

# IRC-Client

IRC Group: Arne Bochem, Benjamin Maas, David Weiss

September 1, 2007

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Description . . . . .	2
1.2	Motivation . . . . .	2
<b>2</b>	<b>Background</b>	<b>2</b>
2.1	The IRC protocol . . . . .	2
2.1.1	Overview . . . . .	2
2.1.2	IRC messages . . . . .	3
2.1.3	Connecting to an IRC network . . . . .	4
2.1.4	Operators . . . . .	4
2.2	Filetransfer with DCC . . . . .	4
2.2.1	CTCP . . . . .	4
2.2.2	DCC . . . . .	5
2.3	XDCC bots . . . . .	5
<b>3</b>	<b>Implementation</b>	<b>6</b>
3.1	Overview . . . . .	6
3.2	IRC . . . . .	7
3.2.1	Required IRC messages . . . . .	8
3.2.2	Persistency . . . . .	8
3.3	Bot communication . . . . .	9
3.3.1	Data collection . . . . .	9
3.3.2	Measures against bots . . . . .	9
3.4	Statistics aggregation . . . . .	10
3.4.1	Robustness . . . . .	10
3.4.2	Bot recognition . . . . .	11
3.4.3	Event logging . . . . .	11
<b>4</b>	<b>Results</b>	<b>11</b>
4.1	Methodology . . . . .	11
4.2	Diagram notes . . . . .	13
4.3	Regions . . . . .	13
4.4	Offered and transferred data . . . . .	14
4.5	Record data rate . . . . .	14
4.6	Online time . . . . .	17
4.7	Operating systems . . . . .	18
4.8	Conclusion . . . . .	18

# 1 Introduction

## 1.1 Description

The purpose of this project was to implement an IRC (Internet Relay Chat) client, which is able to connect to IRC servers, join channels and fully automatically retrieve the IP addresses of, and other information about XDCC bots. The word “bot” stems from the word “robot”. In the context of IRC, a bot is an automatic client with some specific function or functions, which can range from greeting new users to offering files for download.

## 1.2 Motivation

XDCC bots are used to serve files (mostly software/movies etc.) and usually run on hacked computers. By collecting their IP addresses, we can gain some insight, as to which parts of the world are most oftenly targeted by botnet owners, who control a large number of compromised machines (“zombies” or “bots”). These machines are usually tied together in a “botnet”, and used for purposes like distributed denial of service (DDoS) attacks, in which the target is bombarded with useless data from the bots, to fill up its bandwidth, sending spam, or breaking into even more computers.

Since network security became more ubiquitous in the western hemisphere (USA, Europe), and copyright holders became more active to prevent unauthorized copying (collaboration with law enforcement etc.), it should prove interesting to see where these bots come from. The current assumption is, that they are equally distributed around the world, but it is quite possible that some areas of the world are targeted more than others. Some Asian nations have a high amount of home-users with very fast broadband connections, which should be ideal targets for botnet owners. One example of such a country is South Korea, which has one of leading broadband infrastructures in the world. [18]

Still, analyzing the origin of XDCC bots will only provide a partial view of the situation, since there will still be a significant number of bots that evade detection, simply because they are not used for offering files via IRC, but for other malicious purposes.

# 2 Background

## 2.1 The IRC protocol

This section will explain basic IRC concepts which are relevant for our IRC protocol implementation.

### 2.1.1 Overview

Internet Relay Chat (IRC) is an online chat system. The IRC protocol is described in RFC 1459 [7]. RFC 1459 has been updated by RFC 2810, 2811, 2812 and 2813 [7], but these documents have little practical relevance today. Protocols that allow IRC to be used for purposes other than text chat will be introduced in section 2.2.

Figure 1 gives an overview of the IRC architecture. Two IRC networks (IRC

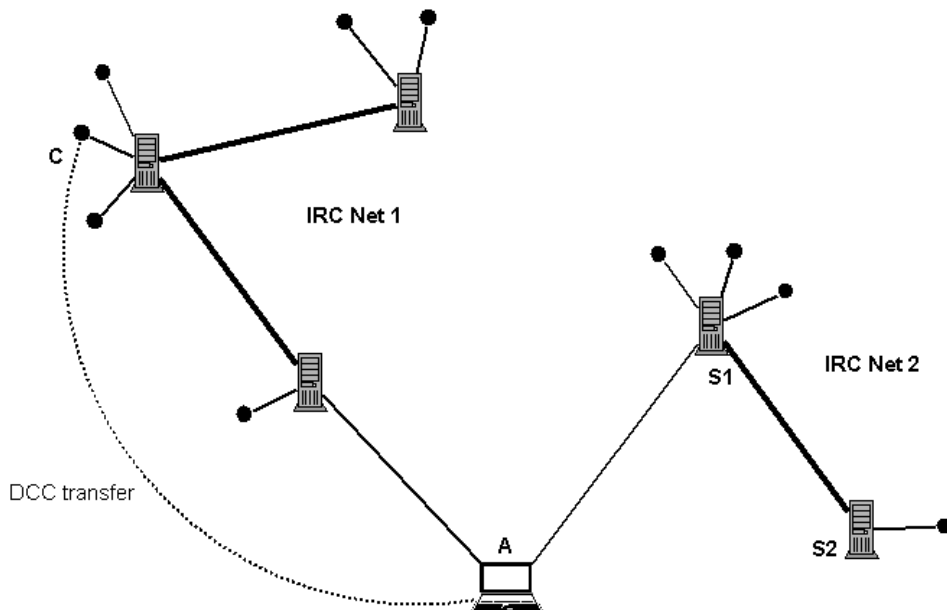


Figure 1: IRC architecture

Net 1 and IRC Net 2) are depicted; in reality there is a large number of independent IRC networks. Networks are organized as client-server systems. In figure 1 the servers  $S1$  and  $S2$  form an IRC network. Clients may be connected to several IRC networks simultaneously, such as client  $A$ . The links represent TCP connections. The servers' main task is to provide for message exchange between the clients. For example, if client  $A$  wants to send a (private) message to a client connected to  $S2$ , the message is sent via  $S1$  and  $S2$ . From a client's point of view there is only direct communication with one server per network. In particular it does not have to regard server-to-server communication at all.

