# An Analysis of Internet Chat Systems

Christian Dewes
Universität des Saarlandes
cdewes@net.uni-sb.de

Arne Wichmann
TU München
aw@net.in.tum.de

Anja Feldmann
TU München
anja@net.in.tum.de

## Abstract

In our quest to better understand network traffic dynamics, we examine Internet chat systems. Although chat as an application does not contribute huge amounts of traffic, chat systems are known to be habit-forming. This implies that catering to such users can be a promising way of attracting them, especially in low bandwidth environments such as wireless networks.

Unfortunately there is no common protocol base for chat systems. Rather there are a multitude of protocol variants whose specifications, with some exceptions, such as IRC and ICQ, are unavailable or ill defined. In addition, chat systems are often layered on top of other application protocols like HTTP. Therefore there is no simple way of even identifying chat traffic. In this paper we show how to separate chat traffic from other Internet traffic and present the results of an extensive validation of our methodology. Using our methodology we gather a week long trace of all chat traffic that crosses a 155 Mbit/s link from the Saarland University to the Internet and present an initial characterization.

## Categories and Subject Descriptors

C.2.3 [**Computer-Communication Networks**]: Network Operations

## General Terms

Measurement, Experimentation

## Keywords

Chat, IRC, Network Measurements

## 1. INTRODUCTION

Chat systems, and especially Web-chats, are used by a sizable number of people around the world for exchanging ideas and/or discussions. Considering the spectrum of Internet applications, chat systems occupy the extreme, short-lived interactive end, while stable, non-interactive applications like the Web occupy another.

Given that chat as an application does not generate a lot of traffic it is not surprising that chat systems do not contribute a large fraction of traffic to the Internet. Therefore one may ask why is chat a worthwhile target for traffic characterization. Our primary motivation is that chat offers computer mediated communication. This means that the measured "behavior" is expected to mainly depend on human behavior (maybe influenced in some ways by packet delays). Furthermore chat is used by a large number of users. It has the potential of being habit-forming [1]. Since chat systems need little bandwidth, they are an interesting and attractive wireless application. A wireless application that has gained extreme popularity in Asia and Europe is SMS (short message service) and is used by some users in a manner similar to chat.

In today's Internet there are various chat systems in use which differ in a number of aspects. Internet Relay Chat (IRC) [2] is one of the oldest systems still in use. From a high level perspective an IRC-system consists of a network of servers which form a spanning tree among them, that is used as the backbone of the network. A client using an IRC-system connects himself to one of these servers and the messages will travel along the backbone to each of the connected servers. The IRC protocol which regulates message exchange is well defined and easy to understand. Accordingly we chose IRC as a starting point for our analysis.

While IRC is still a widely used protocol, a significant share of today's users appear to be using Web-based chat systems. There seem to be two main motivations: ease of use and ease of access. Everyone is familiar with the user interface of a Web browser, and most portals are offering Web-based chat systems. Unfortunately these systems vary widely in terms of visual appearance and technical realization as well as in terms of protocol. Nevertheless we have identified a number of common characteristics of Web-based systems that can be used to separate traffic generated by them from other traffic. As a result we can analyze Web-based chat systems and compare their characteristics to those of IRC-systems.

Other popular chat systems include ICQ [3] and AIM [4] (AOL Instant Messenger). These differ from the above systems in that they use UDP as transport protocol and that their main goal is the exchange of short messages rather than the exchange of opinions or longer discussions. Most chat systems are open systems. In contrast Gale [5] is a fairly new system which encrypts its messages cryptographically and supports even more decentralization of the server functions than IRC.

The main problem in analyzing chat traffic is the multitude and the diversity of systems and protocols. In addition IRC differs significantly from Web-chat and there is no unique Web-chat system. Indeed most Web-chat protocols differ widely and are usually not well documented. Some systems do not only use the Web browser as their user interface but realize their protocol on top of HTTP. This implies that it becomes rather difficult to consistently separate Web-chat traffic from regular Web traffic and other TCP traffic. Therefore we first have to address the challenge of capturing chat traffic before we can present the results of our analysis.

Our approach for separating Web-chat traffic from other network traffic is to first collect traffic generously, then identify candidate chat traffic, and finally disregard known non-chat traffic. We start by collecting all network traffic that satisfies some rather general criteria. We then keep all connections which match a number of criteria derived from our analysis of IRC and a basic understanding of Web-chat systems. Close inspection of the resulting traffic has shown that some other protocols may have similar behavior. These are easy to identify and can therefore be excluded from further analysis. In order to show the validity of this approach we present the results of an extensive validation and a statistical characterization of the collected chat traffic.

Another problem with this approach is imposed by resource limitations. The amount of traffic to process exceeded the disk, cpu, and memory resources of our server machine. Accordingly we further subdivided the work into two filter steps, the first checks for the inexpensively computable criteria while the second performs the computational expensive ones. This pipelining enables us to cope with the amount of data.

The remainder of this paper is structured as follows: first, in Section 2, we will review some details of IRC and Web-chat systems. Next we present our methodology for collecting chat traces (Section 3). In Section 4 this is followed by an account of our experiences in realizing these ideas. As we propose to use quite a complex heuristic Section 5 discusses our approach for validating their appropriateness. In Section 6 we present the results of analyzing the properties of IRC and Web-chat usage. Finally, in Section 7 we summarize our experience and suggest future research directions.

## 2. BACKGROUND

### 2.1 IRC

Internet Relay Chat (IRC, [2]) is a rather widely used chat protocol that has been around for about 15 years. Between 06/28/2003 and 07/28/2003 five big IRC-networks (probably the five biggest) together had 457,190 users on average [6]. Each IRC-network consists of a set of connected servers that are typically listening for client connection requests on port 6667. An active client is usually connected to one server and uses a specific nickname to identify himself within the IRC-network. While a user may change her nickname, nicknames have to be unique within the network.

There are a number of commands a client can use within an IRC network. One of the most common one is `PRIVMSG`, for sending a message to a specific user or channel. A typical form of this command is: `:nick1 PRIVMSG nick2 :Hi!`,

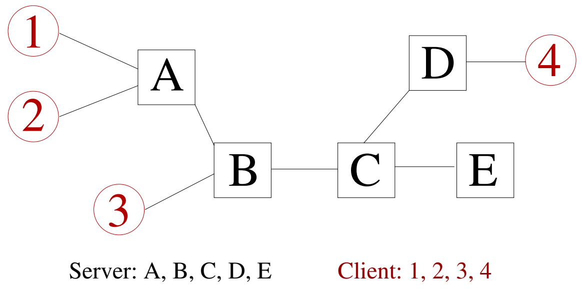


**Figure 1: Simple IRC-Network**

where `nick1` is the nickname of the sending client, and `nick2` the nickname of the receiving client. More details regarding the IRC protocol are specified in the IRC RFC [2].

Discussion channels are the core of life on IRC since they provide the abstraction for meeting other users and are available for a huge range of topics. Most IRC users participate in at least one such channel. The channel information needs to be kept consistent throughout the IRC network and includes user and server information. In addition it contains a list of optional channel attributes, e.g., the list of channel operators, who may remove people from a channel or change the channel attributes, or a 'private' attribute which ensures that the channel is not listed in the channel list.

Another use of IRC is as a medium to share data. To facilitate this it contains a separate sub-protocol for data transfer. Data transfers do not involve the IRC servers. Rather they happen directly between the two clients using TCP ports negotiated via special IRC protocol messages.

Each server of the IRC network has a list of servers that it may connect to. From these the servers identify a spanning tree among themselves in order to reduce the complexity of message routing between them. Figure 1 shows an example of a simple IRC-Network. For each server the list of servers as well as a list of patterns or the explicit list of clients and servers that are allowed to connect to it are maintained by an IRC operator. IRC operators are responsible for the administration of the IRC network. Furthermore an operator can override actions of channel operators and, say, remove a certain user from the network.

### 2.2 Web-chat

There are two main drawbacks of the widespread use of IRC by the general population: the main user interface being tty-based and a lack of understanding of the operation of the IRC system. The first can be overcome by using a known user interface. The second can be tackled by reducing the complexity. These two concepts seem to be the reason why Web-chats are so popular. They are much simpler to use, relying on a Web browser as user interface, and they have a much simpler structure than IRC, as they usually consist of only a single server. Although this implies that all clients, which want to participate in a specific channel, have to connect to the same server, this is conceptually simpler.

However in sharp contrast to IRC there are a large number of disparate Web-chat systems, each based on a different protocols using a wide range of port numbers. These systems and protocols do not only vary widely but are usually not well documented.

We distinguish three classes of chat-systems: the **HTML-**

**Web-chat** systems use the browser as user interface and HTTP as underlying application protocol; the **applet-Web-chat** systems use an applet window as user interface which is downloaded to the client via HTTP and a custom application protocol; the **applet-IRC-chat** systems use a Java or Javascript applet as front-end to a well known application protocol, IRC.

