

Code Governance

«Code» as Regulation in a Self-governed Internet Application from a Computer Science Perspective

vorgelegt von Diplom-Informatiker Kei Ishii

von der Fakultät IV – Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
– Dr. Ing. –

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender: Prof. Dr. Bernd Mahr

Berichter: Prof. Dr. Bernd Lutterbeck

Berichter: Prof. Dr. Hans-Ulrich Heiß

Tag der wissenschaftlichen Aussprache: 7. Juli 2005

Berlin 2005

D 83

Contents

Main Hypotheses	15
Introduction	17
I Internet, Governance, and «Code»	21
1 Internet Governance and «Code»	23
1.1 Internet Governance	23
1.2 "And How Shall the 'Net Be Governed"?"	23
1.3 Governance	25
1.4 «Code» as Regulation Modality	26
2 «Code» Governance: An Empirical Analysis	29
2.1 The Choice of the Object of Analysis	29
2.2 Main Hypothesis	30
2.3 Two Models of Regulation	30
2.3.1 "Lex Informatica": Key Aspects of a Regulation System	31
2.3.2 Types of Rules	33
2.4 Technical Notes on the Analysis	34
II Empirical Analysis of the Internet Relay Chat	37
3 The Internet Relay Chat	39
3.1 The Functional Perspective – Using the IRC	40
3.1.1 A hypothetical user scenario	40
3.1.1.1 Channels	41
3.1.1.2 Private Messages	42
3.1.1.3 CTCP (Client-to-Client Protocol)	43
3.1.1.4 IRC Commands	43

3.1.2	Size and Growth of the IRC	44
3.2	A Conceptual View of the Technology of IRC	46
3.2.1	”Polity”: Structural Overview of the IRC	46
3.2.1.1	IRC Server	47
3.2.1.2	IRC Client	47
3.2.1.3	IRC User Bot	48
3.2.1.4	IRC Service	49
3.2.1.5	IRC Network	50
3.2.2	”Politics”: A Processual View of the IRC	52
3.2.2.1	The Client-server architecture	52
3.2.2.2	Inside the IRC server process	53
3.2.3	”Policy”: IRC Server Installation and Configuration	54
3.2.3.1	Installation of the IRC server	55
3.2.3.2	IRC server configuration	58
3.2.4	Technical environment and code distribution	59
3.3	Main Social Roles in the IRC	60
3.3.1	IRC administrator	61
3.3.2	IRC Service administrator	61
3.3.3	Coder	62
3.3.4	IRC user	62
3.3.5	IRC operator	63
3.3.6	Channel operator	64
4	«Code» Governance in IRC Channels	65
4.1	Principal Channel Design	66
4.2	Numbered Channels	68
4.2.1	Functional Design	69
4.2.2	Technical Implementation	70
4.2.3	The ”Maximum Users Per Channel” Channel Property	72
4.3	Named Channels - a Major Change in «Code» Governance	77
4.3.1	Names, modes, and the channel operator	77
4.3.2	«Code» Evolves – Further Changes in Channel Design	81
5	Sanctions in the IRC	87
5.1	The /kill Command – Immediate Sanction	88
5.1.1	Functionality and Implementation	88
5.1.2	Changes in the /kill command	89
5.2	The K-Line – Entry Denial Sanctions	92

5.2.1	Functional description and technical implementation	92
5.2.2	Config lines complementing K-line sanctioning	94
5.3	Access to K-lines for IRC operators	97
5.3.1	First experiment – Undernet’s /kline and /addline commands	98
5.3.2	/kline and exceptions in the EFnet	100
5.4	Beyond /kills and K-lines	101
5.4.1	Delegating sanctioning power – Channel modes, kicks and bans	101
5.4.2	Non-Sanctioning «Code» Remedies	102
6	Nickname and Channel Ownership	109
6.1	Nicknames, Channel Names, and Early Ownership Policies	110
6.1.1	Nicknames	110
6.1.2	Channel names	112
6.2	Policy Changes With Bots and Services	113
6.2.1	Channel Control with the Eggdrop user bot	113
6.2.2	The NickServ service bot	116
6.3	IRC Channel Registration Services	118
6.3.1	Channel Registration in the DALnet: ChanServ Service	119
6.3.1.1	Channel registration	120
6.3.1.2	Channel management features	121
6.3.2	Channel Registration in the Undernet: The X/W Service	123
6.3.2.1	Channel registration	123
6.3.2.2	Channel management: The X and W channel service bots	125
7	Controlling the Controllers?	129
7.1	IRC Operator – Power and Control	129
7.2	Nominating IRC operators and IRCop Netiquette	132
7.3	Notices and Logs	135
7.3.1	Notices	136
7.3.2	Logging	140
7.4	The Undernet UWorld Service	140
7.4.1	About the UWorld Service	141
7.4.2	Functionality	142
7.4.2.1	UWorld commands	143
7.4.2.2	UWorld automatic functions	146
7.4.3	UWorld User Access	148
7.4.4	Control in UWorld: Information, Notices, and Logs	149

8 IRC Network Issues	155
8.1 «Code» Architecture Shapes the Social Constitution	155
8.1.1 IRC: Topology, Data Distribution and Technical Rationale	155
8.1.2 DNS: Topology, Data Distribution and Technical Rationale	158
8.1.3 Comparison between the IRC and DNS architectures	160
8.1.4 Architecture as Constitution	161
8.2 The "Great Split": The Forking of Anet and EFnet	162
8.2.1 Open-server servers vs. closed-server servers	163
 III Some Notes on the Concept of «Code» Governance	 165
9 Lex Informatica Revisited	169
9.1 Framework	169
9.2 Jurisdiction	170
9.3 Content	172
9.4 Source	173
9.5 Customized Rules and Customization Processes	174
9.5.1 Rule customization on the «code» level	175
9.5.1.1 Source code access	175
9.5.1.2 System access	178
9.5.1.3 Interface access	179
9.5.1.4 User access	180
9.6 Primary Enforcement	181
 10 Rule Types in «Code»	 183
 11 Outlook	 185
11.1 Validating and Refining the «Code» Governance model	185
11.2 Computer Science Implications from «Code» Governance	188
 IV Appendix	 191
 12 IRC Chronology	 193
12.1 1989 – The Birth of the Internet Relay Chat	193
12.2 1989-1990 – Copyright, named channels and the "great split"	194
12.3 1990-1992 – Growth and Development	195
12.4 1993 – The first successful fork: Undernet	196

12.5	1994 – Another fork: DALnet	196
12.6	1996 – IRCnet forks off EFnet	196
12.7	The IRC Source Code Copyright	197
13	Tools for the Examination of the IRC Source Code	199
13.1	In Search of the Right Tool	199
13.2	A Makeshift Solution	202
13.3	The Analysis	203
14	List of IRC server source code packages	205
14.1	Onet	205
14.2	EFnet	205
14.2.1	Standard	205
14.2.2	+CS	205
14.2.3	+th and Hybrid	207
14.3	Undernet	207
14.4	IRCnet	209
	Bibliography	211

Contents

List of Tables

2.1	Lex Informatica (Reidenberg, 1998, p.569)	31
3.1	Configuration lines in server version irc2.1.1 (Oct. 1989)	59
3.2	Social Roles in the IRC	61
4.1	Common commands in connection with IRC channels (Pioch, 1993)	68
5.1	Changes in the /kill command code	92
6.1	DALnet Channel Service Officials	122
6.2	X/W user levels	125
7.1	IRC operator privileged commands	131
7.2	Notices sent to IRCops in irc server version irc2.1.1	138
7.3	Uworld commands and access level	149
7.4	Uworld commands triggering notices	151
14.1	Onet server versions	205
14.2	EFnet server versions 2.5 and 2.6	206
14.3	EFnet server versions 2.7 and 2.8	206
14.4	EFnet +CS (comstud) versions	207
14.5	EFnet server versions +th and HybridUndernet server versions u2.9 and u2.10	208
14.6	Undernet server versions 2.8	208
14.7	Undernet server versions u2.9 and u2.10	209
14.8	EFnet server versions 2.7 and 2.8	210

List of Tables

List of Figures

1.1	Four Regulation Modalities (Lessig, 1999a, p.88)	27
3.1	Initial messages when joining an IRC network (Undernet, on Jan 6, 2004) . .	41
3.2	Example conversation in the IRC channel "#hottub"	42
3.3	Snapshot of a channel list in the Undernet	44
3.4	IRC growth from 1988 to late 1995	45
3.5	Elements of an IRC network	47
3.6	Tree topology	50
3.7	Undernet network (around 1993).	51
3.8	Basic IRC client-server configuration	52
3.9	Model of the server process	54
3.10	IRC server installation steps	55
4.1	IRC Timeline – Numbered Channels	69
4.2	Linked list for channels	71
5.1	Flood block with /ignore	104
5.2	Flood block with /silence	105
6.1	Steps to Set Up an Eggdrop Bot	114
8.1	Spanning tree topology	156
8.2	Tree topology as a "tree"	159
8.3	Querying a Domain Name	160

List of Figures

List of Algorithms

1	The <code>struct Channel</code> data structure (slightly simplified)	70
2	Channel limit check in <code>m_channel()</code>	73
3	<code>is_full()</code> and <code>UnLimChannel()</code> functions	74
4	<code>MAXUSERSPERCHANNEL</code> directive	74
5	Startup option (commented out) for <code>maxusersperchannel</code>	75
6	<code>OPER_KILL</code> directive	91
7	<code>LOCAL_KILLS_ONLY</code> definition	91
8	I-line	94
9	"Restrict" configuration line (R-line)	95
10	"Dump" configuration line (D-line)	97
11	<code>DYNAMIC_CONF</code> directive (activate <code>/kline</code> and <code>/addline</code> commands)	99
12	"Exception" configuration line (E-line)	101
13	The <code>/ignore</code> command	103
14	Eggdrop configuration lines (examples)	115
15	"Operator" configuration line (O-line)	130
16	"Quarantine" configuration line (Q-line)	163

List of Algorithms

Main Hypotheses

1. «Code», the technology itself, has been identified as a regulation modality next to law, market and norms.

1. When «code» can be understood as regulation, then there must be distinct features which are similar to other regulation systems and other distinct features which are unique to «code»:

1. **The analysis of the source code of an self-organized, self-governed Internet application reveals distinct «code» features: Structures and changes in the source code which achieve some regulator's end. These features form the «code» regulation system of the application, it's «code» governance.**

- a) As a self-organized application, it is ensured that the participants of the application themselves shape the source code, and thus are their own regulators.
- b) The choice of a self-governed system let me neglect the impact that the law and market modalities have on the setting.
- c) An Internet application is positioned at the end of the Internet, directly between the user and the lower layer technical infrastructure, as well as distinct and distinguishable from other applications.

Main Hypotheses

Introduction

The starting point of this work is the notion that the software and hardware underlying the technically mediated communication and interaction – called «code» – constitutes a regulation system, similar to but distinct from the legal system.

The Internet continues to change the ways that people interact with each other, with impacts on all levels of their personal lives, social groups and society at large. This leads to the still unanswered question of how governance structures have to be changed in order to cope with the various problems that these new settings create. We already evidence the ongoing discussions in areas like copyright, privacy, or Internet commerce.

One constant in all these situations is the existence of the technology, as its effects and impacts on society are the actual source of all discussions. Interestingly though, the very shape of the technology – the hardware and software itself – has only recently been recognized as important source of regulation in the sense that it shapes online activities by either constraining or enabling its users. The concept of technology as regulating means has found attention in the form of "code is law"¹, mainly among legal scholars. As a consequence, this concept is only studied as far as «code» can be used to serve legal objectives and purposes.

In the light of another regulatory discourse around the topic of (Internet) "governance" though, narrowing «code» to a mere legal enforcing tool appears too limiting. The term governance has made it possible to think about regulatory quandaries not in exclusively legal terms, but to include social norms, market forces, and physical or technical architecture as sources of regulation. From this perspective, the "code is law" turns into "code as a regulation modality", or "code regulates".

But what does it mean that "code regulates"? It is the purpose of this work to evaluate the concept of «code» as a regulation system. To this behalf, I take the legal system as 'model regulation system' to empirically explore the key concepts of «code» in the context of an Internet application. I have chosen an existing self-organized, self-governed Internet application, the Internet Relay Chat, to understand how the participants use the «code» to govern themselves, and how the «code» in turn governs the setting.

The Internet Relay Chat (IRC) is one of the major application in the Internet, albeit not as generally known as the world wide web, email or even file-sharing applications. Nevertheless,

¹Lessig (1999a)

hundreds of thousands of people use the IRC at any time of day, any day of the week. It allows to 'chat', to exchange short text messages in real time with other users inside so called 'channels' (sometimes also referred to as "chat rooms").

The main feature for my analysis is that the IRC is a self-organized, self-governed application. "Self-organized" here means that the entire application structure, a network of IRC servers, is run and maintained by volunteers without any overarching corporate or governmental structure directing them. They cooperate to form an IRC network because and as long as they choose to.

Equally, the software, started by one individual, is constantly maintained and extended by volunteering individuals forming their own institutional environment, again with no company or governmental organization paying or directing them.

In the same venue, "self-governed" means that the IRC participants themselves manage their common affairs, resolve disputes, create and enforce their own set of rules, with no legal or market interests or interference. For the social order of the Internet Relay Chat, law or market forces are not driving forces for its governance. Instead, a large part of social order in the IRC is determined by «code», the technology that constitutes the application. The «code» defines what is possible in the IRC and what is not, it determines the possibilities and constraints of all participants, from the simple user to IRC officials. Changes in the «code» might alter the governance characteristics and the constraints and opportunities for the IRC participants.

At the core of my examination of the governance structures and dynamics of the Internet Relay Chat lies the analysis of the *source code* of the IRC server, the determining application component of an IRC network. By treating the source code as an regulatory expression of the IRC principals, I hope to find distinct features which establish the «code» as a distinct regulation modality, a notion that I call «code» governance.

* * *

This work is consists of three parts.

In the first part, I develop the notion of «code» as a regulation system, by outlining the discussion which lead to this concept ("Internet Governance and «Code»", chapter I), and then present my hypothesis and agenda for the main part, the analysis of the Internet Relay Chat (chapter 2).

This analysis proceeds in the six chapters of part II. Chapter 3 gives an introduction to the Internet Relay Chat, including its use, its main technical components and main social roles of the participants. Chapter 4 examines the main entity in the IRC, the channel. By working out its main features and properties, and their changes over the succession of source code versions, I show how this «code» defines the governance environment, implementing constraints and opportunities for the users to shape the social order in channels.

Sanctions are a major mechanism in most social settings. Chapter 5 reviews the sanctioning mechanisms as they are designed and implemented into code. Again, the development of these tools show how the IRC copes with changing environments, such as its successive growth, creating new governance challenges to be met with adequate regulation tools.

As in other Internet applications, the desire to 'own' names, to hold them for a longer period is prevalent in the Internet Relay Chat. Chapter 6 traces the development of adequate policies and mechanisms to put them into «code», and enforce them.

The IRC employs officials – IRC operators – to manage the daily affairs of the servers and networks, which may include applying sanctions to other users. Chapter 7 sketches the norm rules guiding them, as well as «code» based mechanisms implemented to heighten the transparency of their actions. Also explored is a «code»-based tool which give IRCops and other users unprecedented power in one of the IRC networks.

Finally, chapter 8 joins two chapters dealing with network affairs: the first section examines the relationship between architectural structure of an application like the IRC, and its governance impact on the social setting therein. The second section traces the succession of events which led to the split up of an IRC network due to fundamental policy differences between the principals.

Part III presents the results of the empirical explorations.

Part I

Internet, Governance, and «Code»

1 Internet Governance and «Code»

The agenda that I have chosen for this work is to further the understanding of the «code» part in the concept of «code is law» as popularized by the leading cyberspace law scholar, Lawrence Lessig¹. This concept is part of an ongoing discussion about the regulation and regulability of the Internet. To understand this background of Lessig's concept, a short overview of this discussion is given.

1.1 Internet Governance

The growing use of the Internet from the beginning of the 1990s has spurred the existing discussion and research of social and societal implications of information and communication technologies. Many contributions continue to add to the ever growing body of research and opinions on how these technologies could change everything from the individuals' daily life to the foundations of society at large.

A considerable part of these discussions are concerned with *regulative* issues: Topics like copyright, privacy, content control and others are hotly disputed topics, and remain largely unsolved, whereas other areas, for example commercial contracts, have found a *modus vivendi*, supported by legal regulations and technical solutions.

In parallel to those specific policy issues, another 'meta' issue has received growing attention: *how* and *by whom* is and should the Internet be regulated?

1.2 "And How Shall the 'Net Be Governed"?

In early contributions (especially legal scholarly ones), regarding how the Internet should be governed, the view prevailed that the Internet did not pose any new challenges to the existing (legal) regulation regime; some even claimed that the Internet did not necessitate any changes in the regulatory approaches at all². This stood in contrast to a similarly extreme view of technologists who deemed that legal regulations could always be circumvented by technology: "The net treats censorship as damage and routes around it"³; and that they should not 'rule'

¹For example in Lessig (1999a)

²See for example Lessig (1999b), referring to a speech by Judge Frank Easterbrook.

³This is attributed to John Gilmore; see his website at <http://www.toad.com/gnu/>, 29 Oct 2004.

the Internet:

That's the kind of society I want to build. I want a guarantee – with physics and mathematics, *not with laws* – that we can give ourselves real privacy of personal communications.⁴

These were the extreme ends of the discussion, which both had their merits: There were a number of issues which could be resolved by legal means, through court decisions, and changes in statutory law. These developments usually are subsumed under the topic label of *cyberlaw*⁵. Other issues became non-issues, because of the technical development. But for many issues, such as copyright or the "freedom to tinker"⁶, the tension between "rule of law" and technological advancements remains to this day witnessed in countless contributions, newspapers, web pages or scholarly articles and books.

At some points, legal scholars began to consider the possibility of alternative regulative means. An early contribution for example concluded that for some issues, "'bottom up" rule making processes are also workable"⁷, for which he named "unilateral self help, [...] contracts, private associations [...] and the development of customs"⁸ as examples. Other scholars observed that in the existing Internet, alternatives to legal regulations had already emerged⁹ which would challenge the notion of government (and thus law) as a "bureaucratic single-provider institution"¹⁰.

Especially influential (and highly controversial¹¹) were the papers by David Johnson and David Post¹² who made a theoretical argument on why alternatives to legal regulation in the Internet was not only possible, but should also be actively fostered:

"This new boundary defines a distinct Cyberspace that needs and can create new law and legal institutions of its own. Territorially-based law-making and law-enforcing authorities find this new environment deeply threatening. But established territorial authorities may yet learn to defer to the self-regulatory efforts of Cyberspace participants who care most deeply about this new digital trade in ideas, information, and services. Separated from doctrine tied to territorial jurisdictions, new rules will emerge, in a variety of online spaces, to govern a wide range of new phenomena that have no clear parallel in the non-virtual world. These new rules will play the role of law by defining

⁴Gilmore (1991), my emphasis, K.I.

⁵See for example Johnston et al. (1997); Rosenoer (1997).

⁶Edward Felten, <http://www.freedom-to-tinker.com/>

⁷Hardy (1994, p.1054)

⁸ibid.

⁹E.g. Valauskas (1996) (arguing that the Internet communities have created a "separate law of Cyberspace" in form "cyber-etiquette protocols"); Oberding and Norderhaug (1996) (claiming that there is no need for a separate jurisdiction in Cyberspace, because Internet communities and organizations like the Internet Engineering Task Force (IETF) already provide regulatory norms).

¹⁰Hadfield (2000, p.2)

¹¹See for example Goldsmith (1998) (contending that the Internet is anchored in real space, and thus can readily regulated by national governments), Radin and Wagner (1999) (arguing that any self-governing Internet or "anarcho-cyberlibertarianism" setting cannot exist without the backing of contract law)

¹²Johnson and Post (1996b,a)

legal personhood and property, resolving disputes, and crystallizing a collective conversation about core values.”¹³

More and more legal scholars began to search for alternatives to a direct legal rule in the Internet. Such novel concepts are often not confined to one topic area alone. In parallel to the Internet specific discussions, consideration about regulatory alternatives to law were pursued in other areas as well. For this, the term ”governance” emerged.

1.3 Governance

The term *governance*¹⁴ is used in as different contexts as international relations (global governance) and in corporate environments (corporate governance). Correspondingly, in the context of the regulation of the Internet, the term *Internet governance* had been established.

Common to all approaches subsumed under the label of governance is the recognition that situations exist where the ”monopoly on coercive ordering and dispute resolution”¹⁵ held by the legal system leads to unsatisfying regulatory solutions. Therefore the legal-centric perspective is given up in favor of a multi-actor, multi-regulatory approach which still includes legal means, but not as a sole provider of regulation. Of the various definitions of the term governance¹⁶, I found the one given by the United Nation Commission on Global Governance most clarifying:

”Governance is the sum of the many ways individuals and institutions, public and private, manage their common affairs. It is the continuing process through which conflicting or diverse interests may be accommodated and cooperative action may be taken. It includes formal institutions and regimes empowered to enforce compliance, as well as informal arrangements that people and institutions either have agreed to or perceive to be their interest”¹⁷

This definition points to a number of important topics:

REGULATORS – The definition recognizes a broad range of possible policymakers, from individuals to institutional, and both public and private entities. And importantly, the definition does not prescribe any relationship (hierarchical or not) between them.

¹³Johnson and Post (1996b)

¹⁴According to Grewlich (1999, p.250), the term governance came from corporate law.

¹⁵Hadfield (2000, p.3)

¹⁶See for example MacNeil (1999) (”Governance ranges from issues of how a sovereign governs its subjects to how communities and institutions govern themselves to how individuals govern their daily lives.”); McTaggart (1999) (”In this thesis, ”governance” is used to describe the people, institutions, rules, and principles which made the Internet what it is and influence its evolution.”; followed by a short etymology, its uses in corporate law, international relations etc.); ”Governance.” Wikipedia. 2005-04-19 <http://en.wikipedia.org/wiki/Governance>. (”Although the term governance is often used synonymously with the term government it tends rather to be used to describe the processes and systems by which a government or governor operate.”)

¹⁷Commission-on Global-Governance (1995)

DYNAMIC PROCESS – Regulatory issues can shift due to changing conditions and environments, or due to unforeseen effects of the regulation itself, especially in complex situations where many regulators instituting various regulatory means are involved.

REGULATORY MEANS – The kind of rules, institutions or regimes through which the common affairs are governed is not limited to the legal system, but encompasses various other means.

Accordingly, *Internet* governance is concerned with the ways that multiple actors and institutions in the Internet apply multiple regulatory means to manage their common affairs.

The governance definition serves as a first step in an interesting direction, poses more question than it can answer, such as the relationship between the regulators, or – and thus we return to Lessig's concept of «code is law» – the question of what the "regulatory means" are.

1.4 «Code» as Regulation Modality

Lessig's notion of «code is law» is based on a model in which he identifies four distinct regulatory means or "modalities"¹⁸. One of these modalities in Cyberspace is «code».

Lessig actually develops this model independent of the Internet. He starts by asking: What constrains individual behavior? He defines constraints as the "constraining effects of some action, or policy"¹⁹.

He then identifies *four* distinct kinds of constraints that limit individual behavior: Market, law, (social) norms, and architecture (figure 1.1).

Basically, *law* denotes the threat of sanctions by the legal system, *market* the constraints put up by the price or the (non-) availability in markets, *(social) norms* as constraints put up by some "community", and finally *architecture* as those constraints that physical objects, laws of nature etc. might put up²⁰.

The important modality which establishes the agenda of this work is hidden in the last one, architecture. In his application of this model to Cyberspace, Lessig identifies this modality as *code*:

"And finally, an analog for architecture regulates behavior in cyberspace—*code*. The software and hardware that make cyberspace what it is constitute a set of constraints on how you can behave."²¹

¹⁸Lessig (1999a, pp.85-90); Lessig (1998, pp.662-4)

¹⁹Lemley (1998, p.662)

²⁰In Lessig (1998, pp.667-70), gives a number of examples to illustrate these four constraints, providing examples from smoking to abortion.

²¹Lessig (1999a, p.89)

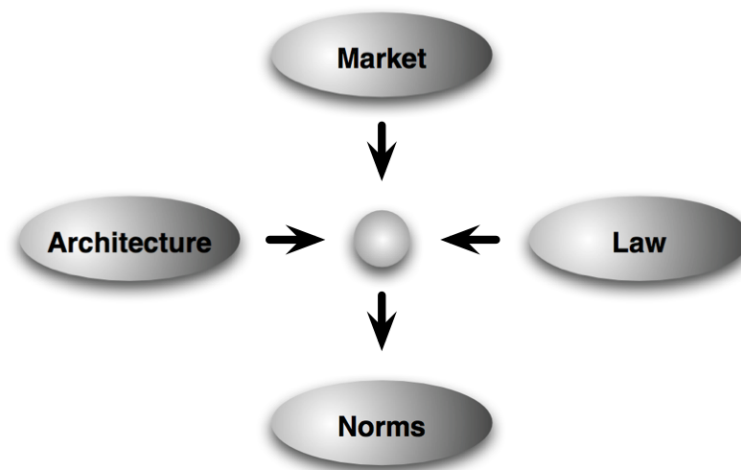


Figure 1.1: Four Regulation Modalities (Lessig, 1999a, p.88)

«Code»

«Code», as used in this work, points to the hardware and software as regulation modality. In order to distinguish this specific meaning from the code in "source code", or the "legal code", «code» in the meaning of a regulation modality will be written with quotation marks.



This point sets the stage for the explorations that I pursue in this work. What is a code modality? How does code constrain, with what consequences on the governance of a social setting? And what does this mean for those who shape the code, programmer, software engineers, computer scientists?

While in the legal scholarly debate this model is widely accepted²², they (understandably) concentrate on the relationship of «code» and law, that is, how law can effectively employ «code» to serve legal objectives, or again in Lessig's terms,

"how law might be used to regulate the architectures of cyberspace so that the architectures of cyberspace might better advance the ends of law—so that it might, that is, become more regulable"²³.

As computer scientist, this is not the direction I will take. My starting point is not the "ends of law", but rather the technical side of the «code» modality, and the «code» side of Internet governance.

²²Of recent articles, see for example Katyal (2003), Kesan and Shah (2004).

²³Lessig (1998, p.676)

2 «Code» Governance: An Empirical Analysis

2.1 The Choice of the Object of Analysis

In order to explore the notion of «code» as regulation modality from a computer science perspective, I have chosen to empirically analyze a *self-organized, self-governed application in the Internet*.

In order to explore such a novel conception like «code» as regulation modality, an *empirical* analysis of an existing governance setting in the Internet seemed sensible; any theoretical considerations would have afforded a deeper legal understanding (since regulation theory is largely an area of legal scholarship).

The next problem arose with the choice of an appropriate setting. My starting point in technology afforded access to the technology itself, i.e. the source code of the Internet setting, which excludes most commercial or corporate settings, and thus the market modality influences. Also, I excluded settings where the influence of law could not be neglected, again due to my focus on the technical side of governance. Strong institutional influences by legal objectives could veil otherwise visible «code» structures; there are studies which show the institutional impact on the resulting technology¹.

Since some kind of group is necessary for a governance situation to arise, this left (in Lessig's model) a setting where norms and «code» had to play an important role: a self-governed application in the Internet. As a last criteria, I needed access to the source code, if possible in different versions, of the application, in order to be able to trace «code» adjustment to social changes. Therefore a setting where the source code is developed by the participants, a self-organized application was necessary.

Given these criteria, I found the ideal object of analysis in the *Internet Relay Chat (IRC)*. This Internet application allows users to employ in real-time group discussions. Created 1989 by a Finnish student, it since then has grown to serve hundreds of thousands of user in many IRC networks. It is

¹Kesan and Shah (2003b)

SELF-ORGANIZED: These networks are run and maintained by volunteers, and large networks also develop their own IRC software, which is available as source code.

SELF-GOVERNED: Users and maintainers are spread around the world, only connected through the Internet so that the influence of law on the social order is negligible. Also due to its voluntary nature, no corporate organization exists, so that neither market nor (corporate) law influence exist.

AN INTERNET APPLICATION: As an application in the Internet, the social setting is entirely based on the *software* of the application. The IRC principals therefore develop and use their software which makes it probable that governance structure in the source code can be traced.

2.2 Main Hypothesis

The choice of the object of analysis already has set some parameters for my main hypothesis pursued in this work.

Main hypothesis

The analysis of the source code of an self-organized, self-governed Internet application reveals distinct «code» features: structures and dynamics in the technology which serve as regulation system, similar to, but distinct from the legal system. These features form the «code» governance of the application.

Specifically, I claim that there are discernible structures in the «code» which do not serve the immediate applications end – the real-time discussion between users – but are (also) governance mechanisms, to help manage the participants their common affairs.

These affairs can include the stability of the network, both technical and social (prevent or resolve disputes), coordination and management between users in the discussion groups, securing contributions for the network etc. In addition to norms guiding the participants, the «code» creates a regulatory environment, constitutes a *regulation system* of the application.

In order to be able to discern these «code» features, I need some concept of what a *regulation system* is constituted of. For my methodological approach, I have chosen two model which help me to pinpoint governance structures in the application code.

2.3 Two Models of Regulation

Recognizing regulative structures in «code» needs some idea of what regulation constitutes of. To this behalf, I have chosen two legal scholarly concepts which elucidate some basic aspects

of regulation and regulation systems.

In the first model, the author explains the «code is law» notion by looking for key aspects of the legal system in «code», or "Lex Informatica". These key aspects offer a framework for a regulation system which I will further examine using the example of the Internet Relay Chat.

The second model is concerned with what kind of rules exist. The author distinguishes five types of rules each serving different purposes in a regulation system: expressing substantive objectives, or determine the amount for sanctions, etc. An identification of such rule types in the IRC «code» is another strong indication for the existence of a «code» regulation system.

2.3.1 "Lex Informatica": Key Aspects of a Regulation System

Similar and in parallel to Lessig's concept of «code is law», Joel R. Reidenberg developed the concept of *Lex Informatica*, suggesting that information policy notes are formulated through technology². The core of his article is a comparison of Lex Informatica with the "key concepts of a legal regime"³, such as contents, source of the regulation system, or its primary enforcement.

	Legal Regulation	Lex Informatica
Framework	Law	Architecture
Jurisdiction	Physical Territory	Network
Content	Statutory/Court Expression	Technical Capabilities Customary Practice
Source	State	Technologists
Customized Rules	Contract	Configuration
Customization Process	Low Cost Moderate cost standard form High cost negotiation	Off-the-shelf configuration Installable configuration user choice
Primary Enforcement	Court	Automated, Self-execution

Table 2.1: Lex Informatica (Reidenberg, 1998, p.569)

These key concepts serve me as framework for my explorations of the IRC «code» regulation system. I therefore briefly introduce each of these concepts as outlined by Reidenberg⁴:

FRAMEWORK – This is the "basic building block" of the respective regulation. Reidenberg names *law* as basic framework of the legal system, and *architecture*, rather "architectural

²Reidenberg (1998)

³ibid., p.569.

⁴All citations from Reidenberg (1998, pp.569-573).

standards” as the framework of the Lex Informatica, because these standards ”define the basic structure and defaults of information flows on a communications network”.

JURISDICTION – The scope of the respective regulation regime. In the legal system, laws are generally limited by the physical territory. As equivalent, Reidenberg names ”network” as the scope inside which the information rules are applicable, without any further specification.

CONTENT – This is how the ”substantive content” of the regime are expressed or derived from. In the legal system, the ”statutory language, government interpretation, and court decisions” form the main content. Reidenberg offers ”technical capabilities and customary practices” for the «code».

SOURCE – The source of the ”default rules”. In law, the *state* with its ”political-governance process ordinarily establishes the substantive law of the land”. In Lex Informatica, it is the ”technology developer and the social process by which customary uses evolve”. Reidenberg distinguishes between two sources: the ”technologists” create the ”technical standards” and products, and the user ”adopts precise interpretations through practices”.

CUSTOMIZED RULES – In law, *contracts* are an instrument which gives the governed (persons, organizations) a means to ”deviate from the law’s default rules and to customize the relationship between the parties” (inside legal constraints). Reidenberg likens it to the *configuration* of technology. And as in the legal realm, configuration is only possible if the ”architectural standards support the deviations”.

CUSTOMIZATION PROCESS: In both regulation systems, there are several kinds of customizations which differ in the cost for the parties involved. Reidenberg claims that the processes ”show a number of significant differences” between the two systems, but only identifies a ”wider range of options” in Lex Informatica, compared to the legal system.

PRIMARY ENFORCEMENT – Here the two regulation systems differ considerably. While legal rules have to be enforced by ”juridical authorities”, and violations have to be ”pursued on an ex post basis before the courts”, Reidenberg sees the enforcement in Lex Informatica as ”automated and self-executing”: The design can ”prevent actions from taking place without the proper permissions or authority”, through mechanisms like cryptography, passwords, etc. In contrast to law, this is an ”ex ante enforcement” implemented in the capabilities of the technical system.

Taken together, this model serves as a good framework for the exploration of the IRC «code» governance. In chapter 9 I return to this model and examine these key aspects in the light of the results of my empirical analysis.

2.3.2 Types of Rules

While Reidenberg's model enables me to analyze the general regulation system features, the "types of rules" as suggested by Robert C. Ellickson⁵ provides a concept of *rules*. In this case, the comparison with the law does not help: Laws work by "threat[ening] *ex post* sanction for the violation of legal rights"⁶, is put down in writing, either as court decisions or statutory law, and "depends primarily on judicial authorities for rule enforcement"⁷. «Code» on the other hand works differently: it is 'built-in' into the technology one uses and, as Reidenberg puts it, "allows for automated and self-executing rule enforcement"⁸, as often not even as a visible sanction, but invisibly preventing some actions⁹. Therefore some kind of abstraction from these features was needed to understand how «code» specifically regulates.

Ellickson embeds the "types of rules" into a larger context, integrating types of sanctions, controllers and rules into a "comprehensive system of social control"¹⁰, on top of an extensive empirical analysis of the behavior and disputes between cattlers in Shasta County, California. Seen from Lessig's modality model, he examines the interplay between law and social norms.

For my analysis, I apply only this part of the larger model which gives me some understanding of the different forms that regulation can take. Ellickson distinguishes five types of rules¹¹:

SUBSTANTIVE RULES: These are rules which "define what primary conduct [...] is to be punished, rewarded, or left alone", which in law would state a "you shall not..." kind of rule. One example in cyberspace would be the limitation of the number of members in an AOL chat room to twenty-three¹².

REMEDIAL RULES: These rules determine the "nature and amount of the sanction", in case that a substantive rule has triggered one. In law, monetary fines, imprisonment would be examples. In the case of the AOL chat room, the remedial rule might be a simple entry rejection.

PROCEDURAL RULES: In order to apply sanctions, someone¹³ (a controller) has to decide if it is applied or not. For this, the controller needs a basis upon which she can decide. Procedural rules determine "how controllers are to obtain and weigh information" for their sanctioning decision. Court rules (code of procedures) are an example for such rules.

⁵Ellickson (1991)

⁶Lessig (1999a, p.89); emphasis in original.

⁷Reidenberg (1998, p.572)

⁸ibid.

⁹Lessig (1998, pp.677-80) suggests here the distinction between "objective" and subjective" constraints.

¹⁰Ellickson (1991, p.123)

¹¹The following description including all quotes follow Ellickson (1991, 132-6).

¹²Lessig (1999a, 68-9)

¹³In «code», this also could be a "something", some technical routine.

The «code» equivalent might be monitoring systems, or AOL collecting information about members and their activities inside the system¹⁴.

CONSTITUTIVE RULES: In addition to rules which affect the conduct of individuals, other rules have to specify the conduct of the controllers. Constitutive rule "govern the internal structures of controllers". In law, the organizational structures of governmental entities come to mind; Ellickson also gives the example of a constitutive norm, in which controllers should build a "continuing relationship" between themselves rather than acting as "loners".

CONTROLLER-SELECTING RULES: Finally, these rules determine the "division of social control labor among the various controllers". This is most evident when considering the above given definition of governance, where many kinds of controllers "manage their common affairs"¹⁵. Also, in most settings, the four regulation modalities are somehow coordinated, either explicitly or implicitly. One example for a controller-selecting rule might be the predominance of law over other modalities, implicitly assumed by most legal scholars¹⁶.

These five type will help me to identify candidates for «code» regulation in the source code of the Internet Relay Chat, when they formulate substantive rules, determine some amount of sanctions applied, or information for officials the help them in the sanctioning decision; and «code» that structures the controllers organization, or coordinates between different kinds of controllers.

2.4 Technical Notes on the Analysis

Before I proceed to the empirical analysis of the Internet Relay Chat, some technical remarks regarding my strategy are due.

My main object of analysis is the source code of the Internet Relay Chat. The IRC consists of many networks which, independent of each other, provide chat functionality over the Internet. These networks in turn consist of IRC servers who relay the messages of the IRC users between themselves. Therefore, the main technical component of an IRC network is the server.

My explorations therefore rely mainly on the analysis of the source code of the IRC servers. Such server source code are only developed by a few of the larger networks. Of these, I have

¹⁴Lessig (1999a, p.69)

¹⁵See above on page 25

¹⁶See for example Lessig (1998, p.666): He sees the non-law modalities "as each *subject* to law [...], each itself an object of law's regulation" (emphasis in original).

included those of four networks in my analysis: Onet, EFnet, Undernet, and IRCnet¹⁷. In the case of the EFnet also, several server series exist which are developed and used in parallel.

In total, I have found the source code for approximately 200 server versions. They are listed in chapter 14 of the appendix. This server code was then examined for «code» related features, in single server versions and through a succession of versions, in order to track changes such as extension or addition of features, or their removal, and between different network versions, comparing similar features with different design or implementation. These examinations were largely done 'manually', by reading the source code. Some technical help, especially for the longitudinal analyses were applied. A short account of the tools used is presented in chapter 13 of the appendix.

In addition to the server code, the source code of two bots were examined as well: the Eggdrop user bot¹⁸ and the Undernet UWorld service bot¹⁹. In both cases only one version was examined.

Next to the source code, the second source of information of the IRC were various documents, both from the networks themselves as well as from other sources. This collection has not been done in a systematic matter, but rather as continuing search for interesting information related to governance structures and events which could be also traced in the source code. The sources used are listed in the bibliography. A problem with these documents was that they sometimes were updated, and older versions not available any more; others disappeared entirely. I have tried to only use those which are still accessible; but all used documents are on file with me.

Notation

Finally here are some technical remarks regarding this work.

DATE FORMAT – I have chosen here two formats: For a full date including the day, the format follows the international standard date notation²⁰, which is YYYY-MM-DD (four year digits, two month digits, two day digits)²¹. When examining the source code and documents, the various date formats were a continuous source of confusion²², leading me to use this notation. I have made one exception: when only month and year were necessary, I chose to write the full month name, followed by the year (e.g., "May 1995"). This occurs mostly in connection with the server code packages. Here it is sometimes

¹⁷I have also examined features of a fourth network, DALnet. Unfortunately, only a few server versions were available, so in this case my explorations rely solely on documents available on this network.

¹⁸see below chapter 6.2.1

¹⁹See below chapter 7.4

²⁰ISO 8601 of the International Organization for Standardization (ISO)

²¹See <http://www.cl.cam.ac.uk/~mgk25/iso-time.html>.

²²Although most of these sources were written in English, their authors came from many countries, so that all kinds of date notations were used.

confusing to track the dates of the many versions, so in order to somewhat alleviate this, I chose to put the date into a slightly more readable format.

”ALGORITHM” – Direct quotes from the source code, when several lines long, are put into a table-like structure called ”algorithm”. The word processing software used for this work²³ offered this functionality, together with an automatic generation of a ”table of algorithms”, so that I made use of this feature²⁴.

CODE QUOTES – Apart from the ”algorithms”, short quotes of source code as well as other text that appear verbatim in the IRC (commands, channel names, etc.) are set in the `typewriter` font.

SOURCE CODE CITATIONS – The notation for quotes from the source code packages has the following format: enclosed in square brackets, the name of the source code as listed in appendix chapter 14, followed by the path to the file in this package and the file-name, and optionally followed by a colon and the lines inside the file. For example, [irc2.8.21+CSr20/include/comstud.h:19-27] points to lines 19 to 27 of the file ”comstud.h” in the the folder ”include” of the source code package ”irc2.8.21+CSr20”.

²³LyX 1.3.5, a front end word processing software for the L^AT_EX typesetting system.

²⁴The «code» constraint in this software was the difficulty, and my inability to change the name ”algorithm” into something more appropriate. This would have necessitated a direct code change in both LyX and L^AT_EX, a task that I chose not to undertake for this feature.

Part II

Empirical Analysis of the Internet Relay Chat

3 The Internet Relay Chat

In this Part II, I analyze the Internet Relay Chat, a self-organized and self-governed application in the Internet. It may be not as widely known as the world wide web, e-mail or file-sharing applications; still it remains an important application in the Internet, used by hundreds of thousands of people at any time of day or night.

The Internet Relay Chat, or short IRC, offers its users a medium to exchange text messages in real time, both within a group of people, or between single users. Group conversations take place inside of so-called *channels*, of which literally thousands can exist in an IRC network, and can be shaped by its users.

The whole application is maintained and developed by volunteers, with no overarching corporate entity or other single organization behind them. Moreover, "the" IRC encompasses thousands of independent *IRC networks*, from small one-server 'networks' to large ones made up by up to 100 servers connected to each other. Each of these networks form their own social setting, with administrators and their helpers managing the servers and network, and its users interacting by using the IRC network, contributing in various ways to it, or causing disruptions and conflicts which have to be remedied or resolved.

All these actions are mediated by the IRC «code», the IRC software, which is developed by the participants themselves. «Code» offers them a powerful means to shape their own governance environment, to manage their common affairs, and to cope with changing social conditions. The main objective of this work is to examine this «code» in its relationship to and interactions with the social setting of these IRC networks.

So what is the Internet Relay Chat? I have chosen to give an three-partite answer:

- Section 3.1 begins with a hypothetical scenario where the steps to become an IRC users are outlined. Along the way, important concepts such as *nicknames* and *channels* are introduced. In addition, this section gives basic information about existing IRC networks, such as size and growth.
- Section 3.2 provides the *technical foundation* of the IRC: the IRC network architecture and the IRC server and IRC client as its main components, the protocols involved, as well as the basic configuration settings of the software.

- Although I mainly examine the technical structures and dynamics, it is important to shed some light on the social side of the IRC to understand the governance situation. Section 3.3 offers a first glimpse by outlining the main social roles: IRC administrator, service administrator and coder; the IRC user, as well as IRC operator and channel operator.

This overview of the functional side, the technical structure and the social setting sets the foundation for the examination of notable features and development with regard to the IRC «code» governance in subsequent chapters.

3.1 The Functional Perspective – Using the IRC

3.1.1 A hypothetical user scenario

As a first introduction, I describe how a user connects to an IRC network, what basic functions are available to her, and what she gets displayed on the screen. In addition to giving an impression on how users experience the IRC, it gives me the opportunity to introduce some basic concepts of the IRC, such as IRC client software, channels, IRC commands, etc.

There are basically four steps necessary to get involved into a chat conversation on the IRC: 1. installing the client software and choosing an IRC network and server; 2. connecting to that server; and finally, 3. entering and using the IRC.

Installing IRC client software: The first step in using the IRC is to acquire and install the *IRC client software*. This is an program similar to web browsers or e-mail programs, offering a user interface to the IRC and handling all necessary connection procedures and data transfers to the IRC servers. IRC clients exist for many computing platforms¹, and with different feature sets; some offer a text-only interface, while recent ones sport graphical user interfaces.

Choosing an IRC network: An IRC network is a set of interconnected servers which together form an entity whose members can chat with each other. In order to connect to one of these networks, one needs to find the connection information for one of its IRC servers; this is similar to finding an URL for a website. Most IRC client programs come with a default list of servers to which one can connect. Other sources exist which provide up-to-date information about the IRC networks². And not at least, many larger IRC networks maintain their own websites where current connection information on servers can be looked up.

Connecting to the IRC: Once the IRC network and server has been chosen, the next step involves the actual connection procedure. The first thing that a user might encounter is the

¹For Windows systems, mIRC (<http://www.mirc.com/>) is said to be the most popular IRC client; for Macintosh, Snak (<http://www.snak.com/Snak.html>) and Ircle (<http://www.ircle.com/>) are the most popular ones.

²Examples for such websites are <http://irchelp.org/> and <http://irc.netsplit.de/>.

rejection of connection. Many IRC networks have specific policies regarding who may connect, and from where. A common reason for these policies concerns the number of clients that one server can concurrently accept. Especially in large networks, servers of one country may decide to reject clients from other countries (based on the domain name of the client), so that the users of the network are more evenly distributed between the servers of the network.

Once a server accepts the connection request, the user has to identify herself by choosing a *nickname*. This is a pseudonym that every IRC user has to choose and is referred to inside the network. This nickname has to be unique inside the network; if another user with the same nickname already exists, the connecting user has to choose another nickname. Only when a unique nickname has been found, the user is granted entry to the IRC network.

Using the IRC: Once entered, the IRC server immediately sends a number of text lines which contain basic information about the network, the server, and any other information that the server administrator deems important, such as pointers to help information, usage policies etc. (figure 3.1).

```
*** Welcome to the UnderNet IRC Network, zyxwvu
*** global: File not found
*** If you have not already done so, please read the new user information with /HELP NEWUSER
*** Your host is Amsterdam.NL.EU.undernet.org, running version u2.10.11.07(pre3)
*** This server was created Mon Sep 13 2004 at 21: 09:54 CEST
*** Amsterdam.NL.EU.undernet.org u2.10.11.07(pre3) dioswkgx biklmnopstvr bklov
*** WHOX WALLCHOPS WALLVOICES USERIP CPRIVMSG CNOTICE SILENCE=15 MODES=6 MAX-
CHANNELS=10 MAXBANS=45
+NICKLEN=12 MAXNICKLEN=15 are supported by this server
*** TOPICLEN=160 AWAYLEN=160 KICKLEN=160 CHANTYPES=#& PREFIX=(ov)@+ CHAN-
MODES=b,k,l,imnpstr
+CASEMAPPING=rfc1459 NETWORK=UnderNet are supported by this server
*** There are 56028 users and 81380 invisible on 35 servers
*** 100 operator(s) online
*** 459 unknown connection(s)
*** 49991 channels formed
*** I have 5009 clients and 1 servers
*** Highest connection count: 5026 (5025 clients)
*** - Amsterdam.NL.EU.undernet.org Message of the Day -
*** This service is provided by EuroNet Internet & Wanadoo - http://www.wanadoo.nl
*** Type /MOTD to read the AUP before continuing using this service.
*** The message of the day was last changed: 2003-7-10 21:21
*** on 1 ca 1(2) ft 10(10)
```

Figure 3.1: Initial messages when joining an IRC network (Undernet, on Jan 6, 2004)

Once inside the IRC, there are three basic methods by which a user communicate with others: channels, private messages, and CTCP messages.