### 2.1.2 IRC messages

It is worth mentioning that IRC messages are encoded using 8-bit character codes. IRC commands like “join”, “nick” or “whois” are easy to remember and can be typed quickly. This makes it possible to use IRC over a simple TELNET client. A typical IRC message would look like this:

```
:hans!x@A0Hell-FFB8BDE5.w.pppool.de PRIVMSG gabi :Hallo !
```

Messages consist of three parts that are separated by whitespaces. The first part (prefix) indicates the message's origin. IRC servers will add correct prefixes to all messages they receive from clients. Therefore, clients do not need to specify a prefix when sending a message. The second part is the IRC command. During conversations the “privmsg” command is used to send text messages to other users. Some other commands will be described below. The last message part consists of a number of parameters (depending on the command). For the “privmsg” command a recipient has to be given as a parameter (message target). Each IRC user is identified by a nickname, which has to be unique for the respective IRC network. The message target may be a nickname. However,

IRC conversations usually take place in channels (chat rooms). It is also possible to send a “privmsg” message to a channel. In that case, the IRC servers replicate the message, and deliver it to all users that previously joined the channel. The exact IRC message syntax is described in RFC 1459 [7].

### 2.1.3 Connecting to an IRC network

A user that wishes to communicate via an IRC network needs to register a connection with one of the network’s IRC servers. This process can be divided into the following steps:

1. Establish a TCP connection to the server.
2. Send a “user” message specifying the name under which the user is known to his or her operating system (user name/ident), and the user’s real name. Note that the server has no way to verify this information.
3. Send a “nick” message specifying the desired nickname.

After connection registration, the user is able to join channels using the “join” command, and to communicate with other users. The server may regularly send “ping” messages to test if a client is still active. The network is left gracefully with the “quit” command.

### 2.1.4 Operators

Some IRC users (channel operators) can remove users that act abusively from a channel (“kick”), and prevent them from rejoining it (“ban”). The “mode” command is used to set the operator status (user mode “+o”) and other user/channel modes; the servers keep track of who has which modes.

Network/server operators can disconnect users from their respective servers (for example using the “kill” command), and prevent them from reconnecting to a certain server (“k-lined” users), or prevent them from reconnecting to the network at all (“g-lined” users). Those measures are often applied automatically. For example, when a user sends a certain amount of messages to a channel within a short period of time, he or she might be kicked. Such triggered measures are usually temporary.

## 2.2 Filetransfer with DCC

The Direct Client-To-Client (DCC; sometimes also written out as “Direct Client Connection”) protocol and the Client-To-Client Protocol (CTCP) operate on top of the IRC protocol. There is no official authoritative specification, but there are several documents from different authors describing the protocols [11]. Many IRC clients implement both protocols.

### 2.2.1 CTCP

CTCP allows to embed special characters like linebreaks in “privmsg” and “notice” messages. (The difference between “privmsg” and “notice” is that “privmsg” messages are not supposed to be sent as automated replies to avoid loops.) More importantly, CTCP defines a number of request/reply pairs. For

example, a user may query the name and version of another user's IRC client using

```
PRIVMSG klara :[001]VERSION[001],
```

where “[001]” denotes the ASCII character with the octal representation “001”.

A CTCP capable client like mIRC [14] will reply accordingly:

```
PRIVMSG thomas :[001]VERSION mIRC:v6.21[001].
```

### 2.2.2 DCC

DCC is used to establish a direct TCP connection between two clients. This connection can then be used for efficient file transfer (among of other purposes). The connection is negotiated through special CTCP requests. Assume that in figure 1 client *C* wishes to send a file to client *A*. The following steps are necessary:

1. *C* chooses a TCP port number for the file transfer and listens on that port.
2. *C* sends the CTCP request  
DCC SEND [filename] [C's IP address] [port number] [file size].
3. *A* connects to the respective IP address/port number.
4. *C* sends the file over the DCC connection.

That means *C* acts as a DCC server. However, *A* could also be the DCC server (passive DCC). There are various passive DCC mechanisms [3]; here is how the “Reverse/Firewall DCC” scheme would work:

1. *C* sends the CTCP request  
DCC SEND [filename] [dummy IP address] 0 [file size] [‘token’,  
as a session identifier].
2. *A* chooses a TCP port number for the file transfer and listens on that port.
3. *A* sends the CTCP request  
DCC SEND [filename] [A's IP address] [port number] [file size]  
[token].
4. *C* connects to the respective IP address/ port number and sends the file.

## 2.3 XDCC bots

IRC bots are clients that run without user interaction. Bots may carry out numerous tasks such as channel maintenance or entertainment. After all, the client we have written for data collection is a bot. This section will focus on bots that are used for automated file transfer. Within the scope of this project, we are especially interested in bots that are used to distribute pirated content (“warez”). Fserve, which is part of the mIRC client, and, more importantly, XDCC seem to be the most commonly used bots for that purpose. An open XDCC implementation can be found at [12]. Below we will describe how “warez” are spread by XDCC bots.