To gain some insights into how a typical Web-chat system operates we sketch a simple one: In our example the user output of the chat room (or channel) is displayed in a never ending HTML Web page, called the chat window. The HTML page only ends if the user leaves the chat system, e.g., by logging off or requesting another Web page. At the beginning of the chat session the user has to fill out a login form to access the chat page. Usually one of the fields of the login form contains the nickname that the user plans to use during this session. In order to ensure that each user will receive the chat output he desires most systems use session IDs. A session ID is some number or string derived from the login process that is usually transmitted with each request and response using cookies or the URL itself. Users usually enter input using Web forms relying on this session ID. Each line of input will be transmitted to all other users in the same chat room (or on the same channel) using their respective chat window. Apart from this basic functionality there usually are a number other functional elements that can be invoked using buttons or similar functional elements. These can be as varied as changing the text color, changing chat rooms, sending private messages to other users, and articulating various emotions. Usually these functional elements are also realized by using forms and the same session ID as for the main chat room.

Instead of using a never ending HTML Web page some systems use an applet window that is typically implemented using JAVA or a similar programming language. Figure 2 shows an example of such a chat system. You can clearly recognize the functional elements such as the window displaying the chat messages, the place for entering messages, the buttons for sending messages and leaving the chat room, and some additional features such as user and room lists.

Applet-IRC-chat systems only alter the user interface but continue to use IRC as their underlying application protocol. The functionality and the appearance of such systems are comparable to those of the other Web-chat systems discussed above. The main difference is that the client converts each message that a user enters in the applet-window into an IRC message and each IRC message that it receives into a form so that it can be presented in the corresponding applet window. Therefore in effect such systems supply a graphical user interface for IRC. From a networking perspective applet-IRC-chat systems are IRC systems. From an application perspective they are applet-chats.

From a user's perspective Web-chat does not differ substantially from a IRC client with a Web interface. The main difference between IRC and Web-chat appears to be a social one: our experience suggests that, on average IRC networks are usually used by the technically more experienced audience while Web-chats are used by the technically less experienced clientele. We also suspect that IRC users are usually older and they often tend to use IRC in parallel to other activities. On the technical side we observe that IRC networks impose a lesser load on a single server due to the spanning-tree protocol. For example, each of the three big IRC-networks had an average usage of more than 90,000 simultaneous users in the period starting on 06/28/2003 and ending on 07/28/2003 [6]. We would be surprised to find a Web server based system using typical hardware that is able to handle such a load.

## 2.3 Other chat systems

There are a number of other services, which enable interactive communication. Some of the most popular ones seem to be instant messenger applications such as ICQ or AOL and Microsoft's instant messenger services (AIM, MIM) or the Yahoo messenger service `http://messenger.yahoo.com`. These systems are stand-alone client applications which have to be explicitly downloaded and installed on the corresponding hardware. In general instant messenger services resemble applet-Web-chat systems in that the purpose is to send and receive messages with little delay. One of the differences is that only some support channels. On the other hand they add certain functionality, e.g., a user can specify a list of friends (buddy list) to the server of the system and is then notified when one of these friends logs into the system. We expect that communication behavior in these networks is comparable to that in chat systems as long as they support channels. More information about, e.g., ICQ can be found in [3].

## 3. METHODOLOGY

The goal of this section is to identify components of chat traffic that will help us separate chat traffic from other Internet traffic.

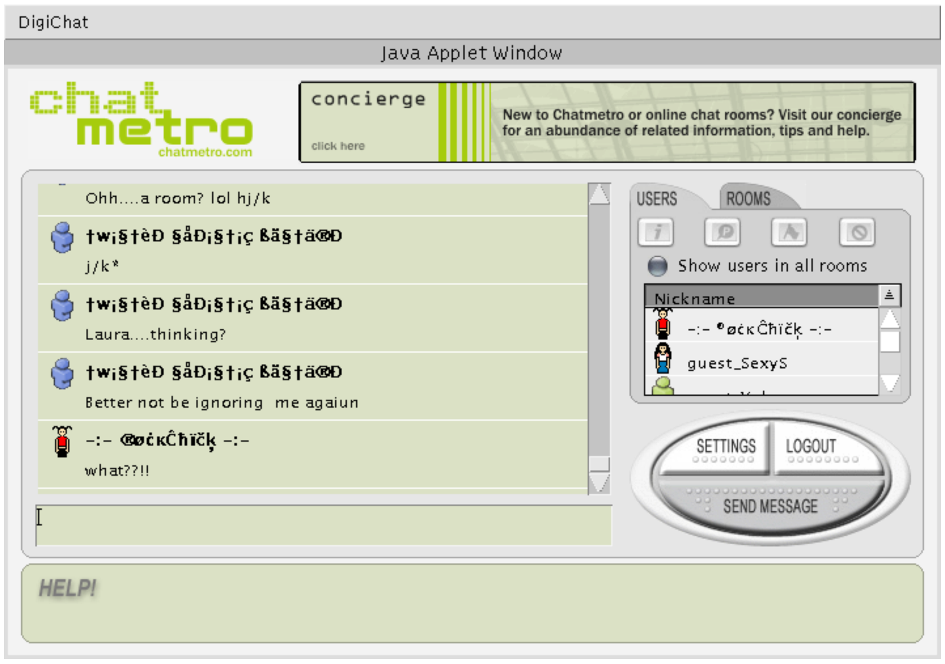
## 3.1 Identifying chat traffic

As discussed in Section 2, the important differences between chat systems and other applications are that chat neither uses a well-defined port nor a well-defined protocol. Rather there exists a wide spectrum of protocols: the well-documented IRC protocol used by IRC-networks or their applet-based user interface equivalents; custom application protocols used by HTML Web-chat systems running on top of HTTP; and custom application protocols used by applet-Web-chat systems. Note that HTML chat systems typically use the port reserved for HTTP (port 80) and therefore it is rather difficult to differentiate Web-chat traffic from other Web-traffic. These application properties make the characterization of chat traffic akin to the proverbial search for a needle in a haystack. Accordingly our approach for capturing chat traffic is to first collect all Internet traffic that could possibly correspond to chat traffic and then split this traffic into chat traffic and non-chat traffic. The attentive reader may find some parallels between this work and work on separating attack traffic from benign traffic in intrusion detection systems, e.g., [7], [8] and [9].

To separate chat traffic from other traffic we can take advantage of the fact that IRC is well specified and usually uses a well known port. Therefore IRC traffic is easy to capture. Relying on a characterization of IRC we identify some basic properties of chat traffic. Combining these properties with our general understanding of Web-chat systems enables us to define a set of filter heuristics.

## 3.2 IRC-derived chat properties

Our experience with IRC systems confirms the common use of the assigned well-known port 6667 for control and



New to Chatmetro or online chat rooms? Visit our concierge for an abundance of related information, tips and help.

Ohh....a room? lol hj/k

†wi§†èÐ §åÐi§†iç ßå§†ä®Ð
j/k*

†wi§†èÐ §åÐi§†iç ßå§†ä®Ð
Laura....thinking?

†wi§†èÐ §åÐi§†iç ßå§†ä®Ð
Better not be ignoring me agaiun

-:- ®øċкĈħïċķ -:-
what??!!

-:- ®øċкĈħïċķ -:-
guest_SexyS

**Figure 2: Applet-Web-chat (chatmetro.com)**

chat messages. Furthermore it is unlikely that an arbitrary client system uses this port as their source port since most systems start at boot with port 1024 and use the port numbers incrementally. Therefore a sizable subset of IRC traffic can be captured with the help of a packet monitor by simply capturing all TCP traffic involving port 6667. As file transfers using the IRC software rely on separate TCP connections between the participating clients we only capture IRC messages and IRC control messages.

One of the striking differences between IRC traffic and, say, general TCP traffic, is that IRC's packet size distribution is dominated by small packets. As an illustration, Figure 9 in Section 6 shows the packet size distribution for datasets IRC1 and TCP. (For details regarding the datasets see Section 4). Of course this does not imply that all chat systems will have exactly the same packet size distribution as IRC. Rather we expect some differences, e.g., some systems build on top of HTTP have to transmit meta information and/or display properties such as fonts, colors, or HTML tags. Nevertheless we expect small packets to dominate the packet size distributions of most chat systems since people seldom type more than a few hundred bytes at a time before sending the message to the channel. Furthermore since chat captures human interaction in the form of a conversation one can expect that most IRC sessions last more than a few minutes.

Another significant difference between IRC and other application protocols is that IRC sends keep-alive messages, PINGs and PONGs, at regular intervals (about every 2 minutes). This limits the maximum packet interarrival time of packets associated with the same TCP connection. Actually, keep-alive messages are partly but not fully responsible for the dominance of small packet sizes. Keep-alive mechanisms are popular since they provide a straightforward error recovery mechanism for broken TCP connections. However, not all chat-systems use keep-alives. Some Web systems are relying on a purely timeout-based system without keep-alives. Subsequently such systems do not tolerate long user idle times.

The receiver of a chat message can be identified in several ways. For example all messages may be transmitted via a single TCP connection to the server which is in turn associated with a recipient. Alternatively each message may explicitly list the recipients of the private message or the channel name, as it is done in IRC. Yet another alternative uses the concept of session ID's. Each session is associated with a certain chat room or recipient and each message includes the session ID. Note that different users on the same channel always have different session ID's.

One of the main features of IRC is the use of a distributed infrastructure of servers. Users participating in a channel can be connected to different servers. While some chat systems such as Gale might use a distributed infrastructure, we did not observe a Web-chat system that used a distributed infrastructure in a manner similar to IRC.

## 3.3 Characteristic properties of Web-chat systems

In Section 2 we distinguish three kinds of Web-chat systems: HTML-Web-chat, applet-Web-chat, and applet-IRC-chat. While the traffic of the latter cannot be distinguished from IRC traffic the other two can be. We discuss some of their specific properties below.