3.1.1.1 Channels

The most important means of communication in the IRC are group conversations inside *channels*³, also sometimes referred to as chat rooms. Users enter (“join”) channels, and then can

³For a detailed account of channels as seen by an IRC coder, see Kalt (2000b).

3 The Internet Relay Chat

submit text lines which are immediately sent to all other members of that channel. Every line that a channel member sends to the channel is immediately displayed by each members' client program, leading to a stream of text lines which, to a novice user, may appear somewhat confusing:

```
<punky> nope
<deepee> You need logs. And proof of chan "ownership" and you need to persauade the irc
ops on the offenders server. A toughie.
<Bob_> Hmm, well looks like I will have to use my influence with BT to get ircop status.
*** BambiEyes (henrik@ppp01.prosalg.no) has joined channel #hottub
<BambiEyes> hi
<Bob_> obviously it will take sometime.
*** BambiEyes has left channel #hottub
<Bob_> Hi BamBi
<deepee> But that wont help unless they are connected through the bt server tho?
<deepee> Do we have any nicks/addresses to go by?
<Bob_> Yea but that way we could force a colide
```

Figure 3.2: Example conversation in the IRC channel "#hottub"

The conversation inside a channel, as figure 3.2 shows, is a continuous stream of text lines. Each message is prepended by the writer's nickname. Channel status messages are interspersed in the conversation, distinguished by the other messages through a leading of three asterisks ("***").

In the example, a user `henrik` from the server `ppp01.prosalg.no` with nickname `BambiEyes`, can be seen to have joined the channel `#hottub`, and, after saying "hi", left it again.

Channels can have various properties, called *channel modes*, which alter various communication characteristics. For example, it is possible to deny entry to most or specific users, or to allow only chosen members to talk on the channel.

These channel modes are decided by the role of the *channel operator*. This role is assigned to the user who created a new channel, just by joining a previously non-existent channel. This means that any user, be it novice or long-time IRC user, can at any time open up new channels in the IRC, and immediately become channel operator for that channel. As channels are the main communication structure in the IRC, later chapters will examine its various properties and uses.

3.1.1.2 Private Messages

In addition to channels, users can also send text messages privately to other single users or a group. Some networks also allow users to send messages to all users in the network; but often, this function is reserved to network officials such as the IRC operators⁴ for announcements etc.

⁴For the role of the IRC operator, see below section 3.3.5 and chapter 7.

Another important use of the private message mechanism is the interaction with automated clients, such as "bots"⁵ and "services"⁶. These connect to the IRC as clients, identified by a nickname, like (human) users, but are programs or scripts which provide diverse functions. Users send commands to the bots via private messages.

3.1.1.3 CTCP (Client-to-Client Protocol)

A special method of messaging is implemented as an extension of private messages. Basically, the Client-to-client protocol or short CTCP⁷ allows two IRC *client programs* to exchange command requests and responses with each other, using the private message mechanism as a transport medium. To IRC servers, these requests and responses appear as if users exchange text messages. But on the client software sides, the request is interpreted, and a response sent back.

One of the main uses for the CTCP mechanism is the *establishment* of a the *direct* Internet connection between two IRC clients outside the IRC network, called the DCC (Direct Client-to-Client) mechanism: once established, a DCC connection is not relayed over the IRC network any more, but two IRC clients connect to each other over the Internet, circumventing the IRC network altogether. With such a DCC connection, users can chat with each other without fear that an IRC server might eavesdrop on it⁸, or exchange large files between the IRC clients instead of being relayed over the IRC network. Another important use of DCC is the interaction with IRC bots⁹.

3.1.1.4 IRC Commands

User interact inside the IRC through commands. These commands are entered in the same way as normal text messages. So in order to distinguish between text messages and commands, the latter are prepended with a slash ('/').

For example, a user joins a channel by issuing `/join #hottub`, leave it with `/leave #hottub`, or sends a private message to another user with `/msg WildThang`.

Other commands allow the user to request information from the system: For example, information about other users is retrieved with the `/who` or `/whois` commands. The `/list` command returns a list of channels with basic information about each one. The following figure 3.3 gives an example for such a channel list¹⁰:

⁵See below chapter 6.2.

⁶See below chapter 6.3.

⁷See Undernet-User-Committee (1997a)

⁸The original document therefore claims it to be "the ultimate in secure chat connections while still in an IRC oriented protocol" (Rollo (1992)).

⁹See below chapter 6.2.

¹⁰The first column after the three asterisks show the channel name, usually prepended by a hash mark ('#'). After the name, the current number of participants are shown, followed by the topic string which is supposed,

3 The Internet Relay Chat

```
*** #de_ce_nu_ 2      salutare
*** #flandra 2      Manchester United RULEzzzz
*** #lovebird 18     03,1Welcome 03,12To03,4LoveBird.03,5Enjoy Your Stay Here.
*** #doar`noi 1      004,120`Wake up.., from yourself...listen..,live...!!`00
*** #KUK-Kanal 1     h?r h?nger alla vi som qillar att suppa kuk!
*** #Ud^peopl 1      Aici intra cine vrea--cu injuraturi si cu ce vreti voi---Channel made by
+asas
*** #gogo 4          baaaaaaa tigani din mikro ?????????????????
*** #MANSON_NI 1     04,1 |[DAVID]| 011,4 MANSON_NIGHTWISH_SEVEN@YAHOO.COM 04,1 |[DAVID]|
*** #oilpunx 2       First they ignore you, then they laugh at you, then they fight you, then you
+win.
*** #eagle_33 7      CINE RADE LA URMA E MAI GREU DE CAP :)))
*** #anime_x 1       bien benido anime_x muy pronto tendremos descargas de series
```

Figure 3.3: Snapshot of a channel list in the Undernet

Yet other commands are reserved to privileged users, such as channel operators or IRC operators, and help with their administrative tasks.

As a rule, all user actions in the Internet Relay Chat are initiated with commands, some of which are examined in detail in later chapters.

3.1.2 Size and Growth of the IRC

Before I describe the main technical concepts, some statistical data on the size and growth of the IRC since its inception in 1988 should give an impression on the rank of the IRC among the Internet applications.

There are a number of indicators which can serve to judge the size of the IRC: the number of users and channels, the number of different IRC networks, and the number of servers, both in the IRC as whole, as well as in each IRC network. Unfortunately, there is only scattered data for the first ten years of the IRC. For data from 1998 on, there are a some websites which offer systematic data¹¹, although their focus lies on current usage, not long-term statistics. Still, from what is available, I can offer a sketch of the size and growth of the Internet.

Figure 3.4 shows the number of users and servers early in the IRC history. I have collected these from various sources, such as "history" documents, as well as some IRC mailing list messages, so the data may be of varying accuracy, and serve only to give an impression of the size and growth in these times.

This data indicates some characteristics of the early IRC. In the first two years, the (at that time only) IRC network was apparently used by those who also ran an server. Therefore it can be assumed that the IRC users constituted a homogeneous, tight-knit community. Then, in the first half of the 1990s, the usership began to show a fast growth which continues to this day: from the 400 users in 1991 through about 5000 in late 1994 to recently more than 1.3 million

but obviously not limited, to give some information about the channel topic. This snapshot was taken at the same session as the initial server messages shown above (figure 3.1), and therefore shows a list of eleven out of 49991 channels.

¹¹See for example <http://irc.netsplit.de/>, <http://www.hinner.com/ircstat/>.

3.1 The Functional Perspective – Using the IRC

Date	Users	Servers	Comments	Sources
1988-08	—	1	first irc server	(1)
1990-05	—	100	dropped down again soon	(1), (2)
1990-07	12	38	source (1) says: on "average"	(1), (2)
1990-09	—	117		(2)
1990-09	41	86	same source, different numbers?	(2)
1990-09	—	12-18 backbones 117 servers	backbones are US only	(5), (6), (7)
1990-10	—	15 backbones	US only	(8)
1990-12		19 backbones	US only	
1991-01	300	—	gulf war fame	(1), (2)
1991-03		135	69 in US, 66 non-US	(2)
1991-04	240		median number	(2)
1991-10	399	120	44 opers; users hit 500 at one time	(2)
1992-03	606	—	(Nystrom 1993)	
1993-04	2300	—	(Nystrom 1993)	
1994-07	—	188	EFnet (irc-20040107/server.list)	
late 1994	5000	—		(2)
1995-10	15000	—		(2)

Sources: (1) Stenberg (1998); (2) Rose and Ian (1999); (5) Rose, Helen (1990-09-08) Backbone Routing 9/7/90. Mailing list IRClst (1991); (6) Lindahl, Greg (1990-09-18) Provisional backbone for eris-free net. Mailing list IRClst (1991); (7) Lindahl, Greg (1990-09-20) eris-free US net backbone list, one change. Mailing list IRClst (1991); (8) Rose, Helen (1990-10-23) Backbone plan 23 October. Mailing list IRClst (1991).

Figure 3.4: IRC growth from 1988 to late 1995

users worldwide on over 400000 channels¹².

Until 1990, there was only one network, later called the "Onet" (for the 'original IRC net'). In September 1990, a dispute between the administrators led to a split of the Onet into two networks, of which only one, the EFnet, survived¹³. From 1993 on, parallel IRC networks appeared: the Undernet (since 1993), DALnet (1994), and the IRCnet (1996); for some time, these four constituted the 'four largest' networks.

From mid-1995 on, other networks also formed¹⁴. One source has a count of 88 active networks in October 1998, with 5 "main" networks (>10000 users each), 10 "big" ones (1000-5000 users), 16 "medium" (100-999) and 57 "small" ones (<100 users)¹⁵. In June 2001, the same source lists over 300 networks. Recent numbers give a count of 679 networks in May 2003¹⁶, and 2428 networks in January 2005¹⁷.

¹²<http://www.searchirc.com/networks> (2005-01-08)

¹³See also below chapter 8.2.

¹⁴Some of these are: AUSTnet, Galaxynet, NewNet, WebChat (1996); Quakenet, RelicNet (1997).

¹⁵Source: <http://netsplit.de/networks.19981013/> (2001-06-22)

¹⁶<http://searchirc.com/> (2003-05-16)

¹⁷<http://searchirc.com/> (2005-01-08)

3.2 A Conceptual View of the Technology of IRC

This section offers a conceptual view on the *technology* that constitutes the Internet Relay Chat. The technical capabilities determine what can be done in the IRC, and they therefore define its governance properties. This affords a basic understanding of what happens on the technical level.

A working model of the IRC from the technical perspective¹⁸ is presented, including:

- The (technical) "polity" of IRC: A structural description of the elements that constitute an IRC network: IRC server, client, user bot, and service bot.
- The "politics" of IRC: A processual view of the IRC network, with the IRC server as main component
- The "policy" of IRC: How policy objectives of the IRC setting are implemented in the software. I will concentrate here on the means of how an IRC admin can change the workings of the IRC server code, and thus influence its «code» policy implementation.

3.2.1 "Polity": Structural Overview of the IRC

One *IRC network* is a set of IRC servers, interconnected in a specific way (topology) and sharing data (data distribution). Each *IRC server*, a node in the IRC network, is a software program running as a process on a host (a computer connected to the Internet), and managed by an IRC administrator. Another software, the *IRC client* is employed by an IRC user to connect to one of the servers in order to communicate with other members of that network.

A special case of IRC client is an *IRC user bot*: Users employ a bot to automate specific tasks executed upon specific events. A special case of an IRC server, but similar to an IRC user bot is the *IRC service bot*: It automates specific tasks, such as channel or nickname registration; but in contrast to a user bot, a service bot is given (privileged) server status by the other servers.

Figure 3.5 shows the basic relationship between these entities of a small IRC network with three servers and two users/three clients.

¹⁸This chapter concentrates on those concepts and structures important for the examination in subsequent chapters. For a detailed account of the technology behind the IRC from actual IRC coders, see Oikarinen and Reed (1993), and Kalt (2000a,b,c,d).

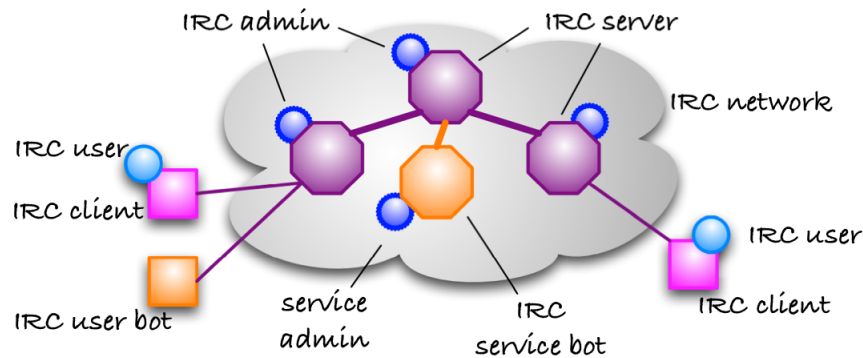


Figure 3.5: Elements of an IRC network

3.2.1.1 IRC Server

The server forms the backbone of IRC as it is the only component of the protocol which is able to link all the other components together: it provides a point to which clients may connect to talk to each other [IRC-CLIENT], and a point for other servers to connect to [IRC-SERVER]. The server is also responsible for providing the basic services defined by the IRC protocol.¹⁹

The IRC server is the central component of the IRC network: It serves as connecting point for the clients, and relays messages to the other servers and clients.

Technically, the server is a process, running on a host connected to the Internet. As a server, it waits for incoming requests from the connected clients and other servers, process these requests and then returns the results to the appropriate users and servers. It also maintains an up-to-date system state of the entire network, including a list of users, channels, other servers in the entire network. In order to keep this information current, all changes made by one server are immediately propagated to all other servers.

Section 3.2.2.2 below presents a processual view of the server.

3.2.1.2 IRC Client

A client is anything connecting to a server that is not another server. [...]

User clients are generally programs providing a text based interface that is used to communicate interactively via IRC. This particular type of clients is often referred as "users".²⁰

IRC client programs have already been briefly introduced in section 3.1.1 above. They are programs which give the user access to the IRC, similar to a web browser giving access to the world wide web. IRC clients exist for many computing platforms²¹, and with different feature sets. Some offer a text-only interface, while many have a graphical user interface.

¹⁹Kalt (2000a, pp.2-3)

²⁰Kalt (2000a, p.3)

²¹For Windows systems, mIRC (<http://www.mirc.com/>) is said to be the most popular IRC client; for Macintosh, Snak (<http://www.snak.com/Snak.html>) and Ircle (<http://www.ircle.com/>) are the most popular ones.

Technically, IRC clients manage the connection to an IRC server, translate the user commands into messages according to the IRC client-server protocol²², and send them to the connected server. The responses from the server are then displayed to the user. One important point here is that IRC networks often change details in their command and feature set, leading to changes in the protocol. IRC client developers therefore continuously adapt their software to cope with these changes.

3.2.1.3 IRC User Bot

With IRC client programs, users interact directly with the IRC network, sending commands and messages, and receiving them. But these direct user interactions can be automated, recorded into or written as programs or scripts which then are executed by the user to achieve some outcomes. In the IRC, client programs which allow the user to automate tasks are called *IRC bots*.

Such automated tasks began with the introduction of a scripting facility in a popular IRC client, *ircII*²³. Created in 1989 as first independent IRC client²⁴, the developer Michael Sandrof included the ability to create and run scripts: Users write a succession of commands into a text file which then can be executed. Also included are features like variables, parameter substitution, and most importantly, the ability of a script to act on a message received from the connected IRC network.

As simple example, the following line changes the default text for a `/join` message:

```
/on ^join * /echo $0 enters cavern $125
```

This rewrites the message before it is displayed to the user, replacing the placeholder `$0` with the name of the user who entered, and `$1` with the channel name. So, the `/join` message on line 5 in figure on page 42 on line 5 would be rewritten from

```
*** BambiEyes (henrik@ppp01.prosalg.no) has joined channel #hottub
```

into

```
*** BambiEyes (henrik@ppp01.prosalg.no) enters cavern #hottub
```

The `/on` scripting command not only reacts on command messages like `/join` in this example, but allows for sophisticated pattern matching on incoming messages. So while the above

²²See below section 3.2.2.1.

²³See <http://www.irchelp.org/irchelp/ircii/>

²⁴The server software versions also included a IRC client program which offered a basic functionality without the advanced features of *ircII* and later developed IRC clients.

²⁵File [irc2.5.1.bu.08/clients/ircII2.02/script/cavern], line 6.

is a very simple example, the scripting facility of the ircII client give users a powerful tool to shape their IRC environment, automating all kinds of tasks. Consequently, all but the most simple IRC client programs nowadays offer some kind of scripting facility similar to that of the ircII.

The scripting facility in IRC clients is geared towards the interactive use, but another class of bots exists which can be seen as IRC bots in a more narrow sense. These are programs which are continuously connected to the IRC, waiting on some messages to initiate a script. This scheme is similar to servers, like the IRC server; but as IRC bots connect to the network as an IRC client, they have no special capabilities (such as IRC servers do) beyond that of an IRC client. Still, their programmability and other features makes them a powerful tool for users and allows them to actively shape their IRC environment. Chapter 6.2.1 examines one specific use of such bots, where users employ them to change the default channel ownership policies of the network.

3.2.1.4 IRC Service

Unlike users, service clients are not intended to be used manually nor for talking. They have a more limited access to the chat functions of the protocol, while optionally having access to more private data from the servers.

Services are typically automatons used to provide some kind of service (not necessarily related to IRC itself) to users. An example is a service collecting statistics about the origin of users connected on the IRC network.²⁶

Services are a kind of cross-section between IRC bots and IRC servers: Like the former, they are automated programs which provide some special functions. In contrast to bots though, services enjoy a status in the network similar to servers²⁷.

Functionally, services are processes similar to servers, waiting for a request which they then process and send a reply. But in contrast to servers, there provide only a few specific tasks that they were created for. For example, some networks have installed a channel service: a central point where user can register new channels to reserve the right (or privilege) to manage them over a longer period. There is one single point, the service, which proceeds all managerial tasks relevant to the channel service. Its maintainer are the only one in the network who can alter its working. This is in contrast to all other IRC functions and services installed by the whole set of IRC servers in the network in a distributed way. Services constitute a single centralized point of power in the IRC network.

Examples for services (examined in detail later on) are nickname and channel registration services²⁸, and the centralized network control implemented by the UWorld service²⁹.

²⁶Kalt (2000a, p.3)

²⁷For detailed information, see for example <http://www.ircservices.esper.net/docs/1.html> (2005-01-15).

²⁸See below chapter 6.3.

²⁹See below chapter 7.4.

3.2.1.5 IRC Network

From the onset on, the IRC has been programmed to be a multi-server network which hides the fact that multiple servers may be involved by posing to the client as a "virtual" server: from the viewpoint of the client, the server network behaves (almost) as if there is only one server present. This certain setup of different computers, processes or (more technically) automata is called a *distributed system*.

Distributed System



In a distributed system, a number of components, connected over a network, provides services in a way that to the users it appears to them as one single entity. This means that the distribution is hidden from the user, in contrast to normal networks, where users are aware of the different components, such as hosts or processes.³⁰

Distributed systems are very common in the Internet, because often one computer does not suffice to provide a specific service for many users. A classic example is the Domain Name System (DNS) in which the resolution process of domain names into Internet names (as well as other Internet-related data) are maintained and served in a distributed manner³¹.

Two characteristics of the IRC server network are important regarding the code governance aspects: The *topology* and *data distribution* concept chosen.

Topology

The IRC servers are connected to each other in a tree topology, or more technically, an acyclic graph. This means that the server are connected to each other so that there is exactly one path to any other server in the tree (see figure 3.6).

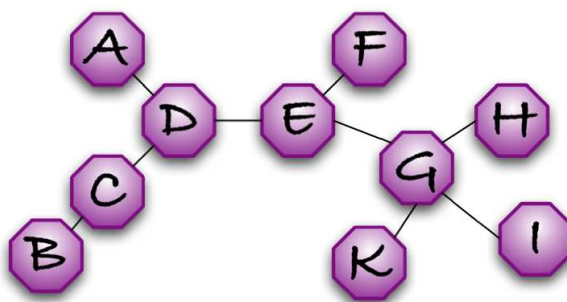


Figure 3.6: Tree topology

³⁰See "Free On-line Dictionary of Computing (09 FEB 02)", according to [http://dict.die.net/distributed system/](http://dict.die.net/distributed%20system/) (2004-03-04).

³¹See below chapter 8.1

A tree topology is often depicted in form of a tree, which gives the impression that the nodes form some kind of hierarchical relationship to each other. And hierarchies indeed normally form a kind of tree shape, with the most powerful entity on top, the next powerful entities on the second level, and so forth. The probably best known corresponding hierarchical structure in the Internet is the network of name servers in the Domain Name System (DNS) which implements an hierarchical structure, with the so called "root server" on top, the "top level domain server" and "country code top level domain server" on the second level, and so forth³².

But equating the tree topology with an hierarchy is somewhat misleading, at least in the example of the Internet Relay Chat. Here the importance of one node is not only given by its position in relation to other nodes in the network, but is dependent on a number of other factors, such as connectivity (the data speed of its connections in the Internet, and in relation to the neighbor nodes).

In addition, the IRC has implemented mechanisms which allow for a dynamic reshaping of the network in case of failure of one node, or disruption of a connection between two nodes.

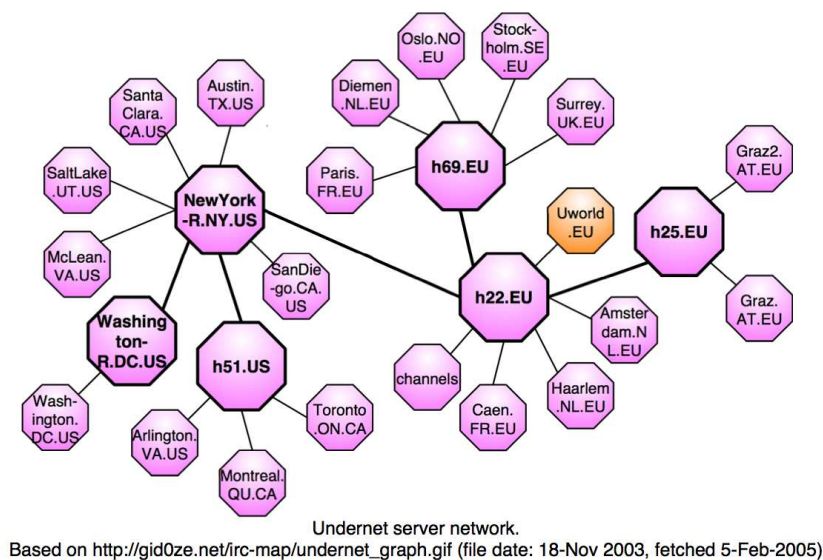


Figure 3.7: Undernet network (around 1993).

Finally, and probably most important, is the relationship of its nodes defined by the specific form of the distribution of the IRC data.

Data distribution

The data of the Internet Relay Chat is comprised of the state of the whole network with regard to its connected servers and users, and the channels. This data is duplicated and maintained

³²See also chapter Section 8.1

in every IRC server; every IRC server holds the actual state of the entire network which is constantly updated by sending every state change to every server in the network.

While at first sight this seems to be very wasteful in terms of bandwidth, this design was chosen exactly in order to save as much bandwidth as possible. At the time of creation, bandwidth was a precious resource, especially since the IRC networks run on university or company hosts, using their bandwidth. Thus an most economical use of this bandwidth was important. The IRC uses the network state information in order to send the main bulk of data, the user messages, only to those servers where the addressees of this message (channel members, receiver of private messages) are connected to.

The design chosen for the IRC server network is a tree topology network, with the state information for the whole network kept current in every single server of the network. We will see in chapter 8 that this design has an important influence on the constitution of the social organization in the IRC: Since every server has the same data, their IRC administrators are more or less on equal standing. There is no single server or group of servers (and their admins) who control crucial data, and thus impose rules on the other servers.

3.2.2 "Politics": A Processual View of the IRC

3.2.2.1 The Client-server architecture

The basic architecture between IRC client and IRC server is shown in figure 3.8. This architecture is known as the *client-server architecture*³³.

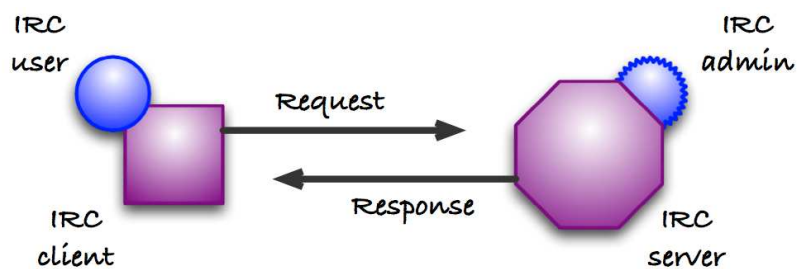


Figure 3.8: Basic IRC client-server configuration

³³Tanenbaum (1989, pp.455-6)

Client-server architecture

This common architecture of many applications in the Internet is based on the principle to separate the requester of a service (the client) from the provider of a service (the server), and to formalize the communication between them in a client-server protocol. In this architecture, the client always initiates the communication by sending a request, while the server never initiates, but only responds to the requests. Often, in order to process the request, or for maintenance tasks, the server can act as a client towards another server, which then constitutes a separate client-server connection. Inside one such connection, only the client acts, and the server responds.

Common applications that use the client-server architecture include the World Wide Web (browser as client, web server as server), e-mail, ftp (file transfer), and the IRC.



In the case of the IRC, the principal design of the IRC client-server protocol has been published two times as Request of Comments (RFC) of the Internet Engineering Task Force: once in 1993, when only one IRC network, the EFnet existed³⁴, and once in 2000, by an IRC coder associated with the IRCnet³⁵. While the principal design is the same in both documents and reflects the basic design of all IRC networks, the specifics in commands, messages and parameters differ from network to network.

3.2.2.2 Inside the IRC server process

Incoming IRC client requests are basically handled in three steps: receive, parse/dispatch, process, and respond. Along these lines, the internal data state is updated, and update messages sent to the other IRC servers:

RECEIVE THE REQUEST – The server maintains a incoming message queue which is provided by the Internet interface of the operating system³⁶. The server polls this queue in regular intervals, and reads in each received request to be further processed.

PARSE/DISPATCH – The read-in request data is a sequence of characters which has to be interpreted according to the format defined in the IRC client-server protocol: The parsing function splits the request data into its functional components, such as the message identifier, the parameters provided by the user, the identifier of the request source, etc. Based on the identifier, the request is dispatched to the appropriate function. For example, the message identifier `LIST`, corresponding to the command `/list`, leads to the function `m_list()` being called in the server.

PROCESS/RESPOND – The function called by the parser does the actual work of the command processing, checking for privileges, changing the appropriate data structures, sending

³⁴Oikarinen and Reed (1993)

³⁵Kalt (2000c)

³⁶In BSD Unix implementations, the so called "socket" interface.

request for data updating to the other servers, notices or messages to other users, and a response to the user who sent the command. In the `/list` example, the function would access the list of channels and send it back to the user.

After handling a request, the server processes some maintenance tasks (such as checking if the connections to other servers are still active, etc.), and then continues with the next request in the incoming queue.

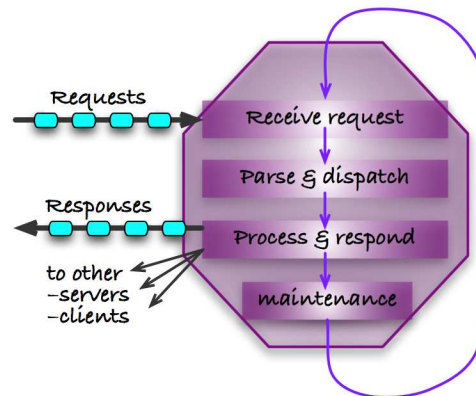


Figure 3.9: Model of the server process

3.2.3 "Policy": IRC Server Installation and Configuration

An important step in the «code» governance setting of a server is how it is installed and set up. Here the IRC administrator makes important decisions on the governance characteristics, such as which users are allowed to join the IRC through that server, what powers the IRC operators acquire, etc.

I sketch the steps leading from the IRC server source code to a running server process. It is assumed here that the IRC admin has set up all necessary prerequisites for the server installation, such as choosing a host, making sure that the software development tools (C compiler, linker, software libraries) exist, and so on.

The entire process can be split up into two phases:

- The *installation itself*, which includes configuring of the source code, and subsequent compiling and linking which results in the binary executable program file of the server.
- The *server configuration*, where configuration files are edited, and startup options determined by the administrator.

3.2.3.1 Installation of the IRC server

The installation of the IRC server begins with the choice of a source code package and possible patches (see below) and ends with an executable binary program file, suitable to be run as IRC server process.

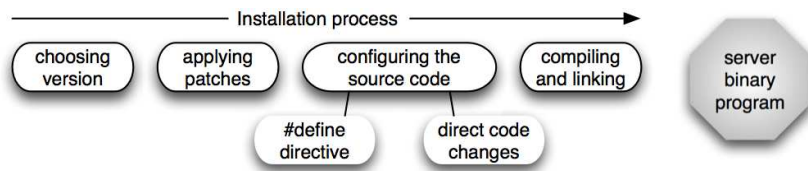


Figure 3.10: IRC server installation steps

A description of each of these steps follows.

Choosing the source code version

In the first step, the IRC administrator chooses the specific IRC source code package. Usually, she can choose between several versions and sometimes even between versions of different code series³⁷. The choice affects what specific features the admin can make available on the server.

In the IRC, the general code policy has been to make new versions backward compatible, to assure that a new version can interoperate with previous versions. Therefore, in IRC networks one can observe that some admins choose not to update their server to a new version, because the older version is perceived as being more stable, or because of the existence or absence of some features.

The same can be said for different server series. The prevalent example is the EFnet, where two series (called ”+CSr”, and ”+th”, which later merges into the ”hybrid” series)³⁸ are available; the servers in these series are interoperable, and therefore can be used concurrently in the EFnet.

Applying patches

In addition to series and versions, there are often a number of so called ”patches”³⁹.

³⁷Server code developed by different individuals or teams.

³⁸See below in the appendix, chapter 14.2

³⁹See [http://en.wikipedia.org/wiki/Patch_\(Unix\)](http://en.wikipedia.org/wiki/Patch_(Unix)) (2005-01-12) for further details.

Patches

Patch files are text files which contain textual differences between two files, or two sets of files. When applied to one file (or one set of files), a new file (or file set) incorporating the changes is created.



The main purpose of this mechanism is to distribute small changes for a larger source code package without having to redistribute the whole package. The distribution of small changes can then even occur in email messages, or in patch files which are considerably smaller than the package itself.

As additional benefit for the IRC, besides the small size of patch files, these patches allow the IRC administrator to install additional functionalities distributed through these patches. Therefore, patches are a method for coders to distribute optional functionalities not incorporated into the main distributed package, a method for IRC admins to choose among different functionalities offered by the available patches.

Configuring the source code

Before the source code is compiled and linked into a binary executable file, the IRC admin configures the source code. This serves various purposes, not the least to determine the compilation and linking settings specific to the computing environment of the server, (software tools, libraries etc.). Besides the general configuration, IRC-specific options are present, some of which are necessary to run the server, and others which change the functionality of the IRC server. These configurations include enabling or disabling of operator privileges, activation of logging facilities or of special commands. Admins configure the source code either by running special configuration scripts provided with the source code package, or manually edit the appropriate files.

Configuration settings are options which are predefined by the coder. It is a means to communicate with the person who installs the software, to present choices affecting how the resulting program works. These choices can be simple technical adaptations to the computing environment, or the prospective uses of the software, but often are also governance choices of the IRC as well.

In the code, several configuration mechanisms exist, different means of entering the options. One of them is the *#define directive*, which is extensively used throughout the IRC server code, so that I find it important to mention here. The other is the *direct source code change*, because in programs where the source code is available, it offers the largest possible means to change the workings of the software, including its governance characteristics, limited only by the interoperability with the other components of the IRC network, such as the other servers and clients.

#define directive

The #define directive is a common construct provided by the C programming language. It allows to associate a identifier label with a specific replacement string:

```
#define label replace-string
```

This directive is evaluated by the C preprocessor (a part of the compilation process) by simply replacing every label with the replace-string in the source code before the compilation process begins. This is used for example to refer to constant values in the source code with a name instead of a value, let's say a hypothetical BigNumber instead of the value 1234567.

In addition, the #define directive can also be given parameters which are substituted when the directive is evaluated, and accompanying constructs such as #ifdef label ('evaluate if label has been #define'd') allow the conditional evaluation of code parts.



An example for the use of the #define directive is shown below in chapter 4.2.3: the 'max-users-per-channel' directive determines the number of users that can concurrently use a channel. The source code as distributed sets this value to 10, but the IRC admin can change it to any other value she sees fit.

The #define labels do not only serve as replacement mechanism. Together with the accompanying conditional directives such as #ifdef, it allows to activate or inactive parts of the source code by bracing them with conditional directives which depend on the status of the #define label (either defined or undefined). For example, one server version includes a directive #define OPER_KILL, which by default is activated. In this state, the IRC server process allows IRC operators of this server to issue /kill commands⁴⁰. But if the IRC admin decides to revoke her own IRC operators the privilege to issue user /kills, all she has to do it to replace #define OPER_KILL with #undef OPER_KILL, thereby deactivate the respective source code parts.

As such, the #define directive is one of the main instruments to allow the IRC admin an coder-provided way to change the «code» governance behavior of the IRC server program.

Arbitrary source code changes

In difference to the #define directive, changes of the source code itself allows the IRC admin to incorporate changes which were not made by the coders. Any changes can be made, and the admin is limited only by her knowledge of the code, and the interoperability with other IRC servers. Regularly voiced concerns over "hacked servers" in IRC mailing lists and other documents indicate that this option appears to be used frequently.

⁴⁰See chapter 5.1 for the explanation of the /kill command.

Compiling and Linking

When the server has been configured, the source code is now processed to create a binary program file which then can directly started. This processing includes the preprocessing and compiling the source code files to get object files, and to link all object files and libraries to get the final binary program.

3.2.3.2 IRC server configuration

Once the installation of the IRC server is completed, the IRC server must be provided with some configuration information. The main location for these settings is the `ircd.conf` configuration file. Additionally, some settings can be provided upon startup of the IRC server..

Configuration lines in the `ircd.conf` configuration file

The central place for the configuration of the IRC server is the `ircd.conf` *configuration file*. This file contains all the settings necessary for the IRC server, such as the servers which it can connect to, which client connections to allow and which to reject, as well as information about the server itself.

The file itself is a normal text file which contains so called *configuration lines* (or short *config lines*). Each line represents one specific setting, such as information about the server and the administrator, a list of other IRC servers which may connect to this one, or to which the local server may connect, users who may connect, or who are banned from the server, etc. Here is an example for a config line:

```
M:BUUSD.BU.EDU:*:Boston University Hoopy Test Server:6667
```

This example is an M-line (defined by the first letter; the letter 'M' stands for 'me', i.e. the server itself), and defines the name of the IRC server. Parameter fields are separated by colons; so this line sets identifies the host as "bucsd.bu.edu", with the (self-chosen) name "Boston University Hoopy Test Server", and the default port on 6667 (the third field with the asterisk is not used).

Table 3.1 on the facing page lists the main config lines of the IRC server version 2.1.1⁴¹.

Config lines roughly fall into three categories:

- *Information*: These config lines contain information about the server, such as the name of the IRC server, or some administrative information such as the administrator's personal details.
- *Server-server*: The main configuration lines in this categories are the so called C/N lines. They determine to which server the local server can connect, and which other

⁴¹The oldest IRC server version for which I have the source code available.

Name	Label in the source code	Category	Description
A	CONF_ADMIN	Information	Administrative information of the server
M	CONF_ME	Information	The server's name
C	CONF_CONNECT_SERVER	Server-Server	Sets up connection to other servers (together with N-line)
N	CONF_NOCONNECT_SERVER	Server-Server	Sets up connection to other servers (together with C-line)
I	CONF_CLIENT	Client-Server	Authorizes clients to connect
O	CONF_OPERATOR	Client-Server	Authorizes IRC operator
K	CONF_KILL	Client-Server	Kill user line

Table 3.1: Configuration lines in server version irc2.1.1 (Oct. 1989)

servers may connect to the local server. It is used to determine the network structure of the IRC network, i.e. which server are connected to which others.

- *Client-server*: These lines affect the ways that IRC clients can interact with the IRC server. For example, the I-line⁴² determines which user groups are allowed to connect to the server, while the all users matching a K-line⁴³ are denied entry. The O-line⁴⁴ authorizes a user to gain IRC operator status.

Configuration lines are meant to let the IRC admin decide about the way how the IRC server 'behaves', and is therefore an important governance mechanism inside the IRC. In subsequent chapters, I will outline the «code» governance features of some of these configuration lines.

Startup options

Finally, next to the configuration files, some settings can be provided to the server process upon start-up. These settings override equivalents made in the installation process, or those in corresponding config lines, such as the location of the configuration file, specifying the debug level, and others. In IRC servers, all important settings are done in the configuration file, so the startup options play a minor role.

3.2.4 Technical environment and code distribution

The ircd server code was written in the programming language C for hosts running Unix operating systems. Both language and operating systems have been and are readily available

⁴²Regarding I-lines, see the subsection on K-lines and I-lines in chapter 5.2.1

⁴³See below chapter 5.2.

⁴⁴See below chapter 7.1.

both as technology (compiler, linker and development environment; computers running an Unix operating system) and as knowledge (such as books and other information sources). This means that there have been and are always a large pool of people who can understand the code, and can make changes to it. It is safe to assume that every IRC server administrator at least has a basic grasp of the inner workings of the server; many who contributed to the *ircd* source code indeed have also at one time been administrating a server, or serving other official position in the IRC community.

The IRC *client* code was initially written by the IRC creator Oikarinen as well, and at first distributed together with the server code. Later on, others have created and maintain independent IRC client software in many computer languages and for diverse computer platforms, so that nowadays the prospective IRC user can choose from a wide variety of IRC client software.

A very important choice of basically all IRC server code, and many of the IRC client code, is that they are distributed as *open source* software⁴⁵. This is important because in this way, anyone can take the IRC code, make modification of it and run a new IRC network with it. This is a situation that does normally not arise in other open source software, because the use of the software is more detached from its development. Here the choice of open sourcing the IRC leads directly to the power of anyone to open up new IRC networks, and implement new features into it.

3.3 Main Social Roles in the IRC

Besides the technical structure, another important part of a techno-social setting are the roles that individuals can assume. In the IRC, we can roughly distinguish two kinds of roles:

- Four roles arising from the general setting: The *coder* who designs and maintains the software involved, especially the server code; the *IRC administrator* who sets up, configures and runs the IRC servers; the *IRC user*, who additionally may maintain an IRC user bot; and the *IRC service administrator*.
- Two roles defined by the technical design: The *channel operator*, who manages and controls a channel; and the *IRC operator*, a administrator appointed official who has maintenance tasks.

As these roles play an important role in my explorations in the remainder of this work, they are outlined here. Table 3.2 lists the main roles in the Internet Relay Chat.

⁴⁵For the events which let the IRC fall under the GNU Public License, see appendix chapter 12.7.

Social Role	Description
IRC administrator (IRCadmin)	Set up, configuration, and maintenance of an IRC server
IRC service administrator	Set up, configuration, and maintenance of an IRC service bot
Coder	Design and implementation of the IRC software
IRC user	Participant in an IRC network
IRC operator (IRCop)	Daily management tasks of an IRC server and in IRC network
Channel operator (chanop)	Configuration, management of and control over an IRC channel

Table 3.2: Social Roles in the IRC

3.3.1 IRC administrator

The most important role in the IRC is that of the IRC administrator. Not only does she set up, and configure an IRC server, which gives her power over the server and in the IRC network, but she also is the one who contributes or organizes the necessary hardware and bandwidth for the IRC. Without the contribution and the work of the IRC administrators, there would be no IRC network, because there is no overarching organization which provides the necessary equipment and bandwidth. This fact also makes the IRC a *self-organized setting*: The existence of the IRC network solely depends on the voluntary contributions of the IRC administrators.

The main tasks of IRC administrators, besides the provision of hardware and connectivity, is the maintenance of the IRC server software: to set up and configure it, to install possible new versions and patches⁴⁶, and to secure its trouble free functioning. For the latter task, the IRC administrators appoint IRC operators (see below) as helpers.

In the network, the IRC admin is part of the group of admins, which form the highest 'political' body of the network, setting up policies, and in general making all decisions regarding the network at large. The technical setting gives all IRC administrators roughly the same decisional power⁴⁷, so that the institutional form of this group varies between the network, and is not predetermined by the technical structure.

3.3.2 IRC Service administrator

The *IRC service administrator* manages an IRC service bot⁴⁸, and has controls the source code and program of the service bot. This is important because a service often *centralizes* functions in an otherwise decentralized IRC network. So changes in the service are much

⁴⁶See above section 3.2.3.1.

⁴⁷Topology and data distribution; see chapter 8.

⁴⁸See above chapter 3.2.1.4 and below chapter 6.3

easier to accomplish than changes in the IRC server, for which every server would have to be changed. On the other hand, the service administrator has central control over the service functionality, whereas normal IRC functions are controlled in a distributed manner by the admins and operators.

3.3.3 Coder

Like in other open source software projects, the IRC code development always needs the infusion of people who dedicate their time and effort to maintain and develop the IRC code base. From the onset on, the source code has been open sourced, not the least to interest others to run their own IRC server, and eventually connect them together to form their own IRC networks. Thus, studying the code as well as running the program has been made possible.

Consequently there has always been an influx of contributions to the code, from single bug fixes to complete rewrites and competing server series, by IRC admins and other interested users. At all times though, there were individuals who have taken over maintenance and coordination duties for an entire version, or have continually contributed large parts of the code; these individuals can be identified as *IRC coders*.

There are a number of individual coders who became widely known, due to their continuing contribution to the IRC in general, not only source code. The initial creator of the IRC, Jarkko Oikarinen, is probably the best known coder. And for many versions, single coders have coordinated the entire source code, or single-handedly created and maintained a server source code series⁴⁹. In other cases, IRC networks have established coding teams or committees⁵⁰ who coordinate the work of the individual coders.

The relationship between coders and IRC admins is necessarily a tight one, and often enough the coders are or have been IRC admins themselves. This relationship is crucial because every IRC admin is always free to accept or reject any code changes, even entire versions. Therefore, in many server code versions one can trace the efforts of coders to offer choices instead of forcing changes on the admins. These choices can come in form of code configuration mechanisms, or by ensuring backward compatibility with earlier versions, so that admins can choose to install new versions, or continue running older ones.

3.3.4 IRC user

Not surprisingly, the IRC user is the most common role in the IRC. It is for the users that the whole IRC exists. But the basic opportunities given to users is quite broad.

⁴⁹The prime example here is Chris Behrens (comstud), who created first developed his own IRC server based on an EFnet version, the irc2.8.21+CS series, and recently has created a new series "written 99% from scratch" (csircd series; see <http://www.comstud.com/ircd/>).

⁵⁰For example the Undernet coding committee (<http://coder-com.undernet.org/>), or the DALnet coders team.

First, most IRC networks are open to anyone to enter and use the facilities; no membership application is required, or fees involved. Given that there exist many IRC networks, users have a broad choice of networks to choose from.

Inside a network, the user can join most channels, but also can create new channels at will, for which she becomes the channel operator (see below); in addition, users can expand their capabilities by running IRC bots⁵¹. Sometimes, she gets promoted to the role of an IRC operator by an IRC administrator⁵².

Finally, from this large pool of IRC users there is a constant flow of contributions for the whole IRC community: Contributions to the code base, new IRC administrators, or members of IRC network committees⁵³ concerned with network management, code development coordination, public relations etc. Also, the wealth of documents and information (introductions, server lists, statistics etc.) about the IRC in world wide web sites, mailing lists, or newsgroups is a contribution by its users.

3.3.5 IRC operator

IRC operators (or short "IRCOps") are users who are appointed by IRC administrators to help in maintenance tasks, and have access to privileged commands. Their duties encompass administrative functions for the maintenance of their local IRC server, maintenance of the whole network, and helping and administering IRC users.

This is an entirely code-generated role in the IRC, as all privileges and commands are given by the IRC system. Once the privileges are revoked, an IRC operator again is a simple IRC user.

It is safe to assume that IRC administrators who appoint the IRC operators, also give themselves operator status when inside the IRC. But with the growing membership and thus growing work to be done, there are much more IRC operators in the network than IRC admins.

A special role constitutes the *local* IRC operators (or "locops"): These are operators whose scope of power is limited to the server to which they are directly connected, or the server they are authorized by⁵⁴.

Finally, the Undernet Uworld has implemented its own hierarchy of users with quasi-IRC operator status and power⁵⁵.

⁵¹See above 3.2.1.3

⁵²On the difficulties of becoming an IRC operator, see below chapter 7.

⁵³The Undernet and DALnet are two examples where such committees have formed. See their respective home-pages at <http://www.undernet.org/> and <http://www.dal.net/>.

⁵⁴For the concept of server locality, see the information box "Locality" below, p.90.

⁵⁵See below chapter 7.4.

3.3.6 Channel operator

The IRC allows any user to create new channels at any time. The user who creates a new channel is appointed *channel operator*, and has absolute power over this channel. She can change a number of channel properties, deny entry or allow entry to the channel, etc. As this role serves a pivotal governance role inside the IRC, the next chapter 4 below will provide more details about this role.

Summary

This section has given an overview over the Internet Relay Chat system. It consists of many networks, each a number of IRC servers connected to each other to provide chat services to the users who connect to it via an IRC client program. Inside the IRC, users mainly communicate with each other in channels, but can also send private messages to each other.