There are dedicated IRC channels for “warez” distribution, that the bots join. IRC users who wish to download such “warez” (downloaders) can use search engines like Packetnews.com [15] to find out the channel names. The channels are normally “moderated” (channel mode “+m”). This means that only users with “+v” (“voice”) user mode are able to send messages to the channel. Typically only the bots have that mode. They regularly send “privmsg” messages to the channel and advertise their files. An advertisement may look like this:

```
** 2 packs ** 3 of 8 slots open, Record: 50.5KB/s
** Bandwidth Usage ** Current: 37.1KB/s, Record: 239.7KB/s
** To request a file, type "/msg [alt]-wick-263 xdcc send #x" **
** To request details, type "/msg [alt]-wick-263 xdcc info #x" **
#1 17x [1.4G] States.Evidence.2006.DVDRip.XviD-VoMiT.tar
#2 26x [705M] The.Silver.Surfer.2007.CAM.XViD-FuZeVcD.tar
```

The first two messages contain statistics and information about the current load. The last messages list the names of the offered files and their size. The third message tells the downloaders how the files can be accessed; the text “xdcc send #x”, where “x” is the number under which the desired file has been listed, has to be sent in a “privmsg” message. Bots usually limit the number of simultaneous downloads (the one above has eight download slots). If all slots are taken, the bot adds the downloader to an internal queue. Eventually the bot uses the DCC protocol (as described above) to deliver the file.

Additionally, most XDCC bots offer some convenience functions. It can be possible to query further information about individual files. The bot may also regularly inform users about the status of their download using “notice” messages.

## 3 Implementation

This section will introduce the IRC client that we have written in C++ to collect statistical data about XDCC bots.

### 3.1 Overview

Figure 2 shows the design of our software. An XML configuration file contains a list of IRC servers and a list of channels for each server. The client will try to connect to these servers and join the channels. It also specifies a number of “identities” (consisting of a nickname, a user name and a real name), which the client will use. The ConfigReader class is responsible for accessing the information in the configuration file. Similarly, gathered statistics are stored in an XML file. The StatisticsAggregator class provides an interface for writing to that file. Both classes use a simple XML library [8].

The Controller class initializes the program. It picks an identity at random (to support several instances of the client monitoring the same networks). Then it creates a BotLogic object and starts an additional thread for each server in the configuration file. That means there is a separate thread for each IRC connection. The BotLogic identifies XDCC bots and communicates with them to obtain their IP addresses and other information. For that purpose it has to connect to an IRC network; an IRC protocol implementation is needed. Every BotLogic object has an IRCWrapper object, which allows to send IRC messages

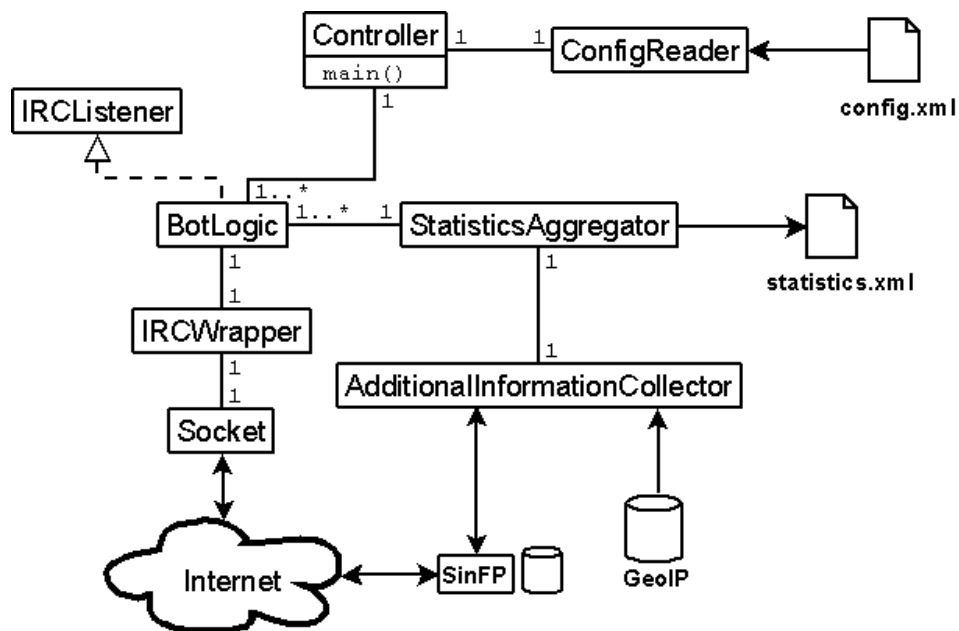


Figure 2: UML class diagram

and provides other “active” IRC functionality. For the basic TCP connectivity Socket instances are used. The Socket class is a wrapper for the POSIX socket library with a simpler interface. Regarding the protocol stack (Socket, IRCWrapper), we wanted the code to be easily reusable. For that reason the IRCWrapper does not call any BotLogic member functions when IRC messages are received. Instead, objects that need to be notified of IRC events have to implement the IRCListener interface. This is similar to the “observer” design pattern [1].

Data obtained by the BotLogic is forwarded to the StatisticsAggregator. The StatisticsAggregator uses the AdditionalInformationCollector class (AIC) to get more detailed information about a bot. Given an IP address and a port number, the AIC tries to identify the country the bot is from. For that purpose the IP address database GeoIP [13] is used, which is reliable and fast, since it does not depend on remote servers. Another way to map IP addresses to countries would have been to query whois servers, but this is slow and the format of the query results can vary wildly, which makes them hard to parse, so lookup via the GeoIP database was chosen for this purpose. The AIC will also try to use the SinFP [9] fingerprinting engine to determine the operating system of said bot, if an open port was provided.

In the following three sections some specific implementation details are discussed.

### 3.2 IRC

For the project’s purpose it was not necessary to implement the entire IRC protocol. First we will list the required IRC commands to illustrate some im-

plementation issues. Then we will explain how the IRC client responds to disruptions.

### 3.2.1 Required IRC messages

The following commands correspond fully to RFC 1459 [7]; fortunately, all IRC networks we encountered seem to comply with the specification — as far as basic client-to-server commands go.