**HTML-Web-chat:** Using HTTP as the transport protocol imposes several problems for a chat system designer: Web pages are often cached and HTTP is, by design, stateless. In order to ensure that no stale chat messages are distributed, appropriate cache-control-headers have to be used. We observed that many chat systems use `Pragma: no-cache` and/or `Cache-control: no-store` and/or `Cache-control: no-cache`. Most chat systems need some state information for session and/or channel identification. Adding state to

HTTP can be accomplished in several ways, e.g., using cookies or appending the ID to the URL. Another HTTP header that contains relevant information is the `Server:.` header. It can be used to identify popular types of chat systems, e.g., `Server: JH-CHAT/1.0`.

So far we have discussed criteria that increase the likelihood that a certain HTTP request/response pair belongs to a chat session. Now we are going to identify criteria that indicate that it is unlikely that a HTTP request/response pair belongs to a chat session. In cases where HTTP is used for transaction it is important that no cached data is used without revalidation. The corresponding header is: `Cache-Control: Must-revalidate`. Use of this header ensures that cached data that is redisplayed is still valid. Since the assumption in chat systems is that the chat room conversation is active and therefore new information will be present, it is more sensible to prohibit caching than to enforce revalidation. Indeed we have not seen this header in use in chat systems. Another commonly used cache control header is `Cache-Control: Private` which is used to increase the privacy of the data. This is another criterion that is not appropriate for simple HTML-Web-chat systems, and we have not seen it in use.

All of the criteria discussed above arise from the use of HTTP as transport protocol. HTML offers additional capabilities: Many chat systems use scripting languages such as Javascript to implement various auxiliary functions such as self refreshing user lists or for changing text attributes (such as size and color). Some systems provide, in addition to the main HTML page window, other applet windows for, e.g., facilitating private conversations, or displaying user lists. Like most applet windows, these are realized in a programming language such as Java. Offering private conversations seems to be a desirable feature and is often realized by creating so-called 'Séparées' (private chat rooms) realized for instance in Javascript via a separate applet window. Other users can be invited to a Séparée for undisturbed discussion.

**Applet-Web-chat:** The most popular programming language for programming applet windows is Java. Accordingly a useful criterion for applet-Web-chat systems is that a Java file has been accessed by the user. Once a user of a applet-Web-chat system has chosen a channel or the name of the user with whom he wants to have a private conversation, a new Java applet window will be opened or an existing one will be reused on the client side. In case of a private conversation a separate window will be opened at both ends so that the communication partner may exchange messages within a private environment.

Yet by itself the fact that a user is downloading a Java applet is rather meaningless. One needs more context. Here we can take advantage of human nature. It is natural that we call a file or script or page that is associated with chat traffic "xxxchatyyy" where "xxx" and "yyy" can be arbitrary strings. Furthermore even if the word 'chat' is not contained in the name it is most likely somewhere in the path. After all people like to use descriptive names. Therefore our last criterion is the use of the word 'chat' in any of the packets. Note that this last criterion is only used in the context of applet-chats and only as an additional criterion.

**Properties of chat systems:** The above discussion has identified various criteria, summarized in Table 1, for identifying chat traffic. The criteria 'IRC' and 'instant messenger', which are not described above, refer to the respective types of applications. While it is easy to imagine how to use the packet size property in a filter it is hard to see how one could automatically recognize a Séparée. Accordingly we partition the properties into two groups: those that are suited for automatic filtering of chat traffic from other traffic and those that may help a human separate chat traffic from other traffic. The latter properties will therefore be used as the basis of our validation.

| Property | Collection | Validation |
|---|---|---|
| packet size | ● | |
| cache-control-headers | ● | |
| Java | ● | ● |
| scripting languages | | ● |
| session ID | | ● |
| séparée | | ● |
| applet window | | ● |
| IRC | | ● |
| instant messenger | | ● |

**Table 1: Properties of chat systems**

## 3.4 Strategy for extracting chat traffic

Based on the properties listed in Table 1, we now devise a heuristic that, starting from a very broad criterion which should include almost all chat traffic but also contains a large amount of other traffic, identifies traffic that cannot be chat traffic, weeds it out, and then iterates this process several times. After some number of iterations some traffic will remain. This traffic satisfies a number of properties typical for chat traffic. As a final step we explicitly remove traffic belonging to applications who's properties are similar to those of chat traffic. The traffic that now remains is assumed to be chat traffic. Naturally those steps that have the ability to eliminate a lot of non-chat traffic and can be implemented efficiently should be applied early in the filtering process. Other filters that need more processing power, more preprocessing, or may not eliminate as much of the traffic should be applied later in the process.

The first step is to identify an appropriate packet filter that allows us to capture all chat traffic. The next step preprocesses the data: we associate packets into flows, which is necessary since most criteria apply to streams of packets rather than individual packets. Each of the following filters decides if the packets of a flow correspond to a chat session or if they do not. This involves a series of filters that gradually remove flows that are unlikely to correspond to chat traffic.

We suggest to use a packet filter that accepts a broad spectrum of packets. The only ports we eliminate are those for which we can almost surely assume that they do not contain chat traffic. These are all privileged ports (ports $<$ 1024) except for the HTTP port (port 80) and ports such as the Gnutella port 6346. We specifically choose to eliminate Gnutella traffic to reduce the amount of data that needs to be inspected. This should not affect any chat traffic since no known chat system uses the Gnutella port. The rest of the unprivileged ports were not excluded since they did not contribute an excessive amount of traffic.

The next step groups packets into flows in a similar way as done by Claffy et al. [10] and Feldmann et al. [11]. Our method differs from the ones cited above in that we consider bidirectional instead of unidirectional flows. Furthermore we suggest to use different timeout values for packets with TCP flags, SYN, FIN, and RST. These flags usually indicate the start or end of flows and therefore offer additional information about the start and end of a connection. In contrast to intrusion detection systems [8] or full protocol analyzer, such as [12], we do not reconstruct the TCP data stream. Rather we filter on a packet by packet basis. Our motivation is twofold: on the one side it is unlikely that the content we are considering, e.g., a HTTP header line or the word 'chat' or the URL of a GET request, is split between two TCP packets; on the other hand the effort and processing power necessary for extracting the payload, reordering the packets, and restiching the content of a flow are quite significant. We acknowledge the drawbacks of these simplifications and realize that in general the quality of the resulting data may not suffice. However, for chat traffic it suffices.

Once the packets are grouped into flows we mark and filter them according to the above criteria. The filter that has the potential of discarding the largest number of flows is applied first. In our case it is the one that checks the packet sizes. Figure 9 shows the packet size distribution for IRC and Web-chat traffic. We see that while small packets dominate IRC and Web-chat traffic, large packets do occur as well. Accordingly the filter cannot be absolute. Rather it has to consider the distribution. We interpret "dominating" to mean that 50% of the packets have a size that is less than some threshold, 300 bytes. As stated we must have access to all packets of a connection in order to apply this criterion. This implies that we have to keep statistics for all active connections. For high-speed links this exceeds the available computational capabilities and memory. Therefore we decided to focus on the start of each flow. Our analysis of Web-chat traffic[1], shows that, i.e., the overall percentage of packets for our trace environment with less than 300 bytes is 93%. If we only consider the first 40, 60, and 80 packets the rate is 90%, 92% and 94%, respectively, indicating that larger packets are more frequent at the beginning of the connection than towards the end of the connection. The need to transfer meta information causes bigger packets to be sent at the start of the connection. Meta information includes, e.g., formatting instructions for the Web page and user lists. Our experiments show that the somewhat arbitrary value of 60 packets seems to result in an appropriate tradeoff between accuracy and performance. Flows that satisfy this filter are marked as candidates for chat traffic. The others are candidates for removal.

But before removing such flows we need to consider our criteria for identifying applet-Web-chat systems: besides the existence of a flow which is dominated by small packets we require the download of a Java applet or the use of the word "chat" on another connection not too long ago. We decided to interpret "too long ago" as meaning 5 minutes. The last two criteria need to be checked for all flows. Since applets are typically implemented in Java the corresponding criterion is the download of files ending with `.jar` (later called *JAR files*) from the same server. Accordingly we mark every flow with a packet that contains a HTTP GET request for a JAR file as an applet-flow. In a similar fashion we mark flows with at least one packet that contains the word "chat" as a chat-word flow. Note that this does not imply that this is Web-chat traffic. Rather this kind of flow is a prerequisite for identifying another flow as chat traffic. Applet-flows and chat-word flows are never eliminated since they are needed for later cross-reference.

If a flow is neither dominated by small packets nor an applet-flow nor a chat-word flow, it is now eliminated. As expected this filter in itself is not sufficient for identifying chat traffic. There are quite a number of other applications, such as news tickers, streaming audio sessions, online games, FTP control connections, peer-to-peer traffic, that generate small packets. Therefore we need to apply specific filters for each class of chat-system.

In order to identify HTML-Web-chat systems we take advantage of the presence or absence of HTTP headers. The first set of criteria is the presence of a suitable cache-control header or known system identification header. At the same time we verify that none of the unsuitable cache-control header is present. Furthermore we check that the requested object is a HTML page and not an image. Images can be recognized by the header `Content-Type: image/*` or if the name of the requested object ends in ".gif", ".jpg", ".jpeg" or ".png". The result of this filter is a list of candidate HTML-Web-chat flows.