Technically, the IRC consists of many components: server, client, bots and services, interconnected to a network in a specific topology and data distribution scheme. The IRC server is the main component in the network, set up and configured by the IRC administrator who has considerable power to change the server behavior since it comes in source code, and different versions and series exist to choose from. Client and server interact with each other through a client-server architecture.

The IRC knows several different social roles: next to the administrators, coders and simple users, the IRC operator and channel operator have specific managerial tasks inside the IRC.

Finally, the IRC networks are set up and run by the principals on a voluntary basis, that is there is no single corporation or overarching organization which runs these networks: The IRC networks are self-organized. Also, the principals of the IRC do rely on social norms and, most importantly, on the «code», the shape and shaping of the IRC technology, to create and maintain the social order: The Internet Relay Chat is a self-governed setting. The goal of the following chapters is to offer some insight on the various way «code» is employed to govern the IRC setting.

4 «Code» Governance in IRC Channels

The previous chapter has laid the foundation of my IRC «code» governance exploration by giving an overview of the technical and social structures of the Internet Relay Chat. Building on this foundation, this chapter now examines the central communication structure of the IRC, the *channel*.

Channels are the main communication structure inside the IRC: They allow users to create and participate in group conversations. Channels can take various forms, from private ones with only a few invited users up to open public fora with tens or hundreds of participants. Some channels are so popular that they have built a "channel community" around it, with web sites, personal meetings etc. Also as a basic principle in all large IRC networks, any user can create new channels at any time, so it is not surprising that there exists a large number of channels throughout the IRC networks. At one time, a website collecting IRC statistics listed over 620,000 channels with more than 1.3 million users¹. Certainly, without channels, there would be no Internet Relay Chat.

My exploration of the «code» governance of the IRC thus begins with the IRC channels.

First, in section 4.1, I set the stage by giving an overview the principal features and functionality of channels in the IRC, and its technical working behind the scenes:

- Channels are identified by *name* and a textual *topic*, and have a set of properties called *channel modes*, allowing users to form the channel to their communication needs.
- Internally in the IRC server code, a channel is represented by a *data structure* and *functions* acting on the data structure, both of which determine the technical functionality of channels.

This description only provides an overview of these features and code design common to all IRC instances, a kind of a 'channel constitution'. The remainder of this chapter concentrates on the 'channel laws', those feature and design details which shape specific policies, and adapt to changing social conditions.

¹<http://netsplit.de/networks/> (2004-12-15)

Section 4.2 begins with the specifics of channels as present in the early versions of the IRC in the first two years of its existence. The analysis of the first channel design in IRC shows how the technology forms the constraints and opportunities that directly govern the user. Specifically, I describe the implementation of a feature which limits the maximal number of users in channels. As we will see, this early channel design is characterized by its lack of configurability, constraining users rather than opening up possibilities for them.

This lack of configurability becomes evident when compared to the later channel design, now standard in all IRC networks: *named channels* (section 4.3). The change from numbered to named channels is accompanied by a complete overhaul of the governance characteristics, the possibilities and constraints given to the users, entirely realized through the change in the IRC server code. I approach the analysis by highlighting the changes made from numbered to named channels, including the introduction of configurable channel modes as well as the role of the channel operator. Also, continuing the case of the "maximal number of users" property from the previous section, I show how the implementation differences lead to the changed characteristics in its use. An overview of further changes in the IRC channel design over the years and in different IRC networks gives a hint of the breadth of innovation potential in «code» rules.

In sum, the examination of the channel facility shows how the principals in the IRC use «code» to shape the social setting of the IRC, and adapt it to changing conditions. The changes are clearly not induced by technical necessity, such as heightened application stability, or scalability, but 'enacted' to shape or influence the social interactions of the IRC users. The «code» is used as a regulation system, and changes in the software are changes in the «code» governance system.

In later chapters, I will recur to certain features of named channels, such as disputes around the 'ownership' over channel names (chapter 6), showing that changes in «code», as typical in other regulation modalities as well, not only maintain the social order in the setting, but also create potential for new disputes and disruption.

4.1 Principal Channel Design

"A channel is a named group of one or more users which will all receive messages addressed to that channel. A channel is characterized by its name and current members, it also has a set of properties which can be manipulated by (some of) its members."²

The basic concept of IRC channels is not a novel one: Users form a discussion group, with individual messages sent to this group being immediately dispatched to all members. Mailing

²Kalt (2000a, p.4)

lists, Netnews newsgroups, or web fora all offer such a functionality, although these discussions do not occur in real-time, but their messages take some time until they reach all recipients. But other real-time discussion groups also existed before the IRC was created, and indeed Jarkko Oikarinen has mentioned some predecessors of the IRC, such as chat rooms in bulletin board systems, and the Bitnet relay chat³.

The basics of channels presented here are well-known concepts; it is in the implementation details that the IRC offers interesting extensions to this basic concept.

The constituting elements of IRC channels are the *channel name*, which identifies the channel; an optional *topic* string, which give a hint of what might be talked about in the channel; and a set of properties of the channel called *channel modes*.

Channel names identify channels in an IRC network. In early IRC versions, channel names consisted simply of a number, like '1', '49283', or '-458' (numbered channels); as standard now in all IRC networks, a channel name consists of a sequence of alphanumeric characters of some maximal length (between 32 and 200 characters). In order to distinguish channel names from other strings (such as text messages, or IRC commands), they are prepended with a hash mark ('#')⁴; for example, a channel named 'hottub' is denoted as #hottub. The main use of the name is to identify the channel in user commands, such as entering (/join #hottub) or exiting a channel (/leave #hottub). Inside one IRC network, a channel name must be unique; there can be only one #hottub in the EFnet. But the same channel name can occur in different IRC networks, so #hottub may exist in the Undernet, DALnet or any other IRC network as well. Finally, a special 'null channel' exists in all networks: this is the first channel that all users enters when connecting to the network. It is special insofar as it does not allow any conversations; but all members of a IRC network are also member of the null channel, therefore a list of users of that channel is equal to the list of connected users.

For each channel a *topic* can be provided. This is a one-line text which is displayed along with the channel name in channel listings. The channel group can provide here any text they see fit, including a short description of the topic of discussions in that channel. For example, the channel #hottub on DALnet at one time displayed the topic "Welcome to Hottub... web-site www.thehottub.net Enjoy your stay and have fun!!!"⁵. But as the example for a channel listing above (figure 3.3 on page 44) shows, there is no restriction on the contents of the topic text.

Finally, channels have certain properties called *channel modes*. These modes determine who and how many users may enter the channel, how the channel is visible to users outside

³The web pages under <http://web.inter.nl.net/users/fred/relay/index.html> (2004-12-16) give an detailed account on the Bitnet Relay Chat.

⁴In some server implementation, special channels have different symbols prepended, such as the ampersand symbol ('&') for channels which only exist locally on one server.

⁵On 16 Dec 2004, found through the <http://irc.netsplit.de/networks> channel search engine.

the channel (channel visibility), and the modes of communication inside. These specific modes (and the means to manage them) are an important governance characteristics in the IRC, and as such differ throughout the IRC software versions and networks. Much of the later discussion in this chapter will be therefore concerned with channel modes.

The IRC provides a number of commands which allow users to gather information about channels, or initiate actions such as entering or leaving channels. Some of the more common ones are listed in table 4.1.

Command	Description
<code>/list</code>	lists channels, number of users, topic
<code>/join #channel</code>	sets your current channel
<code>/leave #channel</code>	leaves a channel
<code>/topic #channel topic-string</code>	changes the topic of the channel
<code>/mode #channel parameters</code>	shows or changes channel modes

Table 4.1: Common commands in connection with IRC channels (Pioch, 1993)

Implementational specifics of commands are important when we examine the governance characteristics. For example, the next section 4.2 shows how a certain property (maximal number of users in a channel) is implemented in the `/join` command. The actual design and implementation, including side effects, of commands is an important topic in a «code» governance analysis.

Based on this principal architecture of IRC channels, the following sections will now examine specific «code» governance structures in different versions of the IRC server code.

4.2 Numbered Channels

In this section I introduce the basic design of numbered channels as it appeared in the first versions of the IRC. This initial design is characterized by its lack of configurability by the users: Channel names were numbers, not text string names; its properties (channel mode) were fixed to number ranges, and could not be changed by the users. In these terms, the design was restrictive in terms of its «code» governance characteristics. This is not due to any malfeasance from the side of the developers; instead, the early versions were a 'first release' software, where principal functionalities were already present, but further innovations in code (the technology) and «code» (the governance characteristics) were continually added.

For my examination, this first design serves as a blueprint against which I will compare the changes made in later versions. Also, I give a first account on what I call «code» *rule patterns*: Specific code patterns which have certain governance impacts on the social setting.

4.2.1 Functional Design

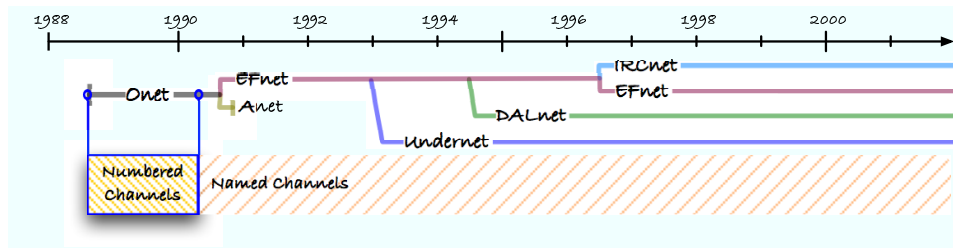


Figure 4.1: IRC Timeline – Numbered Channels

Numbered channels are present from the beginning of the IRC in August 1988, to be replaced by named channels in server version irc2.5+ (July 1990). This channel environment therefore lasted for the initial two years of the existence of the IRC, where the application was in its infancy, with probably less than 50 users on average.

The functional characteristics of numbered channels show themselves in its *name*, *topic*, and *channel modes*.

Channel *names* consisted of a number, either positive or negative ones, such as channels "1", "19408", or channel "-548".

Each channel also possessed a *channel topic*, a short text which is displayed in the channel listing next to the channel number, to give users a hint of what was discussed in the channel, since the name could not provide with such a hint. And since there existed no technology-induced hierarchy between the members of the channel, any member could change the topic text at any time.

The characteristics of a channel, its *channel modes*, was limited to its *visibility* and the *maximal number of channel members*.

Channel *visibility* refers to the ability of users outside the channel to learn of its existence, or to find out who is member of that channel. This visibility property was fixed to number ranges:

- Channels 1 to 999, called *public channels*, were visible to all users. These channels appeared in the list of channels (using the `/list` command), and its members were identified in the list of users (using the command `/who`), where the channel name appeared next to the user name⁶.
- All channels 1000 and up were *secret channels*: While the channel appeared in the list of channels, none of its members would be identified as such in the list of users, making

⁶As another restriction in this design, users could only be member of one channel at the time. The introduction of named channels lifted this constraint as well.

it very time-consuming⁷ to find the user in a secret channel.

- Channels with names in the negative number range were *hidden channels*: Neither the channel nor its users appeared in the respective lists. This ensured that only those having been told the channel name/number could join the channel, if the number was carefully chosen (i.e., rather -234981 than -4).

The channel property being fixed to number ranges, it was not possible to change the visibility of a channel. If for example the members of a public channel wanted to change it to a secret channel, they had to move to one in the range 1000 and up; any user who normally was member of that channel, but by chance not present at that time had to get notified of that move. Such a move (or rather change of channel characteristics) is quite common in the IRC: It regularly happens that obnoxious users disturb discussions in a channel, or that two or more groups inside a channel fight with each other. With the fixed channel properties, there was no other means to resolve such issues than move to another channel. The overall design of channels constrained the users in their ability to deal with such disruptions by themselves.

4.2.2 Technical Implementation

The actual source code reflects the functional design of numbered channels. As my «code» exploration relies mainly on the source code to understand its governance properties, I present here an outline of the technical implementation (without going too much into details). My intention is to show how implementation and functionality are intertwined, not only from a purely technical, but also from the «code» governance perspective.

Channel data structure

The code for channels can be separated into the *data structure* and the *functions* that work on that data⁸. Each numbered channel is represented by one data structure named `struct Channel`, composed of the four variables `channo`, `name`, `users`, and `nextch`:

Algorithm 1 The `struct Channel` data structure (slightly simplified)

```
struct Channel {
    struct Channel *nextch;
    int channo;
    char name[CHANNELLEN+1];
    int users;
}
```

Source: [irc2.1.1/struct.h:190-195].

⁷Looking for a user would mean to enter every of the more than 30.000 channels from number 1000 until the user had been found.

⁸This is a common distinction in programming, institutionalized in the object oriented design paradigm.

⁹The word `struct` is a special keyword in C to denote a data structure.

Certain things should be immediately obvious, while others are somewhat more hidden:

- The variable `channo` serves as channel name, while variable `name` is the channel topic. This can only be found out by examining how functions make use of them.
- As `channo` is of type `int`, integer, it follows that channels can only have numbers as name. Also, most system interpret an integer as being in the range of -32768 to 32767¹⁰, so given the visibility ranges, there were 32768 hidden channels, 999 public ones and 31768 secret channels available.
- The Variable `users` contains the number of actual members in the channel, and is used by the 'maximal user per channel' feature¹¹.

As is obvious for the channel name as *integer* variable `channo`, the implementation defines the constraints (only numbers as channel names). While this example is quite simple, the general principle that ultimately, the constraints and opportunities of a technology-based social setting, its regulation system, is defined by its implementation details, and not only standards and protocols.

The channel list

The data structure `struct Channel` holds the data for one channel. The channel *linked list*¹² is anchored in a global variable¹³ called `channel`¹⁴.

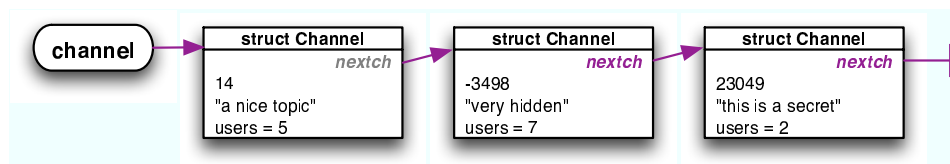


Figure 4.2: Linked list for channels

Functions which access the channel list thus starts with the global variable `channel` and steps through each entry, following the pointer `nextch` to the next entry.

Visibility constraints in the code functions

As an example for implementation details defining governance constraints, I trace the design of the visibility characteristics of channels (public, secret, and hidden channels). As the data

¹⁰This assumes an integer size of 16 bits, which is not always the case. But for the present discussions, this assumption is sufficient, as it should only give an overview.

¹¹Section 4.2.3

¹²In a linked list, each data entry holds a pointer (`nextch` in figure 4.2) which refers to the next entry in the list. The end of the linked list is reached with the pointer of the last entry which contains a special null value ('null pointer'). In the figure, the entry pointer is `nextch`.

¹³A global variable is one with a global scope, i.e. it can be accessed by all functions in the software.

¹⁴[irc2.1.1/s_msg.c:46]

structure shows, no provisions are made in the channel entry which would point towards this functionality. Instead, one has to look into the functions which work on the data structure.

Since the visibility affects the output of the `/list` command, this functionality can be traced to the corresponding function, `m_list()`. Basically, this function steps through each entry of the channel list. For each of the entries, the function now checks the visibility status by calling another function¹⁵, `VisibleChannel()`. When this function returns the value "true", then `m_list()` shows the channel name to the command issuer, otherwise the name is replaced by a asterisk character ("*").

So the visibility condition hinges on the function `VisibleChannel()`. It defines the conditions under which channels are shown, and which are not. Implementation details of this function make clear that the above described number ranges for the visibility (public, hidden, and secret) are hard-coded, that means it cannot be changed once the server process is running. The design does not provide any means to change this channel property.

The major points that have been shown here are the implementation structure for the example of the channel list, distinguished into the *data structure* and *functions*:

- A sequence of *data structure* entries, each holding the information for one channel, and a pointer to the next entry; a global variable points to the beginning of that sequence, the starting point for the functions who operate on this list. Some governance characteristics are implemented here, such as the channel name being represented by a number.
- *Functions* operate on this list, implementing other governance characteristics, such as the visibility property of channels.

In the next section, I examine the other channel property next to visibility in numbered channels: the 'maximum number of users per channel' property. Drawing on its functionality, implementation, and a discussion about it in a mailing list, one gets an impression of the processes ("politics") around such a feature. In section 4.3.1 below, the resolution is shown in the form of a changed channel property.

4.2.3 The "Maximum Users Per Channel" Channel Property

The *maximum users per channel* property. serves as a case analysis of how the IRC 'manages their common affairs': how the principals voiced their discontent with the implementation, and how a resolution was reached.

I first outline the functional and implementation details, and then provide an account of the discussion as it occurred in the then central mailing list for IRC participants. That discussion

¹⁵Actually, this function is implemented as a `#define` directive; for our context though, this is insignificant.

cumulated in a voting which lead to the revocation of the channel limit. The ultimate resolution though followed with the change of the entire channel design, and will be described in section 4.3.1 below.

Functional design

When Jarkko Oikarinen implemented the IRC channels, he considered it necessary to limit the number of users who could concurrently be in one channel, based on his experience that "[a]fter a certain number of people jump in, the conversation often goes to hell."¹⁶ He therefore included into the code a restriction of maximal ten users who could concurrently enter and use a channel. This restriction was absolutely binding for all users, i.e. there was no means to circumvent it. As soon as a user tried to join a channel with already 10 members, she was rejected with the message "Sorry, Channel is full." The only exception implemented by Oikarinen were channels 1 to 10, which he declared as 'unlimited channels', with no restriction to the number of users that could enter these channels. For all channels outside the range 1 to 10, the hard-coded and self-executed substantive «code» rule was that only 10 members were allowed in a channel.

Technical implementation

The 'max users per channel' functionality relies on the variable `users` in the channel data structure `struct Channel`¹⁷ described above. This variable holds the number of users who are in the respective channel, and is continuously kept updated by the system. Every time a user requests to enter a channel (command `/join`), the respective function (called `m_channel()`) checks the limit by calling another function¹⁸ named `is_full()`:

Algorithm 2 Channel limit check in `m_channel()`

```
if (cptr == sptr && is_full(i, chptr->users)) {
    sendto_one(sptr, ":%s %d %s %d :Sorry, channel is full.",
              myhostname, ERR_CHANNELISFULL, sptr->nickname, i);
    return(0);
}
chptr->users++;
```

Source: [irc2.1.1/s_msg.c:619-624].

This code snippet in `m_channel()` shows the conditional statement in the first line calling the function `is_full()`. If the function returns a 'true' value, then the command issuer is sent the message "Sorry, channel is full" (function `sendto_one()`), and the function ends with the `return(0)` statement. If the condition is false, then the line `chptr->users` increments the `users` variable for the channel entry, updating this counter of the actual number of users in that channel, and the user is allowed to join the channel.

¹⁶Oikarinen, Jarkko (1990-05-21) *Channel restriction*. Mailing list *IRClst* (1991) (citing another message).

¹⁷See above algorithm 1.

¹⁸Again, this function is implemented as a directive. And again, this distinction is irrelevant for our discussion.

The function `is_full()` is shown next. It returns the value 'true', if two conditions are met:

- The channel is not one of the 'unlimited' channels. This is checked by calling yet another function `UnLimChannel()`, also shown here. It evaluates true if the channel is in the range 1 to 10.
- The actual number of users is greater or equal a variable named `maxusersperchannel`.

Algorithm 3 `is_full()` and `UnLimChannel()` functions

```
#define is_full(ch, us) (!UnLimChannel((ch)) && ((us) >= maxusersperchannel))
#define UnLimChannel(x) (((x) > 0) && ((x) < 10))
```

Source: [irc2.1.1/struct.h:251,226].

Finally, the variable `maxusersperchannel` is assigned its value at server startup from a `#define` directive unsurprisingly called `MAXUSERSPERCHANNEL` with the default value 10.

Algorithm 4 `MAXUSERSPERCHANNEL` directive

```
#define MAXUSERSPERCHANNEL 10 /* 10 is currently recommended. If this is */
                             /* zero or negative, no restrictions exist */
                             /* If you are connected to other ircds, do */
                             /* NOT change this from default without */
                             /* asking from other irc administrators */
                             /* first ! */
```

Source: [irc2.1.1/struct.h:98-103]

This implementation might seem overly complex for someone not familiar with software programming, but it actually displays a good coding style. The central value `MAXUSERSPERCHANNEL`, appears as a `#define` directive in a source code file which centralizes all such kind of configurable values (in this server version named `struct.h`). This value then is assigned to a variable (`maxusersperchannel` in lower case letters), which keeps open the possibility to implement mechanism which allows to change this value after the source code has been compiled and linked into an executable program.

The source code reveals such a case, although it has not been enabled. Oikarinen made a provision that the value of the `maxusersperchannel` variable could be set as startup option of the IRC server. The IRC admin had to add for example the option `"-c 15"`, to set the channel limit to 15 users, which would override the value of `MAXUSERSPERCHANNEL`. The corresponding code looks like this:

Algorithm 5 Startup option (commented out) for maxusersperchannel

```
#ifndef never
    case 'c':          maxusersperchannel = atoi(&argv[1][2]);
                        break;
#endif
```

Source: [irc2.1.1/ircd.c:100-104].

The first line indicates that this option is disabled. The second line beginning with case assigns the given value to the variable, overriding the assignment from the directive value.

This is another indication that the complexity of the implementation show a good coding style, as it allows for an easy addition of such a functionality (although, in this case, it has not been activated).

Participants' discussion and preliminary resolution

At some time in early 1990, a discussion in the then main IRC mailing list¹⁹ had set off regarding the channel users' restriction. The first mail found is already a reply by Oikarinen, in which he explained the initial rationale behind the restriction, and presented an idea for a (code-based) resolution:

```
From: jto@tolsun.oulu.fi (Jarkko Oikarinen)
To: irclist@tolsun.oulu.fi
Subject: Channel restrictions
Date: Mon, 21 May 90 13:24:32 +0300
[...]
>From: "Matt Crawford" <matt@odjjob.uchicago.edu>
>To: jto@tolsun.oulu.fi (Jarkko Oikarinen)
>Cc: irclist@tolsun.oulu.fi >
>I sent "yes restrictions", with an asterisk: More channels
>with unrestricted membership might be wanted, but I like
>having some channels limited in size. After a certain
>number of people jump in, the conversation often goes to
>hell.
>Crawd{d.
That's true and that's why restricted channels were invented
in the first place... it would be nice if the number of users could
be limited dynamically, everytime a channel is created.20
```

The discussion went on, with some supporting the restriction, others doubting its necessity. Many offered alternative configurations such as:

```
How about upping the number of channels that allow more than 10 users.
Currently, channels 1-10 allow more than 10 users. How about making
it channels 1-100? Would that appease the people who want
unmanageable conversations?21
```

The suggestions concentrated around alternative number range schemes for the limitation, such as: "1-9 have no limit, 10-19 have a limit of 20, 20-29 a limit of 15, and the rest have a

¹⁹IRClis (1991)

²⁰Oikarinen, Jarkko (1990-05-21) *Channel restrictions*. Mailing list *IRClis* (1991).

²¹Peterson, Jan L. (1990-05-21) *Re: 10 users per channel restriction*. Mailing list *IRClis* (1991).

ten person limit."²², or "Negative channels: no limit", "0 to 99: no limit", "100 to 999: limit of 10 users", "1000 and above: no limit"²³.

In the beginning of June 1990, Oikarinen called for a vote in the mailing list which resulted in 17 voting for 'no limits', and 16 (including Oikarinen) in favor of limits. In consequence, Oikarinen gave what amounts to an 'official' permission to lift the user limit restriction.

```
Results of ircvote regarding channel restriction removal:
17 - NO RESTRICTIONS
16 - YES RESTRICTIONS
So, from now on, people can remove the channel restrictions from their
servers. (Wasn't that what we voted about ?) The next irc version
won't have the channel restriction enabled.
I think that the original call for votes wasn't very clear, there
was at least one person who misunderstood and voted YES RESTRICTIONS
even if he meant NO RESTRICTIONS. I hope that it was the only one.
--Jarkko24
```

This highlights the limit of power of one individual, already in this early phase of the IRC. Although Oikarinen was the creator of the IRC, and and IRC administrator as well, he did not force his opinion of let the channel restriction stay upon the others. While this could be attributed to his personal integrity, the main reason is that even as the creator of the IRC, he did not have absolute control over the decisions. IN case that he would have tried to force his opinion, others would have made the changes without his consent, because of their access to the source code. This is a scenario that is well known from other open source projects.

Another fact which may have eased the decision for Oikarinen is that he already worked on a new structure of the channel situation, including the channel user limit:

```
Anyway, I think that it's best if the channel restrictions are removed
now. Tolsun's already running a test server with channel names as strings
and I'm trying to implement a system where first user joining a channel
gets it's 'ownership' and can change the limits and such as he/she likes.25
```

The result of this effort is introduced in the next section (section 4.3.1), where the "maximum number of users" feature turns into a per-channel settable channel mode²⁶.

²²Pelletier, Mike (1990-05-21) *Re: 10 users per channel restriction*. Mailing list *IRClst* (1991).

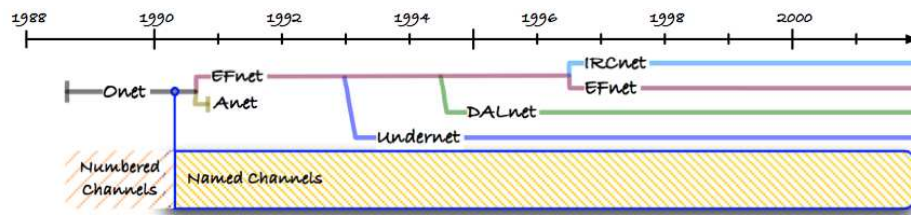
²³Khanna, Sanjay (1990-05-21) *Re: Channel restrictions...* Mailing list *IRClst* (1991).

²⁴Oikarinen, Jarkko (1990-06-07) *Results of ircvote*. Mailing list *IRClst* (1991).

²⁵Oikarinen, Jarkko (1990-06-07) *Re: Results of ircvote*. Mailing list *IRClst* (1991).

²⁶As a side note, one could compare this development to Lessig's account of the AOL chat (Lessig, 1999a, pp.68-9). Here the limitation is set to 23 users in one "chat room" (the AOL equivalent to an IRC channel). What is interesting beyond the fact pointed out by Lessig that the code does set constraints is how differently it is dealt with in the IRC. Because of its different 'code constitution', both in open sourcing the code and in the role of servers and the power of IRC admins inside the system, any one IRC administrator could change the code, and if many were changing it, it would be automatically enforced. Even more, because the server code is in the open source, anyone can start her own IRC server, either as single server-network, or creating a new network with others. Today's variety of hundreds of different IRC network is the result of this policy, something certainly not possible in an environment described by Lessig.

4.3 Named Channels - a Major Change in «Code» Governance



In the previous section I have described how the first design of channels constrained the choice of the user to shape this main communication environment in the IRC: Numbered channels gave only a preset choice of visibility in the IRC, and almost all channels were limited to 10 concurrent members. In addition, using numbers do not have the same expressional power as text names, and the channel topic did not fulfil this function well, since any channel member could change it at any time.

The introduction of *named channels*²⁷ in late summer 1990²⁸ changed this and more. This is the first and probably most influential «code» design change in the Internet Relay Chat. Many code changes have been incorporated since, both to channels and the IRC design at large, but nothing has impacted the IRC as deeply. This principal channel design still prevails today, through all code versions and even new code bases²⁹ of the different IRC networks.

This section examines the design and implementation changes as change in the «code» governance, the constraints and opportunities for users regarding channels.

4.3.1 Names, modes, and the channel operator

The most obvious change is in the channel name. As the name implies, “named” channels have text identifiers instead of numbers. In order to distinguish them from commands, or other text input, channel names are prepended with the hash character (`#`). While previously, only one channel “33” existed, now any of `#33`, `#thirtythree`, `#thirty3` etc. is possible, as well as any other alphanumeric text string³⁰.

This change brought with it the necessity to decide on some policies to handle these new names. Who was to create such names, or coordinate them? What about abandoned names and channels? Such issues had somehow to be addressed.

The initial solution to these issues follows these principles or policies:

²⁷Sometimes also called “string channels”.

²⁸The entire features described here were introduced in two steps, in server versions irc2.5+ (July 1990) and irc2.5.1 (September 1990).

²⁹I.e., server versions which were rewritten from scratch.

³⁰Next to digits and alphabetic characters, a small set of other symbols is allowed.

- *Channel creation*: Any user may at any time create new channels, just by entering into a previously non-existing channel. There is no special command for the creation; issuing a `/join #channel` command creates a new channel, if `#channel` does not exist (otherwise the user simply enters the existing channel).
- *First come-first serve*: No provisions were made regarding any reservation or pre-registration of channels. When a channel does not exist, i.e. a certain name is not used by an existing channel, then any user can create a channel of that name.
- *Channel end-of-life*: As easy as the creation, as quick is the release of a channel (and its name). As soon as the channel becomes empty, i.e. the last user has left the channel, the channel ceases to exist, freeing the name to be used by any other user. This 'no-hold' policy remains the basic «code» policy in all versions of the IRC server code. I will show later on (chapter 6) how at first users, then IRC officials used special programs – bots and services – in order to change this basic policy towards other channel life and control/ownership policies.
- *Member in more than one channel*: With numbered channels, a user could only be member of one channel at a time. This limitation has been lifted with named channels, allowing not only to join several channels at once, but also create more than one channel at one time (and becoming member of all of these).

The elegance of this solution stems from the minimal changes in the use of channels. Channel creation and deletion are quite transparent, as they need no special command or other mechanism. And even the first come-first serve policy appears as a good idea, given the easy handling of channels.

Besides these policies, two new features were introduced with named channels which depart from the former policies of numbered channels in a big way: The introduction of *configurable channel modes* and the role of the *channel operator*.

Channel modes

Each channel has some specific characteristics which shape the ways how it can be used. In the case of numbered channels, these were the visibility (public, hidden, and secret channels) bound to certain number ranges, and the limitation of the number of users in channels. Both were fixed to the channel, so no user could change these characteristics for a channel; if users needed different channel properties, they had to move to another channel.

Named channels introduced an entirely new system for channel properties. Instead of these being fixed, channel modes for named channels were made configurable, and new properties next to visibility and member limit introduced. Each of these channel modes can be individ-

ually changed for each channel, allowing to shape the channel according to the needs of the members:

- *Visibility*: As in numbered channels, the appearance in the net-wide lists of channels and list of users can be set with this mode. If set to *public*, the channel as well as its members appear in both lists; when set to *private*, the members are identified as in that channel, but the channel does not appear in the channel list. Finally, with *secret* channels, neither channel nor members are listed³¹.
- *Invite-Only*: This mode allows to control the membership more tightly than it is possible with the channel visibility. The IRC has a special command `/invite` which allows any user to invite someone to a channel. This is a short text message which say that 'User A has invited me to channel X'. When the invite-only mode is set, then a user must have an invitation for that channel in order to be able to join this channel. Users without an invitation are rejected and cannot enter the channel.
- *Moderated*: Besides the 'invite-only' entry control, this channel mode allows to restrict active participation to specific users. All other users are limited to listen to (i.e., read) the conversation.

Maximum number of users

In the previous section 4.2.3, I had shown how the numbered channels carried a hard-coded property, allowing only at most 10 members in one channel, and how a discussion and vote among IRC principals resulted in the revocation of that limit. At that time, Oikarinen already hinted that this issue would be shortly resolved in a different way.

This resolution came in form of yet another channel mode:

- *Maximum number of users*: For each channel, a limit of users who may concurrently use the channel can be set.

Instead of fixing the limit of the number of users to a certain range or set of channels, or limiting them altogether, this property was made *user configurable* on a per-channel basis, thereby overcoming the need to implement some fixed scheme, as in numbered channels, or as suggested in the above shown discussion. Rather than either leave it fixed, or open it up entirely, the functionality was deferred to the level where an appropriate decision could be made, to the level of individual channels.

³¹Somewhat confusingly, the terminology has changed from numbered channels: "secret" named channels are similar to the "hidden" number channels, whereas the "secret" numbered channels correspond to "private" named channels.

Channel operator

With all these channel modes, who should have the power to change them? The answer to this question comes in form of a new user role, the *channel operator* or short *chanop*.

The assignment of a user to be a channel operator is automatic, and tightly connected to the creation of a channel: The user who creates the new channel becomes the channel operator for this channel. The policy governing the channel creation is extended to the assignment of the operator role. And the lifetime of this assignment is limited to the lifetime of the channel: As soon as the last user has left the channel empty, the channel operator assignment for that channel removed; the next user who (re-)creates this channel then is assigned the new channel operator.

A channel operator has absolute control over her channel. She can change any channel modes to her liking. For example, when the channel is set to *invite-only*, then only chanops can issue invitations, and thus control who may and who may not enter the channel. With *moderated* channels, only chanops can talk, all other members only can listen/read the conversation.

In addition, a chanop can decide (with yet another channel mode) may change the topic of the channel:

- *Topic*: When set, only chanops may change the topic of the channel; otherwise, any member can change it.

Two commands available to channel operators further underline the importance of this role for a channel: The ability to exclude a user from the channel, and the ability to promote other users to channel operator status:

- */kick command*: This command allows chanops to exit any member off her channel. This is an immediate exit from the channel with no long-term consequences. If no entry limitations are activated, the kicked user can immediately reenter the channel. The command resembles the */kill* command available to IRC operators to exit a user from the network³².

The */kick* command complements the other means to control channel membership, visibility and invite-only to control entrance, moderation to control conversation.

Finally, another mode allows the channel operator to promote others chanop status:

- *Chanop*: This channel mode³³ allows a chanop to promote other users to chanop status. All chanops have exactly the same powers, so there is no difference between the chanop

³²See below chapter 5.1

³³This is not implemented as command, but as a channel mode, since it changes the status of the list of channel operators as property of the channel.

who created the channel in the first place and the others who were promoted by the initial chanop.

Basically, this creates a kind of 'two-class community' inside a channel: those with, and those without the chanop status. One can imagine a different hierarchy being formed in channels: one chanop who absolutely rules over the channel; a small group of chanops sharing administrative tasks, or one where every user is a chanop. Since only chanops may talk when the channel is *moderated*, communication settings similar to a lecture (one chanop) or a panel discussion (several chanops) are made possible as well.

On the other side, several introductory texts on IRC warn against giving chanop status to too many or not-to-be trusted users. Apparently, there have been cases where one who created a channel was demoted to user status by another channel operator, resulting in what is called a "channel takeover". Later design changes, some of which are outlined later on³⁴ have also dealt with this problem.

4.3.2 «Code» Evolves – Further Changes in Channel Design

Although one could suspect that such a large change from numbered to named channels, including the new channel modes, the role of the channel operator etc., would necessitate further adjustments to cope with design weaknesses or changing environments, such as growing and different usership, only few changes in the channel design can be found in subsequent IRC server versions. The channel design as implemented with named channels has proven a viable governance environment, basically remaining unchanged in over fifteen years.

Still, over the years there have been some changes made, mostly adding new channel modes to alleviate some of the small shortcomings of the channel design. I will review the first three changes made since introduction of named channels, as they show how those shortcomings in the design were dealt with – by introducing new modes which broaden the actions available to the channel operators. All three new channel modes were introduced with server version irc2.8, released in March 1993, roughly two and a half years after the introduction of named channels and associated channel modes.

Voice

Moderated channels (only chanops can speak in the channel) showed one problem: Everyone who was to be given the opportunity to speak in a moderated channel had to be assigned channel operator status. This certainly was no problem in channel which were created for the sake of one conversation only, but in more established channels, giving channel operator status to all who wanted to speak out was certainly not a good idea.

³⁴See below chapter 6.

The *voice* channel mode is a logical consequence: It gives members in moderated channels the ability to talk to that channel without being assigned channel operator powers. Therefore, in such channels, both users 'with voice' and chanops may speak, but the former have no chanop powers, and the chanops are the only ones who can give or take voice status to users.

Key

The *key* channel mode is another entry control facility in channels. When set, a user has to provide the correct password (key) in order to enter the channel.

This feature expands the entry control mechanisms for channels. Before, visibility (obscure the existence of the channel to outsiders) or invite-only (explicit one-time invitation must be given out by chanop) has been available. Visibility is not effective for unwanted user who have learned of the channel's existence, and with invite-only channels, chanops have to give out an invitation each time that a user wants to enter the channel.

Keys allow for multiple entries, instead of giving out invitations every time. And should the key becomes known to an unwanted user, then only a key change is necessary, instead of moving to another secret or hidden channel.

Ban

This is yet another expansion of the entry control facility, next to visibility, invite-only, and keys. It allows the chanop to enter single users or groups of users into a channel 'ban list', who then are automatically denied entry to the channel. This feature is the channel equivalent of K-lines³⁵ on the network level.

The implementation of the ban feature reveals another interesting «code» governance structure. In the initial implementation, there was no limit to the size of the channel ban list. Chanops could add any number of user or user groups to the list. But the longer the list, the longer it takes for the server to process a channel join request by a user, since the whole list has to be searched for a possible match³⁶. Apparently, the ban list feature has been extensively used, because only two months after the introduction of channel bans, a limit of 20 bans per list has been implemented into the code³⁷. This limitation remains in all subsequent server versions, although the number varies between 20 and 30 bans per list.

This is a small example of how «code» governance mechanism not only affect the social setting, but also affect technical aspects such as efficiency, or scalability. Later chapters³⁸ will provide further examples where coders were confronted with such trade-off issues.

³⁵See below chapter 5.2.

³⁶The processing time is at its maximum for each non-banned user, because for those, the comparison has to take place for each and every entry in the ban list.

³⁷This limitation appears in server version irc2.8.9 (May 1993), as MAXBAN directive. Also introduced here is the limit of the size of one ban entry to 1024 characters.

³⁸Chapter 5.4.2 shows how the change in the design can lessen the computational cost of a governance feature; chapter 8.1 discusses the costs imposed by choosing a technical architecture with specific constitutional governance characteristics.

Summary – «Code» Governance in Channels

Channels are the main structure in the IRC, the most important one for users: Most of the conversation takes place inside channels. They consist of a *channel name*, an optional *topic* string, and a number of *channel modes* which determine how the channel can be used. This structure is determined by the server code, and serves as basic constitution of channels.

The first design of channels were *numbered* channels: Names were numbers, and the channel modes – visibility and maximum number of users – were fixed to certain number ranges.

The next design, *named channels*, has become the constitutional structure of channels in all subsequent IRC versions: Channels are given names (alphanumeric strings), and the channel modes can be set on a per-channel basis, with more modes available then in numbered channels. The new role of the channel operator determines these modes as well as all aspects regarding this channel. A user is assigned channel operator status by creating a new channel. Channels also are deleted automatically by the system as soon as the last user leaves it, thereby also removing the channel operator status; any new user can now re-create this channel and become the new channel operator.

In principle, this design has prevailed through all IRC server versions. Some additions in form of new channel modes were made due to shortcomings, or the need for them due to changing conditions; this chapter has introduced the "voice" and "key" modes as well as the user ban functionality.

From the «code» governance perspective, several points can be made:

CONSTITUTIONAL STRUCTURE: There exists a hierarchy of «code» rules, a kind of layers in the design: On the top is the basic channel design, with its basic functionality, and its name, topic, and channel modes. In this framework, the design of text names (named channels) with configurable channel modes and the channel operator role were amended early on, to form the constitution of all following server versions. Further changes did not touch these principal structures, but changed details, such as new channel modes. It appears that, similar to law, there is a hierarchy of «code» rules present.

RULES TYPES: Here, I try to identify instances of the rule types that were introduced above³⁹.

The channel modes could be seen as *substantive rules*: They allow or disallow certain conduct for users, such as entry denial on certain conditions (maximum number of members reached, password protected etc.) or the ability to find the channel or channel members (visibility). In this course, the change from fixed modes to configurable modes would be a change in the *controller-selecting rule*: In numbered channels, the coder and the IRC admins were those who set these rules, and later on the channel operator. This

³⁹Chapter 2.3.2.

is not only a change, but a deference down the hierarchy in the social roles: Not the coders or admins, but the channel operators decide upon the channel modes, but limited to their channel.

Channel creation and channel operator assignment is also a *controller-selecting* rule, as is the possibility for them to nominate others chanop as well. The problem with the latter is the equal power that any channel operator holds, a weakness in the *constitutive* rule of channel operators: any chanop can take this status from other channel operators—including the creator of the channel. At least with the *voice* channel mode, this problem has been defused with regard to moderated channels. But the lack of a graded chanop status system is a constitutive weakness.

The `/kick` command as well as the ban list are *remedial rules*, as they define the type and amount of sanctions against users (channel exit or entry denial). The decision here is made by those who apply these sanctions, are not put into the `/kick` or ban list «code». Finally, *procedural rules*, are not implemented in «code»: In order to get information to decide over sanctions, the chanop must be present in the channel to get a first hand account about events. There are no built-in facilities which collect event information.

Aside from these rule types, the «code» functions show some recurring patterns, found in different contexts of the IRC. As they appear to be specific to «code» regulated settings, I offer this as an important result of my thesis, under the label of «code» rule patterns:

FUNCTIONAL «CODE» RULE PATTERNS: Some of these patterns are on the *functional* level: types of constraints or opportunities given to users in the IRC. Examples of such rule patterns are the channel modes: the visibility as well as the 'maximum users per channel' property of numbered channels have been *fixed*, allowing no changes by anyone, neither user nor official. In named channels, then *configurable* rule patterns appear: *binary choices* (invite-only, password protected, moderation), *preset choices* (visibility), and those where a more or less *arbitrary value* can be entered (maximum number of users in channel). Binary choices appear to be a subcategory of preset choices.

IMPLEMENTATION «CODE» RULE PATTERNS: Similar to functional ones, these are patterns which recur in different context of the IRC; but in contrast to the former, implementation patterns are specific shapes in source code: coding styles, or features of the programming language, etc. An interesting case study in this regard is the implementation of the "maximum number of users" property. On the functional level, this is a fixed constraint

4.3 Named Channels - a Major Change in «Code» Governance

of users; but on the implementational level, the coding shows a complexity which is intended to offer flexibility in the configuration and expansion of the functionality to those with access to the source code. For example, the number of users has been defined via a `#define` directive, easily visible to those (normally IRC administrators) who configure the source code; the definition of the range of unlimited channels in another `#define` function is not as obvious, but still easily detectable.

A last point of this chapter will be further shown in other contexts as well:

FUNCTIONAL DIFFERENTIATION: The development of the channel modes shows how coders respond to changing conditions or shortcomings with a successive *functional differentiation*, offering new mechanisms rather than changing existing one. In general, there is a tendency to rather open up new opportunities than implement further and stronger constraints. Examples here are the introduction of the channel operator role and new channel modes, as well as making existing one more flexible.

5 Sanctions in the IRC

In any social situation, misbehavior and rule compliance are prevalent problems, and mechanisms against such behavior are an integral part of the governance game. Consequently, the Internet Relay Chat has devised mechanisms which cope with such problems. This chapter examines some of those mechanisms that have been implemented as «code» rules.

Already in its first incarnations, the IRC included two mechanisms which allow IRC officials to sanction users by either exit them from, or deny them entry to the IRC network: the `/kill` command available to IRC operators (section 5.1), and K-lines set by IRC admins (section 5.2). I examine their functionality and implementation as well as their scope and limits. In addition, I outline the various changes and additions to these sanctioning tools. They reflect the continuous efforts to adapt to the changing environment such as the growth of users, accompanied by the growth of IRC servers, admins, and operators.

Issuing K-line bans were for a long time the sole domain of IRC administrators; IRC operators only were allowed to issue `/kills`. Section 5.3 traces a function which bridges this separation, allowing IRCops *controlled* access to K-lines.

Beyond these sanctioning tools, other means have been implemented in order to alleviate the necessity to apply these sanctions. Section 5.4 examines two means, the delegation of sanctioning mechanisms to the channel (operator) level, and user tools which avoid situations where sanctions might become necessary.

This chapter touches a number of «code» governance issues:

- «Code» *remedial rules*¹. `/kill` and K-lines give IRCops and admins a tool to deliver a sanction, but do not implement some policy objectives (substantive rule). This means, the decision of applying the sanctions are in decisional power of the issuer, and thus subject to IRC social norms, outside the realm of the «code».
- *Balance of powers*. The `/kill` and K-line code reveals how coders and admins struggle between giving IRCops power to manage the day-to-day duties, and at the same time retaining control over their actions.
- *Functional differentiation*. The changes in the sanctioning tools show the development of code towards a higher degree of differentiation, allowing for graduated sanctions.

¹See above chapter 2.3.2.

- *Delegation.* Sanctioning power is delegated by providing channel operators means to sanction their channel members, alleviating the necessity of sanctions.
- *Non-sanctioning remedies.* User commands allow to protect against some kinds of misbehavior, thereby diluting the necessity to turn to IRC officials to request sanctions.

5.1 The /kill Command – Immediate Sanction

The `/kill` command is the main sanctioning tool for IRC operators. It allows them to immediately exit a user from the IRC network without recourse, but also with no long-term consequences.

5.1.1 Functionality and Implementation

The functionality of `/kill` is quite simple. An IRC operator issues the command with the nickname of a user who then is immediately disconnected from the IRC network.

The basic format for the command is

```
/kill user
```

where *user* identifies the user to be exited. Upon issuing the command, the user receives a short notice, and then is immediately disconnected from the IRC server which she has been connected to. Though no recourse is possible, the command does not have long-term consequences: The user can immediately reconnect to the server: `/kill` does only disconnect, but has no sanctioning 'memory'. This also means that only users who are currently connected can be exited. The `/kill` command does not allow to issue automatic exiting or entry denials for past or future users.

The technical implementation is straight forward. The command activates the corresponding function `m_kill()`. This function first checks two conditions, then executes three actions. The conditions are:

- *IRCop privilege:* The command issuer must have IRC operator privileges.
- *Existence:* The to-be-exited user must currently exist in the network.

When these two conditions are fulfilled, the following actions are executed:

- *Notices:* A notice is sent to the victim to inform her of the pending disconnection, and the issuing IRC operator receives a acknowledgement of the successful `/kill`. Additionally, all other operators are informed of the successful `/kill` execution. In case of an error, an error message is sent back to the command issuer.

- *Network state synchronization*: A message is propagated to all IRC servers so that they can update their network state by deleting the data entry of the exited user.
- *Disconnection*: On the server where the victim is directly connected to, the connection is severed, and thus the user exited from the IRC network.