- All the commands described in section 2.1.3 are necessary:
  - “user” command: Any user name/ real name can be used. Most servers additionally query the user name using the Identification Protocol (RFC 1413 [7]); however, we did not come across networks that enforce a reply. If necessary, an external Ident server can always be used.
  - “join” command: Whenever a client successfully joins a channel, it is sent one or more “rpl\_namreply” messages, that list the nicknames and user modes of all users on the channel. These replies could be used to identify bots right away.
  - “nick”, “quit”, “ping” and “pong” command.
- As for the “mode” command, we are only interested in the user modes “+v” (bot identification) and “+b” (“ban”).
- The “privmsg” command/CTCP requests are required for the communication with XDCC bots as described in section 2.2.2.
- The “notice” command/CTCP replies are needed to answer well-known CTCP requests (see section 2.2.1). Many IRC servers use these requests to inspect new clients, but a reply does not seem to be mandatory. To be on the safe side, we have implemented an automated “version” reply, which claims that the mIRC client [14] is running.
- The “whois”/“whowas” command has been implemented because it can be used to query additional data about identified bots.

### 3.2.2 Persistency

For data collection the IRC client needs to monitor IRC channels for several hours, maybe even days when bot uptimes are measured. Therefore it is important that the client can deal with disruptions without user interaction. When the IRC server that the client is connected to fails, the TCP connection will be lost. The problem may be temporary (for example, when the server reboots), so the client should try to reconnect for some amount of time. After that, the client should try to connect to a different server (of the same network). In our implementation one alternate server can be set in the configuration file. Some networks will not allow a reconnecting client to use the nickname it formerly had. This is an anti-abuse mechanism [4]. Because of this the client should also be able to pick an alternate nickname.

After successful reconnection all channels have to be rejoined.

### 3.3 Bot communication

This section explains how the BotLogic class obtains IP addresses and other data, and how it is affected by measures against automated XDCC software.

#### 3.3.1 Data collection

The first task is the identification of XDCC bots. As described in section 2.3 bots advertise their files in “privmsg” messages. Therefore bots can be identified by searching incoming “privmsg” messages for typical character strings, such as “To request a file”. It is unlikely that a user that is not an XDCC bot would send such a message. Considering that on “warez” channels bots are usually the only users with mode “+v”, bots can also be identified by their user mode. However, this method seems less safe and so far our client does not rely on it. The PME [17] library is used to sift through the “privmsg” messages.

The advertisements also contain potentially interesting statistics. Currently the sum of the file sizes (all offered files), the amount of transferred data in total, and the bandwidth usage (current usage and peak usage) are recorded. That data is not necessarily correct; but the people who distribute or run XDCC bots have no motive to manipulate the statistics. Our client also measures the bots’ uptimes by recording “join”, “part” and “quit” events. The “whois” command is not used; “whois” replies contain only the bot’s user name and real name. As mentioned before, such information is not reliable at all. The returned hostname is normally masked and useless.

The most important information for us is the bots’ IP addresses. The BotLogic obtains IP addresses as follows:

1. Identify a bot and send the trigger message (see section 2.3). All XDCC bots that we came across used the same trigger; that means it is not necessary to scan advertisements for the trigger message.
2. Proceed as described in section 2.2.2:
  - (a) If the bot uses active DCC, it will send its IP address in a CTCP request. We observed that some bots send an incorrect IP address (for example a private network address [5]). Therefore our client connects to the bot to verify the address. Note that this behavior was not implemented until the end of the project; that means most of the collected data does contain those incorrect IP addresses.
  - (b) If the bot uses passive DCC, a TCP connection is established as well. Then the IP address is queried from the connected socket. All passive DCC bots we encountered used the Reverse/ Firewall DCC mechanism (see section 2.2.2).

It is not sufficient to read the IP address from the “notice” message that is usually sent in response to the trigger command, as not all XDCC bots send that message.

#### 3.3.2 Measures against bots

Even though our client does not attract attention from operators for the most part, it is sometimes mistaken for Bottler [2] or similar automated XDCC software. The small IRC network “Elemental IRC” [16] discourages from using that

kind of software by sending “notice” messages with the following content:

“This is just a reminder that any form of Automatic XDCC Software, (eg. Botler, XDCC Catcher etc.) is NOT permitted here. There are scripts which will detect what you are running and ban accordingly. This is to ensure that all users receive an equal availability to Files. If you have a FSERV or XDCC and notice that you are being hammered, please notify a member of network staff (with logs) who will deal with it. [sic]”

So far our client gets kicked and banned occasionally, but rarely. If this becomes a problem, the code could be changed so that the bots are contacted less frequently. Other networks use a less aggressive mechanism. If a client sends messages to a large number of different targets within a short period of time, the messages are not delivered. Instead, the server responds with an error message.

### 3.4 Statistics aggregation

As noted before, the StatisticsAggregator is used to collect information about bots in an XML file. During implementation, a number of robustness considerations had to be made, since loss of collected data is to be avoided, even in cases where problems occur (no disk space left, irregular termination, etc.). It was also necessary to implement means for the StatisticsAggregator to recognize known bots and add data about events to the relevant “identity”.

#### 3.4.1 Robustness

Generally, even in the case of multiple connections to multiple servers, only one instance of the StatisticsAggregator should be used, but it can be instantiated more than once. Since having more than one instance can lead to problems (e.g. multiple instances with the same filename), some functionality is disabled in these cases, and later instances pointing to a statistics file which is already in use, redirect their method calls to the first instance which uses this file.

To prevent failed disk writes from destroying previously written statistics, on saving a temporary file with the current statistics is created, and, if writing the statistics to disk was successful, atomically renamed/moved into the place of the destination file. This way, in the case of a full disk (or other failures), the original statistics will not be overwritten by a partial, corrupt XML file.

Since the StatisticsAggregator is expected to be used in a threaded environment (multiple servers), it makes heavy use of semaphores to ensure thread safety; concurrently registered events will not try to modify the XML structure in memory at the same time.

Some events might prompt the StatisticsAggregator to do more time intensive work (e.g. OS fingerprinting in the case of a received DCC send). For these procedures, separate worker threads are spawned, to prevent the BotLogic, and general IRC parts of the program, from having to wait for these jobs to complete.