In order to identify applet-Web-chat traffic we look for correlations between flows. In order for a flow to be a candidate it has to be dominated by small packets and it must have been preceded by an applet-flow or a chat-word flow. We furthermore impose the restriction that the two flows have to involve the same client/server pair. This restriction is somewhat stringent and unrealistic. It should be relaxed to flows between the same client and related servers. But since its hard to define and even harder to identify related servers we proceed using the restrictive approach. Since the chat-word flow criterion is a rather weak and general criterion we require that the time difference between the use of the word "chat" and the start of the new connection is not too big, i.e., less than five minutes.

This methodology, see Section 5, is able to capture most Web-chat traffic and reduce the trace size drastically. However, among the flows classified as Web-chat there are still a number of non Web-chat flows. Some of these are can be assigned to a number of known applications with similar characteristics as Web-chat, and so are eliminated using specific rules for the respective application. A list of these applications can be found in Table 2. Other flows are removed using some additional structural filters.

Some of the applications listed in Table 2 use well known port numbers. Among them are emule, eDonkey, soulseek, X11, IP telephony, Napster and HTTPS. The port of Gnutella is already filtered above, but we list Gnutella in Table 2 as it does contain a significant amount of small packets. Other applications, e.g., MMS, RPM and PPCP, identify themselves by frequently using the corresponding abbreviations in the protocol itself. Therefore connections which contain more than a small number of such abbreviations are eliminated. Yet another class of applications contain "keywords", e.g., RTSP contains a Content-Type field, which typically contains the word 'video'. While some SMTP-, SSH-, and FTP-servers listen on high port numbers such protocols are easy to identify using keywords such as `HELO`, `From:`, `To:` and `Subject:` somewhere in the flow, or for `SSH` or `FTP-Server` at the beginning of the connection.

[1]The traces are described in Section 4 and the analysis is discussed in Section 6.

| | | | |
|---|---|---|---|
| emule | SMTP | HTTPS | X11 |
| eDonkey | ESMTP | PPCP | IP telephony |
| soulseek | RTSP | RPM | MMS |
| Napster | FTP | Gnutella | |

**Table 2: Other protocols with small packet sizes**

After applying these filters we are left with an almost clean set of Web-chat traces. The remaining problematic connections are again filtered with structural rules. Among them is a filter that checks the data rate of the connection. Web-chat systems have a fairly low data rate while HTTP connections that use short packets, e.g., because of using non-buffered I/O, do not. Another filter checks that the connection lasts for at least 30 seconds. The assumption is that short connections do not provide enough time for a conversation and therefore are unlikely to be chat connections. Another assumption is that a chat contains interactions. Therefore each direction of the connection should transmit some minimum of data. In our case we check that each direction transmits at least 10 bytes. Any marked flows are eliminated from the chat traffic candidates.

## 4. TRACE COLLECTION AND DATASETS

The methodology described above has been used to collect chat traffic at the University of Saarland. In this section we first discuss how we resolve the hardware resource problems and then present some details about the captured traces.

### 4.1 Partitioning of resources

Hardware, both in terms of memory as well as CPU resources can be a bottleneck in realizing the methodology discussed above. Just consider that one wants to collect at least one week's worth of chat data to be able to study daily and weekday/weekend profiles. Since our methodology implies that most of the Internet traffic has to be captured first and then filtered according to the criteria, the raw data is quite sizable. At the time of the trace collection one week of traffic amounted to about 960 GB of raw data.

The data was captured using `tcpdump` [13] on a dual processor PC, running FreeBSD, with two 1200 MHz AMD processors and 183 GB hard disk capacity. This machine is called the capture machine. In addition we had access to a compute server with eight 650 MHz Intel III processors and 278 GB hard disk capacity. Since the amount of captured data was clearly larger than the available disk capacity and since the resource consumption of other users had to be taken into consideration, it was necessary to process the data in parallel with capturing of the data.

Initially we only used the capturing machine for both capturing and processing. Unfortunately we found that the load due to filtering causes the CPU to become overloaded. When the CPU is overloaded it is unable to service the interrupts for packet capture at the required frequency. Since it is unacceptable to loose a significant number of packets this approach was not successful.

Therefore we decided to transfer the collected traces during the trace collection from the capture machine to the compute server. To enable parallel processing the recording machine split the traces in slices of 100 MB. Note that it takes about 40 seconds to fill a 100 MB slice during typical day time hours.

The various filters of Section 3.4 were implemented in Perl on the compute server. We use Perl in spite of the performance problems since Perl is quite efficient for string oriented tasks and the code is easily revisable.

As the software is not fast enough to process the incoming data in a single step in real time, we resort to a four-step approach during the trace collection and an off-line cleaning step. The first step uses simplified versions of the three simplest filters to mark flows: packet size filter, checks for applet-flows as well as chat-word flows. We choose these since they do not include any expensive pattern matches. Unmarked flows are removed in the second step by eliminating all packets of non-marked flows. The third step marks flows using full versions of the above criteria and the remaining criteria. The last step again eliminates non-marked flows. The last set of filters is applied off-line and comprises of those structural filters that do not affect a lot of traffic, those that need care in terms of choosing the appropriate parameters, and finally the application specific filters. Applying these filters off-line offers many more possibilities for experimentation.

### 4.2 Traces

In this fashion we captured Web-chat traces starting at 12/13/2002, 15:46 MET and ending at 12/21/2002, 15:46 MET from the Internet connection of the Saarland University with the help of a monitoring port that duplicates all packets to our packet monitor. The initial amount of data that was captured is 950 GB. This is before filtering. Interim results amount to 1.2 GB of packet data while the final trace is reduced to 238 MB. The raw trace consists of more than 5 billion packets $(5,150,603,617)$. According to `tcpdump` $1,641,974$ of these are dropped by the packet filter. This results in a drop rate of less than 0.032%. We refer to this dataset as WEBCHAT1. In the end our methodology identifies $3,440$ as Web-chat connections.

The trace IRC1 was captured by extracting all IRC traffic (port 6667) from an interim result that lead to WEBCHAT1. Its size is 192 MB and it consists of slightly more than 4 million packets $(4,075,498)$ in 633 connections. This trace has the same drop rate as WEBCHAT1.

We collected a second Web-chat trace, called WEBCHAT2, starting at 12/4/2002, 13:37 and ending at 12/8/2002, 14:13. This trace was reduced to a size of 350 MB. Its drop rate is similar to the one for WEBCHAT1.

At the same time we used a separate `tcpdump` process to only monitor traffic to a specific set of servers, see Section 5. The resulting trace is called SELCHAT and the data initially amounted to $3,638,427$ packets with no lost packets according to `tcpdump`. The subset of SELCHAT that corresponds to successful connections that last longer than 30 seconds is refered to as SELCHAT*.

To be able to compare the packet size distributions of chat traffic to TCP traffic we use a trace called TCP that start at 03/05/2002, 21:56 and ends at 03/06/2002, 11:29. This trace consists of 91 GB in about 178 million packets. Of these $27,307$ packets were lost, which results in a drop rate of 0.015%.

## 5. VALIDATION

There are two aspects to validating our approach with respect to the goal of identifying all chat traffic. The first question is can our methodology identify all chat traffic? The second question is does this methodology classify non-chat traffic as chat traffic?

We tackle these problems using two common evaluation measures from information retrieval: *recall* which measures the ability of a system to present all relevant items and *precision* which measures the ability of a system to present only relevant items, for more details see, e.g., [14].

### 5.1 Identifying chat traffic – Recall

We start by adapting the definition of recall to our context: A relevant item is a Web-chat connection; To present an item corresponds to locating Web-chat traffic. Accordingly we can define recall $r$ in the following manner:

$$r = \frac{|REL \bigcap FOUND|}{|REL|}$$

where $REL$ stands for the number of Web-chat flows in a trace and $FOUND$ stands for the set of Web-chat flows that our methodology actually finds.

The problem with this definition is that we do not have a trace for which we know the number of relevant Web-chat flows. Therefore we are going to bound the recall from below. Consider the following alternative definition of recall:

$$r = 1 - \frac{|MISSED|}{|REL|}$$

where $REL$ still stands for the number of Web-chat flows in a trace and $MISSED$ counts the number of Web-chat flows that were not located using our methodology:

$$MISSED = REL \backslash FOUND.$$

A lower bound for $REL$ together with an upper bound for $MISSED$ yields a lower bound for recall.

We proceed by considering two traces that are collected independently but during the same time period: The first trace, SELCHAT*, captures only traffic that is very likely to be Web-chat traffic. The second trace, WEBCHAT2, uses our methodology for separating Web-chat traffic from all other traffic.

We proceed with finding a lower bound for $REL$ by concentrating on likely Web-chat sessions within SELCHAT* and by performing a conservative estimate on the number of Web-chat sessions. This estimate is verified by manually checking these connections. To find an upper bound for $MISSED$ one needs to keep in mind that it is acceptable to present flows as Web-chat even if they are not. This later point is addressed in the precision metric not the recall metric. We start by applying our methodology to the WEBCHAT2. Then we need to identify those flows in SELCHAT* that were not labeled as Web-chat by deriving the labeling from WEBCHAT2. These flows are now examined manually to verify the classification. The number of Web-chat flows in this set provide us with an upper bound for $MISSED$. Together this yields a lower bound for recall.