Of some interest in the implementation are the notices. Not only the issuer and the victim are informed, but also all other IRC operators receive a notice of every /kill issued in the entire IRC network. This is an social norm-supporting «code» mechanism, and is examined in detail below in chapter 7.3.

Such norm-supporting mechanisms are important, because the /kill mechanism does not have any policies implemented, like for example the (hard-coded) 'maximum users per channel' rule², where the substantial rule of 'only a maximum of 10 users may concurrently use the channel' is automatically enforced. With a /kill, issuing the command lies entirely in the discretion of the IRC operator. The /kill command is an example for a «code» *remedial rule*³, a rule which prescribes the "nature and magnitude"⁴ of a sanction, but not its substance, i.e. what situation triggers the sanction, like the 10 users limit in the 'maximum users per channel' rule.

It does not follow from the remedial characteristic of the /kill command that it must be free of any implemented limitations. At all times, /kills could be issued only by IRC operators⁵, and over the course of IRC server versions, further mechanisms have been introduced which gave IRC admins – who give IRC operator status to users – choices to limit the scope of the command.

5.1.2 Changes in the /kill command

The initial design of /kill did not have any limits other than the issuer had to be an IRC operator. Any of the IRC operators could issue a /kill against any user in the network.

In the course of the IRC server versions, three different mechanisms were introduced which changed the scope of the /kill command in different ways: *local operators*, the ability of IRC admins to entirely *revoke the /kill command privilege*, and to limit them to *affect only local users*.

Before I turn to each of these mechanisms, it is necessary to introduce the notion of *locality*.

²See above chapter 4.2.3.

³Above chapter 2.3.2

⁴Ellickson (1991, p.133)


⁵An exception from this rule comes with the Uworld service; see below chapter 7.4.

Server locality

Functionally, the IRC network presents itself to the users as a kind of 'virtual IRC server': Users should not be aware of how many servers form the network, or which user is connected to which server etc. The specific structure of the network should be transparent to the user. This is a goal of many distributed systems called *transparency*⁶.

In some cases though, this transparency is breached in order to serve other purposes. Shaping the scope of the `/kill` command is one such example, and is connected with the concept of *locality* with regard to IRC servers.

Locality

 *A user (including IRC operators) is said to be local with regard to an IRC server, when that user has a direct Internet connection to that server. That is, all messages that the client program of the user sends are received and processed by that server, and if necessary relayed to other servers. The opposite is a remote user: All messages that a server receives from a remote user are relayed by at least one other server in the network.*

Local IRC operators (locop)

The first change was the introduction of a new role, that of a *local IRC operator* or *locop*. in server version irc2.6.1 (July 1991).

The main difference between local operators and normal IRC operators is that the former can only issue `/kills` that are *local to the server* which authorizes them. If a local operator is authorized by a server A, only users who are directly connected to server A can be disconnected by the locop, but not users local to any other server in the network. The power of locops thus is confined to the local IRC server.

But for whatever reason, almost one year later, the `/kill` privilege for local operators had been entirely revoked⁷ only to be reinstated after yet another year ⁸. This reflects some disagreements, assumedly among admins and coders, as to who should be given the power to disconnect users from the network. This assumption is reinforced by the other two mechanisms introduced in the same time span between 1992 and 1993.

Revoke `/kill` privilege

IRC version irc2.7.2c of May 1992 introduced another mechanism, this time the ability for IRC admins to allow or revoke the ability to `/kill` users from both IRC operators and local

⁶Transparency thus is a goal for any distributed system. See for example Tanenbaum (1989, p.457) ("To the extent that the [...] client cannot tell that the server is remote, the mechanism is said to be **transparent**"; emphasis in source); see also "Transparency (computing)." Wikipedia. 2005-04-16 [http://en.wikipedia.org/wiki/Transparency\(computing\)](http://en.wikipedia.org/wiki/Transparency(computing)).

⁷irc2.7.2 (May 1992)

⁸irc2.8.5 (April 1993)

operators for one's own server. If the admin activated a `#define` directive⁹ called `OPER_KILL`, then the IRC operators on that server had the power to issue `/kills`. Otherwise, set to the default behavior of the server code, IRCops were not allowed it (algorithm 6).

Algorithm 6 OPER_KILL directive

```
/* OPER_KILL
 *
 * If you dont believe operators should be allowed to use the /KILL command
 * or believe it is unnecessary for them to use it, then leave OPER_KILL
 * undefined. This will not affect other operators or servers issuing KILL
 * commands however.
 */
#undef OPER_KILL
```

Source: [irc2.7.2c/include/config.h:165-172].

It is notable that the *default setting* for `OPER_KILL` is to *disallow* operators to issue `/kills`. IRC administrators had to explicitly allow it before when configuring the server source code, hinting again at problems with the IRC operators' use of `/kills`.

Limit /kill to local users

The third change came ten months after the introduction of `OPER_KILL`, in server version irc2.8 (March 1993). First, the default for that directive changed, so that the server would allow `/kills` by default. Then another directive was introduced: `LOCAL_KILLS_ONLY`. As the name implies, when activated, any operator on that server could only issue `/kills` for local users, i.e. users connected to the same server that authorized the operator. By default though, this restriction was not activated (algorithm 7).

Algorithm 7 LOCAL_KILLS_ONLY definition

```
/* LOCAL_KILL_ONLY
 *
 * To be used, OPER_KILL must be defined.
 * LOCAL_KILL_ONLY restricts KILLS to clients which are connected to the
 * server the Operator is connected to (ie lets them deal with local
 * problem users or 'ghost' clients
 *
 * NOTE: #define'ing this on an IRC net with servers which have a version
 * earlier than 2.7 is prohibited. Such an action and subsequent use
 * of KILL for non-local clients should be punished by removal of the
 * server's links (if only for ignoring this warning!).
 */
#undef LOCAL_KILL_ONLY
```

Source: [irc2.8/include/config.h:407-419].

Table 5.1 on the following page summarizes these changes of the `/kill` command. The new mechanisms, introduced in the time frame of less than two years, hint toward some serious problems regarding the use of the `/kill` command. Apparently, IRC administrators

⁹See above "`#define` directive", page 56

demanded means to limit the power of IRC operators by either confining its scope to local users, or to be able to altogether revoke this privilege. Whatever the causes, the «code» allowed them to find ways to cope with the situation, by expanding the options available to the IRC admin to control 'her' local IRC operators. Chapter 7 examines further mechanisms to control IRC operators' actions.

	IRC operator	Local operator
<i>Initial design</i>	Can /kill any user in the network	--
<i>irc2.6.1 (June 1992)</i>		Introduction of locop role; /kills limited to local users only
<i>irc2.7.2 (May 1992)</i>		/kill privilege entirely revoked
<i>irc2.7.2c (May 1992)</i>	OPER_KILL: /kills allowed for local IRCops if defined. Default: not defined	
<i>irc2.8 (March 1993)</i>	LOCAL_KILLS_ONLY: If defined, IRCops can only /kill local users. Default: not defined	
<i>irc2.8.5 (April 1993)</i>		/kill privilege reinstated

Table 5.1: Changes in the /kill command code

5.2 The K-Line – Entry Denial Sanctions

K-lines allow IRC administrators to ban users for a longer period by denying them entry to the server where the K-line has been issued. As such, they serve as a complement to the /kill immediate exit sanction.

5.2.1 Functional description and technical implementation

With K-lines, IRC administrator can deny entry to the server for single users or user groups.

IRC admins issue K-line bans by adding configuration lines to the IRC configuration file¹⁰. A K-line is a text line which starts with the letter "K", and looks like this:

```
K:*.bu.edu::hoppie
```

In this example, the K-line affects those who connect from any host of the bu.edu (Boston University) domain with the user name hoppie. Any user who fulfil these conditions are denied entry to the server¹¹.

¹⁰Chapter 3.2.3.2

¹¹Entry denial is limited to the server where the K-line is activated. The user can still connect via another server as long as she is not K-lined there as well.

The configuration file can hold any number of such lines. A K-line also is not limited to one user, or one domain: Through the use of wildcard characters¹², groups of users can be K-lined: in the above example, users named `hoppie` from *any host* whose name ends with `.bu.edu` is denied entry. Similarly, some users from a host or domain could be (and have been) excluded by inserting wildcard characters into the user string, such as `ho*` for all usernames beginning with "ho".

Since K-lines have to be written into the configuration file, only those who have write access to that file can issue them¹³, which normally will be only the IRC administrator. Note that this is not a limitation programmed into the IRC server, but a limitation of the 'environment', the access control of the file system of the host that the server runs on. It is possible that the IRC admin give access to the configuration file to other individuals, independent of their user status inside the IRC network.

The internal operation of K-lines is straight forward. Upon startup of the server process, or with a special command¹⁴, the K-lines are read into a linked list¹⁵.

When a connection request comes in, the user data (username and hostname) is checked against all K-line entries in the linked list. In case of a match with a K-line, the connection request is rejected, and the user cannot enter the IRC network through this server.

K-lines, like the `/kill` command, are an example for a «code» *remedial rule*. There are no substantive conditions implemented; instead, IRC admins are free to issue K-lines as entry denial as they see fit, and keep them as long as they deem necessary. The main inherent limitation to the implementation is that a K-line is limited to the server where it is issued. Only if all servers in an IRC network have issued a K-line for a particular user (or group), then it adds up to a net wide ban¹⁶.

Before I turn to important changes and additions to K-lines made throughout the IRC versions, I like to highlight a peculiar duplication in functionality between K-lines and another config line, the I-line.

K-lines and I-lines – Similarity in functionality, difference in intention

There exists another configuration line, the I-line, which offers the same functionality as the K-line: Acceptance or rejection of a entry request by a user. Both config lines are also implemented in the same way. The obvious question is: Why did the IRC coders choose to offer two different configuration lines with the same functionality? The difference lies in the intention

¹²Wildcard characters are placeholders for parts of the string identifying the user, host or domain. In our example, the asterisk in `*.bu.edu` is a wildcard character, with the whole string carrying the meaning: *any hostname which ends in .bu.edu*. See for example "Wildcard character." Wikipedia. 2005-01-02 http://en.wikipedia.org/wiki/Wildcard_character (section "Computing") for further explanations.

¹³See below section 5.3 for a different interface to add K-lines to the file.

¹⁴`/rehash`, an IRC operator privileged command.

¹⁵See figure 4.2 for an example for a linked list.

¹⁶See also chapter 7.4.2.1 for a net-wide ban command, `/gline`.

of each of these lines.

Algorithm 8 I-line

```
# I: authorize clients to connect to your server. You can use domains,  
#   IP addresses, and asterisk wildcards. The second field can contain a  
#   password that the client must use in order to be allowed to connect.  
#  
I:*.bu.edu:*.bu.edu  
I:128.197.*.*:128.197.*.*  
I:fenchurch.mit.edu:xyzy:fenchurch.mit.edu
```

Source: [irc2.1.1/example.conf].

The *I-line* (algorithm 8) has been intended to determine the *general policy* for accepting or rejecting users for the particular server, partly in coordination with the other servers in the network. For example, the IRC administrators may have agreed to each limit their membership according to the top level domain of the connecting users. So one server sets its I-line to deny entry to all users but those from the “.com” domain, another may only allow those from “.edu”, yet another those from “.net” and “.gov” domains, etc. Or, an IRC administrator might manage two IRC servers, and decides to distribute the connecting users between these servers. For example, the admin for the “.com” domain might decide to direct all users of the domain “aol.com” to one server, and the remaining ones to another server.

In this way, the admins express their specific user acceptance policies through the I-line mechanism. K-lines on the other hand allow the IRC administrator to sanction users or user groups *in addition* to the general policy expressed by I-lines.

By separating these two uses, the IRC administrator can easier separate between the differences in intention: basic connection policies in I-lines, and user banishment in the K-lines. Also, in the case that a user connection request is rejected, the server code can return the appropriate reason for the rejection. The functional and technical duplication helps the admins to express and manage the different user rejection policies. From this perspective, a seemingly superfluous duplication serves an important governance means.

5.2.2 Config lines complementing K-line sanctioning

The K-line mechanism itself has changed only very little in the succession of IRC server versions. Further development of the K-line functionality rather came through the introduction of additional config lines, two of which I present here: R-lines and D-lines complement K-lines by offering a slightly different feature set for sanctioning users. These lines extend K-lines in two different directions of the continuum between technical efficiency and «code» functional breadth.

R-lines

An early addition to K-lines comes with server version irc2.5.1.bu.09 (Nov. 1990). The *R-lines* or “restrict lines” work very similar to K-lines, denying entry to single users or user groups (algorithm 9).

Algorithm 9 “Restrict” configuration line (R-line)

```
# Restrict lines
#
# An extended form of K line. These look for a match and run an outside
# program to whose reply determines whether the person should be let on.
# R:<host>:program:username
# It is a good idea to use a full path name for the program. Depending on
# the system, it might follow the instigator's path or accept ~ and such,
# but there are no guarantees.
# The output of the program should be of the form
# 'Y <message>' to let the user in, or
# 'N <message>' to keep them out. In the case of 'N' the message is sent
# as an error message to the user. In the case of 'Y' it is ignored.
#
# The following example means hrose@cs.bu.edu can only get in if the
# program /other/irc/bin/arbitrary does not return an 'N'.
R:cs.bu.edu:/other/irc/bin/arbitrary:hrose
```

Source: [irc2.5.1.bu.09/doc/example.conf:112-127].

The important difference to K-lines lies in the third field:

```
R:cs.bu.edu:/other/irc/bin/arbitrary:hrose.
```

Here, the IRC admin specifies an external program, in the example called *arbitrary*, residing in the folder */other/irc/bin/* on the server host.

If a user requesting a connection matches a R-line entry, the program is called with the username and host as parameter, and has to return either a “Y” or “N”. If “Y” is returned, the server accepts the connection request, and the user is allowed to enter the IRC; when the program returns “N”, the user is rejected. Optionally, the program can return a message string which the server forwards to the rejected user.

In this way, R-lines allow the IRC administrator to create programs with arbitrary criteria for entry denial. According to a comment provided by the coder in the source code, the R-line “allows more freedom to determine who is legal and who isn’t, for example machine load considerations.”¹⁷ The freedom given is quite broad: For example, this program could base its choice on an external blacklist, or even base its reply on weather data.

The practical limit given by the implementation is the processing time the program needs to return a choice: Since the IRC server has to wait for an answer, it is blocked for that duration, and can only resume processing other requests when the program has returned the answer. Repeated waits could seriously affect the overall response time of the server which may lead

¹⁷[irc2.5.1.bu.09/ircd/s_conf:363-365]. Not the use of the term “legal” here; although one cannot speak of the IRC constituting a legal system, the code is seen similar to law.

to disruption of the whole IRC network, certainly an unacceptable condition. This might be a reason why the use of R-line has been discouraged¹⁸. But despite this warning, R-lines are available in all IRC versions of the EFnet and IRCnet; the Undernet has removed this feature only recently¹⁹.

R-lines show interesting «code» governance properties. Here we see one of the few examples where «code» policy decisions are explicitly 'outsourced' to a program external to the server process. The coders have provided an *interface* to the IRC server, to which the IRC admin can attach programs enforcing arbitrary policies as she sees fit.

At the same time, the technical characteristics of the implementation exhibits an important limitation: The longer the external program takes, the less responsive does the IRC server get, and the larger the performance hit on the whole IRC network. One of the constant challenges of coders and IRC admins is to keep the servers fast enough to cope with the demands arising from the large (and growing) usership. Installing a R-line program which bogs down a server might lead to the expulsion of the IRC server (and IRC admin) from the network. It therefore can be assumed that programs for R-line checks never did much more than simple lookups or matches against some filed data.

With R-lines, the breadth of possibilities offered by the external program feature is counter-balanced by the technical efficiency demands of the server and network.

D-lines

D-lines²⁰ as present in the EFnet hybrid server versions²¹ stand on the other side of the feature vs. efficiency continuum from the R-lines. Here, performance issues have been the driving force to implement them into the server code.

Again the basic functionality is similar to that of K-lines: deny entry to the IRC server. The format of the D-line is given as follows²²:

The difference between K-lines and D-lines lies in the 'position' in the data processing of the the user registration inside the server code. As described in chapter 3.2.2.2, the server

¹⁸See for example [irc2.8.17/doc/example.conf].

¹⁹From ircu2.10.10 (April 2000) on.

²⁰Confusingly, other IRC networks have implemented D-lines to serve different purposes: The D-line in the Undernet (ircu2.9.13, Nov. 1994) is called "connect rule" and allows IRC admins to specify under which condition a *server-connect request* should be allowed or denied, and therefore has nothing to do with user entry denials. D-lines in the IRCnet, appearing in irc2.9.5 (February 1998), serve as "auto connect restriction" rule, allowing IRC admins to control the server-server connects. Both implementations of D-lines are not relevant to our discussion.

²¹D-lines appear first in EFnet server version hybrid-2 (April 1997), but are not activated. hybrid-3 (June 1997) is the first version where D-lines are activated (but can be deactivated with the `D_LINES #define` directive).

²²Example from ircd-hybrid-5/doc/example.conf. Earlier versions don't include this explanation, although the functionality is implemented since ircd-hybrid-2.

Algorithm 10 "Dump" configuration line (D-line)

```
# D : dump. Dumps all connect attempts from the matched IP
#      without any procesing.
# First arg is target IP mask, second is a comment.
D:208.148.84.3:bot host that changes domain names frequently
D:128.183.*:NASA users aren't supposed to be on IRC
```

Source: [ircd-hybrid-5/doc/example.conf].

proceeds in four steps: request reception, parsing and dispatching, processing and responding, and maintenance. In the case of a connection initiation by an IRC client, the request reception step is preceded by an Internet connection initiation. Only after this connection has been established can the client send an IRC connection request.

The D-line check already occurs in this connection initiation step, before the client and server exchange any IRC-related data. In this phase, the client is only identified by its IP address (neither username nor hostname are available at that time). The server checks the D-lines against the IP address, and in case of a match, immediately rejects the Internet connection.

In comparison, the K-line (and R-line) checking occurs long after the Internet data connection has been established: The client now has send an IRC user connection request. This request goes through the reception, parse/dispatch, and processing steps, where finally the available user data (username, hostname) are checked against the K-lines, and, the user request rejected (and the Internet connection severed).

Again, as with R-lines above, we can set the functional breadth against the technical efficiency. Contrary to R-lines, the D-line is functionally limited: only IP addresses can be matched, so no rejection criteria based on username or host domain name are possible. On the other hand, the D-line check takes much less time than a K-line check. The former occurs almost immediately: as soon as the client initiates an Internet connection to the server, it is already rejected (in case of a match). In contrast, a K-line rejection can occur only after the Internet connection has been established, and an IRC connection request sent and processed by the server.

5.3 Access to K-lines for IRC operators

The previous sections have introduced two complementing sanctioning means in the Internet Relay Chat: the `/kill` command which exits a user from the network, issued by IRC operators, and the `K-line` which rejects a user's connection request to a server, issued by IRC admins. The separation between the two mechanisms comes quite naturally: issuing an immediate user exit from the network is easiest done from within the network, by a (privileged) user, the IRC operator, which suggests the command interface. A long-term ban on

the other hand needs to be saved between server restarts, so the configuration file mechanism is a natural choice not the least because the IRC admin as highest-ranking official in the IRC controls access to that file. This way, the hierarchy between IRC admin and IRC operator – the former nominating the latter – is underlined by the separation between `/kill` – the ‘weaker’ sanction –, and the K-line as more severe sanction, and if the struggle reflected by the `/kill` command «code» changes²³ are any indication, this hierarchical relationship was not always a harmonic one.

But IRC admins face a dilemma: On one hand, the more power IRC operators hold, the more duties can be delegated to them. On the other hand, more power means more potential for abuse (as judged by the admin), and the necessity for more control, diluting the usefulness of delegation²⁴. The mechanisms implemented to limit the scope of the `/kill` command – local operator, revocation of the command, local users only limitation – might be interpreted as «code» tools for admins to be able to reduce the costs involved with the delegation, for example having to cope with complaints because of obsessive `/kill` use by one of her IRC operators.

The following example – `/klines` – examines «code» mechanisms implemented to increase the benefit side of delegation. The implementation of this command suggests that it became necessary to apply more and more bans, so that some ways had to be found to allow IRC operators *controlled* access to K-Lines after all. As solution, a command interface to the configuration file was implemented, serving the delegation objective in two ways: The IRC operator is given a means to issue K-line bans from within the IRC; but due to the implementation specifics, this access is limited (command scope, exceptions) and controlled.

5.3.1 First experiment – Undernet’s `/kline` and `/addline` commands

The Undernet was the first network to implement a command which allowed IRC operators to access the configuration file. In Undernet server version ircu2.9.22 (August 1995), the IRC operators had access to two commands: `/kline`, and `/addline`. Both allowed the IRC operator to *add* config lines to the configuration file (and to the config line linked list in the running IRC server). The command format was:

```
/kline user
/addline linestring
```

²³See above section 5.1.2.

²⁴This dilemma is known as the principal-agent problem: ”The principal-agent problem arises when a principal compensates an agent for performing certain acts which are useful to the principal and costly to the agent, and where there are elements of the performance which are costly to observe.” (”Principal-agent problem.” Wikipedia (2004-04-17) <http://en.wikipedia.org/wiki/Principal-agent>.) See also Richter and Furubotn (1999, p.163)

The `/kline` command added K-lines, with further limitations: only one user at a time could be K-lined (i.e., no wildcards²⁵ were allowed), and the implementation required that the user was a current user in the network in order to be `/klined`. One could say that this `/kline` implementation was similar to a `/kill` command, with the addition that the user also was banned for a longer duration, instead of being able to immediately reconnect.

The second command was much more powerful: Any config line (including K-lines) could be added to the configuration file, with arbitrary settings. No other limitation besides IRC operator privilege and a superficial syntax check²⁶ was implemented. For example, while `/kline` was limited to one (existing user), no such limitation existed for `/addline`: any kind of K-lines could be entered.

The ability to use these commands depended on the `#define` directive `DYNAMIC_CONF`, which was activated by default (algorithm 11).

Algorithm 11 `DYNAMIC_CONF` directive (activate `/kline` and `/addline` commands)

```
/* Define this if you want Operators to be able to add lines to ircd.conf
 * WARNING: Do not use this if you have a large number of operators, or
 * if you do not trust everyone to add lines responsibly. These lines
 * are logged, but it's smarter to not allow it unless you need it and
 * can be sure it will be used responsibly. -Cym-
 */
#define DYNAMIC_CONF
```

Source: [ircu2.9.22/include/config.h:85-92].

The comment text shows that the coders were aware of the trade-offs associated with these commands. On one hand, the commands let the admins delegate the task of applying K-lines, or add other config lines, to IRC operators. On the other hand there was the potential that IRCops abused their powers, giving admins extra work to resolve the resulting disputes. I must also be considered here that many IRC admins do not know their IRC operators in person, but only from 'inside cyberspace'.

Further aspects of the implementation try to deal with these concerns: The command interface to the configuration file is strictly limited to *adding* config lines; there are no provisions made which could either change or remove these lines. This also means that the added config lines by themselves serve as a record of the actions by the IRC operators. Together with a text comment which included a time stamp and the nickname of the IRCop who added the line, admins could review the action of their IRCops simply by checking on the added lines in the config file. Command interface restriction and logging facility served here as control tool to alleviate the trade-off between delegation benefits and control costs.

²⁵See footnote in text accompanying the use of wildcards in K-lines in chapter 5.2.1 above.

²⁶The function checked that the first character was a alphabetical letter, followed by a colon (the field separator for config lines), which is the common syntax for configuration lines.

In the Undernet, the `/kline` and `/addline` commands did not last long: Only seven months later²⁷, both commands were removed from the server code. Instead, the Undernet introduced a whole new «code» concept for empowering IRC operators and implementing control mechanisms: the UWorld service, which is examined below in chapter 7.4 and which inter alia brings the 'ultimate' sanctioning mechanism: the `/gline` command issuing *net wide* bans.

5.3.2 /kline and exceptions in the EFnet

Either driven by the same needs, or adopting the idea of the Undernet coders²⁸: Around the same time that the Undernet experimented with an IRCops' command interface to the config file, the EFnet coders also implemented a `/kline` command²⁹. Other EFnet server series followed suit³⁰.

In contrast to the Undernet implementation, the EFnet server version never introduced a generic `/addline` command; only `/kline` is offered to IRCops.

The basic functionality is similar to the Undernet version, but some differences exist. The most obvious one concerns the scope of the command: the EFnet version allows IRC operators to issue `/kline` using wildcard characters, therefore are not limited to issue `/klines` for one specific user, but can target user groups. Also, targeted users have not to be connected to the IRC network at the time that a `/kline` is issued. Otherwise the functionality and implementation is similar to the Undernet `/kline` command.

E-lines – exceptions from K-lines

Another new feature implemented in the EFnet creates an additional layer of 'control' for admins, next to the limited command interface, and the logging facilities: The ability to exclude users or user groups from being `/klined` by IRC operators.

E-lines are entered into the config file, therefore accessible only to admins, but not IRC operators. The server source code offers a short explanation for this feature (algorithm 12).

The comment shows one possible use of E-lines: The IRC admin enters a K-line as a general case (every user from the domain `*netcom.com`), and then provides exceptions from

²⁷ircu2.9.30, March 1996.

²⁸Given the timing, the +CSr series could even have been the first to implement `/klines`: `/klines` appeared in the August 1995 Undernet server version; as for the +CS series, I have no exact date when it was implemented, but the first +CS version came out in June 1995, the documentation ([irc2.8.21+CSr20/README.CS] suggests that the `/kline` has been changed in +CSr16 (December 1995), pointing to an earlier first release for that feature.

²⁹Beginning with one of the +CS versions prior to irc2.8.21+CSr20 (Jan 1996)

³⁰Early +th versions and the hybrid series.

Algorithm 12 "Exception" configuration line (E-line)

```

/* E_LINES - Define this if you wish to have lines that bypass
   K: line checking...ie for example:
   You want to K-line all of netcom.com except for
   *cbehrens@netcom.com, use:
   K:*netcom.com::*
   E:*netcom.com::*cbehrens

*/
#define E_LINES

```

Source: ([irc2.8.21+CSr20/include/comstud.h:19-27]).

that case (users whose name match `*cbehrens` are exempted from the K-line). The concurrent introduction of `/kline` and of E-lines suggests another use for E-lines: Admins enter exceptions to K-lines, so that these users cannot be `/klined` by the IRC operators.

5.4 Beyond /kills and K-lines

In the previous section, I have examined a number of sanctioning instruments which allowed IRC officials to cope with misbehaving users. These are examples of remedial «code» rules – they define the quality and severity of sanctions, but the decisions over their application is left to the officials who apply these sanctions. In this section, I examine two other «code» means which do not give IRC admins or operators more sanctioning powers, but have the potential to relieve them of the necessity of some sanctions.

In section 5.4.1, I set the channel environment into the context of sanctions: From this perspective, the empowerment of channel operators can be seen as a delegation of powers to lessen the necessity to issue /kills or K-lines. Section 5.4.2 describes how coders have reacted to a specific kind of misbehavior – flooding – by giving the users a means to address the problem. The initial design remedies the effects for the user, but does not address the side effects, the load put on the network. A later redesign addresses both aspects, showing how new ideas in design can overcome the technological challenges in «code» mechanisms.

5.4.1 Delegating sanctioning power – Channel modes, kicks and bans

In chapter 4.3 above, I have introduced "named channels" as the principal design of channels in all IRC networks. This design includes a number of channel modes which shape the ways that a channel can be used, including access rules and control over the conversation; the channel operator who controls the channel, and some commands exclusively available to the channel operator.

When we compare the channel and the network environment with regard to sanctions, it

becomes clear that these two have very much in common; they almost are the equivalents on the respective level of the IRC:

- User exit: The corresponding command for the `/kill` command is the `/kick` command available to channel operators, with exactly the same functionality on their respective level: to exit a user from the environment (network or channel). They even sport very similar names which assumedly is no coincidence.
- Long term banishment: Whereas the K-line entry denial has been implemented from the beginning on, channel-level bans were introduced later on³¹. But both are meant to banish users or user groups from the respective level. In the case of K-line though, users may still enter the network through a server which did not issue a K-line for them, while the ban list blocks the only entry to a channel.

Shaping the IRC as a two-level environment³² with the introduction of named channels and the accompanying «code» changes gave IRC officials an opportunity to defer authority down to the level of channels, and give them «code» mechanisms to "manage their common affairs"³³, under the 'shadow of'³⁴ the IRC network officials.

This deference to a lower level strongly resembles the political science concept of *subsidiarity*³⁵: The deference of authority to the lowest competent level of authority. By giving users the power to manage their own channels, the IRC officials were relieved of having to resolve all disputes between users. As we will see in the next chapter 6, the tendency to give channel operators a wider array of tools to shape the channel environment continued with the introduction of registration services. The additional power for IRC operators to interfere in channel operations in the Undernet through the UWorld service (chapter 7.4) indicates that subsidiarity was not the only answer to solve the governance quandary in the IRC.

5.4.2 Non-Sanctioning «Code» Remedies

All instruments presented so far in this chapter has been created to give some officials – IRC admins, IRC operators, and channel operators – instruments to dispense sanctions. The last section presents an example where not officials, but the user is given a command to protect

³¹See chapter 4.3.2.

³²It would not be correct to speak of layers here, since the distinction between network/server and channels is more along the line of the 'inside' vs. 'outside' view of technology. For a similar argument, see Frischmann (2003).

³³See the governance definition, above p. 25.

³⁴In legal scholarship, governance concepts which allow private regulation inside a legal framework (private ordering) is sometimes referred to as "in the shadow of the law", in contrast to regulation outside any legal framework (and control). See for example Lemley (1998, p.6).

³⁵See for example "Principle of subsidiarity." Wikipedia. 2005-02-03 http://en.wikipedia.org/wiki/Principle_of_subsolidarity.

herself against a specific misbehavior. Such user-empowering means are well known in Internet applications, such as filter against junk mail, or pop-up window blocking in web browsers. Such means are not sanctions per se, but are implemented to *avoid* situations where the user has to rely on officials' action for remedies.

The example chosen protects IRC users against a specific type of misbehavior called "flooding":

«Flooding is the rapid repetition of words, symbols, ctcip commands or other contacts designed to overpower a user and force a disconnection. This is called 'flooding someone off'. Not only is this very annoying, but it also interferes with the workings of [...] servers.»³⁶

From early on, users were given a command block flooding by blocking all messages from flooding users. But this early design – the `/ignore` command – did not help with the side-effects, the load that a flood puts on the entire network by taking up bandwidth between, and processing time in the servers. Much later on, a new design – the `/silence` command – was introduced with the same functionality, but which also addressed the side-effects of flooding.

This small examples show how changes 'behind the scenes', in the technical implementation, has important influence on the governance impact of a command, even when the functionality for the user does not change.

The /ignore command

the `/ignore` command allows users to block all messages from other users, to 'ignore' them altogether. This feature has been available from the onset of the IRC on.

Functionally, the command is issued to manage a list of users who are to be ignored. She can also limit this block to specific types of messages (only private messages, or only ctcip messages³⁷) (algorithm 13).

Algorithm 13 The /ignore command

```
/IGNORE [<nickname>|<user@host> [[-]<message type>]]
  Suppresses output from the given people from your screen. IGNORE can
  be set by nickname or by specifying a userid@hostname format. Wildcards
  may be used in all formats. Output that can be ignored includes MSGs,
  NOTICES, PUBLIC messages, INVITES, ALL or NONE. Preceding a type with a
  "-" indicates removal of ignoring of that type of message.
```

Source: Pioch (1993).

The important technical aspect is that the command is implemented *in the client program*. This means that `/ignore` is a feature of the IRC client, while the server is ignorant about it. Factually, the user (i.e. the IRC client) helps herself to manage flooding by selectively ignoring the incoming messages from the network.

³⁶Kzoo and LadyDana (2001)

³⁷See above chapter 3.1.1 for a short description of these message types.

Internally, the IRC client holds a list of users whose messages are to be ignored. When the client receives some data from the 'ignored' user, it drops the data instead of displaying it to the user. All message data still travels through the network, but is ignored at the destination (figure 5.1).

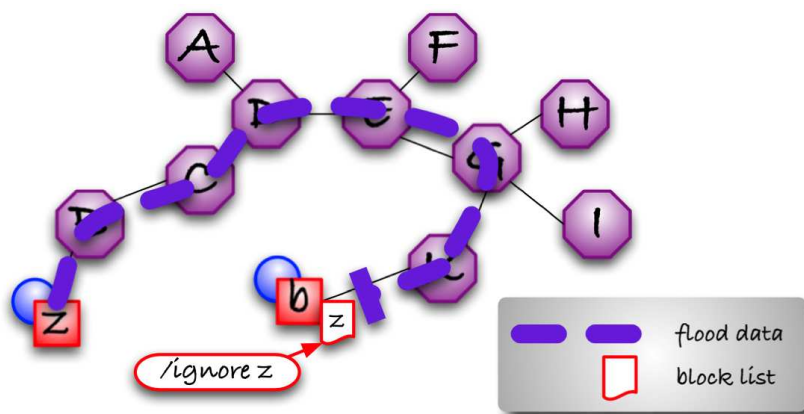


Figure 5.1: Flood block with /ignore

Description: User/client *z* (on server *B*) tries to flood user/client *b* (on server *K*). But *b* has added user *z* to her ignore block list, so user *b* does not see any of the flood data. The flood data adds considerably to the system load for all servers in the path between *z* and *b*, and this can lead to the disruption of the whole network. When network officials encounter such a situation, it is regularly reason for a sanction (either a */kill* or a *K-line*).

From the user perspective, the flooding problem is solved: The command gives her the ability to filter the flooding by ignoring messages from the offending users³⁸. The client simply drops all unwanted data. Also, the form that this filtering takes only depends on the implementation in the client program itself. More sophisticated filtering capabilities can be imagined³⁹.

On the other hand, the side effect of flooding, a heightened server and network load is not addressed at all. IRC officials still need to identify the misbehaving user, and then issue sanctions against her.

The /silence command

In May 1994⁴⁰, the Undernet implemented a command into its servers which offers the same functionality as the */ignore* command, but differs in their technical implementation to cope with the network load effects of flooding: the */silence* command.

³⁸Obviously, this command can be used in situations other than flooding as well, where a user wishes to stop messages from specific users.

³⁹Such filtering is an ideal candidate for scripting facilities in the IRC client; see chapter 3.2.1.3 for a short description of scripting in clients.

⁴⁰ircu2.8.19.U.3.2

From the user perspective, nothing changed⁴¹: The user calls the command with the offending user as parameter in order to stop receiving messages from her. Like /ignore, messages from flooding users are therefore blocked.

The important difference to the /ignore command lies in the implementation details. The whole mechanism is handled by the servers. Instead of keeping the list of blocked users in the client program, the silence mechanism keeps the lists on the server side, that is at the local server of the blocked user. If for example, user b blocks user z who is connected to a server B, then the blocking list *on server B* is updated (figure 5.2). Consequently, all messages that user z on server B sends to user b are already blocked by server B, and do not travel over the IRC network.

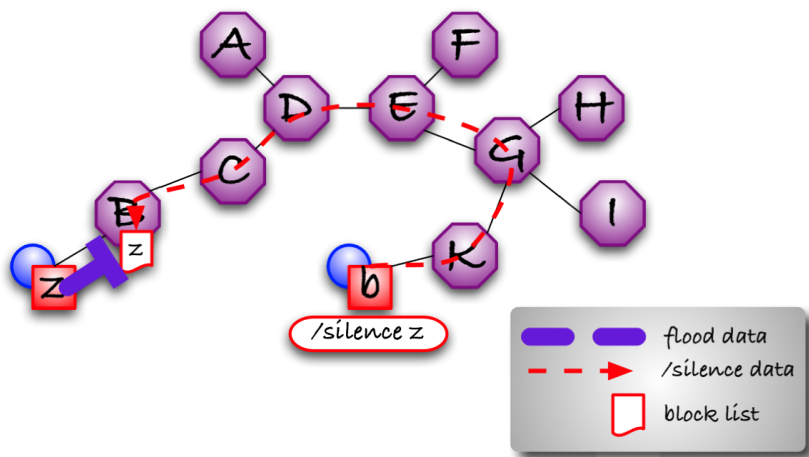


Figure 5.2: Flood block with /silence

Description: User z (connected through server B) tries to flood user b (connected through server K). But because b issued the command /silence z, server B has user z in b's silence block list. Now server B blocks the the all data bound for b, and no flood data goes through the servers connecting z with b.

The main advantage from the perspective of the IRC network is clear: The flooding is stopped at its source, the server where the flooding user is connected to. IRC officials do not have to act by themselves, because the user herself who has been flooded remedies the situation by issuing a /silence command.

The technical cost of this feature lies in the checking mechanism: For every message, the server has to match the sending user against the silence list of all recipients. In order to keep this computing cost on the server side in check, the number of users who can be silenced is limited to 15 entries in the silence list.

⁴¹Also, the /ignore command is still available to her.

This trade-off between higher computing cost versus limiting flooding costs appears to have been favorable for the Undernet, according to the creator of this command, Carlo Wood:

Carlo: Now the SILENCE command did strike even myself as a pretty brilliant solution if I may say so ;) giving the solution to a problem that had existed for 10 years: Make the problem of being flooded the problem of the one that is BEING flooded - give him/her the means to do something about it.

Carlo: Afterwards it looks very trivial :).⁴²

From the perspective of «code» governance, this example shows how two mechanisms with the same functionality, but differing technical implementation can have a different impact on the governance situation of the system. This is an indication that the architecture, determined by its functionalities and often put down in standards and protocols, does not provide the whole governance picture of a system, but that changes in technical details can have an important impact on the governance characteristics of a setting⁴³.

Summary – Sanctioning in the IRC

Sanctioning is an important issue in any social setting. It therefore is no surprise that a number of rule types are apparent in the «code».

From the onset on, the Internet Relay Chat had two mechanisms implemented: The ability to exit users from the network (`/kill` command), and to issue long-term bans (K-line). Both are an example of «code» *remedial rules*: They define the amount of sanctions given, but do not specify when the sanction has to be issued.

As principal design, a separation of power had been upheld which is an *controller-selecting rule*: while the immediate exit command has generally been available to IRC operators, IRC admins did reserve the right to issue bans for themselves.

As certain times, the server code changed the *constitutive rules* of IRC operators, or let the admins decide about changing them: by adding a limited operator role (local operator), restrict `/kills` to server-local users, or even denying local operators its use altogether. On the other

⁴²Undernet-User-Committee (1997b)

⁴³Another interesting aspect is the positioning of the technical functionality in the system. While the `/ignore` command positions its function *at the end* of the system — in the client —, the `/silence` mechanism is positioned in the servers itself. This is an example for an 'exception' from the end-to-end argument discussed in the Internet governance literature (Saltzer et al. (1984); van Schewick (2004)). In short, this argument concerns the placement of functionalities in a multi-layered system, arguing for placing these functions towards the endpoints of the system, i.e. towards the user (see van Schewick (2004, pp.87-107) for an extended analysis of the argument). The 'exception', or rather trade-off then lies in line with the argument made by the original authors that the end-to-end argument "provides a basis for discussion and analysis of trade-offs" rather than "solving the design problem" (Reed et al., 1998). Since the IRC is not strictly a multi-layered system, further work is needed to examine how the end-to-end argument can be applied to applications like the IRC.

hand, the later addition of the `/kline` command, allowing IRC operators to *add* K-lines, is indication of the necessity to expand their power, albeit in a controlled way (only additions of lines).

With K-lines, the prevalent issues were the expansion of the functionality of bans. Next to the intentional separation of functionally equivalent mechanisms (I-line vs. K-line), further config lines expanded the functionality: R-lines allowing arbitrary condition checks (and therefore *substantive* «code» rules to be implemented) for rejection, but at the cost of slowing down the server process, and D-line allowing a very fast rejection, but only based on IP addresses.

Finally, deferring power to sanction down the hierarchy (to the channel or user) was instituted to allow lesser sanctions at a more appropriate place: `/kicks` and bans out of a channel, and `/ignore` or `/silence` other users. This is a *controller-selecting* principle which resembles the subsidiarity rule in political science.

From the «code» governance perspective, a number of topics were touched:

INTERFACES: Sanctioning mechanisms and features explored in this chapter were accessible through different interfaces with different characteristics. The file interface, exemplified by the configuration file, offers easy access to the IRC admin, but is inaccessible to others, unless another interface is created, like the `/kline` command interface. The latter has been used to implement control mechanisms: limiting access to the file (only adding K-lines), and automatically add further information (added when and by whom). Also, the R-line example shows the widening of the K-line functionality by creating an *program* interface: the server explicitly starts that program to receive its judgment about acceptance or rejection of a user connection request.

IMPLEMENTATION VS. FUNCTIONALITY: The R-line interface also showed a specific characteristic often encountered: the trade-off between arbitrary condition and computing cost. The more refined the functional capabilities of a feature, the more costly it may be to execute it. On the other hand, the example of the `/ignore` vs. `/silence` command shows that new ideas in code design and implementation can benefit both overall technical efficiency and increased functionality.

DEFERENCE: Although the IRC is a self-governed setting where no external regulating force is present, this does not mean that there is no hierarchy among the principals of the setting. The limitation efforts of the `/kill` command show the struggle of IRC admin how much power the IRC operators should be given. But the other case has also be shown: the `/kline` command gave them powers previously reserved to IRC admins. While this might be not considered as deference, the empowerment of channel operator

5 *Sanctions in the IRC*

to issue `/kicks` and bans, and of the user to `/ignore` or `/silence` are examples how principals on lower levels are given means to address unacceptable behavior by others. «Code» allows to design such mechanisms, as well as create additional roles (local operator), in order to shape the social setting.

RULE TYPES: These points can also be interpreted in the context of the rule types: In the case of the interfaces, the file vs. command interface points to a controller-selecting rule; `/kline` changes this rule, but formulates the constitutive rule of limited access and implements a procedural rule by recording time and issuer (logging). The program interface of the R-line allows the IRC admin to externalize substantive rules into the program, much easier changeable for the admin than changing the server source code itself, but at the price of increased computing cost.

The computing cost might be interpreted here (R-line) as constitutive rules for admins, limiting their ability to implement costly checking procedures, rules which can quickly change when the means are found to lower this cost.

Finally, deference is the empowerment of controllers down the social hierarchy, a controller-selecting rule. Software seems to be a well-suited tool to decentralize power to a certain extent, a tendency that was already present in the previous chapter on channel management.

6 Nickname and Channel Ownership

There are two classes of identifiers which are unique inside an IRC network: Nicknames, by which every IRC user is identified inside the network, and channel names. The examination of these identifiers in the IRC provides an interesting case study on how the IRC networks use «code» to shape the policies regarding the control ("ownership") over nicknames and channel names. I will trace how the IRC networks have treated the issue of ownership of nicknames and channel names, mainly by means of «code».

Section 6.1 gives a basic description of nicknames and channel names, together with the policy in the first IRC network, the Onet. Their policy was a first-come first-serve, paired with a 'no-hold' policy: as soon as a name was not used anymore, anyone could take that name and use it.

Apparently, the need for a longer-term hold over names arose over the time. As section 6.2 shows, this was first accomplished by tapping into the potential of the client-server protocol of the IRC: So-called *bots* were created which automatically hold the names for the user, thereby effectively circumventing the no-hold policy.

Probably as reaction to such bots, some IRC networks created official services offering some kind of name registration and reservation. The examination of channel registration services in two different IRC networks in section 6.3 shows how «code» enabled these networks to express and enforce different policies through their respective implementation: either as ownership policy with no conditions attached, or as a temporary control bound to an elaborated registration process, and various conditions for the use of the service.

From the «code» governance perspective, it is notable how «code» enables the participants to express their specific policy views: Bots help users to shape their own environment, overriding the default policies of the network. The channel registration services show how «code» makes it possible to design and enforce a fine-tuned 'ownership' policy apart from a simple choice between absolute control over a name, or a 'no-control' policy.

6.1 Nicknames, Channel Names, and Early Ownership Policies

This section explains the characteristics of nicknames and channel names, together with the initial 'ownership' policies in the early IRC network.

6.1.1 Nicknames

Nicknames serve as pseudonyms for IRC users, identifying them inside the network: In channels, each text line is prepended with the nickname of the sender; users address private messages to others via nicknames, and other commands take nicknames as parameter, e.g. the `/whois` command which returns user information for a given nickname.

Every user has to choose a nickname for herself when entering the IRC network¹. The system ensures that the nickname is unique: If the provided nickname is already used, the user has to provide alternative names until a unique one can be assigned to her; only then can the user enter the network.

Once granted entry into the IRC, the system allows users to change the nickname at any time (provided that the new one is unique as well). Each nick change is notified to all members of the channels the user is on, so that no confusion arises on the identity of the user with the changed nickname. Such nick changes are often used to express a mood or other sentiments.

Not all users choose to change nicknames. Specifically, well known people in the IRC identify themselves through nicknames: the original creator of the IRC, Jarkko Oikarinen, for example is known as "WiZ" in the IRC.

Technical implementation

Internally in the server, a linked list² holds a data entry for each user in the entire network. This entry contains the actual nickname, as well as user-/hostname from which she connects, and other related information. As with other IRC state data, this linked list is kept synchronized between all servers. When a new nickname is chosen by some user (on entry, or nick change), this linked list is consulted to check for a possible nickname clash.

From this implementation follows that nicknames are only unique among IRC servers which are interconnected to form an IRC network. Therefore a nickname taken in one network may be free in another one.

Nicknames themselves consist of a combination of letters, digits plus a few other symbols³,

¹See also above chapter 3.1.1.

²See above figure 4.2 on page 71 for an example of a linked list.

³For example, [irc2.8.21/ircd/s_user.c:218-222] defines the allowed characters range as:

```
** Nickname characters are in range
** 'A'..'}', '_', '-', '0'..'9'
** anything outside the above set will terminate nickname.
```

and are limited to a maximum of nine characters in most networks⁴. This amounts to some 50 quadrillion (a 5 followed by 16 zeroes) combinations⁵, but as often the case with such kind of identifiers, there are much fewer 'interesting' combination, like existing words or names⁶.