Also, some signals (SIGTERM, SIGINT, SIGABRT, SIGSEGV) are caught to enable the StatisticsAggregator to save all currently collected data in the case of irregular termination. In the case of a caught SIGSEGV, an additional “.segv” suffix is appended to the statistics filename, to prevent possibly corrupted data from overwriting the last known good state.

Additionally, the signal `SIGHUP` is caught, to provide an easy to use interface for prompting the `StatisticsAggregator` to save the current statistics in environments where no interactive user interface is usable.

### 3.4.2 Bot recognition

The `StatisticsAggregator` keeps a table of known bots' nicknames, usernames and hostnames, mapped to the corresponding bot. By default, when receiving an event or being asked whether a certain bot is known, it will check the nickname map. If nothing was found, the hostname map will be consulted. Unless explicitly configured that way, the username map will not be used, since many XDCC bots have use the same string as ident, while nicknames and hostnames should be mostly unique.

Functionality to change lookup order on a global, per server, and per channel basis is also included.

### 3.4.3 Event logging

The `StatisticsAggregator` provides logging facilities for the following events:

- Seen bot: Triggered when a known or unknown bot was seen sending a message.
- Join: Triggered when a known bot has joined a channel.
- Part: Triggered when a known bot has left a channel.
- Quit: Triggered when a known bot has disconnected from a server.
- Stats: Triggered when a known bot has advertised statistics about its bandwidth, offered data, or transferred data.
- Received DCC send: Triggered when the client has received a DCC send request from a bot. This will make the `StatisticsAggregator` invoke the `AdditionalInformationCollector` to retrieve information about the OS running on the bot's host and about the country the bot is from.

Each logged event will be added to the "identity" of the bot that triggered it, including a timestamp and any additional information, that might be appropriate.

## 4 Results

### 4.1 Methodology

Data collection was carried out over multiple (sometimes overlapping) sessions and on multiple different computers, where in each session a batch of different channels on multiple servers was observed.

In total, 68 channels on 18 servers were observed. The target channels were selected, by searching for channels with a high number of XDCC bots on Packetnews [15], as well as searching for channels with high numbers of users (popular channels) with topics indicating the presence of XDCC bots on Netsplit [6].

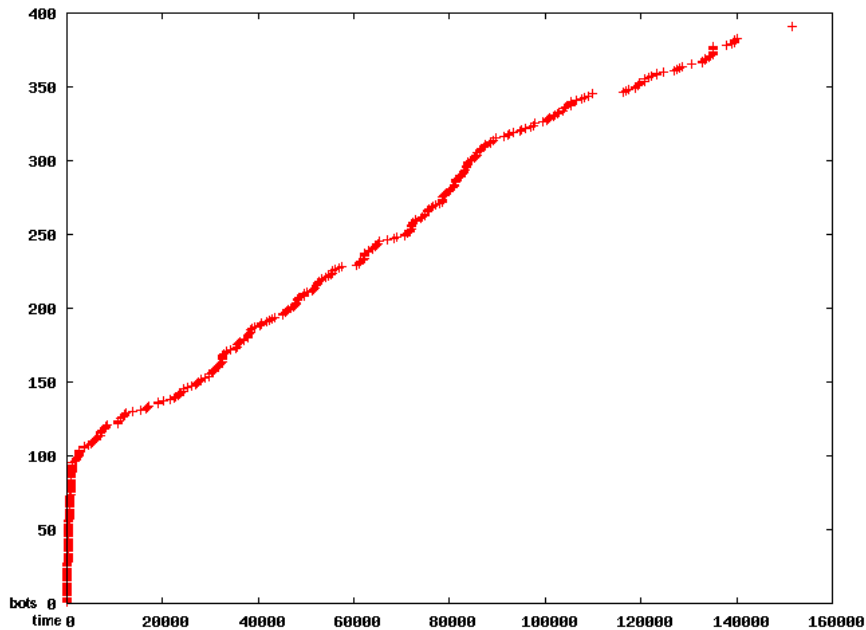


Figure 3: Number of detected bots, *time* seconds after the start of a collection session.

Data collection was carried out over a period of about 242 hours (10 days). With overlap between sessions removed, the data collection period is 226 hours (9.4 days).

After the first data collection sessions, it was noticed that the fingerprinting mechanism gave quite unreliable results (see the “Operating systems” section for more information). Since using the fingerprinting engine requires some special conditions (e.g. the client running with root privileges), it was not enabled for most of the later data collection sessions.

Since at the start of data collection all bots in a channel are unknown, new bots are detected at a much higher rate in the beginning. Later, only hosts with fresh bots, or hosts that were previously offline, are detected as new bots. As can be seen in figure 3, a big number (in this example, approximately  $\frac{1}{4}$ ) of bots is detected right at the beginning. Afterwards the growth roughly corresponds to a somewhat slow growing linear function. This means that starting new collection sessions on so far unobserved or newly found channels will generally yield a higher number of collected bots, than observing a single channel set for a longer time.

The collected data includes the following information:

- Nickname, ident/username, and hostname of each bot, as sent by the server in IRC messages.
- Online and offline events, with timestamp and any additional information like part-messages, and channel.
- IP addresses, countries of origin, and fingerprinting results of bots, including timestamp of the collection.

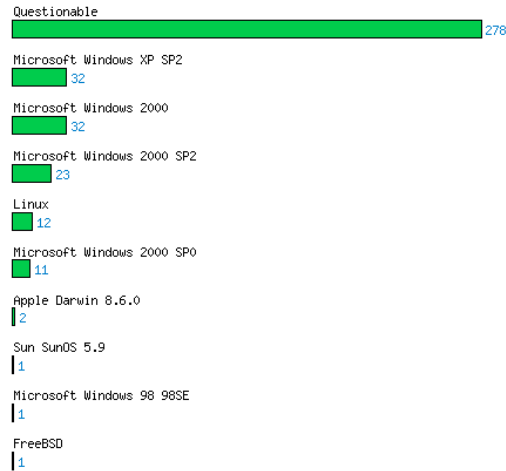


Figure 4: Distribution of XDCC bots by country (Top 10)

- Bandwidth, storage, and transfer statistics of bots as advertised by them.