**Bounding the number of Web-chat sessions:** To solve the problem of identifying a good source of Web-chat traffic we identify a number of hosts which are known to host chat-servers of different kinds of Web-chat systems. The Web site `webchat.de` [15] rates Web-chat systems as well as channels by the number of users. We choose the top 15 Web-chat systems and the servers that host the top five channels. In addition we include 15 hosts that we identified as operating Web-chat systems during our classification work.

We then collect all traffic from our university to these 35 hosts forming the trace SELCHAT. This trace contains quite a bit of non chat traffic. After all the monitored hosts do not only provide Web-chat services. Accordingly some traffic is due to the retrieval of images and other HTML files. In addition not all connection attempts to the servers succeed. In order to increase the likelihood of including only Web-chat traffic we, in trace SELCHAT*, only consider traffic of successful connections that last longer than 30 seconds. 30 seconds seems a reasonable lower bound for something that is related to human behavior. Humans are hardly able to complete any human-to-human interactions within such a small time-span. Even in our trace of known Web-chat servers within a sample of more than 450 connections that had a duration between 30 and 180 seconds, less than 50 correspond to Web-chat connections. On the other hand, of the more than 550 connections that had a duration greater than 180 seconds, more than 450 correspond to Web-chat connections. Keep in mind that typical times for downloading a Web object is rather short, well below 30 seconds [16]. Therefore one might speculate, at least for reasonable short connections, that the shorter a connection lasts the more unlikely it is to be a chat connection.

The trace SELCHAT consists of $28,586$ connections. 367 of these connections are unsuccessful which leaves us with $28,219$. Only $1,045$ of these have a duration greater than 30 seconds leaving us with slightly more than $1,000$ connections in SELCHAT*.

To derive a lower bound for the number of Web-chat sessions within SELCHAT* we consider the 483 connections that were classified as chat by our methodology. These were checked manually and all of them are indeed Web-chat connections which is encouraging. We also verified 44 additional connections. Therefore we can say that SELCHAT* contains at least 537 Web-chat connections.

**Bounding the number of misses:** The trace WEBCHAT2 was captured at the same time as the trace SELCHAT and contains 1514 Web-chat connections. Correlating the connections identified as Web-chat from WEBCHAT2 with the connections to the chat servers in SELCHAT* left us with a significant number of connections, 562, that we found in the SELCHAT* trace but were not identified in the WEBCHAT2 trace. On the other hand 483 connections were successfully identified. The remaining connections are to other Web-chat systems.

The large number of unidentified connection definitely warrants closer inspection. Manual inspection of the 562 connections showed that at most 44 of these are active chat sessions. While these chat-session involve nine different servers most connections are to three servers. Twelve connections are to an applet-Web-chat server which encrypts the chat data. Therefore the data does not contain the word 'chat' in any of its name or protocol parts. Furthermore the JAR-file is downloaded from another server. Twelve other connections are from another applet-Web-chat server which, while not encrypting its data, downloads the JAR-file most of the time from another server. Therefore our current methodology will miss some of these. The third bothersome server, which involves 14 connections, uses an applet-IRC-chat system. Unfortunately it frequently splits IRC messages across packet boundaries, which causes our packet based heuristics to fail. Furthermore most of the undetected connections only contain a small number of messages. Since the applet also transmits some additional data our methodology is not always able to identify these connections as chat connections.

The packet size criterion is the main cause for missing the six remaining shorter HTML-Web-chat connections. In the beginning of a chat connection the client and the server have to exchange meta information. This implies that at the start of a chat connection some number of larger packets have to be transferred. If the chat connection itself does not last very long then these initial packets dominate the packet size distribution. Other artifacts that we have observed are a lost SYN-packet in WEBCHAT2 which causes the correlation to fail and the repeated transmission of a commercial advertisement within the Web-chat page.

Overall we can conclude that the number of actual Web-chat connections that were missed is rather small, 44, of which 24 stem from applets, 14 from applet-IRC and 6 from HTML-chat. Given that we missed 44 out of at least 483+44 relevant Web-chat connections we can now calculate a lower bound for recall:

$$r = 1 - \frac{|MISSED|}{|REL|} > 91.7\%.$$

Intuitively, recall tells us the probability that a given chat connection is found by our method. Should one decide not to count the encrypted applet chat the recall improves to 93.7%.

### 5.2 Identifying non-chat traffic – Precision

We again start by adapting the definition of precision to our context: A relevant item is a Web-chat connection; To present an item corresponds to locating Web-chat traffic. Accordingly we can define recall $p$ in the following manner:

$$p = \frac{|REL \bigcap FOUND|}{|FOUND|}$$

where $REL$ stands for the number of Web-chat flows in a trace and $FOUND$ stands for the set of Web-chat flows that our methodology identifies .

We again face the problem that we do not have a good independent way, besides manual testing, for separating the relevant from the non relevant flows. Due to the number of flows > 1500 in traces like WEBCHAT2 performing manual checks becomes awkward. Therefore we approach this problem probabilistically and ask what is the expected precision:

$$E[p] = E[\frac{|REL \bigcap FOUND|}{|FOUND|}] = 1 - E[\frac{|WRONG|}{|FOUND|}]$$

where $WRONG$ stand for the set of flows that is classified as Web-chat flows even though they are not.

While we could just choose some number of flows at random, test them and determine the precision from these tests this does not do justice to the variety of different kinds of Web-chat systems. Therefore we use stratified sampling. We divide our population along the $n$ categories and criteria discussed in Sections 2 and 3 and summarized in Table 1 and estimate the precision for each category independently:

$$E[p] = 1 - \sum_{i=1}^{n} E[\frac{|WRONG_i|}{|FOUND|}] = 1 - \sum_{i=1}^{n} E[p_i] * \frac{|FOUND_i|}{|FOUND|}$$

$$= 1 - 1/|FOUND| * \sum_{i=1}^{n} E[p_i] * |FOUND_i|$$

where $p_i$ is the precision for class $i$ and $FOUND_i$ stands for the set of identified Web-chat flows within category $i$.

The partitioning of the flows is integrated into the Web-chat software. Once the connections are classified we randomly pick a subset within each class for testing. In order to limit the number of tests but achieve a reasonable coverage we, within each class, picked at least 10% of the flows. To ensure a reasonable number of checked connections we added further tests if the number was below five. When we found a significant failure rate we tested more connections to understand if the error rate was due to a systematic problem in the approach or due to strange flows.

The partitioning into categories occurs in two phases: In the first phase we distinguish HTML-Web-chat and applet-Web-chat systems. Then we divide the HTML-Web-chat systems according to their properties as follows: First we split off chats with séparées, then we group the rest according to the following features into feature classes: chats with session IDs (ID), chats with scripting languages (SL), and chats with characteristic HTTP-headers (HTTP). A feature class can be any combination (X+Y) of the above features X and Y. The group ID+HTTP is split one more time depending on whether the flow contains the word 'script' (SC). Since the feature SL dominates the feature ID we combine the groups SL+ID and SL back into the single group SL. The results are shown in Table 3. Overall we classified 465 HTML-Web-chat connections and tested 119. Only a small number of these failed. This yields an estimated precision of 93.1% for HTML-Web-chat systems.

The applet chat systems are partitioned in a similar fashion: IRC front-ends (IRC); Instant Messenger front-ends (IM); JAR files located on the same server and the first half of the first 30 packets are shorter than 20 bytes (JARlen); other connections with JAR files on the same server (JAR); no JAR file but a GET-request which includes the word 'chat' and more than half of the first 30 packets are shorter than 20 bytes (smallen); no JAR file but a GET-request which includes the word 'chat' but not smallen (chat), other chats (other). The results are summarized in Table 4. Overall we found 1,049 Applet-Web-chat connections and tested 272 of them. With 22 failed tests we derive an estimated precision metric of 93.1%.

Using a weighted average calculation based upon the estimated precisions yields an overall precision of 93.1%.

### 5.3 Separating chat from non-chat traffic

In this section we have analyzed how likely we are to actually locate chat traffic as well as how many additional connections we might classify as chat traffic. Combining the results we find that using our approach we can expect to locate about 91.7% of all real chat connections and that we expect that at least 93.1% of all connections we identify are indeed chat connections.

## 6. RESULTS

Contrary to other applications, including Web [16], telnet [17, 18], ftp [18, 19], etc., the statistical properties of chat traffic have, to the best of our knowledge, not yet been examined. Since chat systems are known to be habit forming [1] and since they have fairly limited bandwidth requirements

| | Séparée | ID+HTTP | ID+HTTP+SL | ID | ID+HTTP+SC | SL+HTTP | SL | HTTP | total |
|---|---|---|---|---|---|---|---|---|---|
| measured: | 21 | 107 | 9 | 40 | 30 | 3 | 229 | 26 | 465 |
| tested: | 5 | 27 | 9 | 17 | 12 | 3 | 20 | 26 | 119 |
| failed: | 0 | 1 | 4 | 6 | 0 | 2 | 0 | 8 | |
| E[precision] (%): | 100 | 96.30 | 55.56 | 64.71 | 100 | 33.33 | 100 | 69.23 | 93.10 |

**Table 3: Precision for HTML-Web-chat systems.**

| | IRC | JARlen | smallen | JAR | IM | chat | other | total |
|---|---|---|---|---|---|---|---|---|
| measured: | 34 | 254 | 240 | 70 | 255 | 98 | 98 | 1049 |
| tested: | 5 | 100 | 50 | 20 | 50 | 22 | 25 | 272 |
| failed: | 0 | 10 | 2 | 5 | 0 | 0 | 5 | |
| precision (in %): | 100 | 90 | 96 | 75 | 100 | 100 | 80 | 93.13 |

**Table 4: Precision for applet-Web-chat systems.**

such statistical properties may be relevant for sizing wireless networks. Furthermore it is always of interest to understand how the properties of one application differ from those of another. Among the metrics of interest to us are such basic properties as chat session duration and interarrival times as well as the activity within the chat connections.