Initial nickname policies

In the initial IRC network, the Onet, the possibility of disputes over names have apparently not shaped the nickname policies. Rather, simple considerations (such as probable confusion when two users are identified by the same nick) must have guided the implementation. The policy has three cornerstones:

UNIQUENESS – As mentioned above, only one member of a network can use a specific nickname at a time. From the functional perspective this is certainly what a user would expect. Technically though it is not a strict necessity for the implementation, but this condition simplifies the technical design of handling users in the server⁷. An attempt to register to an already used nickname is called a *nickname collision*.

FIRST COME-FIRST SERVE – Anyone is may use any currently unused nickname at any time. This policy comes quasi 'naturally' with the technical design: The server-internal user list only holds the used nicknames. And in order to fulfil the uniqueness condition, this list is searched for a match with a new nickname, accepted when no match has been found. Any other policy would require additional code.

NO HOLD – As soon as the user leaves the IRC network or changes to a new nickname, the old one is released and can be taken by any other user. Only the used nickname is bound to the user, and only for the duration of its use. Again, this is a side-effect of the technical design: when a user exits the network, the internal user entry is deleted, and the nickname thus 'forgotten'; with a nick change, the old name is overwritten by the new one.

** In addition, the first character cannot be '-'
** or a Digit.

⁴A notable difference here is DALnet, which allows a maximal length of 30 characters.

⁵62 choices ('A' to 'z' and '_') on the first position, and 73 (62 plus '-' and the ten digits) on each of the following eight positions equals $62 * 73^8 \approx 5 * 10^{16}$ different names.

⁶The probably best-known fight surrounding 'interesting' identifiers concerns the Internet domain names. For a comprehensive source about this controversy see <http://www.icannwatch.org/> (which by the way is organized similar to the IRC, as unincorporated, volunteer effort based on open source software: http://www.icannwatch.org/about_us.shtml, 2004-04-18).

⁷For one thing, the nickname suffices to uniquely identify a user; otherwise other data, such as the user-/hostname from which the user client connects had to be provided, or an internal unique identifier defined and implemented.

In the server code, this basic policy has been principally unchanged in all server versions, and the EFnet has until recently upheld it as only policy. Other networks have changed their policy regarding nicknames, but not through server code changes, but through the use of services; section 6.2.2 below presents an example for a nickname service.

6.1.2 Channel names

The subject of channels, including their names, have been examined in detail in the previous chapter 4. Here I will only repeat the points that are important for the discussion of channel names, which is quite similar to the treatment of nick names.

Channel names identify channels in the IRC network. Like nicknames, they can consist of a combination of letters, digits plus some symbols. The name of a channel is chosen when created: the first user who joins a channel with an unused name creates that channel (and the user becomes channel operator of the channel). In contrast to nicknames though, a channel name cannot be changed after its creation. But as it is easy to create another channel, and then notify all users to move to the new one, this is not a serious limitation.

Technical Aspects

Similar to the user linked list, servers internally maintain a channel linked list with each entry representing a channel in the IRC network; the channel name is one value in this entry. And similar to nicknames, a channel name taken in one network may be free to take in other networks.

Initial channel name policies

Similarity of shape and implementation between nicknames and channel names continues in the initial policies:

UNIQUENESS – Channel names have to be unique in a network.

FIRST COME-FIRST SERVE – The first one to use a name holds it as long as the channel exists.

The specific design of channel creation though shows one difference to nick names: Channels are created in the same way that one enters the channel: the command `/join #channel` creates the channel if it does not exist yet in the network; otherwise, the user simply enters the existing channel. Therefore users will never encounter the rejection of a channel creation due to a name clash, as can happen when a user changes her nickname.

NO HOLD – The channel does only exist as long as at least one user is inside it⁸. But as soon as the last user has left, the channel ceases to exist. Any user now can (re-)create this

⁸This does not necessarily be the channel operator; any user is fine.

channel, and thus becomes the new channel operator.

The most controversial channel policy is the last one. For some kinds of channel usage patterns – when it is used for a quick chat – this policy might have been sufficient. But early in the IRC, stable communities formed around channels with specific names⁹. For these groups, the policies were in no way optimal: They had to ensure that at all times at least one of the members stayed in the channel, otherwise someone else might take the channel name, threatening to displace or even destroy the channel community.

Changes of the policies came from two sides: Users gained the means to circumvent these policies with the help of *IRC user bots*, shaping their own channel environment independently of the network policies; and some networks implemented new policies, mainly with the help of *IRC services*.

6.2 Policy Changes With Bots and Services

Users began early to look for means to change the policies of nicknames and channels. Especially with channels, users employed so-called *user bots*, programs which stay around the clock in an IRC network to fulfil some tasks, such as to keep the channel from being deleted if the last user leaves it.

Similarly, some IRC admins administered a *IRC service bot* (a kind of admin-run bot) which offered an modest version of nickname reservation: it kindly asked users to release nicknames which were reserved by another user.

Both examples show an important principle in an environment that the IRC constitutes: the possibility to run programs which are not part of the server network, but attach to one of its interfaces to offer some services not provided by the network itself. And as the IRC client-server protocol constitutes such an interface, simple IRC users can set up such programs (bots) and change the official network policies.

6.2.1 Channel Control with the Eggdrop user bot

«IRC bots are particularly important on IRC networks without channel registration services, such as EFnet and IRCnet, and on networks that may prevent your channel being registered due to certain registration requirements, such as Undernet. On these networks, keeping a channel running smoothly without some kind of IRC bot would often be impossible.»¹⁰

Bots can be called an 'all-purpose automated IRC client server'. Like a server, they run 24 hours a day. Like an IRC client, they connect to an IRC network, so from the network side

⁹A very interesting account on a channel community in the early 1990s, see Lawrie (2005) and other web pages on the #gb channel website (<http://uknet.com/gb/>).

¹⁰<http://www.egghelp.org/whatis.htm> (2003-05-16).

they appear as a user/client. They are automated, because they are set up to fulfil automated tasks (scripts) on behalf of the user. And they are 'all-purpose', because they can be freely programmed to do anything that a bot coder (often the user herself) is able to implement.

To give an impression of what a user bot is, I first outline the installation and setup process for a popular IRC user bot named *Eggdrop*; this shows the technical similarity of bots and IRC servers. Both are to be configured in source code as well as through a configuration file; both run as a server, waiting for incoming requests to be processed.

I show how such a user bot helps to hold a channel, and provides other channel management tools.

Eggdrop – Installation, configuration, basic usage

The main web site offers the following "short short version" for the set up of the Eggdrop IRC bot¹¹:

- 1) Download eggdrop1.6.17.tar.gz from the eggheads ftp.
- 2) Telnet and FTP to the shell.
- 3) Upload eggdrop1.6.17.tar.gz via FTP.
- 4) In telnet type `tar zxvf eggdrop1.6.17.tar.gz`
- 5) Type `cd eggdrop1.6.17`
- 6) Type `./configure`
- 7) Type `make config` (compiles all modules) or `make iconfig` (allows you to select the modules to compile).
- 8) Type `make`
- 9) Type `make install DEST=/home/name/botdir`
- 10) Switch to the botdir and edit the sample config file eggdrop.conf, then rename it to something appropriate (e.g. botnick.conf).
- 11) Type `./eggdrop -m <config file>`

Figure 6.1: Steps to Set Up an Eggdrop Bot

Step 1) to 9) concerns the installation of the Eggdrop, and is basically the same procedure as outlined for IRC servers¹²:

- In steps 1) to 5), the version is chosen, downloaded, and the source code packages unpacked at the appropriate location in the file system¹³.
- Steps 6) runs the automatic source code configuration script setting up the needed tools to compile and link the program. Here, the user could also make direct source code

¹¹<http://www.egghelp.org/setup.htm> (2005-04-19)

¹²See above chapter 3.2.3.1.

¹³Actually, the description assumes that the host where the Eggdrop will run is not the same as the host where the installing user is working, which leads to step 2) (the "shell" denotes the shell program of the host where Eggdrop will run) and the uploading and 'telnetting' in steps 3) and 4). Running Eggdrop on another host is necessary when "clones" (multiple IRC client connections from one host) are not allowed on the network: see below chapter 7.4.2.2.

changes. Additionally, source code patches and so-called "modules" are offered through the web site which modify Eggdrop in specific ways¹⁴.

- Step 7) to 9) then compiles and links the program, and installs it to the given place in the file system.

In step 10), the main configuration of the Eggdrop takes place. This is similar to the configuration of the IRC server through the config file, editing the configuration lines (chapter 3.2.3.2). Eggdrop features more than 170 configuration options, grouped in 25 categories, from "basic settings", "files and directories", "log files" to the configuration single modules, one of the the "irc module". Examples for such settings are shown in algorithm 14.

Algorithm 14 Eggdrop configuration lines (examples)

```
# What is your network?
# 0 = EFnet# 1 = IRCnet
# 2 = Undernet
# 3 = DALnet
# 4 = +e/+I/max-modes 20 Hybrid
# 5 = Others
set net-type 0
# Set the nick the bot uses on IRC, and on the botnet unless you specify a
# separate botnet-nick, here.
set nick "Lamestbot"
# Set the alternative nick which the bot uses on IRC if the nick specified
# by 'set nick' is unavailable. All '?' characters will be replaced by random
# numbers.
set altnick "Llamab?t"
```

Source: File [eggdrop1.6.17/eggdrop.conf]. (Lines beginning with a hash mark ('#') are comment lines)

This example shows three settings. The first, `net-type`, lets the Eggdrop know which network it is connected to, in order to use features and commands which are specific to the respective IRC network. The second setting, `nick`, provides the nickname of the user bot in the network. As this nickname may be already taken, the third setting, `altnick`, provides a mechanism to automatically search for an unused nick, as described in the accompanying comment.

The last step 11) finally starts the Eggdrop bot which connects to the preset IRC network (also to be provided in the configuration file), and awaits commands from the user.

Using Eggdrop to secure a channel

In tune with our discussion of channel policies, I concentrate here on the channel-related features of the Eggdrop.

¹⁴See <http://www.egghelp.org/files.htm> (2005-04-19), sections "Patches" and "Modules".

In order to issue commands to the Eggdrop, the user interacts with the user bot through the DCC feature¹⁵. The first step to secure a channel is to let Eggdrop join the channel in question. For this, the user issues the command `.+chan #channel`. Once joined, Eggdrop stays in the channel until the user directs the user bot to leave the channel, or some external conditions arise¹⁶. Eggdrop joining the channel alone effectively circumvents the "no-hold" channel policy: When the last user leaves the channel, the user bot still stays in the channel, so that the channel is not deleted in the IRC network.

The Eggdrop offers further channel related features: When Eggdrop is made channel operator, it can be directed to execute a broad selection of channel management tasks: keep a set of channel modes even when other channel operators change them; maintain a list of users who are automatically banned from the channel as soon as they enter (banlist); automatically give (auto-op) or take (auto-deop) specified users channel operator status upon joining the channel, etc. All these settings can also be preserved (i.e. are present when the Eggdrop is quit, and later restarted) by entering them into the Eggdrop configuration file.

In this way, the Eggdrop bot (and other user bots as well) offers many channel-related features which may not be available per se from the IRC network.

Bots in the IRC

This section has barely touched the features and capabilities offered by bots. Already this short examination has shown how the power that a user can exercise through bots is a very important point in the IRC «code» governance: Users, without being an IRC operator, administrator, or coder, can change the «code»-implemented policies set by the IRC network¹⁷, and in this way also actively 'voice' their opinions and concerns over policy choices of the IRC network.

6.2.2 The NickServ service bot

The second example for early 'ownership' policies via «code» introduces a concept similar to bots: *IRC services*¹⁸. Like bots, they are single programs which connect to an IRC server in order to provide their service. But unlike IRC bots, they do not connect to the network as an IRC client, but as an IRC server¹⁹. This has the advantage that the IRC service receives all messages sent between the servers (especially those which synchronize the internal IRC network state); also, when the service sends requests to other servers, no authorization has to

¹⁵Direct connection between clients, but initiated over the IRC network; see above chapter 3.1.1.3.

¹⁶Such conditions include stopping the Eggdrop process, or disconnection of Eggdrop from the IRC server.

¹⁷This may be one of the reasons why many IRC servers actively prohibit the connection of bots (see for example Paulsen and Fleckenstein (1997)), and sometimes even have implemented sophisticated "bot-detection" routines to enforce the user bot prohibition.

¹⁸See also above chapter 3.2.1.4 for a short introduction to services.

¹⁹In some implementation, a special 'IRC service' category has been introduced into the server code in order to separate them from IRC clients and IRC servers.

be provided as it is treated as a server.

This section introduces an early service installed in order to cope with the nickname 'ownership' problem. The *NickServ* service was created by Armin Gruner and Anton Hartl in July 1990²⁰ (one of them also administering an IRC server²¹), and existed until around 1993-1994 as "NickServ@services.de". Unfortunately, I have found no source code version of the service, so that the following description relies on second hand information in documents and mailing list messages.

Functional description

The NickServ was quite non-intrusive: Users could register a nickname by sending a private message²² to NickServ. When another user used the registered nickname, the NickServ would send a warning to that user that the nickname has been registered with NickServ:

```
-NickServ- -----
-NickServ- ! Attention - Nickname "Yurik" is already allocated by
-NickServ- ! vuong@mips1.info.uqam.ca (The Roaming Soul).
-NickServ- ! This may cause some confusion. Please choose another nickname.
-NickServ- ! If you are the real Yurik, but you are logged into a different
-NickServ- ! computer, you should use the ACCESS command to tell NickServ
-NickServ- ! about this. Type /msg NickServ@service.de help ACCESS.
-NickServ- -----23
```

The user was free to ignore these messages; the NickServ did not enforce any ownership claims over nicknames, but instead served as a kind of 'friendly reminder' service.

Technical description

Although I have no source code available, the functional working suggests the following design²⁴: When a user connects to the IRC network, or changes her nickname, a message is sent to all IRC servers informing them of this change. Having a server status in the network, the NickServ received these messages as well, and checked the new nickname against the list of registered nicks. In case of a match, the warning shown above was issued to the user.

Governance issues

Many EFnet users apparently enjoyed the service, leading to over one thousand registered nicks (at a time where users were counted in the hundreds²⁵). But although coded as an *informational* mechanism rather than one enforcing some kind of nickname ownership, there were disagreements about that service.

²⁰Hartl, Anton (1990-07-31) *IRC Service Proposal*. Mailing list IRClst (1991)

²¹One of the maintainers of NickServ, Armin Gruner, has been an IRC administrator (http://irc.leo.org/irc_leo_org_de.html (23 Jan 2005)), contributed to various server code versions (irc2.5+), and maintained the IRC server code version irc2.6 and irc2.6.1.

²²See chapter 3.1.1.2

²³Vuong, Daniel (1992-01-07) *Forwarded mail...* Mailing list Operlist (1992).

²⁴The server code version irc2.6.1, coordinated and maintained by one of the NickServ administrators (Armin Gruner) contains special provisions for IRC services. It is therefore possible that NickServ made use of these provisions.

²⁵See above figure 3.4 on page 45, user counts in 1990 and 1991.

In the *irc* mailing list I have found a discussion thread where an IRC admin admittedly issued a `/kill` command against the NickServ in May 1991, for which he cited "annoyance" about the service:

```
And now, to clarify: what I did yesterday was not an attempt to compromise
the "security" of NickServ, nor was it sabotage. It was me finally reaching
my annoyance threshold after seeing "Nickname is reserved" fill my screen
once too often.26
```

There followed a debate in the mailing list, leading to an (unofficial and informal) "vote" against the NickServ service. But apparently the service must have been revived afterwards, because in May 1992 there is a mail in the *operlist* mailing list which mentions NickServ@services.de²⁷. But at the latest in 1994, NickServ ceased to exist:

```
Archive-name: irc-faq
Last-modified: 1995/11/28
Version: 1.53
[...]
(17) What was NickServ? Is NickServ ever coming back?
NickServ was a nickname registration service run in Germany. It
was a bot that told people who used a registered nickname to stop using
that nickname. NickServ has been down since the Spring of 1994.
It is not likely that NickServ will be back.
Remember, nicknames aren't owned.28
```

The examples of both the Eggdrop bot and the NickServ show how the IRC «code» can be expanded not only by changing the server code, but also by connecting to the IRC interfaces (client-server for bots, server-server for services). This is especially important for users, who otherwise have not «code» means to influence the policies implemented in the IRC.

The next section turns to two examples of channel *registration* services, which not only give warnings, but install a full channel registration and control/ownership environment. The differences between them show how different policies are realized with such services, partly in code, and partly with administrative procedures accompanying the «code» structures.

6.3 IRC Channel Registration Services

Channel registration services change the no-hold policy upheld in the Onet and EFnet networks: They allow users to obtain a longer term control over a channel.

In this section, I examine two examples of such services²⁹, and compare them in their functionality and the policies behind them. These services were created by the Undernet and

²⁶Wisner, William (3 May 1991) *Re: NickServ, NoteServ: Goodbye*. Mailing list *IRCl*ist (1991).

²⁷Casey, Jonathan B. (12 Jan 1992) *(FLAME FREE) NickServ on 2.7, New Email address and Volunteering*. Mailing list *Operlist* (1992).

²⁸irc faq (1995)

²⁹In this section, I concentrate on *channel* services only. DALnet also offers a nickname service, while the Undernet has kept the no-hold policy for nicknames, so that the comparison of services is only possible with the channel services.

DALnet based on their philosophy to better support their usership³⁰, and these services can be seen as an expression of this philosophy. They each offer not only registration of channels, but various channel management tools, such as a channel operator hierarchy, additional channel modes etc.

The DALnet *ChanServ* service and the Undernet *X/W* service were both put into service in 1995³¹. While the DALnet has implemented what amounts to a full channel ownership with no strings attached, the Undernet has installed a mix of administrative procedures and technical service bot to enforce a temporary control privilege over a channel which has to meet a number of conditions for its registration and use. Each network, using the same basic technical mechanism (IRC service bot), has built a service offering which differs in policy, technical implementation, and administrative procedures.

This section concentrates on the *functional* side of these services, as has been found in various documents on the websites of the networks, and elsewhere, since again (like in the case of the NickServ), no source code packages were publicly available for these services. This may be one side effect of the main property of services in general: they are run *centrally*, on one host, while IRC servers are *distributed*, and their source code has to be distributed to others as well.

6.3.1 Channel Registration in the DALnet: ChanServ Service

DALnet, created in 1994 as user friendly alternative to the EFnet, runs a number of services, including those which provide ownership to nicknames and channels. The services are one of the hallmarks of the DALnet:

DALnet stands out as being the largest IRC network with services. It was indeed the first to have successfully implemented *****Serv* services for its users back in 1994. The most well used of these services are !NickServ, !ChanServ and !MemoServ.³²

Their *ChanServ* channel registration service offers what amounts to a full ownership of a channel, i.e. the control of a user over the channel for the duration of her ownership, with next to no conditions attached. The registration procedures are largely automated, and therefore all substantive policies are implemented into the service; no judgment or interference by officials or users is necessary.

³⁰That is, serving them better than the EFnet, which upholds the original policies regarding channels and nicknames (above section 6.1); but there is indication that channel-related services have been recently introduced or at least tested there: see (Riedel, 2001, ch. 11 "Services") and the Chanfix page on the EFnet web site, <http://www.efnet.org/chanfix/> (2005-01-19)

³¹ChanServ in late January 1995 (PJKevin and Dalila (2004)), and the X/W service in February 1995 (Brown (2003)).

³²PJKevin and LadyDana (2004)

Next to the registration and ownership enforcement, the service offers additional features for channel management, such as nominating lower ranking channel officials with some power, special ownership succession procedures, etc.

The following description is largely based on the help documents available on the DALnet website (PJKevin and Mystro (2004); PJKevin and quen (2004)). All command descriptions are directly quoted from these two, if not referenced otherwise.

6.3.1.1 Channel registration

Consistent with the philosophy of the DALnet to give users the equivalent of an ownership over a channel, the registration process is fully automated. In order to register a channel, a user simply sends a `register` command to the ChanServ³³:

```
/chanserv register #channel password description
```

This command sends a message to ChanServ³⁴ with the to-be-registered channel name, a password which subsequently authorizes one as the owner of the channel, and an obligatory short description of the channel. This registration follows the "uniqueness" and "first come-first serve" policy³⁵: registration is rejected only if the channel has already been registered³⁶. If successful, the user is assigned the special role of *channel founder*. She now owns the channel and has absolute control over it, enforced by the ChanServ.

Similar to the registration, the founder can release the channel with a simple command:

```
/chanserv drop #channel
```

An interesting situation arises in connection with the nickname registration. In order to register a channel, the user needs to be registered with the NickServ nickname registration service. The NickServ implements an expiration policy: If a user left the nick name unused for 30 consecutive days, the nickname expires³⁷. Since ChanServ identifies the founder through the registered nickname, the channel would automatically expire as well. For such a situation, ChanServ allows the founder to name a successor who is assigned the channel founder role if the original founder's nickname expires:

```
/chanserv set #channel successor nickname
```

³³In order to register a channel, the user must also use a nickname registered with the NickServ nickname service.

³⁴The command `/chanserv` has to be explicitly implemented on the IRC client as it is unique to the DALnet. If not implemented, the user has to use the generic command `/msg chanserv@services.dal.net` instead.

³⁵See above section 6.1.2.

³⁶The description does not make clear what happens if the channel already exists in the DALnet.

³⁷PJKevin and LadyDana (2004), chapter 1 "Requirements, Abilities and Responsibilities".

The succession process is quite elaborated, but fully implemented in the NickServ code; no human intervention is necessary:

If the founder's nickname expires, and that channel has a successor, the following will occur: A memo is sent to the channel's successor with an AUTH code. The successor must use the AUTH code to authorize himself in the channel within ten (10) days. If the successor does so, a random password is generated that can be used to identify and become the founder. If the successor's nick expires, or the successor doesn't take any action within ten (10) days, the channel expires as normal. A user can prevent himself from being added as a successor to a channel if he has enabled the NOOP option on his nickname.³⁸

Another interesting feature is the ability to transfer ownership to another user:

```
/chanserv set #channel founder
```

Information: The FOUNDER argument of the set command will allow the user that uses this command to change the channel founder to himself.

Note: The nickname that the user is using must be registered or the command will not work³⁹

This command contains a peculiarity: There is no mention of any action by the original founder other than passing the channel password. The consequence is that ultimately, the password is the only authorizing information necessary for a channel. If a user somehow gets access to this password, no further action from the founder, such as explicit transfer to the other user, is necessary to become the new founder. Warnings in the documentation to keep the password secret and never share it with anyone else supports this view⁴⁰.

6.3.1.2 Channel management features

Once the channel has been registered, the channel founder has access to a number of features for the management of the channel. They roughly fall into two categories: nominating channel officials, and special channel properties and commands.

Channel officials

ChanServ allows the founder to nominate other users for two additional official roles in the channel. This results in a five-level hierarchy shown in table 6.1.

On top is the founder, with all powers of the other roles, and the ability to nominate super ops.

The *super op* (SOp) acts as a representative of the founder: she can name other to be auto ops, give or take ChanServ-enforced permanent channel bans, may kick all users out of the channel (with the exception of the founder; but not excepting other SOps).

³⁸PJKevin and Mystro (2004), chapter 11.16 "Setting the successor of the channel"

³⁹PJKevin and Mystro (2004)

⁴⁰See for example PJKevin and Mystro (2004): "Under no circumstances should you give out the password to anyone. DALnet will NOT help with takeovers if you have shared your password".

Role	Description
Founder	Owner of channel, absolute control
Super op (SOp)	Nomination of AOps.
Auto op (AOp)	Automatically assigned chanop privileges; override bans and invite-only for oneself; take chanop status
Channel operator	(regular channel official)
User	(regular user)

Table 6.1: DALnet Channel Service Officials

The *auto op* (AOp) acts as a superior for normal channel operators: She can assign herself chanop status, either automatically upon joining the channel, or by requesting it from the ChanServ. Also, she can override an invite-only mode or channel ban for herself, can take away chanop status or kick everyone out of the channel (provided no superior is present in the channel).

Channel properties

The ChanServ enhances the normal channel modes in certain ways: access can be limited only to users who have a nickname registered with the NickServ service; changes be limited only to ChanServ-backed officials (AOp, SOp, founder); or channel membership limited to these officials, etc. Also settings more specific to the hierarchy can be made: It is for example possible for the channel founder to set the maximal number of members in the channel to some limit which then cannot be overridden by any of the lower officials: as soon as one of the latter changes this limit, ChanServ immediately resets it to the founder-given value.

A last peculiarity of the ChanServ implementation concerns its relationship to the channel. In contrast to channel bots⁴¹ and the X/W service examined in the next section, the ChanServ does not keep the channel open by joining it as a pseudo-user. It is well possible that the channel ceases to exist (all members leave the channel), and a user different from the founder recreates the channel, thereby becoming channel operator of that channel. But the ChanServ keeps all information about the channel, including all channel modes and the ban list; and as soon as the founder chooses to reuse her channel again, it simply retakes the channel with the help of the ChanServ and reinstates all modes and bans as saved within the ChanServ. It serves as a 'backup copy' of the channel state, and enforcer of the registration when the founder deems it necessary.

⁴¹See above section 6.2.

6.3.2 Channel Registration in the Undernet: The X/W Service

The Undernet shares a similar philosophy like the DALnet, to build a user friendly alternative to the EFnet. The most distinct feature of the Undernet management are its various committees which coordinate everything from code development (coders-com), network management (routing-com) and channel service management (cservice) to public relations and documentation (user-com). Each is constituted of a group of volunteers who contribute and manage contributions from the usership-at-large.

Consequently, the approach that the Undernet has taken regarding the channel registration service differs from that of the DALnet: an elaborated mix of committee-evaluated and «code»-implemented registration process, with the resulting channel control understood as a temporarily assigned privilege rather than a full ownership. The institutional side is represented by the channel service committee (CService), and the technical side by the X/W channel service bots⁴².

6.3.2.1 Channel registration

[16:50] <TheBeast> "Channel registration is not meant as a means to start a new channel. It is meant for previously established channels to have an opportunity to have some stability. If you are first starting a new channel, then just start using your channel and give it time to see if a reasonable user base develops to justify registration." [L62] ⁴³

The difference between the DALnet ownership and the Undernet philosophy expresses itself most vividly in the registration process. Whereas in the former registration is just one command away, in the Undernet, a number of conditions have to be met, and a lengthy application process passed. The whole policy is written down in a special "Channel Service acceptable use policy" (AUP) ⁴⁴.

The two main conditions are:

ONE PERSON PER CHANNEL REGISTRATION: A person is only allowed to register one channel. The Undernet policies emphasize that this has to be a real person, not an unique user/host pair. This also is an indication of their policy of responsible control instead of absolute ownership.

ESTABLISHED CHANNEL ONLY: The channel has to be already established, meaning that it enjoys a stable and continuing usership.

⁴²Actually, these bots were replaced in 2001 with the CMaster bot. From the functional side, there appear not many differences, so that I have limited my examination to the X/W service bots.

⁴³Undernet-CService (2003)

⁴⁴Undernet-CService (2002)

In order to show that the second condition is met, the applicant has to name ten other users who are regular members of the channel and agree that the applicant should be the one who registers the channel.

The application is done through a special web page on the CService web site, which triggers a longer process including both technical checks and review by the Channel Service Committee. The committee reviews the application for its merits, and decides on acceptance or rejection of a registration. The whole process is described as follows:

After your registration form is received by the CSC, the supporter email addresses are checked and each supporter is mailed a copy of the application. The applications is then posted to the registration WWW pages under new applications. They are left in this category for 10 days, during which time other people can object to the channel by using the web pages. The application then moves to the "Pending" section. Here, the applications are also reviewed by CSC to make sure they meet the criteria for registration. If the application is approved, you will receive an approval email and instructions on where to obtain the channel manager's FAQ. The channel is added as soon as it is approved. The entire registration process takes 10-15 days to complete.⁴⁵

Only if the committee has approved the registration, the applicant is assigned *channel manager* for the registered channel and given access to the X/W channel service bots. But in tune with the overall philosophy, the responsibilities for the channel only start here. The channel manager and other officials nominated by her, as well as the channel users have to show that they regularly use and maintain the channel. This includes regular appearance of the channel manager in the channel, as well as continuous use of the channel by its members. The committee appears to regularly check on the activity of the channel, assumedly also with the help of the X/W channel service bots. The AUP emphasizes:

Channel Services regularly monitors all registered channels for activity. If a registered channel is not active, X will be removed from the channel. Channel managers are expected to be active in the channel. If you are gone for more than 21 days, X can be removed from the channel or in very active channels, a new channel manager can be elected by the high level ops.⁴⁶

In addition, the Undernet regularly arranges so-called "opschools", IRC sessions where applicants or channel manager are educated about the registration, the use of the bots, and "other stuff"⁴⁷.

Finally, to underline the commitment of the Channel Service committee to their policies, their web site also contains a "Channel Service committee guidelines" document⁴⁸ explaining their institutional structure including the what they call "ethos", as well as organization and procedures.

⁴⁵Undernet-CService (1997)

⁴⁶Undernet-CService (2002)

⁴⁷See <http://cservice.undernet.org/main/opschool/>; transcripts of past sessions are also made available.

⁴⁸Undernet-CService (1998)

Taken together, the Undernet puts a heavy emphasis on the application of institutional structures and procedures in order to allow users a conditioned temporary control over channels.

6.3.2.2 Channel management: The X and W channel service bots

Once registered, the channel manager is responsible for all activities in the channel. In turn, she is given control over this channel through the X and W channel bots: She can name channel officials, and a number of commands are available to enforce the actions of those officials, and in general to manage the channel⁴⁹.

Channel officials

Similar to the DALnet ChanServ bot, X/W allows the channel manager to nominate other channel officials. But in contrast to the former, it has a 500 level hierarchy, grouped into 9 categories; the channel manager as the highest official occupies level 500 (table 6.2).

Table 6.2: X/W user levels

Level	Title of Official	Main Powers
500	Channel Manager	Sets privileged X/W modes
450	Trusted Channel Admin	Set main X/W modes; tell X/W to join/part channel
400	Userlist Admin	User list access (add/remove/modify); expanded X/W status information
200	Userlist Operator	Clear expired bans in ban list; kick user groups
100	Channel Operator	Give or take chanop status, suspend user's X/W access for specified time
75	New Channel Operator	Ban and unban channel users
50	Channel Regular	Kick single channel users; change channel topic
1	Minimum Access	Login and logout into X/W; basic X/W status information
0	Everyone else (i.e., not in user list)	Mostly commands for channel information

The rationale behind the 500 levels lies in commands which could affect other users and officials, such as channel kicks or bans. These commands can only applied to users *lower* in the level than oneself. For example, an "userlist admin" can nominate other official roles. But if she has level 444, she can only nominate others up to level 443. As one consequence, there can be only one channel manager (level 500), because she – as highest ranking official – can nominate others only up to level 499. This system allows to build fine-tuned hierarchies inside one level, where officials with higher level values (but same title and commands available) control those with lower level values.

⁴⁹For the following details of the X/W service bot, see Undernet-CService (1999).

X/W features

As for the commands or powers available, they are mostly similar to those available in channels, or those in the DALnet ChanServ: to set channel modes, to kick users from the channel, or to ban them for a longer period⁵⁰. Additionally, commands exist to manage the user list, ban list, and X/W modes.

It is not necessary to list all these commands here; it suffices to say that they are again enforced by the X/W service bot on behalf of the channel officials. Instead, two examples, a command and a set of X/W modes reiterate a point made above in the chapter on sanctions⁵¹: How «code» can be shaped to retain a balance between the power given to lower-level officials and the control over them:

SUSPEND COMMAND: The *suspend* command gives officials the power to block access of lower level officials to the X/W for a specified duration. The command allows to specify the time in units of seconds to days, so from short-term to sanctioning to a long-term demotion is possible. A corresponding *unsuspend* command allows to lift the suspension as well. This command is positioned fairly low in the hierarchy (level 100 and up), so is implemented to serve as general power control tool between the channel operators.

OFFICIALS FLOOD CONTROL: Chapter 5.4.2 has introduced the problem of flooding: users harassing other users by sending a rapid succession of messages to them. The X/W enforced modes *FloodPro* (kicks, topic changes), *MassDeopPro* (taking operator status from a user), and *NickFloodPro* (nick changes) deal with such situations: each of them can be set to a numeric value. If a user or official sends more than 'value' commands associated to each of these modes in a 15 second time frame, then she is automatically kicked from the channel, and suspended from the X/W. In the case of FloodPro and MassDeopPro, the associated commands are operator-only ones, so this feature again is mainly geared towards control of channel officials, rather than users. In contrast to the suspend command though, these modes can only be changed by high ranking officials⁵².

The X/W offers the officials not only commands and feature to help manage the channel, but also to manage the relationship between the officials. And above all this looms the control of the Undernet channel service committee which enforces the philosophy of the channel registration as a temporarily given privilege given to those who maintain a usable and popularized channel.

⁵⁰Undernet-CService (1999) contains a list of all X/W commands.

⁵¹Chapter. 5

⁵²FloodPro: channel manager only. MassDeopPro, NickFloodPro: Trusted channel admin (level 450).

Summary – Nickname and Channel Name Ownership

This chapter has examined the ownership issue of identifiers in the Internet Relay Chat: Nicknames and channel names. For both, the initial design of the IRC determined a simple set of policies: They had to be unique in the network (uniqueness policy); the first one to take a name claimed it for the duration of its use (first come-first serve); as soon as the name was not used anymore, it was free to take for anyone else (no-hold). These policies were mainly chosen out of technical reason, but upheld by some networks, although users and channel groups voiced the need for policies which would allow for a long-term use of a name.

As is often the case in such an environment like the IRC, technical means were found which gave users the ability to 'reformulate' these policies: *user bots*, programs which run continuously to provide specific services possibly not available in the servers. In the case of channels, the application of such bots allowed to hold on channels, thereby circumventing the no-hold policy, in addition to many other channel management (and other) tools that such bots offer the user. Notably, these features are available through a regular connection to the IRC (the IRC client-server protocol), without any changes in the server code.

This potential that bots offer was also exploited by the IRC admins. Some networks deviated from the original name policies and installed *service bots*, officially approved bots which connect to the other servers as a special server, thereby sharing their authority.

Two different implementation for *channel registration services* were examined: the DALnet NickServ, which implements a fully automated channel name *ownership* scheme, with next to no conditions attached to its use. In contrast, the Undernet X/W service employs an sophisticated registration procedure which includes technical as well as committee-approval elements, and enforces a policy which gives a temporary hold over a channel under the condition of continuous active use and responsible management of the channel, which is regularly monitored by the Undernet. Using the same technical mechanism of bots, the both network realized a differing set of policies for the registration of channels.

The topic of bots served to show some aspects of the «code» governance topics examined in previous chapters:

IRC CONSTITUTION AND INTERFACES: The initial policies for names in the IRC were apparently a byproduct of the implementation, a consequence of the technical structure. The formulation therefore happened entirely in «code». Similarly, but apparently unexpected, was the emergence of user bots, empowering users to actively shape their own policies and override those of the network, it is based on the «code» constitutional structure of the IRC, enabled by the existence of the client-server *interface*: the power of user bots does not use any addition in the IRC server code, but is a result of detaching the IRC client from the interactivity of users who instead programmed it to act on their

behalf. Similarly, *service bots* provide their services not through special facilitation in the server code, but simply by being authorized to use the server-server interface, to provide their centralized service. The principle of centrality in a distributed network, examined later on, here also posed a problem in my analysis: No source code for either service bots was available for examination.

IMPLEMENTATION «CODE» RULE PATTERNS: It is worth mentioning how the set up and configuration of the Eggdrop user bot resembles that of the IRC server as outlined in chapter 3.2.3. All elements – from patches and source code change to configuration file changes – are present here. The breadth of functionality offered here makes the user almost an IRC admin herself⁵³.

RULE TYPES: The different approaches to channel registration have shown the application of several rule types: In the DALnet ChanServ, fully automated due to its principle of full ownership, implements its substantive rules in «code». Not only registration and release of the channel, but the succession policy and nickname expiration all are examples where the policy is formulated and enforced through the ChanServ service bot.

In contrast, the Undernet channel registration service has chosen to formulate their registration policies in a mostly administrative way, where ultimately the members of an Undernet-official committee decides over the merit of an application; the X/W service bot comes into play only after the registration has taken place.

After the registration, both ChanServ and X/W offer the registrar substantive and remedial rules similar to those in normal channels, but offering more options and better enforcement due to the central power of the service bots, as well as controller-selecting and constitutive rules (channel official hierarchies; suspend and flood control against channel officials in X/W), much finer grained as those available in normal channels.

⁵³Indeed, the Eggdrop web site also features a script which allows to connect several bots to a "botnet" network: see <http://www.eggghelp.org/netbots/index.htm> (2005-04-26).

7 Controlling the Controllers?

IRC operators or IRCops are special users who have privileged access to a number of commands in order to help maintain servers, network, and users. As such, it is the single most powerful role inside an IRC network.

The role of the IRC operator is entirely «code»-based: without the provisions in the server code, there would be no such role. As a consequence, their powers as well as the checks and balances depend on the «code» rule design agreed upon by the IRC administrators and coders. Each IRC admin has the power to assign and revoke users the IRCop privilege.

Section 7.1 gives an overview of the IRC operator role, including an example of how a network related command can be abused to take over a channel.

Section 7.2 gives an impression of the reputation of and norms surrounding IRC operators, as well as (lack of) nomination process for them. Judged by this, there appears a certain 'mystic aura' around the IRC operator role: a closed group with no clear rules how they can be joined, and discrepancies between what they should do, and how they actually behave.

In section 7.3, I show how «code» can help to provide the transparency in a system in order for the participants to enter into disputes in the first place. The extent of information about what is happening in the IRC – here, the actions by IRC operators – determines how informed the participants are. This is also reflected in the IRC code, in *information* available to participants, *notices* automatically sent by the servers upon certain actions and events, and *logging* facilities in the servers.

Finally, section 7.4 examines UWorld: a service bot introduced in the Undernet which expands the powers of IRC operators and other users to control channels, and issues sanctions in ways not possible in other networks. In order to check these powers, UWorld has a separate access system as well as notices and log facilities implemented to heighten the transparency of some actions.

7.1 IRC Operator – Power and Control

The IRC operators occupy the role of the main officials inside the IRC: users appointed by IRC administrators to take over server- and network- and user-related administrative tasks, and for these purposes given access to privileged commands.

7 Controlling the Controllers?

This role has been present in the IRC from its inception on, and is apparently based on a similar role in the predecessor of the IRC, the Bitnet Relay Chat, as the description of their "Relay Operator" shows:

The day-to-day maintenance and operation of a Relay server is carried out by the "Relay Operator", an individual or group of individuals who are responsible to the institution's network administrative authority. The Relay Operator is also responsible for implementing corrective action to temporarily or permanently terminate a user's access to the Relay in cases of violation of the guidelines described in other sections of this document.¹

In a similar fashion, IRC operators manage the day-to-day operations of an IRC server and the IRC network, and can temporarily terminate a user's access.

Technically, in order to become nominated an IRC operator, one IRC administrator adds an specific config line (O-line) into the configuration file, providing the IRCop's nickname and a password (algorithm 15).

Algorithm 15 "Operator" configuration line (O-line)

```
# O: authorize operators. Fields are, in order, host name the operator must
#   be logged in from (wildcards allowed), operator's password, operator's
#   nickname. The first example allows me to become an operator from any
#   machine in BU.EDU by typing /oper wisner foo.
#
O:*.bu.edu:foo:wisner
O:bu-cs.bu.edu:bar:jsollyxcode
```

Source: [irc2.1.1/example.conf:27-33].

Once nominated this way, the IRC operator connects to the IRC network as normal user, and then issues the `/oper` command (see figure 7.1) in order to be recognized as IRCop by the server.

The main duties of IRC operators lie in three areas, in the order of importance²:

SERVER MAINTENANCE: The main IRCop duty concerns the smooth operation of the server of the IRC admin who nominated her. Various conditions from server code bugs to increased system load can lead to interruption of the operation, which immediately affects the entire network. The IRCop helps by restarting or entirely stopping the server process in case of disruptions, next to other server-related tasks.

NETWORK MAINTENANCE: In addition to server-related duties, IRC operators play an important role in the administration of the entire network. In fact, the group of IRC admins and IRC operators constitute the highest organizational level in an IRC network,

¹Bitnet-Relay (1986)

²See for example DALnet (2003); also Riedel (2001) for a detailed account of the network-related duties

since no corporation or other single entity above them exists. As part of these maintenance tasks, IRCops have access to commands to review the network structure and make changes in the interconnection between servers.

USER ASSISTANCE: IRC operators are also available for help and support of other users, but this duty is seen as least important: "[helping the users with problems] is done primarily in a IRCoper's "free time" if they choose so training and experience as a helper is not the primary focus or major need"³. When users threaten stability or operability of a server or the entire network though, actions against them become more important as they touch the higher ranking duties of IRCops (server and network maintenance).

To fulfil these duties, the IRC operators have access to privileged commands listed below (table 7.1).

Server-related commands		Network-related commands		Other commands	
Command	Description	Command	Description	Command	Description
/restart	Restarts IRC server process	/trace server	Returns info about all servers between local one and server	/kill nick	Exists user <i>nick</i> from network
/die	Stops IRC server process	/connect host port	Local server connect to <i>host</i> on <i>port</i> . Only hosts specified in configuration file allowed.	/wall message	Sends <i>message</i> to all connected users in the network
/rehash	Rereads ircd.conf configuration file into server process	/squit server	Breaks link between two remote servers ("server quit")	/oper nick password	Authorizes oneself as IRC operator. <i>nick</i> / <i>password</i> must match ircd.conf

Table 7.1: IRC operator privileged commands

From the user perspective, the most visible power of IRCops certainly is the `/kill` command already examined above⁴. But the power with the greatest potential for disruption are the network-related commands, since they can affect literally thousands of users, for example if an IRCop breaks the link between two servers in the middle of the network: In all channels, the membership is split in two, so a possibly large part of the members suddenly disappear from the channel (net split). The following show case illustrates this power.

Show case: Gaining channel operator status via /squit

In order to give an impression o the power of such commands, one mailing list message give details on an abuse of the `/squit` command to gain channel operator status (called a "channel takeover"):

"[H]e /SQUITed his server to create a chanop who then deoped all of the users ont he channel except himself and then signed off. After that, he AGAIN squited a server (his personal server this time i believe) and restarted ops on the channel."⁵

Here is the step-by-step reconstruction of the channel takeover:

³DALnet (2003)

⁴Chapter 5.1.

⁵randall@sumter.cso.uiuc.edu (1991-10-14) *Spewbabe's abuse of SQUIT*. Mailing list Operlist (1992)

7 Controlling the Controllers?

1. The starting point was the channel `+hottub`⁶, where a user `spewbabe` was banned from (the complaining user apparently had IRC operator status on his server).
2. `spewbabe` countered first by severing the link between his local server and the rest of the network with `/split` (net split). This left him with an empty channel `+hottub` on his side of the net split.
3. Since empty channels get deleted by the system, `spewbabe` could recreate the channel, and automatically gain channel operator status.
4. The server rejoined with the network, and the the two channel parts were put together. Because `spewbabe` was channel operator on his server, he was now channel operator on the channel where he had been banned from in the first place.
5. He now "deoped" (take operator status) from all other channel operators, leaving him the only chanop on the channel.⁷

Although this might sound complicated, the other alternative, to `/kill` all users in the channel⁸, would have taken much more time. More importantly, it would have resulted in a notification of all IRC operators of 20-30 `/kills` by one IRCop, arousing strong suspicions of the legitimacy of such an action. The way it actually happened though, only one to two `/split` notifications have been seen, which certainly did not get any attention by the IRC operators. So it strikes a good foresight (or a deliberate set up) that the user who sent this description to the mailing list logged the events as happened to document this misbehavior.

7.2 Nominating IRC operators and IRCop Netiquette

Positioned between the IRC administrator and the simple user, the IRC operator is in an awkward position: on one hand, she derives her legitimacy only through the assignment by *one* administrator in the network without any visible process (vote, vetoes etc.) or coordination between the admins. On the other hand, duties and powers given to her encompass the whole network. This discrepancy is reflected in the documents on IRCop policies.

"How do I get to be an IRC operator?"

The documents of IRC operators indicate that, despite the power and importance of this role, the large IRC networks have not chosen to implement more formal procedures for their nomi-

⁶The plus sign also denoted a named channel for a short time before it was superseded by the hash mark. So `+hottub` is the same as `#hottub`.

⁷According to the short description, the user did another `/split` ("his personal server"). This appears strange, since he already gained sole channel operator status in the first place.

⁸according to the description containing around 20-30 members

nation, leaving it to the individual IRC admins. Accordingly, these documents draw an somehow idealized but non-informative way on how to become an IRC operator: Build up trust with IRC officials, always be helpful in the network, etc. All texts agree in one point though: The best way to never been assigned IRC operator privileges is to ask for it. For the Undernet:

«How can I get to be an IRCop?

The easy answer to this question is that if you go around asking IRCops how you can get to be one, you probably will never be one. There is no list of potential IRCops or any kind of application procedure. There are actually two ways to become an IRCop: Start your own Undernet server or get asked by an admin to be an IRCop on their server. Starting a server is not practical for many people, so the latter is the more common way. Here it becomes a bit of a Catch-22 – an admin will only ask someone they trust to be an IRCop on their server, but if they think someone is cozying up to them just to get an O:line, they won't take that person seriously. You must be asked by a server admin, and asking an admin to make you an IRCop is a sure way of never becoming one. It is actually very difficult to become an IRCop on Undernet, but please don't think this means you can't help. #userguide, #Help, #mirc, #cservice, and the many other help channels are always on the lookout for well-intentioned people who really do just want to help out without looking for a quick way to get an O:line and become an IRCop. »⁹

And similarly, for DALnet:

The position of an IRCop is not one which can be applied for, it is granted by Server Administrators and tends to be given to people they have known for a number of years either on IRC, or in real life. Therefore, if the role of IRCop is your goal, you are likely to become disappointed and disillusioned over time. [...] There is also one sure fire way of ensuring that you never become an IRCop and that is by 'shopping' for an O:line, in other words, continually asking one or more Server Administrators to make you an IRCop.

This lack of transparent criteria and procedures of such an important official role leaves an impression of obscurity and secrecy which finds its continuation when looking at policies and norms which are to guide IRCop behavior.