## 4.2 Diagram notes

At this time, we observed a total of 1766 unique bots.

Now some of the statistics, which can be extracted from the collected data shall be shown.

The “country” UNKNOWN represents all bots, for which no country information could be retrieved from the IP address. These bots sent IP addresses from the ranges 10.\*, 172.16.\*, 172.26.\*, and 192.\*. These bots are most likely misconfigured. Still, some of them claim to have transferred a non-zero amount of data, which leads one to assume, that they were recently reconfigured or the way the hosts are connected to the internet was changed.

Since not every bot advertised all possible information about itself (amount of offered data, transferred data, record data rate), the total number of bots will vary over the different diagram types.

Data offered statistics were advertised by a total of 748 bots, data transferred statistics by 442 bots, record data rate statistics by 649 bots. We have online time data for 1637 bots with country information. Fingerprinting results are available for 393 bots.

For the most diagrams, countries with less than 6 (arbitrarily chosen threshold) bots were ignored, since they are mostly irrelevant for meaningful observation. In the case of mean online time (Figure 11), countries with less than 10 bots were ignored. This higher threshold was chosen to compensate for the bigger data set.

“Other” is a catchall for countries which did not make it into the diagram.

## 4.3 Regions

As can be seen from the pie chart (Figure 4), the highest number of bots is in the USA and South Korea, followed by Germany, the UK, Taiwan, China,

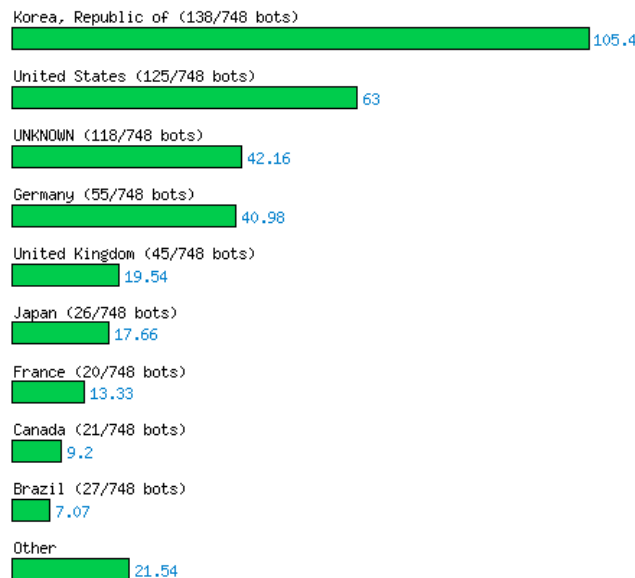


Figure 5: Total amount of data offered by country in GiB (Top 10)

Japan and France.

Now for some more detailed information about offered/transferred data per country.

#### 4.4 Offered and transferred data

Most XDCC bots publicly advertise the amount of data they offer, as well as the amount of data transferred by them so far. These statistics will give an overview, about how active the bots from different countries are.

Looking at these charts, it will become apparent, that South Korean bots store and transfer a vast majority of data. South Korea is at the top of each single data offered/transferred diagram. Even though the USA contain a higher number of bots, they share the lower places with Germany, the UK, Japan, France, Canada, Taiwan, the Russian Federation, Turkey, Spain and Brazil.

In the charts pertaining to the total amount of data (figures 5 and 6), the bots from South Korea transfer/offer almost twice as much as the USA, which holds the second place in both cases.

The differences in the charts for mean amount of data offered and transferred per bot (figures 7 and 8) are smaller. The bots from South Korea are still at the top, while the USA rank lower than on the charts before, which can be attributed to the fact that, while they contain a large number of bots, those bots are not used all that much.

#### 4.5 Record data rate

The bots also publish the record transfer speed in their advertisements.

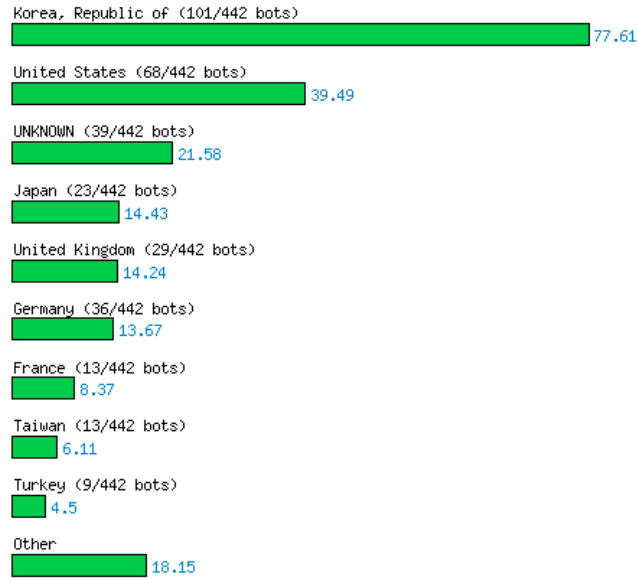


Figure 6: Total amount of data transferred by country in GiB (Top 10)

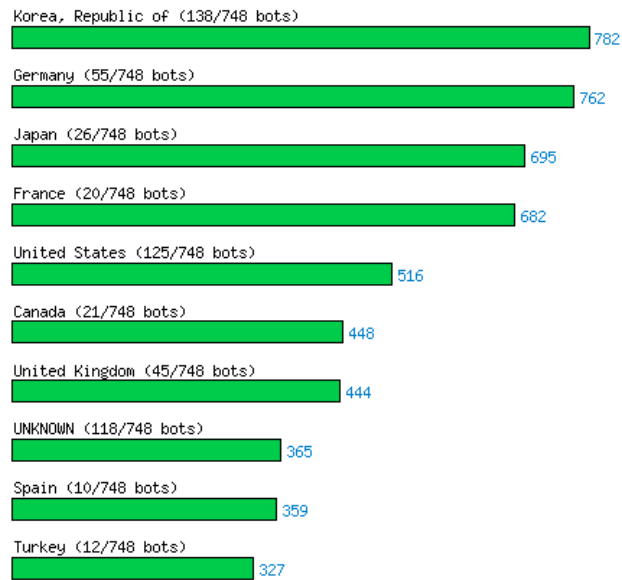


Figure 7: Mean amount of data offered per bot in MiB (Top 10)