Before discussing the results of our analysis we want to explicitly list some of the caveats of the underlying data and the analysis. First of all we only capture data at a single location, the exit point of an university campus network. Before generalizing the results it is necessary to verify the results using traces from several other locations in the Internet as well as different kinds of locations, e.g., at an Internet Service Provider (ISP) entry point. Second, the number of observed Web-chat connections even over a whole week is limited. Within each traced hour we only observe a small number of connections. Third our methodology of separating Web-chat traffic from non chat is not 100% accurate, see Section 5. Some connections ($< 8.3\%$) are missing from our analysis and some additional ones are included ($< 6.9\%$). To further reduce the uncertainty regarding if a connection is indeed a chat connection we focus for our analysis only on those connections that last for more than 30 seconds[2]. Therefore the following results represent an initial step but more work is needed for a good understanding of Web-chat properties.

In order for us to compare the characteristics of Web-chat to those of IRC we have to ensure that we are using similar abstractions. IRC uses a single TCP connection during the duration of a chat session, even if the user switches between multiple channels. This is not necessarily the case in Web-chat. Therefore we introduce the concept of a chat-session to capture the timespan that a user is chatting at the same server host. For IRC, a chat-session just corresponds to the duration of the IRC connection which equals the duration of the TCP connection to the server. This is not the case for Web-chat. For example in HTML-Web-chat if a user changes his channel the old Web page is closed and a new one is opened since the session ID is changed. This implies that more than two TCP connections are used. Accordingly we merge connections between the same client and the same server into sessions much in the same way as one usually merges packets to flows [10, 11]. The main difference is that we use a rather large timeout of one hour in order to account for users that are silent for extended time periods, e.g., over lunch. The advantage of using chat-sessions is that the results for IRC and Web-chat sessions are now comparable.

[2]See Section 5 for a more detailed discussion regarding the choice of 30 seconds.

For the analysis we collected a week long trace, WEBCHAT1, which contains 3,440 connections that are classified as Web-chat. These can be summarized into 1,530 Web-chat sessions. The corresponding IRC connections are the subset, IRC1 of the raw data of WEBCHAT1 that corresponds to IRC traffic. It contains 782 IRC connections. In terms of metrics of interest we first analyze the basic properties of chat-session that are considered relevant for traffic modeling such as session duration and interarrival times. Next we examine the dynamics within the chat connections. To this end we consider interarrival times of packets within chat connections as well as packet size distributions. Finally we examine the ratio of send vs. received bytes of chat-sessions in order to understand how data is flowing between participants.

## 6.1 History of traffic characteristics

Traditionally traffic modeling has its starting point in telephony and has been based on Markovian assumptions about traffic arrivals and exponential assumptions about holding times (e.g. [20, 21]). With the emergence of high-speed packet networks and their heterogeneous mix of services and applications this does not necessarily hold any longer, e.g., [22, 17, 23, 24, 25, 19, 26, 27, 28, 29, 30].

If events are recurring "at random points in time" or if the future lifetime has the same distribution as the current, then one might expect to observe an exponential distribution [31]. Furthermore if a large number of independent traffic streams with suitable regularity conditions are multiplexed the resulting stream approaches a Poisson process [20]. Therefore it is often assumed that human behavior at large leads to Poisson processes. If data is consistent with an exponential tail we expect to see a straight line for the tail end of the complementary cumulative distribution function (CCDF) on a log linear plot. On the other hand heavy-tailed behavior has been commonly observed in all kinds of aspects of Internet traffic, e.g., [19, 27, 28, 30, 11, 32, 33]. If data is consistent with a heavy-tailed distribution we expect to see a straight line for the tail end of the CCDF on a log log plot. Furthermore it has been shown that the superposition of ON/OFF sources converges to a self-similar process [27]. An alternative way of providing a structural explanation for generating asymptotically self-similar traffic is given by a

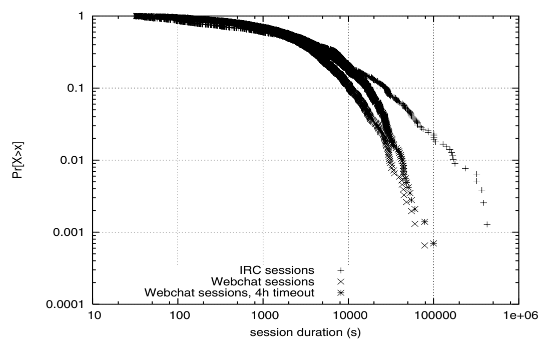


**Figure 3: CCDF of session durations (log log scale).**

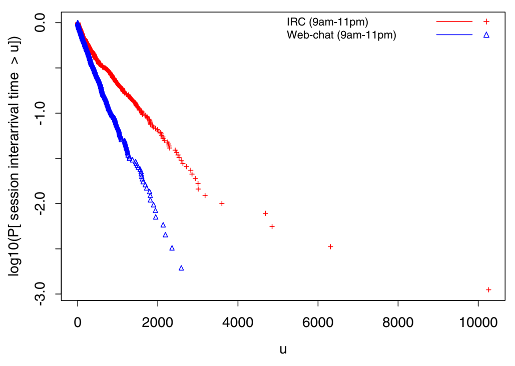


**Figure 4: CCDF of session interarrival times (log linear scale) during the day**

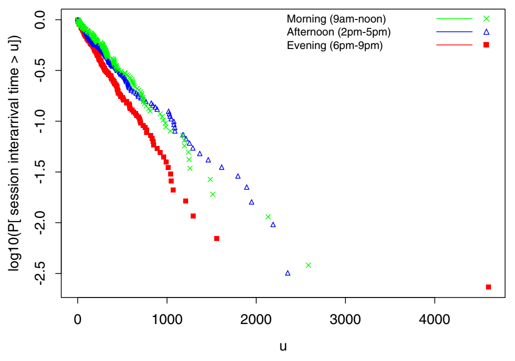


**Figure 5: CCDF of Web-chat session interarrival times (log linear scale) during the morning/afternoon/evening)**

construction due to Cox [34, 35] also known as immigration death process of $M/G/\infty$ queueing model. It assumes that sessions arrive according to a Poisson process, that the distribution of session lengths/sizes is heavy-tailed, and that packets/bytes are transmitted at a constant rate. Such behavior has been observed for the pre-Web days for FTP and Telnet [19]. For Web this approach does not suffice since, while the durations are heavy-tailed, the interarrival times of Web connections [30, 36] are not consistent with an exponential distribution. On the other hand it is still applicable for modem calls, who's arrival process is consistent with an exponential distribution, and Web connections within such modem calls. A more general approach, e.g., for Web traffic, is given by Kurtz's construction [37, 36]. How about chat traffic, are the session interarrival times consistent with an exponential distribution, are the session durations consistent with a heavy-tailed distribution, and what is happening within a session? These are the questions that we examine next.

## 6.2 Session durations

Figure 3 plots the CCDF of the duration of Web-chat and IRC sessions, respectively, on a log log scale. This plot immediately demonstrates that chat sessions tend to last a significant amount of time. Indeed the median for both IRC and Web-chat sessions is roughly half an hour. Overall IRC sessions seem to last longer than Web-chat sessions. While this could be an artifact of our use of an timeout of only one hour in the session definition we think that this is rather unlikely. Indeed the relatively small difference between the CCDF for the Web-chat sessions with one hour timeout vs. the one using a four hour timeout strongly supports this. We note that less than 1% of the Web-chat connections last for more than 8 hours. In contrast, 10% of the IRC sessions last longer than 8 hours, and 2% last longer than 100 000 seconds, more than a day. This suggests that there is a significant population of IRC users that continue their IRC session not just during the day but also throughout the night. Indeed, a few of the sessions were active during the whole trace collection period. The most likely explanation is that most of the Web-chat, but not all IRC, users log out of their system while sleeping.

Both IRC and Web-chat traffic include some rather long lasting sessions. Furthermore IRC session durations have a heavier tail than Web-chat ones. But we consider the current evidence as not sufficient to claim that session durations are consistent with heavy-tails or exponential tails. Both on the log log plot and on a log linear plot the curves are not straight enough over a sufficient range of time scales. Since some of the connections, especially the long ones, may be idle almost all of the time, the distribution should not be used to infer what the durations of actual chatting looks like (except as an upper bound).

## 6.3 Interarrival times of sessions

In this section we explore the frequencies with which users connect to chat servers. Accordingly we identify the starting time of each chat session, i. e., the time when a user connects to the chat server and derive the interarrival time of consecutive logins. In order to eliminate the significant activity shifts during the day and during the night we first disregard chat sessions started after 11pm and before 9am. Note that the session interarrival times during the day are significantly shorter than those during the night. Figure 4 shows the plot of the CCDF of the session interarrival times on a log linear scale. Both lines feature a more or less straight line. This is a first indication that interarrival times are consistent with an exponential distribution.