IRC operator 'etiquette'

Documents paint a mixed picture regarding the behavior of existing IRC operators, interspersed with "etiquette"-like expectations on how they should behave. The earliest reference for the responsibility of IRCops is short, and rather factual than providing any guidance:

Obnoxiousness is not to be tolerated. But operators do not use /kill lightly.¹⁰

One reason for this terseness might be that back then, the number of participants was small, so that they knew each other well, and the difference between simple users, IRC operators and

⁹Undernet-User-Committee (2001)

¹⁰[irc2.1.1/doc/MANUAL]; see also Reid (1991) who discusses power, etiquette and behavior of IRC operators.

7 Controlling the Controllers?

IRC admins were blurred; most users in the early IRC were IRC admins, and therefore also IRC operators as well¹¹.

An early attempt to formulate an "Etiquette Guide" for IRC operators was made at the time when the EFnet emerged. But despite its title, it contains less 'etiquette rules' than rather a description of the commands available to them. Only unspecified norms are given, such as:

/kill and /wall are special operator commands. You should use them with care, and only if absolutely needed.¹²

And, to dilute the normative character even further, the last paragraph states:

Please let me know if there should be any additions to this guide. Again, this is not MANDATORY, this is just a GUIDE. Please conduct yourself as an IRC Operator would...you are looked upon for assistance, both emotional and mental.¹³

This guide was updated in January 1991, in May 1992, and again in December 1994. The last rewrite, claiming to be "more a reflection of the current state of IRC", is fairly pessimistic on the effects of norms, or such a guideline. See for example the comment accompanying the /kill command explanation (last line):

/kill is a special operator command. The format is as follows:
/kill NICKNAME comment.
Comment can be a phrase of almost any length (within reason) and should be used for specifying the reason of the kill.
Example:
/kill jonathan hey, this is fun!¹⁴

And even more explicit:

Don't bother wasting your time with #twilight_zone. There are many operators on many hub servers (including many found in this channel) which purport to know something but in fact know nothing. They will most likely ignore you, make fun of you, abuse you, dump your private messages to the channel, etc. In general, many operators (especially true of those on #twilight_zone) are not the sort of people you'd want to do this as a paid job and won't do anything unless it serves them in some way.¹⁵

This view from 1994 did not change in 1997, according to another IRC operators' guide:

From what I've seen, most opers look down on users, make fun of them, and ignore them.¹⁶

¹¹This also may be the reason why in early documents "operator" and "administrator" are sometimes mixed up. See for example [irc2.1.1/doc/INSTALL], where one section title reads «OPERATOR PRIVILEGES: How to become the IRC Adminestor on your site» (errors in the original text).

¹²Internet Relay Chat Operator Etiquette Guide 1991-1994, version 1991: [irc.2.6.1/doc/US-Admin/Operators]
¹³ibid.

¹⁴Internet Relay Chat Operator Etiquette Guide 1991-1994, version 1994: [irc.2.8.21/doc/US-Admin/Operators]

¹⁵ibid.

¹⁶Brinton (1997)

But the author takes a more pragmatic view by adding:

There are a lot of politics that go on in the irc operator community and, whether you like it or not, these politics are here to stay. Fighting this and complaining about it will get you nowhere.¹⁷

As for social norms, he offers more generic ones, such as using common sense when apply a `/kill` against a user, and to give reasons for such actions, help other users, or to regard the social order of ops and admins:

On occasion, ops have their disagreements. There is a bit of a pecking order that exists in the op ranks, usually with hub admins and ops being more "powerful" than leaf admins and ops. It's generally not a good idea to try to win an argument with the people who are providing your connectivity to the IRC network. For that matter, it's generally not a good idea to try to win any argument at all. If you do have a serious problem with another op, and can't resolve it directly with him/her, go to your admin about it. Your admin can then approach the issue with the other op's admin, and if that goes nowhere, with their uplink admin. This is a quick way to make enemies, so make sure it's important to you before doing it.¹⁸

Apparently, the role of IRC operators always remained a source of disputes, controversies and power struggles. Part of this may also be attributed to the growth of usership and number of servers connected, leading to the need of more IRC operators to manage the ever growing network. The creation of other network with the explicitly stated goal to offer a more user-centered network (Undernet, DALnet) can also be seen as reaction to these problems.

7.3 Notices and Logs

The previous section has given an impression of the contentious role of the IRC operator. Appointed by nomination by one IRC administrator based on no formal or even discernible criteria, but given power over the network and the users. Supposed to act for the common good (mainly network stability, but also user help) of the network, but apparently pursuing other interests as well, for which they even actively abuse their powers.

Beyond «code» tools which implement mechanisms to check these powers, such as limiting the scope¹⁹ or functionality²⁰ of IRCop commands, an important prerequisite for any control is the capability to *observe* the actions, to monitor the behavior. In a social setting like the IRC, the ability to monitor the controller offers a kind of counterbalance to their powers, and a means to foster a 'community' to enable mutual monitoring, or behavior correction through informal control²¹ by the community.

¹⁷ibid.

¹⁸ibid.

¹⁹Local operator role; `/kills` only for locally connected users or entirely revoked: see above chapter 5.1.2

²⁰Limits in K-line access to IRC operators: see chapter 5.3.

²¹See Ellickson (1991, p.131).

7 Controlling the Controllers?

In the IRC, there are three groups who may engage monitoring IRC operators: IRC admins (in addition to their «code» means to limit the powers), other IRC operators (as peer control), and users which are the most affected group of IRCops' actions. Hierarchically, these correspond to a top-bottom, peer, and bottom-up control.

These parties do of not stand by themselves: One could well imagine that a dispute by a user is easier resolved if some records are available to other IRCops or admins which can help to evaluate the claims. Similarly, mutual information about actions between IRC operators, together with tight communication connections between them (channels, mailing lists) might facilitate norm building and mutual monitoring between the IRCops.

In any case, in the IRC server code, two mechanisms can be distinguished where the servers automatically transmit information about IRCops' actions to these groups:

Notices are sent automatically upon the execution of certain actions, with the main audience being IRC operators²² and users.

Logs are written into files by the server, and therefore are primarily an IRC administrator tool to review logged actions after they happened.

7.3.1 Notices

"The hole point of KILL's being visible to everyone is that, if needed, futher explanations can be asked."²³

One mechanism to make system actions transparent to others are *notices*. These are text messages sent by the server when some action – such as a command by some user – is executed. Depending on the implementation, notices can be sent upon the successful execution, but also in case of errors, or other conditions.

An important characteristic of notices is their target or audience. In some cases, this audience is hard-coded to a fixed group (e.g., all IRC operators), in others a more variable group is given access to it. And finally, the type and amount of the contents of the notice might differ.

Basic functionality and implementation

The functionality of notices can be shown by reviewing the sequence of actions when an IRC operator issues a `/kill` command²⁴. The corresponding function `m_kill()` in the server code first checks two conditions (Is issuer an IRCop? Is the target user in the network?) and (upon passing the conditions) executes two actions (database synchronization, disconnection of the target user); in all cases, notices are sent out.

²²which includes IRC admins who mostly will be IRCops as well

²³Savela, Markku (1993-02-20) *Oper should not just sign on to IRC and then disappear*. Mailing list *operlist* (1993).

²⁴Chapter 5.1.1

One kind of notices is self-evident, as it derives from the command itself: Those sent to the command issuer, either on the successful execution, or due to some error or rejection, and possibly those sent to the target(s) of a command as well, in the case of a `/kill` the affected user.

My focus here lies on the other notices which do not directly follow from the command itself. Of the many code lines in the function which send messages to users (command issuer, target user) as well as servers (database synchronization), one code line sticks out:

```
sendto_ops("Received KILL message for %s. Path: %s!%s", user, cptr->host, oldpath);25
```

In this example, the function `sendto_ops()` sends the message "Received KILL message for", followed by the nickname of the exited user and a host path to all IRC operators²⁶. The host path is a sequence of the names of all hosts from the receiving operator to the host where the command has been received, followed by the nickname of the issuer. Therefore all information (action, issuer, target user) about every `/kill` is sent to all IRC operators present in the network.

A notable detail in the implementation is the presence of a dedicated function for sending those messages to all operators, `sendto_ops()`, indicating that it is used in many places throughout the code. Table 7.2 on the following page shows all places where the function is used.

Consistent with the primary duties of IRC operators, all but the last two messages give notice about server connections. The last message is a command which allows users to send an (arbitrary) message to all IRCops. Therefore, the notice with the `/kill` command in this server version is the only which informs IRCops of a user-related action.

As example for the use of these notices, one user created a "top ten list" of the IRC operators who issued the most `/kills`²⁷. This was used as an argument in a discussion about the usefulness of the `/kill` command. The apparent importance of a discussion about this command is expressed by someone else in the same discussion:

"Amazing isn't it that the biggest discussion for quite a while on operlist has been of the KILL command."²⁸

It appears that the existence of these notices created the awareness about problems which then could be discussed, and leads to resolutions like the change of the `/kill` commands as described above in chapter 5.1.2.

²⁵[irc2.1.1/s_msg.c:807-808]

²⁶Function `sendto_ops()`, defined in [irc2.1.1/s_msg.c:154-168].

²⁷monie's matte green (1992-12-16) *Re: KILL*. Mailing list *Operlist* (1993)

²⁸Watts, Andrew (1992-12-17) *Kills etc etc*. Mailing list *Operlist* (1992).

Message sent to IRCop	Associated outcome of action	Location
Connection to <host> established	Automatic connection to other server succeeded	ircd.c:159 main()
Received unauthorized connection from <host>	Connecting user or server rejected	s_bsd.c:223 read_msg()
Lost server connection to <host>	Connection to user disrupted	s_bsd.c:241 read_msg()
Connection to <host> established	Connection request by user or server approved	s_bsd.c:247 read_msg()
Link with <host> established	Success of a IRCop-issued '/server' command	s_msg.c:755 m_server()
Received KILL message for <nick>. Path: <user>!<host>	IRCop issued '/kill' command	s_msg.c:807 m_kill()
<nick>: <message>	User issued '/woper' command	s_msg.c:1378 m_woper()

Table 7.2: Notices sent to IRCops in irc server version irc2.1.1

Notices allow the recipients to get a better overview about what is happening in the network, and can help fostering the 'community' between the participants of an IRC network.

Quantitatively, one can observe a continuous growth of notices sent to IRC operators. Whereas in the whole irc2.1.1 code (October 1989) only three `sendto_ops()` are used, irc2.4 (May 1990) already uses it in 14 places throughout the code, and it grows steadily in the EFnet to 65 in hybrid-6.0 (January 2001). This indicates an active use of the notices, and necessity or demand for more fine grained notices about the network.

Changes in notice mechanism

Next to the growth in notices, some changes in the mechanisms are notable. For a long time, all notices could only be received by the IRC operators; normal users did not have access to them. But with the beginning of the irc2.7 code series, the coders made it possible that a number of notices could also be received by users who wished to do so.

The functionality was implemented by expanding a mechanism called *user modes*. Similar to channel modes²⁹, it allows the user to configure characteristics concerning her representation in the IRC. For example, an "invisible" mode takes the user out of the list of users returned by the `/who` command. Also, IRC operator status is implemented as an user mode. User modes are displayed or changed with the `/mode` command³⁰.

The irc2.7 series (January 1992) introduced the *server notice* user mode, the ability to receive server notices. When set, the user was receiving all notices that IRC operators also received. If for example an IRCop issued a kill, all these users were receiving a notice like

²⁹See above chapter 4.3.1.

³⁰For a description of these modes and command, see for example Pioch (1993) (section 2.4, "Channel and User Modes").

```
*** Notice -- Received KILL message for user123. Path: svr3!svr2!svr1!someOper
```

informing them on a `/kill` issued by the IRC operator `someOper` for the user `user123`. So beginning with this server version, users had access to the same notices that before were reserved to the IRC operators.

The feature was implemented through a change in the `sendto_ops()` function. Where before all users tagged 'operator' was sent the message, now all users tagged 'receives server notices' were sent it instead, and no distinction between users and IRCops were made anymore.

The server notice mode allowed user only to either receive all notices, or none at all. An functional expansion of the reception of notices by users was made in server version `irc2.8.21+CSr20` (Jan. 1996). Now every notice is subsumed by the coders under one of six categories. Users and IRC operators can subscribe to the categories, and then receive the respective notices.

Two of them can only be received by IRC operators:

- *(no name)*: Notices which are sent to all IRC operators. These include failed IRCop authorizations, IRCop addition to K-lines (command `/kline`³¹), and reaching maximal numbers of connected clients and servers.
- *C-mode*: Only for IRCops, but they can opt to not receive them. When subscribed, IRC operators are notified of every successful client connection.

The remaining four modes can be subscribed by any user:

- *K-mode*: A notice for every issued `/kill` command.
- *F-mode*: A notice for every nickname collision (a nick change or nick registration collides with an already used nick³²).
- *R-mode*: A notice for every client connection rejection by a server. Rejection can occur on various conditions, for example a prohibition to use bots, to connect more than once to a server³³, or limitations on the username format³⁴.
- *U-mode*: A notice that the maximal number of allowed client connection to a server has been exceeded.

In this way, users and IRC operators can each decide on the kind and amount of server-generated notices that they receive, instead of receiving either all notices or none at all.

³¹See above chapter 5.3

³²See above chapter 6.1.1.

³³These multiple connections are called *clones*; see below section 7.4.2.2.

³⁴Using special characters, or mixed case characters.

7.3.2 Logging

Logging, i.e. saving IRC messages to a file for later review, can occur at various places in the IRC. Generally every user can log her conversations through her IRC client program³⁵. In this way, log files are used to save conversations for retrieval in web sites etc. As example, the Undernet opschool³⁶ saves logs of its channel operator courses which then can be downloaded on its website³⁷.

More important than user log files are those which the server itself generates upon issued commands, or system status events etc. Due to its nature as server-written files (similar to the configuration file), access to them is limited to the IRC administrators. They therefore are a control tool of IRC administrator which allows her to replay events, for example what happened in her absence. Server log files are a valuable tool to control the actions of the IRC operators.

As example of server logs, logging of IRCop actions are introduced in server version irc2.8.5 (April 1993), where every IRC operator authorization could be logged to a file, and, using the Unix-provided "system log" facility³⁸, `/kill`, `/squit`, `/connect` (server connect) and (again) operator authorization³⁹. As other additions, irc2.8.21+CSr20 (Jan. 1996) specifically logged failed IRC operator authorizations⁴⁰, and hybrid-4.3 (Aug. 1997) the `/kline` command⁴¹.

Similar to the notices above, although on a smaller scale, a tendency to collect more information about events can be observed. Successively, the coders have added new logging facilities to keep the IRC administrator updated about what the IRC operators have done. And since the logs are only accessible by the IRC admin, they can be seen as the most accurate record of events.

7.4 The Undernet UWorld Service

This last section of the chapter presents a comprehensive system of power and control, implemented in the Undernet network, known as the *UWorld service*. As the name indicates, it is a service bot similar to the channel service bots reviewed above⁴². In contrast to these

³⁵The show case in section 7.1 above is based on the log file of the complaining user, large parts of which was attached to the mail that he sent to the mailing list.

³⁶See above chapter 6.3.2.

³⁷<http://cservice.undernet.org/main/opschool/>

³⁸The "syslog" facility available on Unix operating systems allows processes to write log messages to central log files.

³⁹[irc2.8.5/include/config.h:201-328]

⁴⁰[irc2.8.21+CSr20] (Jan 1996)

⁴¹See above chapter 5.3.

⁴²Chapter 6.3. This similarity goes even further since one of the services reviewed there is the X/W channel service bot which has strong connections to the Uworld bot.

though, the UWorld offers its commands and functions as expansion for IRC operators' channel support duties. As we have seen above, this user support is generally considered the least important duty of IRC operators, so there are only few tools available to them. By instituting UWorld, the Undernet departed from this philosophy, instead providing a number of commands and automatic functions to UWorld-authorized users (not necessarily IRC operators themselves) to cope with channel-related (and other) issues.

This case study first gives an overview of the UWorld service, its basic setting (section 7.4.1), followed by the main functionalities (section 7.4.2). Then, both the user access system (section 7.4.3) as well as notices, logging and information means as control (section 7.4.4) are examined.

7.4.1 About the UWorld Service

UWorld is a service bot, similar in working to the bots and services examined above⁴³. From one central place in the network, it provides commands and other functionality not available by the servers: the ability of IRC operators to interfere in channel affairs, issue netwide bans, and actively fight specific user behavior (flooding, clones) deemed an offense by the Undernet.

According to a historical account of the Undernet, UWorld appears to have been present from the start on⁴⁴. This may be not coincidental, because the creator of UWorld, Daniel Mitchell ("WildThang") was one of the founders of the Undernet. It seems though that the functionality had not been widely known for some time:

"There's a long debate over the existence and use of Underworld (Underworld was the server name, UWorld was the bot name). It had been linked since the norman* server was first linked back in December 1992, but many opers and admins didn't know what it was for. It was agreed that it provided necessary and helpful services"⁴⁵

Besides its functionality, a main "characteristic" of the service was the non-availability of the source code. Mitchell did not release the UWorld code into the open source, but instead sold binary versions of UWorld through his company, Chatsystems Inc. This may be one of the reasons why there only sporadic mentions can be found of the service, and no documentation or detailed functional descriptions.

The mentions are as short as the following:

"[On what an IRCop is] IRCops also have the ability to use Undernet's services like UWorld, EUWorld, and UWorld2 in attempts to keep Undernet together as a network.

⁴³See chapters 6.2 (bots) and 6.3 (services).

⁴⁴See quote below, according to which Uworld linked to the Underworld in December 1992. Undernet formed itself at the same time.

⁴⁵Mirashi and Brown (2003): "[Uworld] had been linked since the norman* server was first linked back in December 1992" (Undernet formed itself in late December 1992).

7 Controlling the Controllers?

”[Secondary duties] may include [...] using UWorld, UWorld2, or EUWorld to resolve channel problems.”⁴⁶

The UWorld code went through three revisions, but technical problems as well as deliberate attacks on the service led the Undernet principals to think about alternatives. One such alternative was EUWorld, a independently programmed, but functionally equivalent to UWorld which provided the services mainly on the European side of the Undernet. Later, another alternative emerged in form of an open sourced generic service bot framework named GNUWorld⁴⁷ which offered to replace both UWorld (as well as the X/W channel registration service). In May 2001, the UWorld replacement GNUWorld ”CControl” was ready, which Mitchell opposed to at first. But as problems with the UWorld furthered, CControl was connected to Underworld first in December 2001 to replace EUWorld, and finally in May 2002 also on the US side, replacing UWorld which delinked in April 2002.

Although the source code of UWorld was not released to the open source, Mitchell did release the source code of an older version, because ”numerous people have shown lots of interest in obtaining it”⁴⁸. As for the code itself, he explicitly points to the poor quality of the code, and the lack of documentation:

”This code was hacked/thrown together over about 7 years of tossing ideas around, learning more about C, and is quite honestly NOT something I would dare to consider professional quality code.[...]I apologize for the lack of documentation or instructions.”⁴⁹

Despite these claims, I have based the following explorations on this source code, partly because no other version was available, and partly because, developed as a closed source software, it includes some interesting «code» structures which might not be available in openly developed software⁵⁰.

7.4.2 Functionality

The main functionality of the UWorld can be roughly separated into two categories:

COMMANDS: UWorld offers a number of commands, such as banning users, or changing channel settings or membership.

⁴⁶Undernet-User-Committee (2001)

⁴⁷<http://www.gnuworld.org/>

⁴⁸UWorld 2.0+ wild source code, file [uworld/README]

⁴⁹ibid.

⁵⁰It could be interesting to compare the UWorld code with its successor, the GNUworld CControl module. Next to «code» governance differences, such a comparison might for example support the results given by Kesan and Shah (2002, 2003a) who argue that different institutions (they studied university, firms, consortia, open source movement) incorporate different value sets into the code.

AUTOMATIC FUNCTIONS: UWorld traces all activities in the entire Undernet, and can initiate actions according to its settings. This includes denying entry to banned users and detecting clones (multiple user connections from one host) and floods (rapid succession of data being sent over the network).

7.4.2.1 UWorld commands

The commands available through the UWorld are notable as they break with many philosophies and policies upheld in other IRC networks through the server code: Direct channel interference by IRC operators and network-wide bans were both not available, although there was no technical reason which would have hindered their implementation (as the UWorld itself shows). The first two subsections below trace the implementation of these functionalities in the UWorld.

But the two other functionalities presented here are as notable as the first ones: The ability to "masskill" may be comparable to the simple `/kill` command, but "nuking" users, flooding users with messages for the purpose of disconnecting them, definitely is not. Another command, "mycmd" allows to submit arbitrary commands, circumventing any UWorld-implemented control structures (validity checks, notice and log facilities), and therefore can be seen as the most powerful command in the UWorld, for which one could imagine many uses with strong governance implications, as it hands the unchecked power of the UWorld into the hand of the issuer. The existence of the latter two commands might be attributed to the lack of an open source peer review.

Channel related commands

Most IRC networks follow the principle of no-interference in channel affairs, as long as the network stability is not affected. But apparently this has not stopped IRC officials to interfere in an indirect way, as shown above⁵¹.

UWorld implements a number of commands which allow to *directly* manipulate channels in various ways: There are commands which override any channel modes set by channel operators (command `opcom mode`); the commands `clearop` and `clearbans` delete the list of channel operators and the channel ban list, respectively. As for the users of a channel, `opcom kick` allows to exit any user from a channel, `deop` takes channel operator status away from the user, and `reop` gives channel operator status to a user. In this way, any aspect of the channels (channel mode, operators, membership) can be controlled from the Uworld.

From the technical perspective, the UWorld actions do not need any special provisions on the IRC server side. The UWorld commands are converted into appropriate server messages and sent into the network. Since the UWorld is identified by the others as an IRC server itself,

⁵¹The show case in chapter 7.1.

all such commands are readily accepted without further authorization checks. The only user authorization which takes place is that of the UWorld itself⁵².

Besides the power that UWorld gives its users over channel affairs, the other rationale to implement these commands are its use by the X/W channel service⁵³. Rather than enforce its policies itself, the X/W service connects to UWorld and uses its commands. This relieves X/W from having to duplicate the commands and services already offered by the UWorld, as well as the control structures (access control system).

G-lines: Undernet-wide user bans

In chapter 5 I have shown how long term sanctions – K-lines – have been user entry denials for one server, initially to be issued only by IRC admins, only later one to be given limited access to for IRC operators (`/kline` command, chapter 5.3). The UWorld `gline` command breaks with both principles: It allows *UWorld users* (which do not necessarily have to be IRC operators) to issue *Undernet-wide* bans (called G-lines⁵⁴). These G-line bans can be managed only through the UWorld: although G-lines are active in the IRC servers next with K-lines, they can only be set or changed through the UWorld; there is no direct access of IRC admins or IRC operators to the G-lines in their own server.

G-lines are time-limited: UWorld code sets a limit of ten days for a ban. Interesting here is the immediate sanctioning function built into the command: If an UWorld user issues a `gline` command with a duration longer than ten days, then the UWorld *immediately* sanctions this user by g-lining her for twenty days. UWorld users with a very high access level (see below) though are exempted from the ten day limit, they may issue G-lines with arbitrary duration.

As another important detail, G-lines are associated with two configuration files in the UWorld: the *G-line file*, and the *G-line exception* file. Every time a `gline` command is issued, it is also saved to the G-line file. In case of a restart of the UWorld, this file is read in, thereby preserving the G-lines in case of disruptions. Also, UWorld admins⁵⁵ can change G-lines by directly editing this file. And similar to the exceptions to the K-lines in the IRC server⁵⁶, UWorld allows its admins to supply an G-line exception file: none of the users or user groups in this file can be G-lined by UWorld users.

Note the subtle differences in control over these functionalities, through the interface offered to these two files:

- G-lines can be issued by UWorld users and are automatically *added* to the G-line file.

⁵²See below section 7.4.3.

⁵³See above chapter 6.3.2.

⁵⁴This is confusing since "G-lines" are not configuration lines in an IRC server config file. The name is instead derived from the functional similarity to K-lines.

⁵⁵Not Uworld *users*. This is similar to the difference between IRC admins and IRC operators: Only the Uworld admins have access to the file system of the host where Uworld runs, and can therefore change the G-line file (as well as all other such configuration files).

⁵⁶E-lines; see above chapter 5.3.2

But that file can only be *edited* by the UWorld admins; deletion of G-lines occurs automatically after the set duration.

- Exceptions may only be made by UWorld admins by editing the G-line exception file; no command interface exists which would allow UWorld users to change its content.

Technically, the IRC server code has been changed in order to accommodate the G-lines. But, as mentioned above, neither IRC admins nor IRC operators have direct access to them on their server, but need to access them through UWorld, an interesting exception from the principle of absolute control that the IRC admin has over her own server.

Kills, masskills, and nukes

The first two of these commands are rather a 'convenience' function for IRC operators, but give UWorld users who are not IRC operators the same functionality: The `kill` command is the same as a `/kill`, whereas `masskill` allows to provide more than one user or one user group at once, living up to its rather disgusting name.

More noteworthy than these commands is the `numnuk` command: It allows to nuke⁵⁷ a user, to send hundreds of messages in rapid succession with the intention to disconnect her from the server. Nuking users is considered a severe offense in the IRC:

"[Are nuking allowed in Irc?]

"Nuking are definitely not allowed in IRC.If anyone caught nuking people he/she will have a straight ban from the sever by an IRcop.Ban means a forever ban from that sever,meaning you can never enter that sever again. So please think twice before you want to do this"⁵⁸

Even among those who might consider attacks in the IRC as not per se illegitimate, nukes have a low standing:

"Nuking is not clever and certainly not '3lit3'(elite) unless you wrote the software yourself, or at least designed it for someone else to write and compile. And nuking is never as big or as admirable as being able to win a battle intelligence and wit. Nukers are generally the tools of lamerz who have no other means to get notice and respect."⁵⁹

One can only imagine that this command has been implemented to simulate nuking attacks in order to design safeguards against it. As indication of this intent stands the fact that this command is only available to UWorld users of the highest authorization level (see below).

⁵⁷This term is sometimes interchangeably used with flooding. Nuking though is generally used for user floods, whereas flooding also occur in channels, or against servers; also the intention of nuking is generally described as disconnecting users.

⁵⁸<http://www.geocities.com/adazmy/Whatirc.htm> (2002-08-07)

⁵⁹<http://www.geocities.com/adazmy/Strategies.htm>(7 Aug 2002)

Mycmd – A command to execute arbitrary commands

Commands like the mycmd command can be often seen in software which is in the process of development, as developer back door to the system: mycmd allows to send arbitrary messages to the UWorld.

With normal UWorld commands, the service receives the command string and the parameters, constructs a well-formed⁶⁰ message string, and sends this string to the appropriate targets (servers, users). In addition, it executes various checks (authorization etc.) as well as control mechanisms (notices to others; logging etc.). In contrast, the mycmd command sends the given parameter string *verbatim* out to the servers; neither message string construction nor checks etc. are executed.

For example, an UWorld user would issue the command in the form:

```
gline 600 anuser@anhost.com Test
```

UWorld transforms it into the message string

```
GLINE * +anuser@anhost.com 600 :Banned  
(+anuser@anhost.com) until 1104534600 (Test)61
```

and send this string to all Undernet servers, resulting in a G-line in all servers for that user. Also, the gline function would check a number of conditions, create log entries, and send notices to operators.

With the mycmd command, the issuer gives the second message string as parameter, and the effect is the same as in the first command string: a gline is issued. The difference is that all condition checks, log entries, notices part of the command processing inside the UWorld are circumvented in the second case, since no command processing takes place.

Therefore, the mycmd command is the most powerful command in the UWorld: it allows to issue arbitrary commands which are then send with the authority of the UWorld service bot to the servers and users, but without any built-in checks, notices or logs. The only condition built-in to mycmd is that only UWorld users with the highest authorization level are allowed to issue this command, which is quite understandable. Still, neither mycmd nor numnuk commands would be imaginable in an open sourced program.

7.4.2.2 UWorld automatic functions

In addition to commands issued directly by the users, UWorld includes a number of functions which are automatically executed based on some set conditions. As service bot, UWorld has access to the same global state information available to all IRC server in the network, and

⁶⁰I.e., conforming to the appropriate communication protocol.

⁶¹The whole string actually consists of only one text line, and has been separated into two to fit into the text layout.

receives all network messages. UWorld can therefore be instructed to act in several ways on such messages. Examples implemented in UWorld are given below: Dealing with clones and flood protection.

Dealing with Clones

When more than one client from one host connect simultaneously to an IRC network, these clients are called *clones*. Clones are considered offensive in IRC networks:

- ”I) Clones can be loaded to all attack one target with a flood of data, usually in the form of excess DCC requests or ICMP data packets. Many Floods are only effective when used by 3-4 at once.
- ”II) Clones can use your nickname when you are offline to cause trouble and start a war in order to ‘tarnish’ your name.
- ”III) Clones can be ‘cloaked’ behind your nickname so that, to everyone else, whatever they do seems as if you had done it.”⁶²

UWorld provides a clone checking facility by tracking the number of clients for each site connected to the network. It can be directed to react in several ways:

- *User warning:* UWorld sends back a message to the user who exceeded the allowed-clients-per-host count a text which looks like this:

 ”WARNING: Multiple connections from a single user host are considered clones. If this continues, you risk being banned from the entire Undernet. (Think about it.. Is it worth it?) Undernet will not tolerate flooding or clones. Your host has been added to the logfiles”⁶³
- *Group notice:* A notice is sent out to a special oper-only channel that a possible clone has been detected.
- *Auto-sanction:* The connecting client is automatically glined: the client is disconnected from the network, and a netwide entry denial for 10 minutes issued.

Also, UWorld manages a number of lists which further influence the conditions and actions in case of clones:

- A list of hosts is kept from which no clones are allowed at all. This list can only be changed by editing a UWorld configuration file. The UWorld command interface only allows to examine the list (`showshost`) and to reload the file (`loadshost`).
- Another list contains all hosts from which clones are always allowed. Again, this list can only be changed by editing a configuration file, and the command interface is limited to examination (`showmhost`) and reloading (`loadmhost`) of the list.

⁶²[http://www.geocities.com/adazmy/Strategies.htm\(2002-08-07\)](http://www.geocities.com/adazmy/Strategies.htm(2002-08-07))

⁶³Uworld 2.0+wild, file [uworld/nicklist.c], line 1246

7 Controlling the Controllers?

- A third list contains all hosts which are automatically glined, independent of the current clone-related setting. As with the lists above, this is configuration-file only changeable (commands `showagl` and `loadagl`, resp.)

Further functions are implemented into UWorld, suggesting that this feature has received much attention in design and implementation, and must have been of quite some importance for the Undernet officials.

Flood protection

This feature is a self-protection mechanism for UWorld. It protects the service bot against being made non-responsive because it is flooded with (bogus) service requests. Basically it consists of recording the number of requests made in a given time frame, and if this number exceeds ten requests, the originating client is disconnected from the Undernet (`kill`). This feature can be activated or deactivated from the UWorld command interface (command `floodprotection`). With another command (`floodlog`), UWorld can be directed to write every request that it receives into a special log file. This allows the UWorld admins to inspect the flow of requests, and initiate appropriate actions against flooders.

7.4.3 UWorld User Access

Given the various functionalities offered by UWorld, it comes at no surprise that the coders have implemented an access control system to UWorld. The main features of this system is a level-based access to the commands, similar to that of the channel services⁶⁴, and its independence from the normal IRC operator authorization.

The access to commands of the UWorld is determined by a level value assigned to each user. Normal Undernet users have an access level of zero (0) which means that access to UWorld is altogether denied. Negative access levels even can trigger further actions, such as being automatically banned from the Undernet. Positive values give access to UWorld commands, with higher levels authorizing more commands. In addition, some commands require IRC operator status in the Undernet on lower levels.

The levels are written into a special UWorld user list. It contains all users who have a status in UWorld different from the normal Undernet users status (i.e., level zero). On startup (and upon the `reload` or `loadusr` commands) the list is read in from a users configuration file; UWorld admins therefore can give users access to UWorld by adding them with an appropriate level to this file. While UWorld is running, a command interface allows changes to this list: the command `addusr` adds a user to the list, with the given access level. Similarly, `remusr` removes a user entry.

⁶⁴See above chapters 6.3.1 and 6.3.2.

A special case is the IRC operator status, which is treated as independent of the level system. Normally only set when a user is IRC operator, a non-operator user can be assigned IRC operator status for the UWorld by issuing the `makeop` command. This status is valid only inside the UWorld; but the powers given can be similar to that of a normal IRC operator, so that this status can be considered an 'UWorld IRCop' one.

In order to gain access to the UWorld commands, a user authorizes herself with the `vrfy` command. If this user matches an entry in the UWorld user list and has provided the correct password, she is now recognized as authorized user and can further issue UWorld commands according to the access level given.

UWorld distinguishes roughly six access levels, denoted in table 7.3. As usual, with higher level comes access to more and more powerful commands. The above mentioned all-purpose `mycmd` command for example requires the highest level 10.

LEVELS	COMMANDS
1	help, reop, vrfy
1 + oper	version, wibble, leave, join, users, stats, bans, testban, loadglines, showbans, clones, chanlevel, clearchan, clearops, clearbans, clearmodes, gline, mgline, rline, remailgline, remgline, autoban, mautoban, remautoban, kiltel
6	DCC dmsg/dwhois/dccwalkwalklist, info, servs, protect, unprotect, scan2, nofloods, operin, operrem, showoperchan, operadd, showkillchan, remkillchan, killchan, scan, lusers, lastcomm, lastlog, opcom
8	spew, actanal, ipscan, uidgline, tsc, floodlog, secureoper, clearkick, snoop, kickass, dogline, strictscan, reconnect, jupe, makehelp
9	nohelp, makeop, opersuspend, chansuspend, nokill, autokick, autocore, floodprotection, logjoin, loghost, addusr, remusr, showagl, showmhost, showshost, showsuspend, showcsuspend, loadchansuspend, loadkillchan, loadsuspend, loadshost, loadmhost, loadagl, loadnogl, showfloods, loadusr, dokill, reboot, masskill, check, reconnect
10	nickserv, dkill, reload, killlog, noopers, maxses, maxsite, autogline, warnclone, maxclients, mycmd, make-mybots, killmybots, checkmask, showhost, nickdump, filenick, numnuk, numnuk2, numnuk3, kill, deop, dome, exit

Table 7.3: Uworld commands and access level

Another interesting feature is the ability to deny IRC operators access to UWorld. The command `opersuspend` suspends an IRC operator's access for a given duration, if the IRCop has access level 7 or lower. Any user with level 8 and up cannot be denied access to the UWorld.

7.4.4 Control in UWorld: Information, Notices, and Logs

As the description so far has shown, the UWorld service expands the power that an Undernet official can have inside the network, allowing its users to change channel settings, issues bans, etc. To control the use of its power, the coders of the UWorld have implemented control mechanisms, similar to those already examined in the IRC servers: In addition to *notices* and

logging mechanisms, access to *information* is equally important, as the UWorld is a centralized service, and therefore information not as readily available as in the distributed server network.

Information

UWorld offers a little over 20 (out of 122) commands which are used to get various information, from a help text and UWorld version string to the contents of the many access lists. The amount of information depends on the access level of the user.

On level 1, the default level for IRC operators, there are only a few informational commands available. The three important commands are `users`, which shows the list of UWorld users including their access level, `showbans`, which lists all active glines, and `clones`, which lists all hosts that UWorld considers to have clones (multiple simultaneous client connections from one host).

For further access to information, users need level 6 or higher. Level 6 for example allows to see the list of all oper-only channels managed by the UWorld, and also the list of the last 15 commands that were issued through the UWorld. Level 9 gives access to almost all lists managed by UWorld: The list of auto-glined hosts, the hosts from which no clones are allowed, and the hosts from which clones are allowed; the list of suspended IRC operators and suspended channels; and the list of potential flood hosts. Finally, users with the highest level 10 are allowed to see the list of every host that UWorld knows of.

Taken together, IRC operators are given the basic information (user list, active glines, possible clones), whereas for further status information of the UWorld, at least level 6 is necessary, which has to be explicitly assigned by an UWorld user of level 9 or 10.

Notices

The main source of information about actions of UWorld are sent via notices, special messages to a specific user group or a special channel. UWorld employs two main ways to send out notices: `wallops` and `#UWorld.floods`. *Wallops*, short for 'write to all ops', sends a notice to every IRC operator logged into the network. This allows the IRCops to stay informed of the UWorld actions.

The other way is the UWorld-managed channel named `#UWorld.floods`. This is a oper-only channel, that means only IRC operators (and those UWorld users assigned quasi-oper status) may enter this channel. As with the `wallops`, UWorld sends a text line if it executes certain actions.

In addition to these two main ways, there are a few commands where users connected through a DCC connection to UWorld receive a notice. The following table lists all commands which trigger a notice, together with the level needed to issue the command, and the kind of notice sent by UWorld (table 7.4).

`Wallops` are the default method to send out notices. The channel method as well as the DCC

COMMAND	LEVEL	WALLOP	#U.F	DCC
clearchan	1+oper	X	X	
gline	1+oper	X	X	
rgline	1+oper	X		
remgline, remallgline	1+oper	X		
autoban, mautoban	1+oper	X		
remautoban	1+oper	X		
killchan	6	X		
opcom kick opcom mode	6+oper or 7	X	X	
secureoper, kickass, dogline (toggles)	8	X		
jupe	8	X		
operadd	9	X		
opersuspend	9			X
chansuspend	9			X
reboot	9	X		
masskill	9	X		
exit	10			X

Table 7.4: Uworld commands triggering notices

notices are used only for a few commands. Also, easily detectable from the table are the most powerful commands on the lower levels: `clearchan`, `gline`, and both `opcom` commands trigger both wallops and `#uworld.floods` channel notices, informing both recipient groups.

Not so obvious is the distribution of notices in relation to the access level of the command: While the important commands on level 1 + oper are at least walloped (6 out of 25), in the higher levels fewer ones send out notices (on level 8: 4 out of 15; on level 9: 5 out of 32). On the highest level 10, only the `exit` (stopping the UWorld process) sends out a DCC notice; none of the commands are walloped or sent to the special channel.

Logging

In addition to notices, UWorld maintains a number of log files, some of them all the time, some conditional to activation via UWorld commands. None of these log files are accessible through the command interface, therefore only accessible by the UWorld admins. In the available code version though, most of the log facilities are not fully implemented, but only stubs with no actual function. Six logging facilities are fully implemented, three of them are always turned on, one can be disabled at compile time (a debug log), and two can be explicitly turned on or off via the command interface.

Of the three logs always turned on, the first one is a global logging facility recording every⁶⁵ UWorld command received, including the issuer, all parameters and a timestamp. This allows

⁶⁵Almost: the commands `dccwall`, `wall`, `dmsg`, `dlist`, `dwho`, and `umode` are ignored.

to replay the whole usage of the UWorld service.

The other two logs are similar, but limited to specific commands: One creates log files for each occurrence of the `gline` command. It is obvious that this must have been of importance for the coders and UWorld admins: One file contains all data of each `gline` command issued in one day, with the date as part of the filename. This allows the admins to check all `glines` issued on one day. The other log records every `KILL` message sent over the Undernet, which includes those issued by IRC operators with the `/kill` command. So both logs together allow a good overview of both short-term and long-term banishments in the Undernet.

These logfiles are only accessible to the UWorld admins; none of the other officials, including IRC admins or IRC operators, have access to these files.

Next to the debug log, which is similar to the general log file, there are two which again log only specific situations, corresponding to the above mentioned functions: Flooding, and Clones. The flood log is similar to the general log file: it records every message received by UWorld. But in difference to the former, the flood log must be turned on explicitly via a specific UWorld command (`floodlog`, level 8 command); in addition, the log is written to a distinct file. I assume that the log was turned on when a flood against UWorld was detected, and turned off afterwards.

A similar mechanism has been implemented for clones. This log records every time that a new user connection is considered a clone (multiple connection from one host). This log recording is bound to further conditions: It must be compiled into UWorld (via `define` directive, per default turned on), the log functionality must be turned on (`tsc` command), and UWorld must consider the new connection a clone (which can be tuned in various ways). In any case, this allows the UWorld admins to review which connection attempts UWorld has judged to be a clone; such information can be used to fine-tune the clone detection mechanism.

In sum, UWorld employs a number of logs to trace its general operation (general log) as well as specific commands or situations (`glines` and `kills`; `floods` and `clones`). Logs can either run all the time (general log, `glines`, `kills`) or turned on or off through the command interface (`floods`, `clones`). In the case of UWorld though, none of them is accessible through the command interface, but can only be accessed by the UWorld admins.

Summary – Checks and Balances for IRC Operators

IRC operators are the most important officials in the IRC, where they manage the day-to-day operations, help to maintain the server and the network, and to support the users. For this, they have access to some privileged commands. The social status of IRC operators in the IRC appears to be contentious, apparently due to the non-transparent way that they are nominated, and the discrepancy of how they should act, and how they actually do.

Previous chapters have explored the ways how an IRC administrator can check the power given to IRC operators. Here a 'softer' variant of control has been introduced: server notices, sent to other IRC operators and later also received by users made the actions issued by IRC operators more transparent in the network. Also, log files help the IRC admin trace back the actions by her own IRC operators, detect possible misuses or allow evaluation of complaints from users. The continuous growth in implemented notices and logs in the server versions indicate that the need for transparency in network events was growing, especially when considering the growth in usership, with which came the growth in servers, admins, and IRC operators.

The UWorld service bot of the Undernet, presented in the last section, combines many of the points made in this and the previous chapters into one single show case: A large step in differentiating the mechanisms available to maintain a social order, with both «code» remedial rules applied by officials as well as substantive «code» rules; a fine-grained access system, matched by both notices and logging facilities built-in to ensure some control of the UWorld users; and last not least, implemented as a centralized service bot, not in a distributed manner through server code changes.

From the «code» governance perspective, some issues are apparent:

TRANSPARENCY (1): The main rule examples of first chapters—*notices* and *logs*—are example for «code» *procedural rules*, as they give the 'controllers' of the IRC operators means to obtain information about their behavior. These kind of information are important in «code» based social settings, as there are otherwise few means to understand what is happening in the system, who initiated some actions, etc. The description of the status of IRC operators has give some impressions on the problem to control their behavior, and the apparent lack of success with normative guidelines etc. The «code»-based mechanisms which enhance the transparency can be a valuable mechanism in promoting a common understanding of norms, and may lead those in power (IRC admins) to apply sanctions in case of abuse.

CENTRALITY IN DISTRIBUTED SYSTEM: The simple fact that one central service bot can offer such a breadth of features just by acting different from the other servers shows the potential of such centralized services. Even more than the channel services reviewed above, the UWorld offers its users unprecedented power, such as allowing them to interfere into channel affairs, or offering network-global bans, both of which are not available even to IRC operators as highest ranking official in the IRC network. Also, UWorld has its own user access system independent of the IRC servers. In the version examined, only the service admin could give users access, again centralizing this power in the hand of those who run the service.

7 *Controlling the Controllers?*

TRANSPARENCY (2): As for the transparency of the UWorld, a mixed picture must be drawn.

On one hand, the notices and logs implemented follow the trend depicted for the normal servers: most actions, especially the powerful ones on the lower levels, are accompanied by notices and logs. But overall, due to lack of documentation and access to the source code, the power of those with high level access appears to have remained largely unknown. This might have been part of the reason to replace the UWorld service with another one, this time developed in an open sourced project.

RULE TYPES: Next to transparency as serving procedural means, the UWorld system replicates on the network level what was already pointed out with the registration services on the channel level: implementation of substantive rules (dealing with clones), and remedial rules (commands), giving Uworld users much finer grained power for example over channels than were available to IRCops. The Uworld access system as well is similar to those in channel services. A major difference though in the controller-selecting stems from the closed source development, combined with the centrality of service bots: The power accumulated here for the highest access levels is unprecedented, especially when considering the the lower level powers already encompass anything available in other networks. «Code»-based constitutive or even only procedural rules fail because there is no control instance of the controllers.

8 IRC Network Issues

In this chapter, two issues on the IRC network level are considered. The first section 8.1 claims that minor changes in topology and data distribution of a distributed system can have a large impact on the governance structure of the application. By comparing the IRC to the Domain Name System, I suggest that at least part of the power struggle of the latter derive from the chosen architecture, as does the apparent balance between IRC administrators.

The second section 8.2 and last one of the empirical part of this work gives a short account of the first documented disruption of a network, a major disagreement on a IRC-constitutional value which led to a split up into two networks. In this way, both networks were given the opportunity to evaluate the viability of their policy choice.

8.1 «Code» Architecture Shapes the Social Constitution

In this section, I will explain how the choice of the basic code architecture of the Internet Relay Chat defines a specific part of the the IRC constitution, the relationship between the IRC servers and thus between the IRC administrators.

Comparing the IRC architecture to that of the Domain Name System (DNS) I suggest that detail differences between these two – both chosen out of simple technical considerations – have large consequences for the relationship between the servers and its administrators: In the IRC, the architecture leads to a much more balanced power distribution than in the DNS, where the so-called "root" server(s) and those who manage them hold the most powerful position of all servers. In this way, the «code» architecture shapes the social constitution of the application in question.

8.1.1 IRC: Topology, Data Distribution and Technical Rationale

The technical rationale of the IRC architecture was determined by one main consideration: Created at a time (1988) where many links between the Internet were measured in *kilobits* per seconds, a real-time communication application had to cope with a limited bandwidth situation. Thus the prime concern was to minimize the bandwidth needed by the IRC. As

another factor, the creator Jarkko Oikarinen planned a system which should serve about 100 users¹ over these limited data connections (one has to compare this to the around fifty to sixty thousand users in the large IRC networks nowadays!).

Based on these considerations, Oikarinen chose the topology of a *spanning tree*, and a data distribution scheme where each server holds the the entire network status, constantly synchronized between the servers.

Topology: Spanning tree

The main characteristic of a spanning tree² is its lack of cycles between servers: there is no way between the nodes of such a tree which leads from one node through others back to that node. As consequence, there is always exactly one way between two arbitrary nodes in the tree (see figure 8.1).