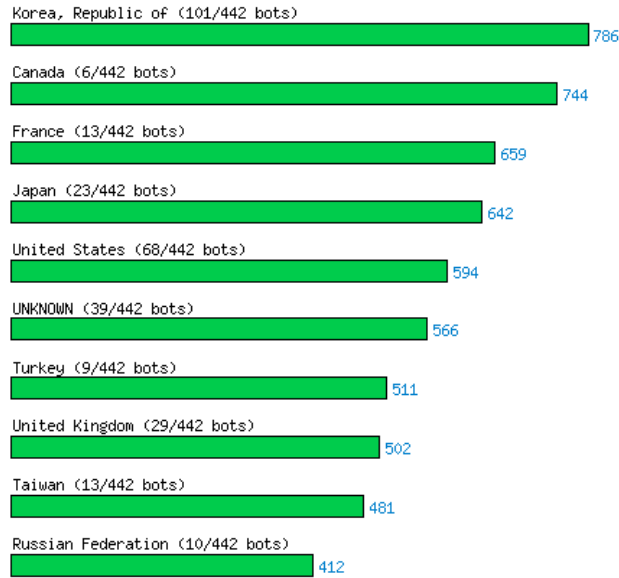


Figure 8: Mean amount of data transferred per bot in MiB (Top 10)

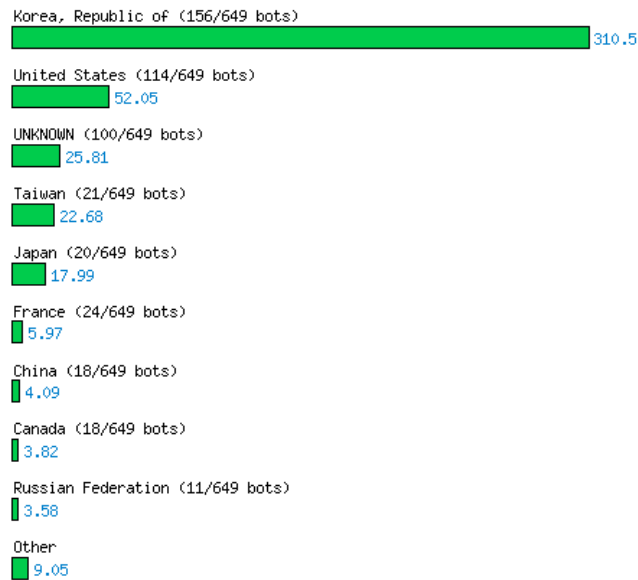


Figure 9: Total record data rate of all bots in MiB/s (Top 10)

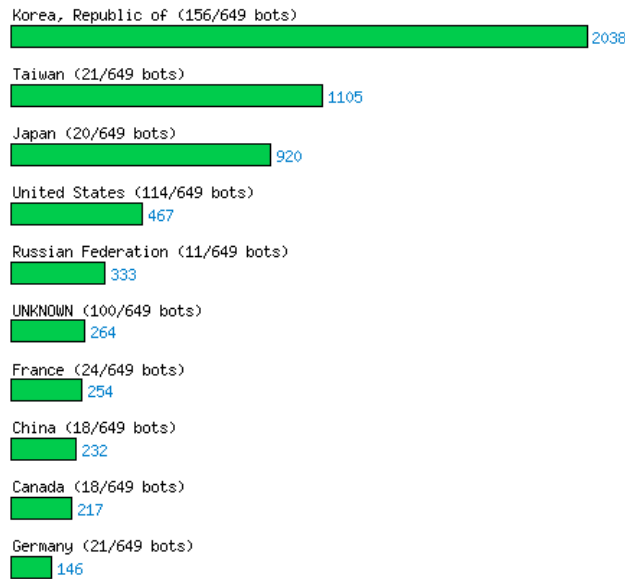


Figure 10: Mean record data rate of all bots in KiB/s (Top 10)

In the case of the total record data rate diagram (figure 9), the total record speed of South Korean bots is almost six times that of the bots in the USA, who are placed 2nd. This coincides with the fact, that fast broadband connectivity is very common in South Korea. The relatively high total record data of the USA is mainly made up from the large number of bots in that country. Third place is taken by currently misconfigured bots, which probably worked some time ago. Taiwan and Japan (4th and 5th place respectively) both contain a comparatively low number of bots, but still have a high total record data rate, which again stems from the high availability of fast broadband connectivity in both countries. [19] [20]

The mean record data rate diagram (figure 10) is, again, lead by South Korea by quite some margin. On average, South Korean bots have twice the record rate as those from Taiwan, which is placed second. Japan is placed third. In this diagram it becomes apparent, that bots from the USA are generally slower than those in South Korea, Taiwan and Japan, and the USA only placed high in the total record data rate diagram, because of its high number of bots. Bots from the USA have about half the mean record data rate as those from Japan.

Obviously a high number of fast broadband connections in a country leads to faster bots and, as can be seen in the case of South Korea, make PCs from the country a more common/popular target for attacks.

## 4.6 Online time

In this diagram, countries with 10 or less bots were excluded, since they are not really statistically relevant in this significantly bigger data set.