To further disregard the impact of time of day effects we consider three smaller subsets of Web-chat sessions: **morning** contains the interarrival times of those sessions that are started after 9am and before noon, **afternoon** consists of those that are started between 2pm and 5pm, while **evening** gathers those that are started between 6pm and 9pm. The

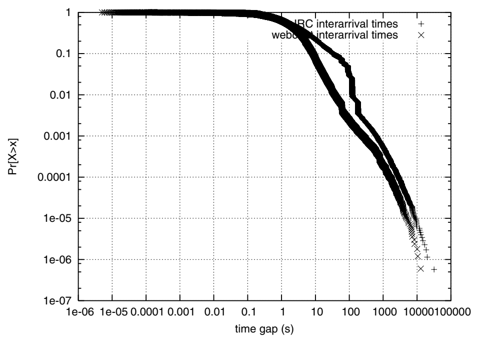


**Figure 7: CCDF of packet interarrival times for IRC with and without PINGs and Web-chat.**

CCDF's of each subset of session interarrival times, see Figure 5, again show a more or less straight line on a log linear scale. Similar observations hold for IRC sessions.

Just checking the interarrival times for consistency with exponentially distributed interarrivals does not suffice for a Poisson model. We furthermore have to check if the data is consistent with independent arrivals by checking for correlations in the data. Figure 6 plots the auto correlation function (ACF) for each of the three subsets of interarrival times. While there is some correlation it hardly ever exceeds the approximate 95% confidence limit lines added to the plot. Again we observe a similar behavior for IRC sessions.

## 6.4 Interarrival times of chat messages

Does the above observation with regards to consistency of interarrival times with an exponential distribution also hold for the interarrival times within the client-server communication. In order to investigate this aspect we inspect interarrival times of packets from the same session in the network. For this calculation we ignore all packets that do not contain any data such as pure acknowledgement packets. In this case packet interarrival times should correspond to message interarrival times under the assumption that most messages fit into a single packet. Extensive manual inspection of Web-chat connections during the validation phase, see Section 5, strengthens our belief that it is indeed the case that most messages fit into a single packet.

Special consideration has to be taken as many chat systems use keep-alives to check whether their clients are still active. Keep-alives can dominate interarrival time distributions. For an example in terms of IRC see Figure 7 which shows the CCDF of the packet interarrival times in IRC with and without PING messages. After eliminating the keep-alives the curves start to deviate at interarrival times greater than 120 seconds and we observe that the interarrival times span a huge range of values. Unfortunately it is not quite that simple to remove keep-alive messages from Web-chat. As their influence while visible around 60 seconds, is not as strong as for IRC, we did not eliminate these. This implies that we will observe a smaller number of larger interarrival times. Even after removing the keep-alive messages from IRC some periodic components seem to remain at about 100 and 120 seconds. One possible explanations for these are automatic bots and the ISON feature which generate messages in regular intervals. The ISON command allows the client to query the server if anyone from a list

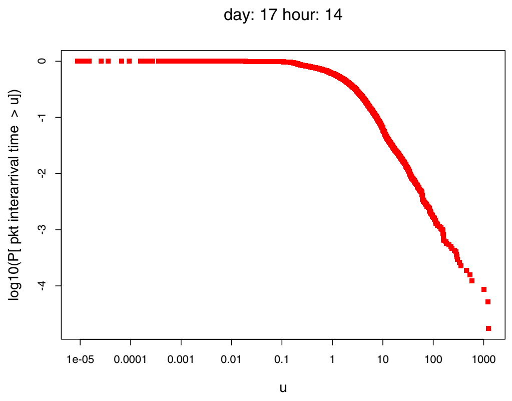


**Figure 8: CCDF of packet interarrival times during 2pm to 3pm on the 17th.**

of people is currently on-line. Note that some chat systems do not use any keep-alives since otherwise interarrival times greater than 120 would not be observable.

Typical interarrival times for Web-chat messages, see Figure 7, are between one and ten seconds while typical interarrival times for IRC messages are significantly higher. This does not imply that the number of messages in a Web-chat channel is higher than the number of messages in an IRC channel. Keep in mind that IRC sessions usually last longer than Web-chat sessions. On the other hand we note that larger gaps between packets do happen from time to time and correspond to user idle times, for example when they are away for lunch, etc.. We found interarrival times of up to four hours in Web-chats and up to ten hours in IRC.

The overall interarrival time distribution of messages is by no means consistent with an exponential distribution. Still there are many possible explanations among them that the superposition of various time varying exponential distributions [38] can create a distribution that is consistent with such a distribution. Accordingly we partition time into one hour time periods and investigate the interarrival distributions for each hour separately. Figure 8 shows an example for Web-chat. Overall we note that the interarrival time distributions are still not consistent with an exponential distribution. Rather they are consistent with a heavy-tailed distribution. On the other hand the impact of keep-alives for some of the periods is more dominant. Overall we note that our results indicate that the interarrival times of chat messages are not consistent with an exponential distribution which, e.g., is in line with the results from Paxson and Floyd [39] for telnet.

## 6.5 Packet sizes

Figure 9 (a) shows the CDF of the packet size distribution for TCP, Web-chat, and IRC traffic. For TCP this shows us the typical behavior: more than 39% of the packets are pure acknowledgements; more than 25% fully utilize the available MTU sizes, e.g., 1500, 576, 1420, 1492, 1480. This plot shows that, in a typical TCP connection the median packet size is about 113 bytes. Considering the packet size distributions for Web-chat and IRC traffic we already notice that the typical packet size appears to be much smaller. There hardly are any full packets.

In order to distinguish the contribution of the acknowledgements vs. partially filled packets to the small packets

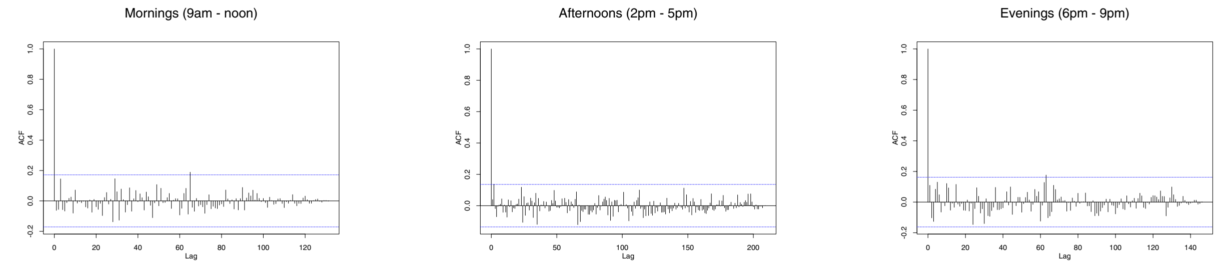


**Figure 6: ACF of session interarrival times for subsets morning/afternoon/evening**

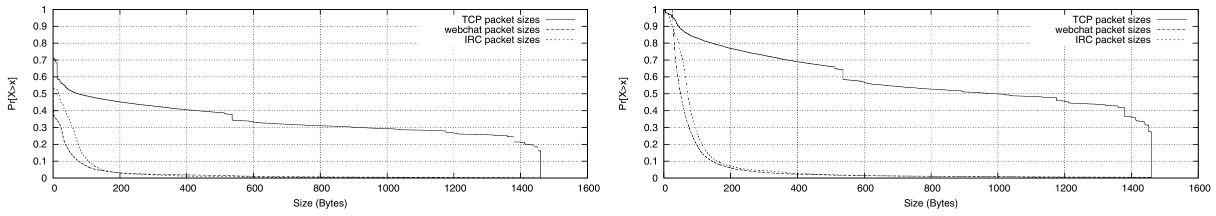


**Figure 9: Packet size distributions: (a) including acknowledgements (b) excluding acknowledgements**

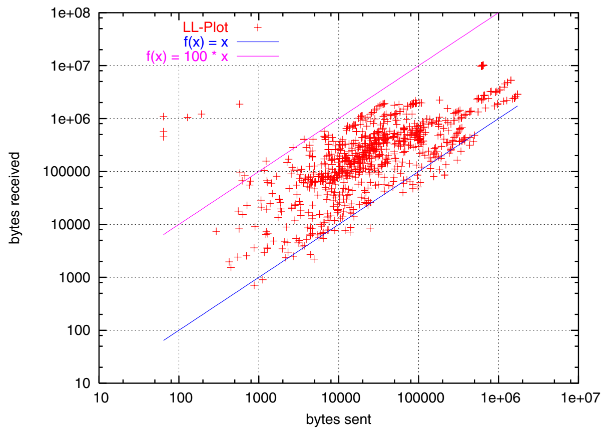


**Figure 10: Sent and received bytes (Web-chat)**

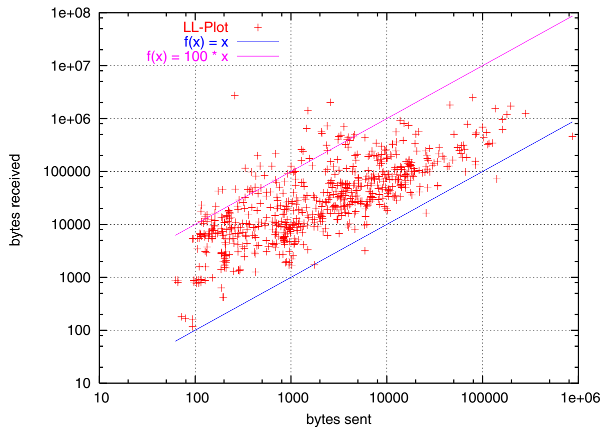


**Figure 11: Sent and received bytes (IRC)**

we recomputed the packet size distribution for TCP, Web-chat, and IRC after eliminating all packets that contain no data. The resulting distributions are shown in Figure 9 (b). Now it is apparent that the IRC and Web-chat distributions are dominated by small packets. Less than 8% of all chat packets have packet sizes of more than 300 bytes. Moreover 50% have a size of less than 100 bytes. On the other hand, more than 70% of all TCP packets contain more than 300 bytes. These specifics of the packet size distribution motivates our approach for separating chat traffic from other traffic starting from the packet size criterion.