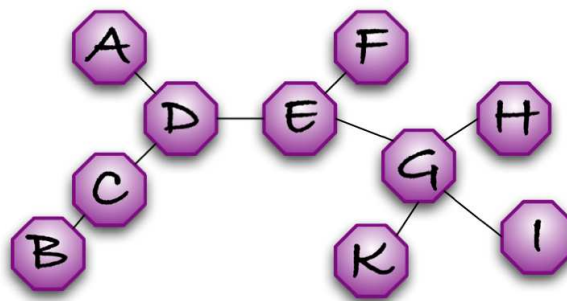


Figure 8.1: Spanning tree topology

This minimizes the computing cost for the servers (nodes) in the network: There is no need to determine for example a fastest or best route once the network is set up. Also such a tree topology allows for comparatively easy dynamic configuration, i.e. if one host ceases to work, then the remaining hosts can relatively easy form into a new tree.

Data distribution: Global state duplication and synchronization

The second choice by Oikarinen concerns the information about the network topology available to a server. Again in order to minimize the data traffic and thus the bandwidth requirements, Oikarinen chose a technique where each IRC host holds information about the whole network so that it can decide for each data packet to which other (directly connected) hosts to forward it to³. Thus only the data packets necessary are sent over the IRC network.

¹Undernet-User-Committee (1996)

²See for example Perlman (2000, p.531).

³This resembles the broadcast algorithm that Tanenbaum (1989, p.308) describes in connection with the spanning tree. He claims that this algorithm "makes excellent use of bandwidth, generating the absolute minimum number of packets necessary to do the job". But the disadvantages are the same, as well: "The only problem is that each [node] must have knowledge of some spanning tree for it to be applicable". In the case of IRC this knowledge is the state of the whole network which has to be constantly updated in each host.

If for example three users connected each to servers A, E, and B (figure 8.1) are in one channel, then it is not necessary to send the communication data between them to the servers beyond server E, that is servers F to K. In order to send the messages in such a way though, server E in the example has to know that there are no channel members in servers F to K. As soon as for example a user in server K joins the channel, server E has to relay the channel messages to server K. This way, every server has to have the up-to-date status information about the entire network.

IRC architecture: Advantages and disadvantages

The advantages of the choice of the IRC – spanning tree and up-to-date global status information in all servers – have been mentioned: a low computing cost in determining the path between servers, and minimizing bandwidth usage by sending the messages only to those servers who need them.

Another not so obvious advantage is a consequence of the data distribution model: Since every server holds the same data (network status), none of the server has an advantage over the others, and thus no single server or admin has a power advantage over others. If for example a server in the middle (for example server E in the figure above) decides to quit service, there is no major loss of service (other than a net split, see next paragraph). When server E quits, server F connects to D and G, creating a functioning network again. In other words, due to the data distribution scheme, there is a *non-hierarchical* relationship between the servers (and thus admins)⁴.

The major disadvantage of this architecture is a general sensitivity for so-called "net splits": As soon as a link between two servers or one server (especially those in the middle of the network) fails, then the IRC network splits into two parts and has to be reconnected. But also, these two parts become desynchronized: The status information in one half does not match the status information in the other half, so that upon reconnecting the two parts, all servers has to mutually synchronize their global network status information. With a few servers and tens or a few hundred users, this is no problem; but with fifty servers and ten thousand users, this becomes a major source of annoyance for the network. In other words, the IRC does not scale very well. It is only due to the continuous growth in computing power and Internet stability and bandwidth growth that the IRC could cope with this problem, but such net splits are apparently very common and a constant source of disruption of the communication.

⁴This is a principal statement which skates over details of hierarchical nature: the disruption of a leaf server (servers with only one connection to another server) affects much less users than that of a hub server (those with more than one connection). Also, hub servers with a high-bandwidth connection are more valuable as those with a low-bandwidth connection etc.

8.1.2 DNS: Topology, Data Distribution and Technical Rationale

There are several reasons why I have chosen to compare the IRC architecture to the technology behind the domain name system: The regulatory issues around the many aspects of the Domain Name System (trademark issues, global management of the name spaces) are the prime example of the Internet governance discussion, so that the term "Internet governance" has almost become synonymous with the DNS issue. But also, both IRC and DNS have been designed out of technical considerations alone, before their respective popularity led to governance challenges. Above all, both share the same network topology.

One of the main function of the domain name system is the assignment of a host name to the IP address of that host⁵. The rationale behind the technical domain name system as known today was the distribution of the management of this association. Instead of all host administrator sending name or assignment changes to a central place, and regularly downloading the updated file ('hosts.txt') back to the host, the DNS built up a server system where the data is distributed in a way so that it 'localizes the changes': Instead of managing them at one place, the institutions (universities, companies, Internet providers etc.) which decide on the names also enter them directly into their "name server". In this way, the whole DNS system is always up-to-date, because the changes are immediately available.

The second feature was the introduction of the hierarchical name scheme: The name consists of several parts, separated by dots. Each of these parts are managed by a different name server, with the rightmost invisible "root" of all names being managed by the "root" server, the next top level domain (TLD) by the top level domain servers, etc. down to the level of the institutions which name single hosts. Taken together, these structures determine the topology and data distribution of the DNS.

Topology and Data Distribution

The DNS server topology is a tree structure, and thus shares the same characteristic as the IRC topology (no cycles in the path, exactly one way between servers). Now if one has seen pictures of the DNS server topology, one might wonder why the above figure 8.1 of the IRC does not resemble a "tree", with one node on top, and the others growing in a triangle form down from the root (see figure 8.2).

The reason is that the graph of the IRC topology and this 'tree-like' graph are identical in their interconnection, but the servers just in different positions in the pictures. For example, server E in both examples connects to servers D, F, and G, etc. And the reason that the DNS network is depicted in such an hierarchical form lies in its data distribution model.

The data distribution derives directly from the said intention of the system: to allow a distributed editing of the name-IP address assignment in the context of the domain name scheme.

⁵See for example "Domain name system", Wikipedia 2005-04-25 (http://en.wikipedia.org/wiki/Domain_name_system).

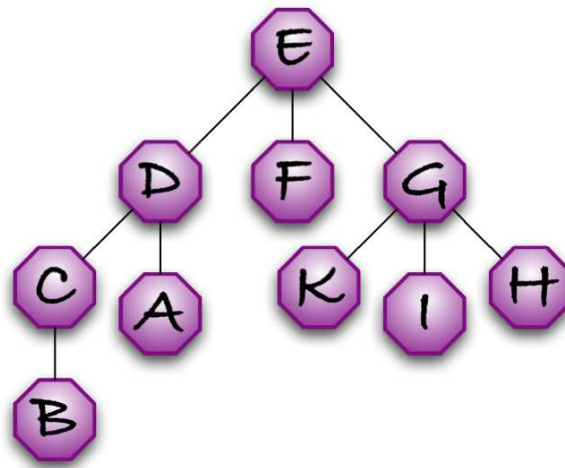


Figure 8.2: Tree topology as a "tree"

On each domain level, a server is assigned one or more unique names of this level: on the top level domain, each of the names "com", "org", "net" etc. are managed by one server. Similarly, the names of the next levels are also assigned to a server. Finally, on top of the server tree sits the "root" server.

The system now works as follows (see figure 8.3): To find out the IP address of a domain name, the user queries⁶ a so-called "name resolver" which is the DNS client of the user host; the name resolver now directs the query to the root name server⁷.

Now the data distribution scheme comes into play: The root server only holds the addresses of the top level domain servers. The only duty of the root name server is the redirection of the query to the appropriate top level domain server, so for a domain name ending with "de" (for Germany), the root server forwards the query to the name server which manages the "de"-domain. Similarly, the top level domain server only redirects the query to the next level server. This goes on until one server has a match for the entire domain name of the query: then the associated IP address is returned to the user.

The DNS data distribution scheme therefore differs considerably from that of the IRC: whereas in the latter, all data is duplicated in and synchronized between all servers, the actual name to address assignment data is only held by the lower level name servers. The upper level servers including the root server only hold the IP addresses of the DNS servers on the next lower level, which is a fairly small list.

⁶Indirectly, through the program that is used, such as a web browser etc.

⁷Normally, the name resolver queries first the local name server which in turn queries the root server.

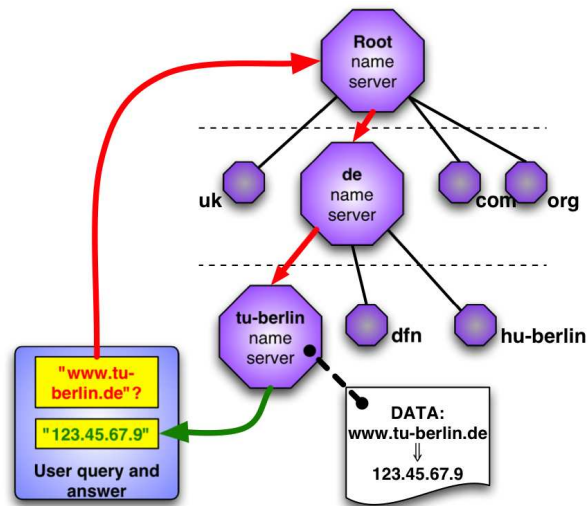


Figure 8.3: Querying a Domain Name

Advantages and disadvantages

The principal advantages and disadvantages immediately follow: The changes in the names and assignments are done locally, and do not need any propagation mechanisms to other servers, since every query directly comes down to the server which holds the name and assignment⁸. This also means that there are no synchronizing problems like in the IRC. Every server only holds the information necessary for its role in the DNS tree.

The main technical disadvantage is the necessity for queries to start at the top of the DNS server tree, i.e. with the root servers. This means that in principle every single query in the entire Internet has to pass through the root server, which leads to ten thousands of queries per second⁹; the system load of the lower level servers should be considerable as well. Consequently, many technical improvement strive to lessen the load, such as caching mechanisms, distribution of servers¹⁰, etc.

8.1.3 Comparison between the IRC and DNS architectures

The previous description of both the architecture of the IRC and DNS has been limited to the technical side. Both design choices have their clear rationales, and serve them well, despite the disadvantages which come with the choices.

Both also share the same topology, but differ in the distribution of the data: In the IRC, all servers have the same data (the network state), and in order for the IRC to function, this

⁸Here again a principal statement is made, which skips over details such as caching mechanisms.

⁹A RIPE report from 2003 speaks of two to three thousands of queries per second in two (of the thirteen) root servers: <http://www.centri.org/docs/2003/09/centr-ga19-ripe.html> (2005-04-25).

¹⁰"The root" is actually served by thirteen root servers, distributed over the world.

data has to be synchronized between the servers at all times. In contrast, the most frequented servers in the DNS network only hold a small part of the entire data (higher level domain servers which only hold the addresses of the next-lower level domain servers), while the name-to-address assignment data is held by the many lowest level servers. But in this system, the data of the higher level name servers is much more important, as they determine the way which the user queries take in the server tree. It is in the sole discretion of the root server (and thus those who control it) which server receives all queries for the respective top level domain; the change of one IP address would redirect all data for a whole top level domain to another server.

When one reviews the manifold articles about the political struggle about the control of the domain name system, it becomes apparent that there is one major point of power, the key to the entire domain name system:

”The Internet relies on an underlying centralized hierarchy built into the domain name system (DNS) to control the routing for the vast majority of Internet traffic. At its heart is a single data file, known as the root. Control of the root provides singular power in cyberspace.”¹¹

Ultimately, the combination of the tree topology coupled with the data distribution scheme of the domain name system ensured that those who control the root servers hold the ultimate key for the entire system.

The situation in the Internet Relay Chat is an entirely different one: Although employing a tree architecture as well, the distribution of all data between all servers ensures that in case of a failure of one server, all other server can recover by reconnecting the remaining servers to form a new network (with following resynchronization of its data). Therefore, the topology and data distribution create a network ”where each server acts as a central node for the rest of the network it sees”¹², so all servers can be considered root from their point of view.

8.1.4 Architecture as Constitution

I conclude that the technical architecture is an important factor for the governance of any Internet application like the IRC or the DNS. For the latter, it might be interesting (although, given the power struggle surrounding this issue, only of academic interest) to try out alternative topology and data distribution designs for the domain name problem which go beyond providing alternative or multiple roots¹³. In the case of the IRC, the equal distribution of powers through the topology and data distribution architecture, despite their technical shortcoming of poor scalability, has to this day served as a stable constitution of this Internet application.

¹¹Froomkin (2000, p.1)

¹²Kalt (2000a, p.3)

¹³See Froomkin (2000, p.39) (text surrounding footnote 61). The solutions he suggests though point all to institutional changes on top of the same technical system: *ibid.*, pp.171-182.

8.2 The "Great Split": The Forking of Anet and EFnet

The forking of a open source development project into two is widely considered as detrimental to the project as whole, because it splits the manpower available into two smaller fractions. In contrast, the history and the current situation of the IRC application and software highlights the positive side of forks, that is the creation of many different IRC networks with differing policies and service offerings for the user, and thus more code innovations which – through the open source property of the underlying code – can spread through the IRC networks. IRC networks have split up into different networks which are preserved through creating code-based boundaries. But these boundaries are permeable enough so that innovations can spread over these network boundaries.

In the beginning of the IRC, there was only one IRC network which was formed by the initial servers in Finland, those interested in Scandinavia, and the U.S. As of July 1990, 38 servers were counted (see above figure on page 45).

New servers were added by more or less informal coordination. The coordination was necessary to keep the network stable, that is keep the "distance"¹⁴ between the servers as small as necessary. A documentation file in version 2.1.1 states:

```
3) HOW DO I GET CONNECTED??
The official Irc-Network coordinators are: "vijay@lll-winken.llnl.gov"
and "karl@cheops.cis.ohio-state.edu" for the USA. FINISH and EUROPEAN
sites should contact "jto@tolsun.oulu.fi" and he can direct you to your
best connection.
Everyone that has mailed me and goes "where should i connect?" I generally
say what is your PING statistic to <some host>?". You will greatly help
us in the early stage by sending us your PING times to these hosts when
you first send us mail: [...]15
```

But this was rather a goal than reflecting reality, as a posting in the (then) main irc mailing list ("operlist") shows:

```
1) Our primary concern must be establishing proper routing. This can only
be accomplished be creating a WORLD-WIDE backbone of servers. This backbone
should be based on reliability, geographical location, network affiliation
(SURANET, ARPANET, etc), and IRC administation reliability/dedication.16
```

The mailing list contains several efforts to structure the IRC network, i.e. to coordinate the routing between the servers. But in general, there was no strict enforcement of any coordination, especially for servers not in the backbone part of the IRC network.

¹⁴This is not the geographical distance, but the time that a data packet travels between two hosts on the Internet. This is called the "ping time", because of the Unix program "ping" used.

¹⁵[irc2.1.1/doc/NETWORKING']

¹⁶Berlo, John J (1990-04-24) *Operators READ THIS!!!!*. Mailing list *IRClst* (1991).

8.2.1 Open-server servers vs. closed-server servers

Among the server administrators, there were two basic opinions about who should be generally allowed to the network: Open server vs. closed server.

Those favoring open server argued that any server should be allowed without regard to any formal process or criteria. They wanted to uphold the fundamental principle of openness. In order to serve this openness principle, the open server proponents also suggested to change the topology of the network from a tree to a star topology: Every server should connect to one central server.

In contrast, the closed server fraction argued with security and IRC net stability. If anyone was allowed as server, it would be too easy to gain the power of an IRC admin¹⁷, and thus at the same time IRC operator status, ultimately giving all users the same power in the IRC network.

As is often the case, there were two small groups of administrators, with a third big group of admins without a clear opinion towards either side. At first, this conflict was discussed over months in the diverse fora, IRC channels as well as the "irc1ist" mailing list. After the discussions prevailed over months in without reaching a consensus, the closed server proponents decided to exclude the open server proponents from their part of the network.

The open server proponents set up a central server at "eris.berkeley.edu", and allowed any other host to connect to it as server. This network was named appropriately as "Anarchynet" or "Anet". Those favoring the close server concept renamed their network to "EFnet" for "eris free network".

As usual in such heated debate, there was a majority of server administrator who were not sure where to go: Anet or EFnet. Some administrator tried to mediate between these nets by setting up a server which served both Anet and EFnet. While it did not clash with the basic principle of the Anet (on the contrary), it went against the close server concept of the EFnet. This therefore called for strong action, implemented as «code» rule, first as patch of the current server version, then as full blown mechanism in the following server version: the Q-line or "quarantine" line (algorithm 16).

Algorithm 16 "Quarantine" configuration line (Q-line)

```
# Quarantine lines. These lines disallow connections to the
# specified server and drops the link to anyone connecting to them.
Q::they have a server open server:eris.berkeley.edu
```

Source: [irc2.5.1.bu.08/doc/example.conf:109-111].

This mechanism allows the server administrator to set a server to quarantine, that is to disallow any connection or data transfer between the IRC network and this server, as well as

¹⁷It is important to note that the dispute went around open *server* servers, and not open *client* servers. With an server in the IRC network, one is automatically an administrator, while with a client connected to a server, one is 'only' an IRC user.

servers connecting to the net via the quarantined server. Basically it ostracizes the server and all servers connected to it from the network. The interesting feature of this Q-line mechanism is that it only works if all servers invoke the quarantine.

It is interesting to see that the Q-line apparently was only used in that incident. Later server versions still incorporate the Q-line mechanism, but its use is explicitly deprecated. The threat of reactivation of this mechanism is still prevalent.

The Q-line mechanism allows for exclusion, ostracism from the server community, but with the high hurdle of unanimous action. Since the network would break if not everyone invokes the quarantine, this is a strong hurdle to its use. Here, code is effectively used as a supporting mechanism for community binding and decisions.

This event might be an example for Lessig calls the "open evolution" principle¹⁸: After it became clear the the open/closed server conflict could not be resolved through debate, this conflict was "solved" by proof of concept: each side set up the network they wanted, and let the undecided server admins as well as the users decide. In this case, the EFnet survived and thrives today, while the Anet left no traces other than the conflict.

¹⁸Lessig (1999c); in an earlier draft dated March 5, 1999 ("Draft 2"), Lessig used the term "open forking" which in our context fits better than the more generic "evolution".

Part III

Some Notes on the Concept of «Code» Governance

As no better man advances to take this matter in hand, I hereupon offer my own poor endeavors. I promise nothing complete; because any human thing supposed to be complete, must for that very reason infallibly be faulty.

Herman Melville, *Moby Dick*

Introduction

This part concludes my exploration of the «code» governance aspects of the Internet Relay Chat as a self-organized, self-governed Internet application. I revisit the main hypothesis of this work that the analysis of the source code of the IRC reveals distinct «code» features, and that the «code» constitutes a regulation system similar to, but distinct from the legal system. As this work is exploratory in character, no ready-made theory on «code» governance can be offered. As preliminary results, the following findings can be offered: In chapter 9, I reiterate each key aspects of a regulation system according to the "Lex Informatica" model introduced above¹⁹, summarizing my findings under the appropriate aspects. Chapter 10 summarizes the findings concerning the five types of rules as introduced above²⁰. Finally, chapter 11 offers an outlook of further research that could built upon the «code» governance model developed in this work.

¹⁹Chapter 2.3.1

²⁰Chapter 2.3.2

9 Lex Informatica Revisited

In chapter 2.3.1 I have introduced the concept of "Lex Informatica" (Reidenberg (1998)) which offers the key concepts of a regulation system, and compares the legal system and the «code» regulation system, or "Lex Informatica") based on these concepts.

In this section, I apply the results of my empirical analysis to each of these key concepts, thereby developing the "Lex Informatica" side one step further.

	Legal Regulation	Lex Informatica
Framework	Law	Architecture
Jurisdiction	Physical Territory	Network
Content	Statutory/Court Expression	Technical Capabilities Customary Practice
Source	State	Technologists
Customized Rules	Contract	Configuration
Customization Process	Low Cost Moderate cost standard form High cost negotiation	Off-the-shelf configuration Installable configuration user choice
Primary Enforcement	Court	Automated, Self-execution

Lex Informatica (same as table 2.1)

9.1 Framework

Reidenberg describes the framework as the "basic building block" of the respective regulation system. In the legal system, "law" functions as the basic building block, and for Lex Informatica he offers "architectural standards" which "define the basic structure and defaults of information flows on a communications network", with the HTTP standard as one example of such a building block¹.

What defines the basic structure and defaults in the Internet Relay Chat? Beginning with Reidenberg's suggestion it is clear that the communication between the components of the IRC (servers, clients, etc.) have to standardize their communication.

One set of standards, also available for the IRC, are those submitted to an institutionalized standardization body: Two times, 1993 and 2000, such IRC standards have been submitted to the main Internet standardization body, the Internet Engineering Task Force (IETF). In 1993, the IRC client-server protocol is published², and in 2000, four documents describe the architecture, channel management, client(-server) protocol and server(-server) protocol of the IRC³. These standards certainly provide an overview of the "framework" of the IRC.

¹Reidenberg (1998, p.570)

²Oikarinen and Reed (1993)

³Kalt (2000a,b,c,d)

But these documents only provide a snap shot of the IRC setting. As described, the IRC server code has been continuously changed and updated, with many major and minor changes in client-server and server-server protocols, in implementation and functionality. These changes can be seen as a hallmark of a self-organized and «code»-governed setting like the IRC.

An indication of such adaptations of the client-server protocol has been shown for the Eggdrop user bot⁴, where one configuration setting allowed to specify the network to which the bot connected⁵, thereby enabling the special feature of that respective network.

The point made here is that, while standards and protocols *as published* certainly are an important part of the «code» regulation framework, often they do not draw the entire picture of the system. The ultimate reference for the architecture can only be provided by the actual code of the system, in case of the IRC the server source code.



In «code» governance, the framework or the basic building block is the «code», the software and hardware of a system. Standards are important, but provide only an approximated view of the real system.⁶

9.2 Jurisdiction

Jurisdiction denotes the scope of the regulation regime inside which the rules are valid and enforced. In the legal system, the laws are principally defined by a territorial jurisdiction. For Lex Informatica, Reidenberg provides a somehow vague "network" or "network spheres":

"[T]he jurisdiction of Lex Informatica is the network itself because the default rules apply to information flows in network spheres rather than physical places"⁷

Based on the analysis of the Internet Relay Chat, some clarifications can be made.

IRC border (1): Internet Layers

The choice of the IRC as *Internet application* already includes a «code»-jurisdictional limitation: The IRC is positioned in the *application layer* of the Internet⁸.

⁴See above chapter 6.2.1.

⁵See algorithm 14

⁶As an interesting thought that may be worth pursuing is the difference here between law and «code»: In law, the statutory code itself (or a single case) seldom provides a full picture of the legal situation at hand. Instead, only a thorough document analysis including commentaries, other cases and other secondary sources can provide a sufficient picture. In comparison, the ultimate source for the understanding of «code» is the «code» itself, the source code of the software, or the technical capabilities of the hardware.

⁷Reidenberg (1998, p.570)

⁸Here I only examine the technical ISO-OSI model. There are also those which are derived from the technical models in legal scholarly papers, such as a three-layered model in Benkler (2000, p.562) and following Lessig (2001, pp.23-5), but they add no further insight to the matter at hand.

Layering is a commonly used model with networks which helps to reduce the complexity of its design. The best known⁹ one is the Open Systems Interconnection (OSI) model¹⁰ which distinguishes seven layers, from the lowest "physical layer" to the top "application layer".

As an application, the Internet Relay Chat is positioned in the top *application layer*. As consequence, any issues of the lower layers are outside the IRC "jurisdiction", out of its governance reach. On the other side, any changes in these lower layers may affect the IRC, as it uses the services that the lower layers provide.

IRC border (2): Application

Inside the application layer, a second line can be drawn between different application¹¹, although these lines can get blurry sometimes.

Generally, applications such as the world wide web, e-mail, peer-to-peer networks each create their own jurisdictions. Their respective «code» rules are determined by the application code and the standards and protocols which the code implements. Therefore the IRC with its server code and the IRC client-server protocol, can for example be distinguished from the world wide web, with its servers and the http protocol.

But for a specific functionality¹², the border between these two can be crossed: one can imagine for example that an IRC admin provides a web interface¹³ to her IRC network¹⁴. Also, the example of the CTCP functionality¹⁵ inside the IRC shows another blurred line: The messages are exchanged over the IRC lines, but the actual meaning of the messages lie outside its scope. This allows for example to *establish* a direct Internet connection via the IRC; the connection itself then is directly between two IRC clients, thereby circumventing the network for the conversation.

These remark point to the question of how the term "jurisdiction" can be applied, rather than giving final answers. But the lines between application nevertheless play an important role in the question of the «code» jurisdiction.

IRC network borders

Finally, "the" IRC consists of many IRC network which are not interconnected with each other. Each of these forms its own jurisdiction. An indication of a "strong" border is when the IRC server software between different networks do not interoperate. This might be only

⁹ Another early example, written for the predecessor of the Internet, is the ARPANET reference model: Padlipsky (1982, pp.12-3).

¹⁰ See for example Tanenbaum (1989, pp.19, 528-530), Perlman (2000, p.4).

¹¹ A similar argument makes Wu (1999).

¹² It could be interesting if the concept of "functional, overlapping, competing jurisdictions" (Frey and Eichenberger (2000, pp.4-5)) could be applied to the «code» domain.

¹³ This is not to be confused with so-called "web chats", which are single-server chats which can only be used through the web page interface, and not reached by an IRC client.

¹⁴ And indeed, a google search for "irc web interface" shows a number of entries.

¹⁵ See above chapter 3.1.1.3

partially intentional, because the implementation of different network policies and feature sets dictates changes in the server-server protocol, so that lack of interoperability often comes as a by-product of the functional differentiation between the networks.

The "jurisdiction" of the IRC is determined by a number of factors:

- a) Network: *Its position in the application layer according to the network layer model*
 - b) Inter-application: *Its position as an specific application, separated by functionality, protocols and code, although these lines can be blurred*
 - c) Intra-application: *Inside the IRC, each IRC network forms its own jurisdiction, sometimes underlined by the non-interoperability of the server software between networks.*
-



9.3 Content

The content of rules in a legal regime sees Reidenberg as deriving from "statutory language, government interpretation, and court decisions"¹⁶. For Lex Informatica, he sees the content "defined through technical capabilities and customary practices"¹⁷. As example he describes how the SMTP protocol for e-mail defines the rule that the "From" field identifies the sender, with the "customary practice" of mail servers that this field "pertains to the actual person" of the e-mail.

The examination of the IRC has shown how the technical capabilities of the IRC server code builds the foundation of the content of the IRC «code» rules. They determine through their server-client protocol which features are available to the users. The server code also determines all IRC internal structures, like channels, the powers of IRC operators, etc.

A notable exception from this rule are the service bots¹⁸ which as a centralized service in a distributed server network provide additional capabilities not available through the server code. Also important are the user bots¹⁹ which use the client-server protocol in order to offer features not available through the servers, for example changing the channel ownership policies set by the server. In these ways, the main «code» regulatory content in the IRC is determined in the server code, but due to its distributed nature, with the server-server and client-server interfaces present, surprising and unplanned uses are possible, changing the setting which can lead to adaptation in the policies (e.g. by providing official services, like the channel registration services).

The customary practices are guided, as the name implies, by customs or norms. Indications

¹⁶Reidenberg (1998, p.570)

¹⁷ibid.

¹⁸See chapter 6.3 (channel services) and chapter 7.4 (Uworld service)

¹⁹Chapter 6.2

for such norms were in the discussion of the IRC operators' behavior²⁰, but also in discouraging the use of features without removing them from the code²¹. One could also interpret the unsuspected uses of existing features, such as user bots, as "customary practices", a use not intended when designed and implemented, but later used in the new, specific way. These practices form an important element of the dynamics of «code» governance settings.

9.4 Source

The source of regulation in a legal system is given by Reidenberg with "the state" in which a "political-governance process ordinarily establishes the substantive law of the land". For Lex Informatica, Reidenberg suggests both "technologists", the "technology developer and the social process by which customary uses evolve" and who create the "technical standards", and the user who "adopts precise interpretations through practices".²²

With my «code» analysis in mind, we can sketch a slightly more concise picture of the sources of regulation in the IRC.

The first IRC software, and thus the first set of «code» rules were designed by the creator of IRC, Jarkko Oikarinen, adopting from other similar applications not only technical structures, but also governance patterns such as the role of the IRC operator²³. Many of the basic elements that define the IRC remained unchanged from this first version on, such as the topology and data distribution, the IRC operator role, channels, etc. Oikarinen thus is not only the (technical) creator of the IRC, but, from the «code» governance perspective, the "founding father" of the basic "constitution" of the IRC.

The pivotal decision by Oikarinen which determined the "source" of future «code» rules in the software was to distribute the IRC *as source code package*, and subsequently to put the software under an open source licence²⁴. This created two further sources of changes, next to Oikarinen: Other coders who contributed to the source code, and by the provisions of the GPL, made them accessible as source code as well; and the IRC administrators, who have the power to change any aspect of the IRC server, limited only by the interoperability with the other servers in the network.

The availability of the source code also has created a kind of power balance inside the IRC: while changes in the server code *for all servers* is made by the coders, the adoption requires the agreement of the IRC admins. This explains why changes in the source code were implemented so often as configurable options through mechanisms such as patches, #define

²⁰Chapter 7.2

²¹See the R-lines in chapter 5.2.2

²²All quotes from Reidenberg (1998, p.571)

²³See above chapter 7.1

²⁴See appendix, chapter 12.7.

directives, etc.

Being open sourced, coders cannot force IRC admins to adopt these changes, as is common in closed source scenarios, where lack of interoperability or backward compatibility can force those who run the software to upgrade to a newer version, even though it might be not in their interest. On the other hand, IRC admins have much more freedom to deviate from net-wide policies, only as long as the interoperability is ensured. Enlarging the source of rules brings up a different set of governance opportunities and challenges. The kind of openness in the distribution of the software here is the key aspect.

Another source of regulation is the *IRC user*. One point not further examined in this work, but quite probable, is that simple users have made important contributions to the IRC software development process. This is a basic characteristics of any open source development. Lessig (1999c, p.113-115) has named it "universal standing", giving it the status of a «code» governance constitutional value.

The IRC user also have been the source of regulation with another «code» mechanism: the IRC bots. As shown in the example of channel and nickname ownership issues (chapter 6.2), these bots have been employed to change the policy set by the IRC officials (no ownership of channels and nicknames), and may well have been a driving force to implement 'official' ownership policies, such as the DALnet or Undernet services.

In the legal scholarship, such "user regulation" is often subsumed under the label of "digital self-help"²⁵, and in software engineering the users are also recognized to play a more and more active role in the design and implementation processes²⁶. The IRC underlines the importance of the ability of users to contribute to the «code» governance, facilitated here both by the open source code as well as the open client-server interface.



Open sourcing the server code has widened the "source" for code, and thus for «code» governance mechanisms, leading to constant contributions from the usership at large.

Another important source of «code» governance in the IRC came through the open client-server interface which led to the development of the IRC user bots.

9.5 Customized Rules and Customization Processes

This is an important point and a wide field for «code» governance settings. Most hard- and software provide some kind of interfaces through which the users access the functionality offered by the technology, and adapt it towards their needs.

For law, Reidenberg offers the legal institution of "contract" as customized rules: The ability

²⁵See for example Bell (2000).

²⁶See for example IEEE (2004, ch.2.2).

of private parties to customize their interactions through the process of reaching a contractual agreement. This inherently also contain another connotation that appears detrimental when applying this model to the «code»: the notion of the legal system as hierarchical default rule system, with the contractual relationship a "deviation" from the legal default:

"In the legal regulatory regime, private contractual arrangements can be used both to deviate from the law's default rules and to customize the relationship between the parties. Such deviations are only available if the law permits freedom of contract and does not preclude the participants' actions"(Reidenberg, 1998, p.571)

In contrast, The IRC is a self-governed setting, so neither a hierarchical 3rd party controller²⁷ as the government exist, nor is it governed by any overarching organizational entity, such as university, firm, or consortium²⁸. Consequently, I regard the customization rules and processes on both the level of code development and implementation, and on the level of the use of the IRC software.

9.5.1 Rule customization on the «code» level

The examination of the IRC source code has shown that the coder have applied various mechanisms in order to facilitate governance decisions for those who install the server software. I have tried to develop a first step towards a taxonomy of these mechanisms by subsuming them under the four steps where customization of the software takes place: Source code access, System access, Interface access, and User access.

9.5.1.1 Source code access

The IRC server software is distributed as source code package, an archive file containing all necessary files in order to build the server program.

There are several reasons for this kind of distribution method. These include:

- *Distribution file size*: This distribution is the most efficient one: a compressed archive file containing source code can be many times smaller than a (compressed) compiled binary program files, and thus takes up less bandwidth. Also, small changes can be sent as patches (see below), even in e-mails, whereas binary program file patches tend to be larger. Moreover, it may be easier to mirror a smaller source code archive file than a larger binary file.

²⁷See Ellickson (1991, pp.130-132).

²⁸See Kesan and Shah (2003b). While the IRC would fall under their category "open source movement", it differs in that the IRC developers and users are much more tighter coupled than for example open source operating systems, or other application software.

- *Platform adaptation*: It might be necessary to make small changes to adapt the software to the specific configuration of the computer where the software is to be run. The source code can cope at build time with these differences (for example with the auto-configure process; see below), or the installing person can make these changes by hand. With binaries, a distinct file per specific computer configuration has to be built and made accessible, or the program has to cope with it through programmed configuration routines.
- *Licensing*: The main non-technical reason for many open sourced programs is the license. Like many others, the IRC server code is distributed under the GNU Public License (GPL) which necessitates that the distributor gives easy access to the source code, even if the software is distributed as binary program file.
- *Governance*: Another important reason – conforming with the main hypothesis of this work – is to give the installer²⁹ options to change the working of the software, including its governance characteristics.

Here I distinguish several means how the software can be changed: Choice of versions, source code patches, source code configuration, and direct source code changes.

Choice of Versions

This is probably the easiest way to choose between sets of «code» rules. In the IRC, admins have the choice to upgrade or not to upgrade to a new version, and in some cases to choose between different series of software versions. The latter variant can be seen for example in the EFnet, where at times the comstud (ircd+CSr), TH, and Hybrid series were available for the admins to choose from. Also, many different IRC client programs are available to the users.

In the IRC, the choice of versions is an important instrument to give the coders the incentive to create admin-acceptable software. Mailing list messages suggest that admins have been quite conservative with regard to version changes: once a server runs stable, there has to be a good reason to replace it with an potentially instable one. In IRC history, there might have been cases where the version changes have been enforced by making the new version incompatible to the earlier version. Such severe limitation in choice would certainly have been preceded by an extensive debate by all admins and coders.

The choice of versions or concurrent series of software is one means to choose between different «code» rule sets. This choice of versions is not limited to source code versions of «code», but generally available with any product, be it (closed or open source) software or hardware.

²⁹I will refer to the person who installs a software as "installer"; in the case of programs which will install software, these will be referred to as "installer programs".

Source Code Patches

The previous choice of version is a very coarse one: One has to choose a «code» rule set over others, without the possibility to selectively choose specific rules or features. With patches, coders can pack one or several features into one or more patch files which the installer can choose to apply. This is a convenient and often used way not only to distribute features changes, but also bug fixes. The patch files tend to be very small, so that they are sometimes even distributed through mail messages.

Source code patches consist of sets of code lines, where additional embedded data specifies which code lines are to be replaced with the new lines in a file. A special patch program (in Unix systems appropriately named "patch") finds the old code lines, and upon a match, replaces it with the new ones. Thus, an automatic source code change is applied.

In the IRC (and undoubtedly in other applications as well), these patches are also used to give the admin choices over «code» rule functionality.

This patching mechanism also exists for binary program files. The advantage of source code patches compared to binary patches lie in their transparency: Since the patch is simply a text file, it is easy to understand what the patch is doing. In comparison it is usually not possible to understand what a binary patch will change; even reverse engineering does not help, because the contents of the patch is taken out of context, which is even more necessary for understanding the binary code than it is in source code.

Source Code Configuration

Once a version is chosen, and the appropriate source code patches applied, the software needs to be configured. This includes the adaptation to the hardware and software of the computer where the program is to be run, the network environment, as well as other information needed to run the program.

The IRC server configuration comes in several flavors, depending on what the coders chose to use. In the most simple case, it consists of editing one or more files, adding or changing values at certain places according to the instruction given by the coders which in the IRC server distribution is explained in the file "INSTALL". In the case of the IRC server software, this is usually the "include/config.h" file.

A more sophisticated means are configuration scripts. These are small script programs which the admin runs; the script now requests all necessary information from the admin, and makes the necessary adaptations to the source code. Afterwards, the software is ready to be built into the binary program. These scripts can either be written by the coders themselves, or use a prefabricated configuration tool, such as the GNU `autoconf`³⁰. In `irc2.8.21`, for example, the script "Config" has been written by one IRC coder (Darren Reed); recent versions in all major networks have applied the GNU `autoconf`.

³⁰See <http://www.gnu.org/software/autoconf/>.

However the configuration process is done, it is an important step for the admin to decide upon the «code» rule regime that her server employs. In addition to the choice of the server version or series, and the choice of patches, the configuration of the source code allows for a fine-tuning of the various aspects of the code.

It is important to note that all these choices are facilitated by the coders. They have to provide these configuration option for the installer, and has to make sure that the «code» does work properly for all possible configuration settings.

Direct Source Code Changes

The ultimate power that an installer has with regard to the program is to change the source code itself. For stand-alone application, this power is limited only by the programming ability of the installer. In networked applications like the IRC though, the changes must be interoperable with the other servers in the network.

9.5.1.2 System access

In difference to source code access «code» rule patterns, this and the following types presuppose a built binary program. In the case of open source programs, the installer has compiled and linked the software. In the case of closed source software, this is the state in which the user acquires the program.

The "system access" type differs from the next one ("interface access", below section 9.5.1.3) in that it mainly applies to the software before it is started.

The following categories can be subsumed under this type: versioning, patches, and binary program configuration.

Versioning

Similar to its source code equivalent described above, coders can provide several versions of the same software with different properties. In this case, the user can choose between them and thereby chooses different «code» rule regimes.

Patches

Again similar to its source code equivalent. Binary patches change the program and thus the implemented «code» rule regime. But in difference to source code patches, it is difficult to see what changes exactly are applied with the patch. A reverse engineering of the patch itself will not get much information, if any. Only reverse engineering the patched binary could shed some light on the changes. One has to rely on information accompanying the patches.

Configuration

In difference to source code configurations which affect the shape of the binary file, this kind of configuration affects the program in its running state.

Configuration of a binary file come in two flavors: as a configuration file, or as startup options. In the IRC, both methods are used. With the configuration file, in IRC called "ircd.conf", the main configurations of the IRC server are made in form of the configuration lines. Here, a number of policies are determined, for example which groups of users may enter the IRC through this server, which other servers may connect to it, what K-lines are applied, etc. For an IRC admin, the main choices are made in this file.

With the IRC server it is possible to reread this configuration file through the IRC operator `/reread` command or through the restart of the server (command `/restart`). Accordingly, when the server is running, the admin can make changes to the configuration file, and activate these changes through a reread.

This configuration file resembles the preferences file often seen in software which normally is manipulated through an in-program preference pane or similar user interface. The difference becomes obvious in the IRC: the admin is different from the user or the IRC operator. The above examined concept of the `/kline` command for example would not be possible if IRCops or users generally had access to the configuration file. Therefore this allows for a access separation between different roles in the setting.

The other mechanism for configuring a program is to provide startup options. This method is applied when a program is started from the command line inside a terminal program. Normally, the options available here are a subset of the configuration file options. If a program is started with these options, they will override the configuration file options.

9.5.1.3 Interface access

Software (as well as hardware) can provide interfaces as either necessary link to other components of a system, or through which the functionality is expanded. Often functionality-expanding interfaces are referred to as "plug-ins". In the case of IRC, there are two main interfaces in the server, one to other servers in order to form the network (specified in the server-server protocol), and the other to clients through which users connect to and use the IRC (specified in the client-server protocol).

Another kind of interface in the IRC is the R-line feature in irc2.5.1.bu.09 (Nov 1990)³¹: the acceptance of a new IRC user is made dependent on an external program to be provided by the IRC admin. Into this program the admin can make any check she wants; the result (yes or no) will be sent back to the IRC server, which then accordingly decides upon acceptance of the user. Here the IRC server provides an interface to the external program, with a small protocol (reply string: "Y" or "N").

With "R-lines", this interface can be interpreted as subsidiarity rule: The coder transmits the right to form arbitrary criteria for new IRC members' acceptance rules. Without this interface,

³¹See also above chapter 5.1.2.

admins could only use the other mechanisms (I-line, K-line) which are based on checking the user/host resp. user/IP address pair.

System interfaces code rule changes types are a very powerful instrument in the relationship between coders and users. As soon as the interface specification (protocols, application programming interfaces) are made available, the software can be expanded in unforeseen ways. See for example the case of bots (chapter 6.2) which give IRC users quite some power, just by automating the client-server interface of the IRC.

This code rule change type also rises in importance with the tendency to modularize software into components, making interfaces for communication and interaction between these components an integral part of a system. The more and complex the interfaces, the larger the opportunities to create components which use these interfaces in unforeseen ways.

9.5.1.4 User access

On the level of the usage, the available commands determine the available actions. But as the examination of the IRC has shown, there are different ways to implement a certain functionality.

The candidates for these code rule types are: hard coded, conditionally limited, functionally limited, preset choices, settable.

Hard coded rules

This is certainly the strongest constraint that coders can implement: These rules are fixed in the code, and there is no means for the users to change its behavior.

Examples include the numeric channel situation, with the channel visibility property³² hard coded with specific channel number ranges, as well as the "maximum number of users" fixed property³³.

This «code» rule pattern also could be tagged "unconditionally activated", because there is no mechanism which allows someone to choose over the activation of the rule. Instead, as soon as the appropriate situation arises, the code automatically executed its rule.

Conditionally activated rules

In difference to the hard coded rules, this type of rules is activated according to conditions which can be changed by some principals.

Examples include K-lines and ban lists. An IRC official adds K-lines into the server configuration file, and subsequently these users cannot enter the IRC. A similar feature, the ban lists, allow channel operators to set entry denials for channels.

The rule conditions can be coarse as in the case of early K-lines (only banned or not banned

³²See chapter 4.3.1

³³See above chapter 4.2.3

as condition), or quite sophisticated (time-limited bans in later K-lines; R-line, D-line, E-line as further instruments), depending on what the coders have implemented.

Functionally limited rules

In difference to aforementioned types, this type points to the action part of the rule. I am not sure if this can be counted as a code rule type; but this type occurs quite often in a «code» setting. In conditionally activated code rules, especially with commands, coders often limit the action part functionally by allowing only a range of actions.

An example is the `/kline` command. Instead of giving the IRCop full access to the K-lines in the server configuration file, only additions to it are allowed through the command. Full access here is reserved to the IRC admin who has file system access to that file.

Preset Choices

Another technique next to functional limits is to give a preset list of choices from which the user can choose a setting. These choices can either be exhaustive, or limited. The visibility property and numeric channels are one example: users had to choose from the number range of channels in order to get the channel with the desired property (public, secret, or hidden channel). There was no means to change the property once the channel was chosen.

Another kind of preset choices would be one setting, from which these choices are available. The visibility property in named channels is such a choice: the channel operator can choose from three settings.

9.6 Primary Enforcement

Reidenberg compares the legal system's *ex post* enforcement to the *ex ante* enforcement in Lex Informatica, where rules implemented in the hard- and software are automated and self-executed.

Auto-enforcement is not an *inherent* property of «code»; coders either implement an *ex ante* enforcing rule in code, or choose to implement the equivalent of an *ex post* enforcement: The breach of a substantive rule leads only to a warning, or collection of evidence which then can be used to decide over appropriate sanctions.

This is a topic which is detailed in the following chapter 10 on rule types in «code».

Conclusion: «Code» as Regulation System

The "Lex Informatica" model has served me to show that key aspects of regulation system, as they are well-known in the legal system, are also present in the «code», the software as understood as regulation modality: The *framework* consists of the actual code. The *jurisdiction*

is determined by its position in the network layer model, as well as (in the case of the IRC) its status as an application; in addition, each IRC network forms its own jurisdiction inside the IRC. The content, the actual *rules*, are also formed by the code, with norms guiding the practices.

The *source* of the rules were broadened by the decision to distribute the code as open source software, so that all participants of the IRC – from IRC admin to the simple user, in addition to the coders – could contribute to the code. In addition, the open interfaces have allowed for further «code» rules. Finally, *customized rules* and *primary enforcement* are important topics and thus detailed in the next chapter.

Judged from this discussion of key aspects, it is safe to claim that the notion of «code» as a regulation system is indeed a useful one, as it provides a framework for the explanation of the structures and developments as they have occurred in the IRC.

10 Rule Types in «Code»

In the summaries of the chapters in the main part of this work, I already have reviewed the rules types as introduced in chapter 2.3.2. Here I will therefore try to make some conclusions based on these summaries.

I have not found many *substantive rules* which by itself determine rewardable or punishable primary conduct. This may be due to the categories of social control of "reward" and "punishment". «Code» allows for the implementation of substantive rules which are not seen as such, but instead as "objective" constraints¹. Those structures that I have labeled as "constitutional" structure or design might be interpreted as such "objective" rules, such as the inability to name channels (in the numbered channel design²).

But another interpretation is possible: The "maximum users per channel" case has shown how a fixed constraint, the substantive rule of limiting membership in channels to ten individuals, has been transformed into a per-channel configuration setting, to be decided by a channel operator. Coupled with the ability or any user to create new channel, and the abundance of channels (but not names), the need to fix a substantive rule in «code» was not given. The technical abundance might reduce the need for fixed substantive rules in «code», instead offering configurable environments with a set of controller-selecting and constitutive rules in place.

The development of *controller-selecting rules* is especially obvious: In the beginning formed as three-level hierarchy—IRC admin, IRC operator, and user—, the IRC successively expanded this hierarchy: channel operator and local operator; channel manager or founder with the registration services, and their respective channel official hierarchy systems; and the UWorld with its access system.

This was accompanied by *constitutive rules*, to regulate the relationship between the controllers, and check their powers. On the channel level, the "voice" mode is an example, where before one had to give chanop status to all speakers in a moderated channel and therefore unchecked power over the entire channel; the voice mode now allowed to give only a 'voice' when the channel was moderated³.

Another interesting example is the file interface, coupled with the limited `/kline` command, which gave of IRC operators the ability to add, but neither change nor remove (nor

¹See Lemley (1998, p.677).

²See above chapter 4.2

³See above chapter 4.3.2.

read) the K-lines in the configuration file⁴. And the logging facilities allowed the IRC admin to control the use of the command⁵.

