40-bytes datagrams correspond to TCP segments with no data and no options.

The interarrival and interdeparture times statistics are reported in Table 23 and the corresponding models are provided in Table 24. It is interesting to note that interarrival times for IP datagrams follow an exponential distribution. This is similar to the case of interarrival times for valid and invalid sessions as well as to the case of interarrival times for Gnutella messages (type ALL). Just as in those cases, it is conjectured that the superposition of point processes is responsible for this phenomenon.

Table 19
IP layer byte rate (bytes/s) statistics

DIR	Max	Min	Mean	Median	Stddev
IN	249522	0	11536	10961	4075
OUT	176986	0	12668	12037	5722

Table 20
Models for IP layer byte rates (bytes/s)

DIR	Model	$E_{\%}$
IN	$0.89\,\mathrm{LN}_X(x; 9.33, 0.26) +$ $0.11\,\mathrm{GPA}_X(x; 0.32, 0, 2774)$	3.5%
IN (upper 30%)	$\mathrm{GPA}_X(x; 0.27, 12812, 2180)$	2.2%
OUT	$0.86\,\mathrm{LN}_X(x; 9.45, 0.27) +$ $0.14\,\mathrm{GPA}_X(x; 0.87, 0, 1662)$	2.3%

Table 21
Model for IP datagram size (bytes)

DIR	Model	$E_{\%}$
IN/OUT	$0.52\,\mathrm{GPA}_X(x; 0.53, 60, 50.55)$	3.3%

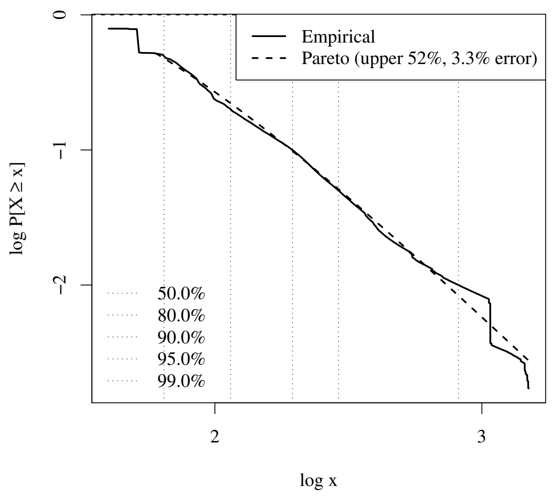


Fig. 11. IP datagram size (bytes)

Table 22
Probability mass points for IP datagram size (bytes)

Datagram size	0–39	40	41	42	43	44
Probability	0.000	0.209	0.000	0.001	0.000	0.000
Datagram size	45	46	47	48	49	50
Probability	0.000	0.000	0.008	0.006	0.000	0.000
Datagram size	51	52	53	54	55	
Probability	0.000	0.258	0.003	0.001	0.000	

7. Conclusions

The work presented here is focused on characteristics and statistical models for Gnutella network traffic. New traffic characteristics are reported for Gnu-

Table 23
IP datagram interarrival and interdeparture times statistics (s)

DIR	Max	Min	Mean	Median	Stddev
IN	11.36	3e-6	8.85e-3	5.75e-3	9.72e-3
OUT	22.05	7e-6	8.24e-3	0.49e-3	14.22e-3

Table 24
Interarrival and interdeparture times models for IP datagrams (s)

DIR	Message	Model	$E_{\%}$
IN	IP datagrams	$\mathrm{EXP}_X(x; 118.76)$	0.8%
OUT	IP datagrams	$0.5\,\mathrm{LN}_X(x; -8.45, 0.26)+$ $0.5\,\mathrm{EXP}_X(x; 61.29)$	1.2%

tella traffic at various layers in the TCP/IP stack: IP level, Gnutella session level and Gnutella message level. Additionally, we provide statistical models for interarrival times, interdeparture times, and size at each level. For Gnutella sessions we model also the session duration.

Another contribution of this work is the design and implementation of a modular measurement infrastructure for P2P traffic. Using the measurement infrastructure developed at BTH, Gnutella traffic from the BTH network was recorded as a link-layer packet trace. Decoding and analysis of the packet trace yielded statistical characteristics and models for sessions, flows and messages, which are the major contributions of this paper.