## 6.6 Transmitted and received bytes per session

Next we examine the relation between sent and received packets in chat and IRC sessions. Figure 10 and 11 plot the number of bytes sent vs. the number of received bytes for each chat session and each IRC session, respectively. Both axes are scaled logarithmically. We note that in a typical session, a client receives about 10 times as much data as he sends. We added the functions $f(x) = x$ and $f(x) = 100 * x$ to Figure 10 and Figure 11 in order to help us gauge the spread of the values.

There are a few outliers: in 6 Web-chat connections a user receives more than a thousand times as much as he sends. These connections are games rather than chat. Moreover there are some Web-chat connections where the user sends more than he receives. This is surprising at first, since usually a server echoes everything a user sends. But once we consider that sending HTML-web-chat data may include a new Web connection for each line that the user is typing, this is not that impossible, especially if this user is more active than most of the other people in his channel.

For IRC we observe 8 cases where a user sends more than he receives. At first this seems strange given that the IRC server usually echos all communications except for private messages. But there is the ISON feature. With the ISON command the client sends a list of people to the server and the server replies with the subset of these that are currently on-line. Therefore these 8 cases are caused by a combination of users sending many PRIVMSG's and receiving few and using the ISON command with a sizable list of people. On the other hand there are many sessions where a user receives more than 100 times as much as he sends. Overall the most notable is again the wide range of different values.

## 7. SUMMARY

In this paper we develop a methodology for separating chat traffic from other Internet traffic and show that it can be rather successful. We find that our approach is expected to miss less than 8.3% of all existing chat connections and to correctly classify at least 93.1%. Given the wide range of ill-defined chat systems we consider this to be rather effective. Our analysis of the collected traces shows that session interarrival times are consistent with exponential distributions while packet interarrival times are not which is surprising given that both reflect human behavior.

## Acknowledgments

We would like to thank numerous colleagues for helping us access and understand the network measurement and configuration data used in this work and for valuable suggestions on how to improve the presentation of the material.

## 8. REFERENCES

[1] K. S. Young, "What makes the internet addictive: Potential explanations for pathological internet use," in *105th Annual Conference of the American Psychological Association*, 1997.

[2] J. Oikarinen and D. Reed, "Internet Relay Chat Protocol RFC 1495," 1993.

[3] ICQ Inc., *Introduction to ICQ*. `http://www.icq.com/products/whatisicq-ger.html`.

[4] AOL Inc., *AOL Instant Messenger*. `http://www.aim.com/index.adp`.

[5] D. Egnor, *Gale Home*. `http://www.gale.org/`.

[6] A. Gelhausen, "Charts of irc networks." Web Page, 2003. `http://irc.netsplit.de/networks/list1uma.html`, the copy we used is available for reference at `http://www.net.in.tum.de/~aw/list1uma.html`.

[7] Y. Zhang and V. Paxson, "Detecting backdoors," in *Proc. of 9th USENIX Security Symposium*, 2000.

[8] V. Paxson, "Bro: A system for detecting network intruders in real-time," in *Computer Networks*, 1999.

[9] M. Roesch, "Snort – lightweight intrusion detection for networks," in *USENIX LISA*, (Berkeley, CA), 1999.

[10] K. C. Claffy, H.-W. Braun, and G. C. Polyzos, "A parameterizable methodology for internet traffic flow profiling," *IEEE J. Selected Areas in Communications*, 1995.

[11] A. Feldmann, J. Rexford, and R. Caceres, "Efficient policies for carrying Web traffic over flow-switched networks," *IEEE/ACM Trans. Networking*, 1998.

[12] A. Feldmann, "Blt: Bi-layer tracing of HTTP and TCP/IP," in *Proc. WWW-9*, 2000.

[13] V. Jacobson, C. Leres, and S. McCanne. `tcpdump`,ftp://ftp.ee.lbl.gov, June 1989.

[14] "Common evaluation measures." The Tenth Text REtrieval Conference (TREC 2001) Appendices.

[15] "Webchat - das verzeichnis deutsprachiger chats." Web Page, 2002. `http://www.webchat.de`.

[16] B. Krishnamurthy and J. Rexford, *Web Protocols and Practice*. Addison-Wesley, 2001.

[17] P. Danzig, R. Caceres, D. Mitzel, and D. Estrin, "An empirical workload model for driving wide-area TCP/IP network simulations," *IEEE/ACM Trans. Networking*, 1992.

[18] P. Danzig and S. Jamin, "tcplib: A Library of TCP Internetwork Traffic Characteristics," tech. rep., USC, 1991.

[19] V. Paxson and S. Floyd, "Wide areatraffic: The failure of posson modeling," *IEEE/ACM Trans. Networking*, 1995.

[20] V. Frost and B. Melamed, "Traffic modeling for telecommunications networks," *IEEE Communication Magazine*, 1994.

[21] D. Jagerman, B. Melamed, and W. Willinger, "Stochastic modeling of traffic processes," in *Frontiers in Queueing: Models, Methods and Problems* (J. Dshalalow, ed.), CCR Press, 1996.

[22] R. Jain and S. Routhier, "Packet trains – measurements and a new model for computer network traffic," *IEEE J. Selected Areas in Communications*, September 1986.

[23] D. Duffy, A. Mcintosh, M. Rosenstein, and W. Willinger, "Statistical analysis of ccsn/ss7 traffic data from working ccs subnetworks," *IEEE J. Selected Areas in Communications*, 1994.

[24] W. Leland, M. Taqqu, W. Willinger, and D. Wilson, "On the self-similar nature of ethernet traffic (extended version)," *IEEE/ACM Trans. Networking*, 1994.

[25] V. Bolotin, "Modeling call holding time distributions for ccs network design and performance analysis," *IEEE J. Selected Areas in Communications*, 1994.

[26] W. Willinger, M. Taqqu, and A. Erramilli, "A bibliographical guide to self-similar traffic and performance modeling for modern high-speed networks," in *Stochastic Networks: Theory and Applications* (S. Z. F.P. Kelly and I. Ziedins, eds.), Oxford: Oxford University Press, 1996.

[27] W. Willinger, M. S. Taqqu, R. Sherman, and D. V. Wilson, "Self-similarity Through High-variability: Statistical Analysis of Ethernet LAN Traffic at the Source Level," *IEEE/ACM Trans. Networking*, 1997.

[28] M. E. Crovella and A. Bestavros, "Self-similarity in World Wide Web traffic: Evidence and possible causes," in *Proceedings of 1996 ACM SIGMETRICS Conference*, 1996.

[29] S. Resnick, "Heavy tail modeling and teletraffic data," *Anals of Statistics*, 1997.

[30] A. Feldmann, "Characteristics of TCP connection arrivals," in *Self-Similar Network Traffic And Performance Evaluation* (K. Park and W. Willinger, eds.), J. Wiley & Sons, Inc. 2000.

[31] N. Johnson, S. Kotz, and N. Balkrishnan, *Continuous Univariate Distributions, Volume 1*. Wiley, 1994.

[32] P. Barford, A. Bestavros, A. Bradley, and M. Crovella, "Changes in web client access patterns: Characteristics and caching implications," *World Wide Web, Special Issue on Characterization and Performance Evaluation*, 1999.

[33] A. Downey, "Evidence for long-tailed distributions in the internet," in *Internet Measurement Workshop*, 2001.

[34] D. Cox, "Long-range dependence: A review," in *Statistics: An Appraisal* (H. David and H. David, eds.), Ames, Iowa: Iowa State Universtiy Press, 1984.

[35] W. Willinger, V. Paxson, and M. Taqqu, "Self-similarity and heavy tails: Structural modeling of network traffic," in *A Practical Guide to Heavy Tails: Statistical Techniques for Analyzing Heavy Tailed Distributions* (R. Adler, R. Feldmann, and M. Taqqu, eds.), Boston: Birkhauser Verlag, 1997.

[36] A. Feldmann, A. C. Gilbert, W. Willinger, and T. G. Kurtz, "The Changing Nature of Network Traffic: Scaling Phenomena," *ACM Computer Communication Review*, 1998.

[37] T. Kurtz, "Limit theorems for workload input models," in *Stochastic Networks: Theory and Applications* (S. Z. F.P. Kelly and I. Ziedins, eds.), Oxford: Oxford University Press, 1996.

[38] A. Feldmann and W. Whitt, "Fitting mixtures of exponentials to long-tail distributions to analyze network performance models," *Performance Evaluation*, 1998.

[39] V. Paxson and S. Floyd, "Wide area traffic: the failure of Poisson modeling," *IEEE/ACM Trans. Networking*, 1995.