As can be seen in figure 11, the bots from Spain were the most stable bots (4.65 hours mean online time). Those from South Korea were also among the

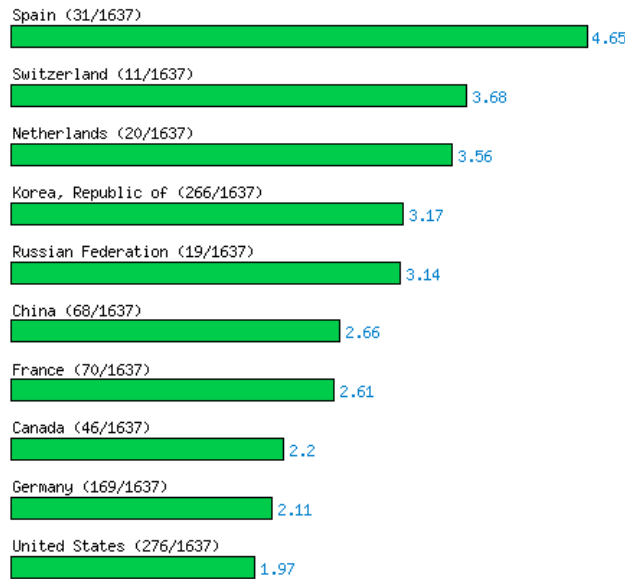


Figure 11: Mean online time per bot in hours. (Top 10)

most stable bots (3.17 hours mean online time, fourth place), while those from the USA ranked a good bit lower (1.97 hours mean online time). Bots from the Russian Federation (3.14 hours), China (2.66 hours), France (2.61 hours), Canada (2.2 hours) and Germany (2.11 hours) are between the USA and South Korea, as far as reliability/online time goes.

## 4.7 Operating systems

Now we shall take a look at the OSes run by the bot hosts.

It quickly becomes apparent, that the OS fingerprinting was less than perfect. The questionable results were mostly Linksys routers, which really should not run XDCC bots and were probably misidentified for some reason. Still, it is quite obvious that most of the “owned” (compromised/hacked) machines run MS Windows (esp. Windows 2000).

There are also a few bots running Linux and other UNIX like OSes. These might be just slightly unusual botnet zombies, or they could be machines of people voluntarily running an XDCC bot for some reason.

## 4.8 Conclusion

While the USA have the largest share of bot running computers (16.4%), South Korea (13.4%) and other Asian countries (Taiwan, China, Japan) also contain a significant number of bots (about 26.5% in total).

Considering the fact, that in the USA, there are about 211M internet connections [10], while in South Korea there are only 34M, the proportion of bot-running PCs in South Korea is much higher.

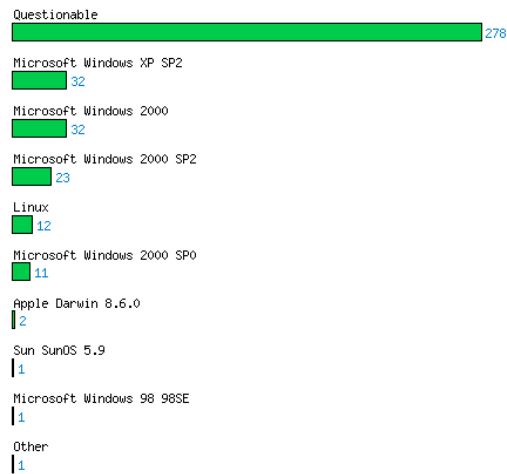


Figure 12: OSes of computers running XDCC bots (Top 10)

Total data offer/transfer statistics (about twice as much data offered/transferred as bots from the USA), as well as the data rate statistics (highest record data rate by a big margin) lead us to consider the South Korean bots as the most active/actively used ones.

From this, we can draw the following conclusions:

- Our data supports the assumption, that a very significant number of bots come from South Korea (and Asia in general). This is especially meaningful, since those bots are the most active.
- Most of the bots seem to run on MS Windows machines. This corresponds to our assumptions.
- Computers from countries with a high number of very fast broadband connections seem to be preferred targets for attacking and running XDCC bots one them.

## References

- [1] Bruce Eckel and Chuck Allison. *Thinking In C++. Volume 2: Practical Programming*. Prentice-Hall, 2003.
- [2] [http://en.wikipedia.org/wiki/Bottler\\_%28IRC\\_client%29](http://en.wikipedia.org/wiki/Bottler_%28IRC_client%29).
- [3] [http://en.wikipedia.org/wiki/Direct\\_Client-to-Client](http://en.wikipedia.org/wiki/Direct_Client-to-Client).
- [4] [http://en.wikipedia.org/wiki/Internet\\_Relay\\_Chat#Nick.2Fchannel.delay](http://en.wikipedia.org/wiki/Internet_Relay_Chat#Nick.2Fchannel.delay).
- [5] [http://en.wikipedia.org/wiki/IPv4#Private\\_networks](http://en.wikipedia.org/wiki/IPv4#Private_networks).
- [6] <http://irc.netsplit.de/>.
- [7] <http://tools.ietf.org/html/>.
- [8] <http://www.applied-mathematics.net/tools/xmlParser.html>.
- [9] <http://www.gomor.org/cgi-bin/sinfp.pl>.
- [10] <http://www.internetworldstats.com/>.
- [11] <http://www.irchelp.org/irchelp/rfc/ctcpspec.html>.
- [12] <http://www.iroffer.org/>.
- [13] <http://www.maxmind.com/app/c>.
- [14] <http://www.mirc.co.uk/>.
- [15] <http://www.packetnews.com/>.
- [16] <http://www.twist3d.com/elemental/>.
- [17] <http://xaxxon.slackworks.com/pme/>.
- [18] [http://www.itu.int/ITU-D/ict/cs/korea/material/CS\\_KOR.pdf](http://www.itu.int/ITU-D/ict/cs/korea/material/CS_KOR.pdf) International Telecommunication Union. Broadband korea: Internet case study, 2003.
- [19] <http://www.elcot.com/mait-reports/Broadband-Taiwan.pdf> MAIT. Broadband in taiwan, 2005.
- [20] <http://www.neca.org/media/taniwaki.pdf> Yasu Taniwaki. Broadband deployment in japan, 2004.