Although the packet trace provided much insight in the patterns of Gnutella network traffic, there are several other questions that warrant further investigation. Foremost, analysis of the degree of long-range dependence is necessary in order to determine the amount of correlation that occurs in the traffic. This issue is relevant in the light of results presented in [58,59], which indicate that certain types of traffic aggregations tend towards a Poisson distribution when the number of sources increases. A Gnutella ultrapeer can be viewed as an application layer router that aggregates traffic flows from many peers. If the results from [58,59] hold in the case of a Gnutella ultrapeer, then it follows that statistical multiplexing gains may occur. It would be very interesting to measure the extent to which that happens, if any.

The models developed here will be used to generate synthetic traffic in a simulator. The idea is to tweak some of the Gnutella messages to carry QoS information about peer connections, e.g., throughput, RTT, packet loss. Thus, a QoS overlay network will be built on top of Gnutella. We plan to use the overlay network to perform QoS routing, which would enable high-quality interactive multimedia services. The real challenge is to ensure that the QoS routing runs smoothly, with little overhead compared to the payload data.

References

[1] R. Schollmeier, A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications, in: Proceedings of the IEEE 2001 International Conference on Peer-to-Peer Computing (P2P2001), Linköping, Sweden, 2001.

[2] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, Z. Xu, Peer-to-peer computing, Tech. Rep. HPL-2002-57, HP Laboratories Palo Alto, `http://www.hpl.hp.com/techreports/2002/HPL-2002-57R1.pdf` (Jul. 2003).

[3] J. Eberspächer, R. Schollmeier, Peer-to-Peer Systems and Applications, no. 3485 in Lecture Notes in Computer Science, Springer-Verlag, Berling, Heidelberg, Germany, 2005, Chapter 2. What is This "Peer-to-Peer" About?, pp. 9–16, iSBN: 3-540-29192-X.

[4] E. Adar, B. A. Huberman, Free riding on gnutella, First Monday 5 (10), `http://www.firstmonday.org/issues/issue5\_10/adar/index.html`.

[5] S. Saroiu, P. K. Gummadi, S. D. Gribble, A measurement study of peer-to-peer file sharing systems, Tech. Rep. UW-CSE-01-06-02, Deparment of Computer Science and Engineering, University of Washington, Seattle, WA, USA (Jul. 2001).

[6] S. Saroiu, P. K. Gummadi, S. D. Gribble, A measurement study of peer-to-peer file sharing systems, in: Proceedings of the Multimedia Computing and Networking (MMCN), 2002.

[7] S. Sen, J. Wang, Analyzing peer-to-peer traffic across large networks, IEEE/ACM Transactions on Networking 12 (2) (2004) 219–232.

[8] N. B. Azzouna, F. Guillemin, Experimental analysis of the impact of peer-to-peer applications on traffic in commercial ip networks, European Transactions on Telecommunications: Special Issue on P2P Networking and P2P Services.

[9] S. Sen, J. Wang, A measurement study of peer-to-peer file sharing systems, in: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment, Marseille, France, 2002, pp. 137–150.

[10] T. Karagiannis, A. Broido, N. Brownlee, K. Claffy, M. Faloutsos, File-sharing in the internet: A characterization of p2p traffic in the backbone, Tech. rep., University of California, Riverside (Nov. 2003).

[11] T. Karagiannis, A. Broido, M. Faloutsos, K. Claffy, Transport layer identification of p2p traffic, in: Internet Measurement Conference (IMC), Taormina, Sicily, Italy, 2004.

[12] T. Karagiannis, A. Broido, N. Brownlee, K. Claffy, M. Faloutsos, Is p2p dying or just hiding?, in: IEEE Globecom 2004 - Global Internet and Next Generation Networks, Dallas, TX, USA, 2004.

[13] Clip2, The Annotated Gnutella Protocol Specification v0.4, The Gnutella Developer Forum (GDF), 1st Edition, http://groups.yahoo.com/group/the_gdf/files/Development (Jul. 2003).

[14] T. Klingberg, R. Manfredi, Gnutella 0.6, The Gnutella Developer Forum (GDF), http://groups.yahoo.com/group/the_gdf/files/Development (Jun. 2002).

[15] A. Singla, C. Rohrs, Ultrapeers: Another Step Towards Gnutella Scalability, Lime Wire LLC, 1st Edition, http://groups.yahoo.com/group/the_gdf/files/Development (Nov. 2002).

[16] A. A. Fisk, Gnutella Dynamic Query Protocol, LimeWire LLC, 0th Edition, http://groups.yahoo.com/group/the_gdf/files/Proposals/Working Proposals/search/Dynamic Querying/ (May 2003).

[17] P. Verdy, Gnutella topology, The Gnutella Developer Forum (GDF), http://groups.yahoo.com/group/the_gdf/message/22187 (Jan. 2006).

[18] Gnucleus, Gnutella web cache, http://www.gnucleus.com/gwebcache (Jun. 2006).

[19] Gnutella protocol development, http://www.the-gdf.org (Dec. 2005).

[20] R. Manfredi, Gnutella Traffic Compression, The Gnutella Developer Forum (GDF), http://groups.yahoo.com/group/the_gdf/files/Proposals/Working%20Proposa% ls/Gnet%20Compression (Jan. 2003).

[21] J.-l. Gailly, M. Adler, zlib, http://www.gzip.org/zlib (Dec. 2005).

[22] P. Leach, M. Mealling, R. Salz, RFC 4122: A Universally Unique IDentifier (UUID) URN Namespace, category: Standards Track (Jul. 2005).

[23] C. Rohrs, SACHRIFC: Simple Flow Control for Gnutella, Lime Wire LLC (Mar. 2002).

[24] C. Rohrs, Query Routing for the Gnutella Network, Lime Wire LLC, 1st Edition, http://groups.yahoo.com/group/the_gdf/files/Development (May 2002).

[25] B. H. Bloom, Space/time trade-offs in hash coding with allowable errors, Communication of the ACM Volume 13 (Number 7) (1970) p. 422–426, iSSN:0001-0782.

[26] A. A. Fisk, Gnutella Ultrapeer Query Routing, Lime Wire LLC, 0th Edition, http://groups.yahoo.com/group/the_gdf/files/Proposals/Working Proposals/search/Ultrapeer QRP (May 2003).

[27] T. Schürger, Horizon size estimation on the Gnutella network v0.2, http://www.menden.org/gnutella/hsep.html (Mar. 2004).

[28] F. Michaut, F. Lepage, Application-oriented network metrology: Metrics and active measurement tools, IEEE Communications Surveys & Tutorials 7 (2) (2005) 2–24.

[29] D. Ilie, Gnutella network traffic: Measurements and characteristics, Licentiate dissertation, Blekinge Institute of Technology (BTH), Karlskrona, Sweden, ISBN: 91-7295-084-6 (Apr. 2006).

[30] L. Torvalds, do_gettimeofday(), Linux 2.6.14 Kernel Sources, arch/i386/kernel/time.c (2005).

[31] D. Ilie, D. Erman, A. Popescu, A. A. Nilsson, Traffic measurements of p2p systems, in: Proceedings of SNCNW, Karlstad, Sweden, 2004, pp. 25–29.

[32] V. Jacobsen, C. Leres, S. McCanne, Tcpdump/libpcap, http://www.tcpdump.org (Aug. 2005).

[33] S. Ostermann, Tcptrace, http://www.tcptrace.org (Aug. 2005).

[34] D. Ilie, D. Erman, A. Popescu, A. A. Nilsson, Measurement and analysis of gnutella signaling traffic, in: Proceedings of IPSI, Stockholm, Sweden, 2004.

[35] D. Erman, D. Ilie, A. Popescu, Bittorrent session characteristics and models, in: D. Kouvatsos (Ed.), Proceedings of HET-NET, Ilkley, West Yorkshire, UK, 2005, pp. P30/1–P30/10.

[36] D. Erman, Bittorrent traffic measurements and models, Licentiate dissertation, Blekinge Institute of Technology (BTH), Karlskrona, Sweden, iSBN: 91-7295-071-4 (Oct. 2005).

[37] G. R. Wright, W. R. Stevens, TCP/IP Illustrated: The Implementation, Vol. 2 of Professional Computing Series, Addison Wesley, Boston, MA, USA, 1995, ISBN: 0-201-63354-X.

[38] V. Paxson, Empirically derived analytic models for wide-area tcp connections, IEEE/ACM Transactions on Networking 2 (4) (1994) 316–336.

[39] R. B. D'Agostino, M. A. Stephens, Goodness-of-Fit Techniques, Vol. 68 of STATISTICS: textbooks and monographs, Marcel Dekker, Inc., 1986, ISBN: 0-8247-7487-6.

[40] J. Maindonald, J. Braun, Data Analysis and Graphics using R: An Example-based Approach, Cambridge Series in Statistical and Probabilistic Mathematics, Cambridge University Press, Cambridge, UK, 2003, ISBN: 0-521-81336-0.

[41] A. M. Mood, F. A. Graybill, D. C. Boes, Introduction to the Theory of Statistics, 3rd Edition, McGraw-Hill, New York, NY, USA, 1974, ISBN: 0-07-085465-3.

[42] D. C. Montgomery, G. C. Runger, 2nd Edition, John Wiley & Sons, Hoboken, NJ, USA, 1999, ISBN: 0-471-17027-5.

[43] J. Beran, Statistics for Long-Memory Processes, Vol. 61 of Monographs on Statistics and Applied Probability, Chapman & Hall, 1994, ISBN: 0-412-04901-5.

[44] A. M. Law, W. D. Kelton, Simulation Modeling and Analysis, 3rd Edition, McGraw-Hill, New York, NY, USA, 2000, ISBN: 0-07-059292-6.

[45] J. Banks, J. S. Carson II, B. L. Nelson, D. M. Nicol, Discrete-Event System Simulation, 3rd Edition, Prentice Hall, Upper Saddle River, NJ, USA, 2001, ISBN: 0-13-088702-1.

[46] D. Ilie, D. Erman, A. Popescu, Transfer rate models for gnutella signaling traffic, in: Proceedings of ICIW, Guadeloupe, French Caribbean, 2006.

[47] F. N. David, N. L. Johnson, The probability integral transform when the variable is discontinuous, Biometrika 37 (1–2) (1950) 42–49.

[48] D. M. Titterington, A. F. M. Smith, U. E. Makov, Statistical Analysis of Finite Mixture Distributions, John Wiley & Sons, 1985, ISBN: 0-471-90763-4.

[49] R Development Core Team, R: A Language and Environment for Statistical Computing, R Foundation for Statistical Computing, Vienna, Austria, ISBN 3-900051-07-0
http://www.R-project.org (2005).

[50] W. H. Press, S. A. Teukolsky, W. T. Vettering, B. P. Flannery, Numerical Recipes in C: The Art of Scientific Computing, 2nd Edition, Cambridge University Press, Cambridge, UK, 1992, ISBN: 0-521-43108-5.

[51] J. C. Lagarias, J. A. Reeds, M. H. Wright, W. P. E., Convergence properties of the Nelder-Mead simplex algorithm in low dimensions, SIAM Journal on Optimization 9 (1) (1998) 112–147.

[52] J. H. Mathews, K. K. Funk, Numerical Methods using Matlab, 4th Edition, Prentice Hall, Upper Saddle River, NJ, USA, 2004, Ch. 8: Numerical Optimization, pp. 430–436, ISBN: 0-13-065248-2.

[53] R. H. Byrd, P. Lu, J. Nocedal, C. Zhu, A limited memory algorithm for bound constrained optimization, Tech. Rep. NAM-08, Northwestern University, Evanston, IL, USA (May 1994).

[54] S. Coles, An Introduction to Statistical Modeling of Extreme Values, Springer Series in Statistics, Springer-Verlag, London, UK, 2001, ISBN: 1-85233-459-2.

[55] N. Brownlee, K. C. Claffy, Understanding internet traffic streams: Dragonflies and tortoises, IEEE Communications Magazine (2002) 110–117.

[56] L. Kleinrock, Queueing Systems Volume 1: Theory, John Wiley & Sons, Hoboken, NJ, USA, 1975, ISBN: 0-471-49110-1.

[57] K. Sriram, W. Whitt, Characterizing superposition arrival processes in packet multiplexers for voice and data, IEEE Journal on Selected Areas in Communications SAC-4 (6) (1986) 833–846.

[58] J. Cao, K. Ramanan, A poisson limit for buffer overflow probabilities, in: Proceedings of IEEE Infocom 2002, no. 1, 2002, pp. 994–1003.

[59] J. Cao, W. S. Cleveland, D. Lin, D. X. Sun, Nonlinear Estimation and Classification, Vol. 171 of Lecture Notes in Statistics, Springer-Verlag, New York, NY, USA, 2003, Ch. Internet Traffic Tends Toward Poisson and Independent as the Load Increases, pp. 83–110, ISBN: 0-387-95471-6.

[60] S. M. Ross, Applied Probability Models With Optimization Applications, Dover Books On Mathematics, Dover Publications, Mineola, NY, USA, 1992, ISBN: 0-486-67314-6.

[61] D. R. Cox, H. D. Miller, The Theory of Stochastic Processes, Chapman & Hall, London, UK, 1965, ISBN: 0-412-15170-7.