Logging, together with notices create some transparency and therefore are also *procedural rules*. Here again is a tendency towards expanding the amount of information available⁶ as well as widening the audience, letting users receive notices formerly available only for IRC operators.

On a more constitutional level, the ability for anyone to obtain the source code and study might be also categorized as *procedural rule*: The example of the UWorld as closed source with its commands on the highest access level (nuking, and the mycmd command effectively circumventing any built-in control mechanisms⁷) might be seen as a breach against the procedural rule of the principal transparency of the technical components of the application. If so, then the replacement of Uworld by a open source counterpart may be seen as remedy of this breach.

Finally, the majority of commands presented in the chapters are *remedial rules*: `/kick` and bans on the channel level, their counterparts on server and network level (`/kill`, K-lines etc.) and the commands made available by the service bots⁸: they all serve to apply sanctions against other users. And again we can observe here a successive functional differentiation, where more and more fine-grained tools are made available, in different contexts (network, server, channel) as well as on different levels of the user hierarchy.

⁴See above chapter 5.3.

⁵See above chapter 7.3.2

⁶For example the number of notices sent by the IRC server; see above chapter 7.3.1.

⁷Above chapter 7.4.2.1,

⁸Above chapter 6.3.

11 Outlook

In the previous chapters I have shown how in the Internet Relay Chat (IRC), the participants have employed «code», the underlying technology of the application, as main means to govern themselves. Specifically, I have interpreted the IRC source code *as a regulation system* by identifying code structures as "rules" which shape the social situation, and changes in the source code as adaptation of the governance situation to changing social contexts. In this sense, the technology itself can be seen as regulation system, similar to but distinct from others, such as the legal system; I have employed the term «code» governance for this theoretical concept.

The goal of this work has been to explore the possible viability of this concept, for which I have chosen the IRC as a self-governed Internet application. While this goal has been generally reached, further research is necessary to explore the viability of this model for other socio-technical settings, as well as to work on its methodological approaches. The next section 11.1 outlines some possible areas which appears to be suited to expand the «code» governance concept.

Once this concept has shown its usefulness, I can see two immediate implications for the academic study of socio-technical settings (section 11.2). For computer science itself, the «code» governance model strongly suggests to systematically incorporate social structures and dynamics into the design process, instead of relying on the requirement analysis and software maintenance processes to cope with these issues. Additionally, my model could prove instrumental in exchange research results in techno-social studies between computer science and other concerned disciplines, giving the former a structured way to incorporate economic, legal, and social science concepts into the technical domain, and at the same time deepen the understanding of technology in law, economics and social and political science.

11.1 Validating and Refining the «Code» Governance model

In this work I have shown that there are indeed structures and changes in the IRC source code which can be linked to the social situation and dynamics of the IRC participants. From this I have concluded that, in the case of the IRC, «code» governance serves as useful model

to explain these code structures and changes as "regulation system", to treat the «code» as "rules", and its creators as "regulators".

I have shown the usefulness of the «code» governance model here for exactly one example, the IRC, which also has been carefully chosen for this exploration, exhibiting these properties:

«CODE» AND NORMS ONLY: The IRC excludes influences from the "law" and "market" regulation modalities, thus leaving «code» and social norms as prevalent modalities.

OPEN SOURCE SOFTWARE: The underlying software of the IRC is open sourced, so that it is in principle available to all participants, for examination or contribution to its development.

INTERNET APPLICATION: The IRC is positioned on the application layer in the Internet; therefore all technical structures of the lower layers have been considered exogenous in my examinations.

As my exploration is limited to one very specific Internet application setting, further work has to be done in order to show a more general viability of the «code» governance model. It should be challenged against other techno-social settings, to show its potential as well as its limits. In the following subsections, I identify three areas which appear worthy challenge for my model: social software, the domain name system, and digital rights management systems.

"Social Software"

The most obvious candidates for validating and refining the «code» governance model would be settings which are similar to the Internet Relay Chat: Internet applications which serve as a communication medium, allowing its participants to communicate and interact with each other, and thus build social groups. Recently, the term "social software" has been proposed for this kind of applications¹; it is described as "software that supports group interaction"². Besides the IRC as one category, instant messaging and Internet forums, blogs and wikis up to social networking software, peer-to-peer networks and multiplayer online games are seen as examples of "social software".

Empirical analyses of such applications should give valuable insights in how the «code» is designed to cope with the social setting, and how and to which extent the participants and stakeholders are taking part in govern their setting through «code», thereby validating and expanding the preliminary results found in this work. Specific attention should be given to the interaction between the social realm and the «code», as these application might differ in their social settings: For example, many online games or social networking sites have a corporate

¹See for example "Social Software", Wikipedia, 2005-08-25 (http://en.wikipedia.org/wiki/Social_software).

²http://www.shirky.com/writings/group_enemy.html, according to "Social software", Wikipedia (2005-08-25).

backing which could influence the ways how the «code» is used for governance issues, with law and market regulation modalities influencing the setting besides «code» and social norms.

In this way, applications in the “social software” category provides a rich field for «code» governance analyses, and help validating and refining this concept.

Domain Name System

The debate around the organization and administration of reserved identifiers in the Internet, the domain name system (DNS) debate is another example where the «code» governance concept should be applied to in order to prove its usefulness.

The DNS debate has developed into what is seen as the major governance issue in the Internet, so that the the term “Internet governance” has almost become a synonym for the DNS debate, the question about *who should control* the distribution and management of identifiers in the Internet. The diversity of the stakeholders, the strong corporate and governmental interests involved, and and the current importance of the DNS for the applications in the Internet makes it an interesting case to apply the «code» governance concept to. Additionally, as outlined above in the comparison between the IRC and the DNS³, the current DNS systems shares some technical similarities with the Internet Relay Chat, which could help applying results found in this work to this much larger setting. In total, the disputes around the domain name system form a formidable challenge to the «code» governance concept, promising valuable insights into this hotly debated issue.

Digital Rights Management

Another important debate concerns the question of “property” and “control” in the digital realm, usually labeled as “intellectual property” or “digital copyright” issues. Particularly interesting in this field is the specific use of the technology. “Digital rights management” (DRM) systems are an example where content holders imprint their specific interests into the technology, thereby trying to enforce what they hold their legitimate control over the distributed content. At the same time one encounters individual efforts to reshape these DRM systems so that they serve their individual interests, or some larger “public” interests. Finally, governments strive to (re-)form the respective regulatory regimes (primarily via law) as to accommodate these different interests.

As these technologies are highly intertwined as part of the legal and societal complex settings, where all regulation modalities—law, market, social norms, code—play an important but varying part, the «code» governance approach could be used to clarify how design and implementation choices of DRMs affect the various stakeholders, and what interests and choices

³Chapter 8.1.

are coded into technology. Such results might lead to regulatory alternatives, for example by alternative «code» design for DRMs, or inform policy considerations leading to a better regulation. At the same time, applying the «code» governance model to the DRM issue should give insights into the power and limits of this model, and let it refine its methodological instruments.

11.2 Computer Science Implications from «Code» Governance

Once a better understanding of the «code» governance model and its application to other empirical settings has been developed, there are some implications for the discipline of computer science. If this model proves useful for grasping the complexities of techno-social settings, then computer science should develop approaches to treat social issues as *endogenous to software and systems design*. Additionally, recent important techno-social research from computer science scholars using their core competencies show that models such as «code» governance promise valuable contributions from computer science towards better policies on technology issues in society.

Recently, the U.S. National Science Foundation (NSF) launched a new program titled "Science of Design – Software-Intensive Systems"⁴, which addresses the ongoing struggle of software engineering and computer science with the complex processes of reliable and systematic system design. In an article in the Communications of the ACM⁵, the NSF authors claim that "[c]omputer science and engineering needs an intellectually rigorous, analytical, teachable design process to ensure development of systems we all can live with"⁶. They give examples for questions to be addressed, such as:

- "How might e-voting systems be designed [...] to protect civil rights and personal privacy while minimizing political bias and security threats?"
- "How might urban planning systems be designed to account for social and cultural diversity, public decision making, and accessibility?"⁷

The questions make clear that one of the predominant problem at hand is to cope with the complexity of the dynamic interdependencies between the larger social and societal contexts

⁴See Science of Design program solicitation at <http://www.nsf.gov/pubs/2004/nsf04552/nsf04552.htm> (2005-06-24).

⁵Freeman and Hart (2004).

⁶ibid.

⁷ibid., p.20.

and the systems-to-be-designed. There is a need for techno-social models adapted to the specifics of the system design. Strangely though, such a conclusion is not reflected by the projects currently awarded under the NSF program⁸: almost none of them give considerations to topics or theories outside the core computer science and software engineering realm⁹.

The results of this work back my conviction that computer science should systematically incorporate the social context into the design process in order to deal with the complexities of today's systems. Any effort like the "science of design" research therefore should be concerned with the social and societal components of the systems on the level of the design and implementation¹⁰. «code» governance constitutes one concept where the incorporation of social contexts into the design and implementation processes to deal with socio-technical complexities have been examined. This approach, if nothing else, shows that computer science and software engineering need to proactively deal with the social contexts in their own discipline, instead of solely relying on other disciplines for concepts and theories.

The inclusion of techno-social contexts into the core of the design and implementation processes could open up another important venue for the computer science research at large. Recently, we can observe contributions from computer science scholars which combine a deep knowledge of the technical workings with models from important new multidisciplinary research areas, such as the new institutionalism¹¹, institutional economics¹² or law & economics¹³. Examples includes works on the impact of architectural principles of the Internet such as the end-to-end on economic innovation¹⁴, and various contributions to the research on the open source phenomenon¹⁵.

Their competitive advantage compared to the works from other disciplines lies in their

⁸The list of awards can be found via the NSF award search (<http://www.nsf.gov/awardsearch/>), keyword "Science of Design". On 2005-09-04, the search returned the awards numbered 0438970, 0438153, 0438923, 0438866, 0438931, 0438948, 0439017, 0438786.

⁹The exception is the "value-based science of design" (<http://www.nsf.gov/awardsearch/showAward.do?AwardNumber=0438931>, 2005-09-04), based on a "software economics" approach of the principal investigators which appears to incorporate some economics-based theories; see also Boehm et al. (2001).

¹⁰This is also suggested by the "value sensitive design" research (Friedman et al. (2003)) which draws its ancestorship from the long tradition of "computers and society" research, such as computer ethics, social informatics, computer-supported cooperative work and participatory design (ibid, pp. 2-3). Their difference to the «code governance» concept lies in their approach taken on the "value" proposition: They treat the so called "values with ethical import" as *exogenous* to the design process, presenting a list of values with a "distinct claim on resources in the design process". Prima facie this appears to be a unnecessary limitation: as «code governance» has shown, system participants find «code» solutions for social conflicts which itself formulates unique value sets (see for example the IRC "ownership" solutions in chapter 6). The «code governance» opens up venues where such endogenous values could be detected and offered as valuable policy debate inputs from the technical perspective.

¹¹See for example Ostrom (1990).

¹²See for example Richter and Furubotn (1999).

¹³The seminal work in this area is Richard Posner, *Economic analysis of law* (1992, Boston: Little, Brown).

¹⁴van Schewick (2004)

¹⁵See for example Gehring and Lutterbeck (2004); Bärwolff et al. (2005), Gehring (2005).

11 Outlook

unique perspective of an intimate understanding of the technical workings and computer science concepts. This leads to better analyses and alternative policy suggestions for techno-social quandaries than solely pursued through legal, economic or political science scholarship. The incorporation of a «code» governance understanding into the system design could further help to foster such important computer science contributions to the "internet governance", "information society" or other techno-social policy debates.

It is to hope, that the example set by those computer science scholarly works—and hopefully by this thesis as well—will be more stringently pursued by the computer science discipline, for better techno-social systems designs, and a deeper understanding of the complexities of the information and communication technologies in society.

Part IV

Appendix

12 IRC Chronology

The analysis of the IRC «code» governance which I attempt here draws heavily on the changes throughout the history of the IRC: I compare the different software versions over the time, noting changes and interpret them as governance changes, linking them to events as found in textual documents on the IRC. It is therefore necessary to sketch a short history of the IRC to give a timeline of events and introduce the different IRC networks.

12.1 1989 – The Birth of the Internet Relay Chat

The beginning of the Internet Relay Chat bears interesting parallels with the probably best known open source software project, Linux. Like Linux, the IRC has been initially conceived and programmed by a Finnish student. And like Linux, the first versions of IRC has been distributed over the Internet (USENET News), with other people getting interested and starting to use and change the software, in addition to interconnect the servers to form the first IRC network. Today, like Linux, the IRC has grown to one of the major projects of its kind.

The IRC was created in the summer of 1988 by the Finnish student Jarkko Oikarinen. At that time, the Internet was not the ubiquitous communication network that it is today. In the corporate and academic world, networks like the Bitnet (IBM) and DECnet (Digital) existed next to subscription-based like CompuServe; in addition, there were many private-run Bulletin Board Systems, which were stand-alone servers to which users connected directly through telephone lines. Most BBS's were usually not continuously connected to each other, but there existed a technology called "store-and-forward" networks, where the server exchanged data by calling each other at specified times of the day (mostly nights, to save phone charges); mainly emails as well as Usenet news were transported through such networks.

Oikarinen had access to or knowledge of many such systems: on one hand a BBS called "OuluBox" (which administered Oikarinen) which allowed real-time discussions (i.e, chats), but did not feature a multi-group (or multi-channel) system. Usenet news, on the other hand, is a multi-group system, but is not a real-time system. Other examples were a Unix application called "phone" (a real-time chat system between two users on Unix systems), other examples the Bitnet chat system as well as the DECnet "phone".

In August 1988, Oikarinen created the Internet Relay Chat as a multi-group real-time dis-

cussion system running on Unix systems¹ over the Internet:

2) Did you come up with IRC just because you were frustrated by the limitations of talk, or were there other reasons?

Kev

I believe that IRC came up because there was a clear need for it. It was not to replace talk.. the purpose of talk is different. The original purpose of IRC was more like to provide similar features that existed on BITNET and DECNET, ie. bitnet chat system and the DECNET phone.

At first it ran as single-server system on a host at the University of Oulu where Oikarinen was employed. But soon he found friends in other Finnish universities (Tampere, Helsinki) who installed IRC servers connecting to his server, and soon the IRC became a multi-server network with a number of Finnish servers. At one time, Oikarinen gave the software to some people in the U.S., and after a while he got messages from people in U.S. universities who successfully ran the IRC software. By interconnecting the Finnish and the U.S. network, the original IRC network (later dubbed *Onet* for "original" network) was born.

12.2 1989-1990 – Copyright, named channels and the "great split"

After only one year (mid-1989) there were already about 40 servers in the O-net, but not many users—in July 1990, per average only 12 users. But the code base was constantly fixed and refined, going through more than 10 versions. In 1989, Michael Sandrof also created the first independent IRC client version, *ircII*, which for a long time remained the preferred IRC client software of most users.

Copyright issues

The first important incident occurred with *ircd* version 2.3 (April 1990), where the distributing person changed the copyright notice without knowledge of the other developers (including Jarkko Oikarinen) to a non-existing "IRC Development Consortium". As consequence, Jarkko Oikarinen (the then copyright holder) released the next version 2.4 under the Gnu Public License (GPL), under which all subsequent *ircd* versions are released².

Named channels

The summer of the same year 1990 saw two further developments which are important especially under the «code» governance aspects examined in this work. First, with the server code

¹At that time, the majority of the hosts in the Internet were running the Unix operating system.

²See also chapter 12.7

version 2.5 and 2.5+, a new kind of channel was introduced: the named channel. The details of this change were examined above³, but this change gave the normal user more power in shaping the communication over the channels, and with it more responsibility, and more conflicts.

The first "Great Split"

The second incident sheds some light on the IRC network management and policy conflicts. It is based on a year long dispute over how open the network should be regarding the acceptance of new IRC servers, which culminated in the first forking of an IRC network, called "the great split"⁴. It resulted in two different IRC networks with different policies: The A-net (for Anarchy net) and the EFnet (for eris-free net, after the main server on A-net, eris.berkeley.edu). Eventually though, the A-net vanished, while the EFnet remained and today still forms one of the largest IRC networks.

12.3 1990-1992 – Growth and Development

EFnet continued to grow, both in the number of users and servers:

- 1991 already over a hundred servers are interconnected.
- 1991 gulf war brings live reports from the Iraq, boosting the usership first by the hundreds, later in the thousands. In late 1994 around 5000, a year later 15000, in 1997 30000, and so on. Nowadays the largest network around 50000 to 100000 users per network.
- The continuous growth poses challenges on system stability and feature sets. This results in constant code development efforts: In July 1991, irc2.6 version is released. The end of the same year brings the irc2.7 version. March 1993 brings the irc2.8 version with a large number of enhancements.
- In May 1993, the IRC client-server protocol (Oikarinen and Reed, 1993) is approved by the Internet Engineering Task Force, and therefore made 'officially' an Internet protocol (Request for comments, RFC).

³Chapter 4.3.

⁴See above chapter 8.2.

12.4 1993 – The first successful fork: Undernet

The Undernet⁵ emerged from the joining of three separate groups of which two started as a test network. In the U.S., Daniel Mitchell ("Wildthang") started a server in October 1992 where bots were tested. But apparently, friends also started to use it to chat with each other, and soon other servers joined this "friends-of-friends network". Around the same time, french server administrator Laurent Demailly ("_dl") and P. Ducrot ("dp") tested the new 2.8 server with a small net. On the Canadian side, Donnie Lambert ("Whizzard") had set up a server for the EFnet, but got his link revoked. So, in December 1992 the Canadian server linked to the french servers, and this Canadian-French net linked to the U.S. network. Thus the Undernet was formed.

12.5 1994 – Another fork: DALnet

Another case is the creation of DALnet. This is the initiative of members of the channel "#StarTrek" on EFnet, fans of the TV series "Star Trek". Similar to the creation of the Undernet, the DALnet founders were annoyed with the lacking stability and user-friendliness of the EFnet, and so some members formed a small network to which others connected to form a new network. The main feature of this network are some services which introduced a kind of nickname and channel ownership (NickServ, ChanServ⁶), as well as a service allowing to leave messages for offline users (MemoServ).

The DALnet forked off the Undernet, and was named after its founder, a user named "dal-venjah". Their main hallmarks are the ChanServ and NickServ services, next to many other enhancements of the server code and the administrative environment.

There are two subsequent server series: First the "dreamforge" series, later the "Bahamut" series which still remains in active server development. Recently, in 2003, DALnet suffered a number of severe attacks which brought the whole DALnet down; their future at that time remained unclear.

12.6 1996 – IRCnet forks off EFnet

Similar to the split between Anet and EFnet⁷, IRCnet resulted from another forking efforts. In the case of the IRCnet, a dispute about the choice of a constitutional technology against net splits lead to the split of the IRCnet from the EFnet. What is interesting here is that the split went along 'cultural' lines: Most of the U.S. server remained in the EFnet, while the large

⁵The following derived from Mirashi and Brown (2003).

⁶See above chapter 6.3.1.

⁷See above chapter 8.2

majority of the European servers formed the IRCnet. In the split, IRCnet took the then newly developed irc2.9 series with it, because the main maintainer (an Australian) decided to go with IRCnet. EFnet continued then with the comstud, +TH and hybrid series.

12.7 The IRC Source Code Copyright

Like most software of the main Internet applications, the software underlying the IRC has been open sourced from the beginning on, and the server software of the main IRC networks has been kept under the GNU Public License (GPL) to this day. Here I will outline how the IRC software came under the GPL, and what the main consequences for the development and use of the software are.

In the beginning of the IRC, the creator Jarkko Oikarinen put a simple copyright into the IRC software⁸. This named him and his employer, the Computing Center of the University of Oulu, Finland, as copyright holder. The copyright notice gave permission for free distribution of the software, but limited the use to non-commercial purposes. Furthermore, the license gave no permission for the modification and distribution of modified versions. This notice remained until version irc2.2 of December 1989. Then an incident convinced Oikarinen to change his mind about the copyright.

This incident happened with the version 2.3. The IRC software was actively developed by a changing group of people, and the IRC software underwent a quick succession of main versions and minor bug fixes and enhancements. Version irc2.2 was quickly followed by two patches (irc2.2PL0, irc2.2PL1), but "had a tendency to die mysteriously very often"⁹. Thus Markku Savela took over and released a number of enhancements named irc2.2msa.x (x being a number). At some point another coder, Chelsea Ashley Dyerma contributed to the IRC code. She and Savela began to work on the new version irc2.3, which would consolidate the changes that Savela (and others) made to the IRC server source code.

On 6 April 1990 at 10:50¹⁰, Dyerma announced on the "operlist" mailing list the release of the irc2.3 version¹¹. On the same day, at 19:27, Jarkko Oikarinen, the creator of the IRC, sent a message¹² on "operlist" where he asks about some changes that obviously were made by Dyerma. He noted that the copyright message in the source code had been changed from

(c) 1988 University of Oulu, Computing Center

to

⁸[irc2.1.1/COPYRIGHT']

⁹[irc2.4/2.4.notes]

¹⁰All times converted to Central European Time (CET).

¹¹Dyerma, Chelsea Ashley (1990-04-06) *irc 2.3*. Mailing list *IRClst* (1990).

¹²Oikarinen, Jarkko (1990-04-06) *New IRC version comments...* Mailing list *IRClst* (1990).

12 IRC Chronology

```
@(#) * Copyright (c) 1990 IRC Development Consortium.\n\nAll rights reserved.\n
```

As it became clear quite fast, no such organization existed.

In addition, some files were changed but there was no information in the header of the file informing of the changes. Only ten minutes later, Oikarinen had found another major change in the code, one that enabled all IRC operators to see all secret and private channels:

I'm not sure (I'm not interested in trying the new irc right now), but does the new server include things which affect irc privacy ?
Like operators being able to see secret/private channels ?
I think I'm going to quit using irc soon if that's true.¹³

Other people, including Markku Savela, commented or asked about these changes, especially the copyright changes. The answer followed promptly by Dyerman, who made these copyright changes, in an apology to Oikarinen:

I can't not even begin to say how much of a real creep I am. I was very wrong in removing the headers, and installing the new. I will not try and hide behind excuses, and try and run from what I have done. I would like to try and explain what the meaning that this whole mess of mine is, if you will take this time to read on...¹⁴

She explained the change as a kind of joke, since so many contributed to the IRC code besides Oikarinen. Also there was some kind of misunderstanding, confirmed by a message of Greg Lindahl to Dyerman and forwarded by her to the operlist mailing list:

Please, Casie, don't be so perturbed at the latest irc flap. I'm partially to blame for the copyright affair; I am the person who contacted WiZ about it and only I have read his email reply.¹⁵

The problematic version irc2.3, released on April 6, 1990, was made non-readable on the next day¹⁶. Server administrator continued to use the previous version 2.2, and Savela continued with some more patches in the 2.2msa series.

This incident obviously convinced the copyright holder, Oikarinen, to change the copyright to the GNU Public Licence (GPL). Version 2.4, coordinated by Markku Savela and Chelsea Ashley Dyerman, was the first IRC software version which was put under the GPL¹⁷. Since this time, all major IRC software versions carry the GPL licence.

¹³Oikarinen, Jarkko (1990-04-06) *Privacy*. Mailing list *IRClst* (1990).

¹⁴Dyerman, Chelsea Ashley (1990-04-07) *irc*. Mailing list *IRClst* (1990).

¹⁵Forwarded message in Dyerman, Chelsea Ashley (1990-04-08) *the latest irc flap (fwd)*. Mailing list *IRClst* (1990).

¹⁶Oikarinen, Jarkko (1990-04-06) *Re: irc*. Mailing list *IRClst* (1990).

¹⁷As an interesting side note, Markku Savela, which gave important contributions to the IRC software, decided to quit contributing to the IRC because of the GPL: He interpreted the GPL in that he could not use his own code which he contributed to the IRC in other (commercial) projects anymore, and thus: "If the above interpretation is true, I cannot modify or contribute to any program that has GNU copyleft. I must have the option to use my code as I wish, even if I allow others the right to use it as they wish." (Savela, Markku (1990-04-06) *Release 2.4 hassles...* *IRClst* (1990)). But as is so often the case with open source projects, his place was taken over by other volunteer programmers.

13 Tools for the Examination of the IRC Source Code

13.1 In Search of the Right Tool

At the time I realized that my hypothesis needed a in-depth exploration of the IRC source code I already had read myself through much of the source code in a non-systematic way, and with no other tools than what a regular Unix environment offered.¹ Nevertheless I ventured to search what kind of source code analysis tools existed, and how they could help me in this specific kind of code examination.

To this behalf, I first initiated some searches in Google and the ACM website, and then traced the resulting source code analysis tools, information about them, information which could help me to find further pointers. I deemed this task finished when subsequent searches only revealed the same tools and information without no new results.

Before I offer the results of this search though, I should specify what I deemed necessary features of such a tool. My starting point were over 200 source code packages of IRC server versions, each of them consisting of between 20 and 140 source code files (among a total of between 50 and 480 files) for the IRC server. The server is written in the C programming language, using BSD style libraries (e.g. sockets)². What I needed was a tool which processed these file to allow both examinations of one version (in order to understand how the server, or a specific feature is implemented) and the differences between two or more versions (in order to understand how a specific feature evolves over the time).

Especially the latter functionality was important for me, since the tracking of changes in functions helps to understand their «code» governance relevance. So I will mostly concentrate here on the search of tools allowing me to compare and analyse a version succession of C source code.

As one final condition, I was restricted to find an either cost-free or reasonably priced solution. As we will see, this ruled out even the evaluation of some interesting sounding products,

¹Which by the way is a quite powerful set of tools, even in the hands of an intermediate user such as myself.

²In early versions, there are file which provide an alternative Unix System V library dependency instead of the BSD one, but these efforts ceased to exist early on. In a current version of the Undernet server code, provisions for other operating system libraries (Linux, OpenBSD, Solaris) are made.

because I had no access to them.

Specifically, I have looked into *version control systems* and *cross referencing tools*.

Version Control Systems

The first idea was to use one of the available version control systems. Such systems are used in software projects to manage changes in files (source code and other files) where many developers may have writing access to. It automates tracking revisions of a file, allows to define version points (a snapshot of the project where the state of all files is fixed) and branching (allowing concurrent versions of one software), and provides tools to manage collisions (multiple developers changing one file at the same time). Of the many version control systems in existence³, I have checked "CVS", the control version system⁴.

A definitive plus of such systems are the ability to define branches. In the course of the IRC server code development, there are all kinds of code branches:

- The changes of the "bu"⁵ versions branched off irc2.5.1. But after the last bu version, the new irc2.6 series was developed on basis of the irc2.5 series, and only selected changes of the "bu" branch were incorporated back into the irc2.6 code.
- Within the EFnet, at least two concurrently developed IRC server branches emerged from one version (irc2.8.21) which continue to coexist. While the development is done independently of each other, there are code parts which are adopted from the competing branch and incorporated into their own code. Similar concurrent versions exist also in the Undernet (where different code series coexist), and presumably in other networks as well.
- Although the server software of different IRC networks are not interoperable, there are many similarities resulting from a common code base (Undernet and IRCnet code branched off EFnet versions; DALnet branched off Undernet, etc.) as well as adoption and incorporation of code innovations into the own software, be it by recoding it, or adapting existing code (verbatim code copying has been done rarely).

While the branching feature would have been helpful, the whole system is geared towards the coordination of source code for actual development, instead of an after-the-facts analysis. Based on constraints in time and knowledge, I found that CVS offered not enough support for my needs, such as the preservation of time stamps when importing the packages (the IRC server packages retain their time stamps, which is an important information of when the file has been last changed). Moreover, CVS did not offer easy support for visualization of tracking

³See for example "Revision control." Wikipedia. 2005-04-04 http://en.wikipedia.org/wiki/Revision_control.

⁴See <http://www.cvs.org/>

⁵"bu" stands for Boston University, students of which maintained these versions.

differences; the addition of such a feature would have necessitated to create another layer on top of the CVS, making the efforts to bring the code packages into CVS too cumbersome.

Cross Reference Tools

Cross reference tools take a number of source code files of one software package and generate documentation files in various formats allowing to browse through the code. Often seen features include an index of functions, variables, definitions and other code objects, and taking specially formatted comments in the code to create an automated documentation.

Existing tools⁶ not only differ by the features they offer, but also by the number of programming languages supported. The most common uses for such tools are auto-generated source code documentation, information gathering for debugging processes, and to a certain extent the exploration of the code structures of undocumented source code. Note that these tools do not allow to compare software versions, but can only applied to one version at a time.

The most promising tools appeared to be Doxygen⁷. At the time of writing in version 1.4.2 (28 Mar 2005), I had evaluated versions 1.2.5 (March 2001) and 1.3.8 (August 2004). According to the manual, it is a "documentation system for C++, C, Java, Objective-C, IDL (Corba and Microsoft flavors) and to some extent PHP, C#, and D", offering to "generate an on-line documentation browser (in HTML) and/or an off-line reference manual (in L^AT_EX)", and configure it to "extract the code structure from undocumented source files". Most of the features are geared towards documentation of actively developed source code, rather than examining a given code package. But the documentation offers also support for such tasks by creating "dependency graphs, inheritance diagrams, and collaboration diagrams".

The output generated by Doxygen is impressive. Once I had found out the proper configuration settings, it generated a thorough documentation of the available source code, including

- a dependency graph for each file (which header files are included);
- for each define directive and function the definition, references to other objects, a list of functions which reference this one, a call graph, and the source code listing;
- for each data structure its field list (and graphical representation), and for each field the definition and a list of functions referencing this field.

Interestingly though, upon trying to use this comprehensive output, I found myself returning to the 'low-tech' variant of using the terminal and the Unix tools (vi editor, ctags, as well as string search tools, such as 'find' or 'grep') to explore the source code. Both the speed as well as flexibility of exploration of the code proved to be better served with the terminal than the Doxygen output⁸.

⁶For a list of such tools, see for example <http://www.doxygen.org/links.html> (2005-04-05).

⁷<http://www.doxygen.org/index.html>. The following quotes are from this page as of 5 Apr 2005.

⁸The fact that I am a rather fast touch typist might have been an important factor here as well.

In conclusion, cross reference tools such as Doxygen offer a sophisticated way to examine a source code package. But the quality of output is geared towards a static documentation, rather than a dynamic exploration of the code. Therefore, such cross reference tools greatly help to get an overview of a source code, which then can be used to further explore the code with more dynamic tools. For my «code» governance purposes though, the output did not prove flexible enough to support me in 'wading' through the source code, so that I basically fell back to using basic tools, such as editor and Unix tools.

Other Tools

There are a number of other tools which aid developers to understand code structures, such as code visualization tools⁹, or software slicing¹⁰. The IEEE Software Engineering Body of Knowledge subsumes these tools under the category of "Software maintenance tools", sub-category "Comprehension tools"¹¹. These tools are more geared towards helping in debug and optimizing processes by offering views on the code where the complexity of the code is reduced. Such tools appeared too large for my smaller needs (from the code complexity, the IRC server code is only a small to medium sized project); in addition, these topics are in active research, where available tools are often just a proof of concept, rather than generally usable software.

13.2 A Makeshift Solution

As none of the available tools appeared to have a viable effort-result ratio, I settled on a makeshift 'solution', using available Unix tools for the process. A mix of small python, awk and shell scripts with a small C program interspersed takes two directories (each containing source code and other files of one software version) and generates some HTML files (and intermediate text files as well).

At the core, the corresponding file in each of the two directory trees is matched and compared. If there are differences, then a HTML file is generated which shows a side-by-side comparison of each line of the files (generated by the GNU 'diff' program). In addition, a overview file is generated in which for each file or file pair in the two directory trees, a status is given (for a file pair: 'identical' or 'differ', for single files: 'added', 'removed'), together with the (linked) position of the file.

As the file positions inside the directory tree has been changed throughout the versions (by creating new subdirectories, or simply renaming them, for example), the scripts make some

⁹See for example http://www.cc.gatech.edu/classes/AY2001/cs7450_fall/Talks/18-softvis.ppt (2005-02-15) for a good overview of existing tools and research.

¹⁰For a list of projects, see for example <http://www.infosun.fmi.uni-passau.de/st/staff/krinke/slicing/node2.html> (2005-04-05); a list of papers offers <http://hissa.ncsl.nist.gov/~jimmy/refs.html> (2005-04-05).

¹¹IEEE (2004, p.10-2)

attempts to locate the corresponding file. If no such file was found, the file is tagged either 'added' or 'removed'. If there was still a corresponding file, then I have manually generated and examined the changes.

13.3 The Analysis

Although the actual exploration was not a linear process as I describe it here, one can nevertheless distinguish some steps of analysis that I went through.

The first step is to identify «code» governance-relevant parts in the software. As the topics in Part II show, there were plenty of alternatives from which I could choose: functions and commands surrounding channel issues, sanctioning mechanisms, ownership issues, checks and balances, and network constitutional issues could be readily identified, and are important governance topics even outside «code» related settings.

Once these functionalities were identified, I examined the actual implementation in more or less arbitrarily chosen single server version. For example, by tracking the processing of a command through the functions in the code, I identified not only the general working of that command, but all details such as options, conditions on its use, notification and logging facilities etc. When comparing to the command documentation (if available at all), more than once it turned out to be incomplete, or even wrong. So the code examination proved the only way to grasp the entire functionality of a command. This kind of source code exploration was done without any tools other than an editor (vi), and the 'ctags' functionality¹².

The next step after understanding the functionality was to track changes throughout the server versions. To this behalf, I generated the HTML files showing the file differences between versions for each adjacent version pair for all server code versions that I had found in the Internet. Now I could trace changes in the implementation through the versions in the side-by-side comparisons for the appropriate files, looking for governance changes in the code.

In sum, my exploration into the IRC server source code was quite 'low-tech', by using standard Unix tools and some make-shift scripts which created a HTML-ized file-by-file comparison of adjacent versions.

¹²The command ctags creates an index of code objects (such as functions, variables, directives etc.) which then can be used by editors (such as vi) to quickly locate that object. For example, when in the course of the code examination a function call occurs, one can issue a vi command in order to go to the function definition. This greatly helps to understand the sequence of processing in the source code.

14 List of IRC server source code packages

This is a list of all IRC server source code packages that I have collected and examined for my work. They are put under the heading of the network for which they were developed.

14.1 Onet

All versions are available from <ftp://ftp.irc.org/pub/pub/irc/server/Old/> (2005-04-27)

Version	Archive filename	Release Date
irc2.1.1	irc2.1.1.tar.Z	1989-10-20
irc2.2PL0	irc2.2.tar	1989-12-17
irc2.4	irc2.4.tar.Z	1990-05-10
irc2.4+	irc2.4+.tar.Z	1990-05-22
irc2.5	irc2.5.tar.Z	1990-07-07
irc2.5+	irc2.5+.tar.gz	1990-07-09

Table 14.1: Onet server versions

14.2 EFnet

14.2.1 Standard

The standard version goes from irc2.5.1 through irc2.8.21.

All versions are available from <ftp://ftp.irc.org/pub/pub/irc/server/Old/> (2005-04-27)

14.2.2 +CS

The main developer of this server code series was Chris Behrens (comstud).

Built on basis of irc2.8.21

14 List of IRC server source code packages

Version	Archive filename	Release Date
irc2.5.1	irc2.5.1.tgz	1990-09-06
irc2.5.1bu8	diffs.2.5.1-to-bu.08	1990-10-22
irc2.5.1bu9a	diffs.2.5.1.bu.09a-to-bu.10	1990-11-12
irc2.5.1bu10	irc2.5.1.bu.10.tar.Z	1991-01-01

Version	Archive filename	Release Date
irc2.6pre19	irc2.6pre19.tar.Z	1991-03-08
irc2.6pre19patched	irc2.6pre19.patched.tar.Z	1991-03-09
irc2.6.1	irc2.6.1.tar.Z	1991-07-04
irc2.6.2	irc2.6.2.tar.Z	1991-09-02
irc2.6.2d	irc2.6.2d.tar.Z	1991-11-06
irc2.6.2f	irc2.6.2f.tar.Z	1992-11-21

Table 14.2: EFnet server versions 2.5 and 2.6

Version	Archive filename	Release Date
irc2.7.1	irc2.7.1.tar.Z	1992-01-14
irc2.7.1e	irc2.7.1e.tar.Z	1992-03-15
irc2.7.1e+4	irc2.7.1e+4.tar.Z	1992-04-08
irc2.7.1e+10	irc2.7.1e+10.tar.Z	1992-04-28
irc2.7.2	irc2.7.2.tar.Z	1992-05-14
irc2.7.2c	irc2.7.2c.tar.Z	1992-05-22
irc2.7.2e	irc2.7.2e.tar.Z	1992-06-03
irc2.7.2f	irc2.7.2f.tar.Z	1992-06-16
irc2.7.2g	irc2.7.2g.tar.Z	1992-08-11
irc2.7.2h	irc2.7.2h.tar.Z	1993-03-24
irc2.7.2i	irc2.7.2h-i.patch	1993-04-02

Version	Archive filename	Release Date
irc2.8	irc2.8.tar.Z	1993-03-28
irc2.8.5	irc2.8.5.tar.Z	1993-04-20
irc2.8.6	irc2.8.6.tar.Z	1993-04-30
irc2.8.7	irc2.8.7.tar.Z	1993-05-05
irc2.8.9	irc2.8.9.tar.Z	1993-05-26
irc2.8.10	irc2.8.10.tar.gz	1993-06-26
irc2.8.12	irc2.8.12.tar.gz	1993-07-08
irc2.8.14	irc2.8.14.tar.gz	1993-09-14
irc2.8.15	irc2.8.15.tar.Z	1993-10-17
irc2.8.16	irc2.8.16.tar.gz	1993-11-09
irc2.8.20	irc2.8.20.tar.Z	1994-06-10
irc2.8.21	irc2.8.21.tar.gz	1994-12-03

Table 14.3: EFnet server versions 2.7 and 2.8

All versions are available from <http://ftp.carnet.hr/pub/misc/irc/ircd/CSr/> (2005-04-27)

Version	Archive filename	Release Date
irc2.8.21+CSr16	21+CSr15-r16.patch	1995-12-09
irc2.8.21+CSr17	21+CSr16-r17.patch	1995-12-02
irc2.8.21+CSr18	21+CSr17-r18.patch	1996-01-04
irc2.8.21+CSr19	21+CSr18-r19.patch	1996-01-05
irc2.8.21+CSr20	irc2.8.21+CSr20.tar.gz	1996-01-01
irc2.8.21+CSr21	21+CSr20-r21.patch	1996-04-04
irc2.8.21+CSr22	21+CSr21-r22.patch	1996-04-07
irc2.8.21+CSr23	21+CSr22-r23.patch	1996-05-09
irc2.8.21+CSr24	irc2.8.21+CSr24.tar.gz	1996-07-05
irc2.8.21+CSr25	irc2.8.21+CSr25.tar.gz	1996-09-05
irc2.8.21+CSr27	irc2.8.21+CSr27.tar.gz	1996-11-02
irc2.8.21+CSr28	irc2.8.21+CSr28.tar.gz	1997-03-06
irc2.8.21+CSr29	irc2.8.21+CSr29.tar.gz	1997-03-08
irc2.8.21+CSr30	irc2.8.21+CSr30.tar.gz	1997-07-03
irc2.8.21+CSr30.5	irc2.8.21+CSr30.5.tar.gz	1997-07-04
irc2.8.21+CSr31pl2	irc2.8.21+CSr31pl2.tar.gz	1998-09-05
irc2.8.21+CSr31pl4	irc2.8.21+CSr31pl4.tar.gz	2000-05-07

Table 14.4: EFnet +CS (comstud) versions

14.2.3 +th and Hybrid

+TH: Developed by Taner Halicioglu. The TH series builds on the 2.8. series, but it is unclear on which specific version, but irc2.8.21 seems a viable candidate. the +th series is the basis of the Hybrid series.

All versions are available from <http://ftp.carnet.hr/pub/misc/irc/ircd/th/> (2005-04-27)

HYBRID: The EFnet hybrid server series takes the ircd2.8/th.v5a.3 and adds "WHO, WHOWAS, and IsMember() code from Comstud's irc2.8.21CSr29." ([ircd-hybrid-2/README.hybrid]).

The main source of the code was: <ftp://ftp.blackened.com/pub/irc/hybrid/old/> (2000-03-22). with newer files also in <ftp://ftp.ircdhelp.org/pub/unix/ircd/> (2003-01-13).

All files are available from <http://ftp.carnet.hr/pub/misc/irc/ircd/hybrid/> (2005-04-27)

14.3 Undernet

2.8 VERSIONS: The Undernet 2.8 versions all derived from their EFnet counterparts. As soon as the EFnet released a new versions, the changes were included and released as "mu"

14 List of IRC server source code packages

Version	Archive Filename	Release Date
hybrid-2	ircd-hybrid-2.tar.gz	1997-04-02
hybrid-3	ircd-hybrid-3.tar.gz	1997-06-07
hybrid-4	ircd-hybrid-4.tar.gz	1997-07-01
hybrid-4.1	ircd-hybrid-4.1.tar.gz	1997-08-04
hybrid-4.2	ircd-hybrid-4.2.tar.gz	1997-08-04
hybrid-4.3	ircd-hybrid-4.3.tar.gz	1997-08-09
hybrid-5	ircd-hybrid-5.tar.gz	1997-09-06
hybrid-5.1b5	ircd-hybrid-5.1b5.tar.gz	1997-10-08
hybrid-5.2	ircd-hybrid-5.2.tar.gz	1998-04-04
hybrid-5.2p1	ircd-hybrid-5.2p1.tar.gz	1998-05-03
hybrid-5.3	ircd-hybrid-5.3.tar.gz	1998-06-03
hybrid-5.3p2	ircd-hybrid-5.3p2.tar.gz	1998-09-09
hybrid-5.3p3	ircd-hybrid-5.3p3.tar.gz	1998-11-03
hybrid-5.3p4	ircd-hybrid-5.3p4.tar.gz	1998-11-05
hybrid-5.3p6	ircd-hybrid-5.3p6.tar.gz	1998-12-04
hybrid-5.3p7	ircd-hybrid-5.3p7.tar.gz	1999-06-02
ircd-hybrid-6.0	ircd-hybrid-6.0.tgz	2001-01-04
ircd-hybrid-6.3	ircd-hybrid-6.3.tgz	2002-02-07
ircd-hybrid-6.3.1	ircd-hybrid-6.3.1.tgz	2002-04-08

Version	Archive filename	Release Date
+th-5a.0	ircd+th-5a.0.tar.gz	1996-11-15
+th-5a.1	ircd+th-5a.1.tar.gz	1996-11-22
+th-5a.3a	ircd+th-5a.3a.tar.gz	1997-02-28
+th-6a	ircd+th-6a.tar.gz	1997-07-22

Table 14.5: EFnet server versions +th and HybridUndernet server versions u2.9 and u2.10

and "U" versions. "mu" and "U" differ slightly in their code.

All versions are available from <http://ftp.undernet.org/index.php?dir=/servers/old-versions> (2005-04-27)

Version	Archive filename	Release Date
ircd2.8.14.mu	ircd2.8.14.mu.tar.Z	1993-10-05
irc2.8.16.U2	irc2.8.16.U2.tar.gz	1994-02-08
irc2.8.16.mu	irc2.8.16.mu.tar.gz	1994-03-04
irc2.8.19.U3.2	irc2.8.19.U3.2.tar.gz	1994-05-05
irc2.8.19.mu1	irc2.8.19.mu1.tar.gz	1994-06-00
irc2.8.20.U4	irc2.8.20.U4.tar.gz	1994-06-05
irc2.8.20.mu2	irc2.8.20.mu2.tar.gz	1994-06-06
irc2.8.20.mu3	irc2.8.20.mu3.tgz	1994-11-06
irc2.8.21.mu3.1	irc2.8.20.mu3-21.mu3.1.patch.gz	1995-01-03

Table 14.6: Undernet server versions 2.8

U2.9/U2.10 VERSIONS: Beginning with u2.9, the code was developed independently of the EFnet versions.

All files but u2.10.11 are available from <http://ftp.carnet.hr/pub/misc/irc/ircd/ircu/> (2005-04-27). u2.10.11 is available from <http://stargate.ukscifi.net/servers/ircnet/> (2005-04-27).

Version	Archive filename	Release Date	Version	Archive filename	Release Date
ircu2.9.13.mu	ircu2.9.13.mu.tgz	1994-11-01	ircu2.10.00	ircu2.10.00.tgz	1997-07-03
ircu2.9.13	ircu2.9.13.tgz	1994-11-01	ircu2.10.02	ircu2.10.02.tgz	1998-03-05
ircu2.9.19	ircu2.9.19.tgz	1995-03-08	ircu2.10.03	ircu2.10.03.tgz	1998-04-08
ircu2.9.19.mu	ircu2.9.19.mu.tgz	1995-03-08	ircu2.10.04	ircu2.10.04.tgz	1998-05-01
ircu2.9.20	ircu2.9.20.tgz	1995-04-05	ircu2.10.05.9	ircu2.10.05.9.tgz	1999-02-01
ircu2.9.20.mu	ircu2.9.20.mu.tgz	1995-04-05	ircu2.10.07	ircu2.10.07.tar.gz	1999-11-06
irc2.8.21.mu3.2	irc2.8.21.mu3.2.tgz	1995-05-03	ircu2.10.07.pl6	ircu2.10.07.pl6.tgz	2000-01-00
ircu2.9.21.mu	ircu2.9.21.mu.tgz	1995-05-06	ircu2.10.07.11	ircu2.10.07.11.tgz	2000-03-08
ircu2.9.22	ircu2.9.22.tar.gz	1995-08-00	ircu2.10.08.01	ircu2.10.08.01.tar.gz	2000-05-07
ircu2.9.21.2	ircu2.9.21.2.mu.tgz	1995-09-07	ircu2.10.08.02	ircu2.10.08.02.tar.gz	2000-05-08
ircu2.9.30	ircu2.9.30.tgz	1996-03-05	ircu2.10.08.03	ircu2.10.08.03.tar.gz	2000-05-09
ircu2.9.31	ircu2.9.31.tgz	1996-06-08	ircu2.10.10.pl5	ircu2.10.10.pl5.tgz	2000-04-01
ircu2.9.32	ircu2.9.32.tgz	1996-08-00	ircu2.10.10.pl6	ircu2.10.10.pl6.tgz	2000-04-01
			ircu2.10.10	ircu2.10.10.tgz	2000-04-04
			ircu2.10.10pl20	ircu2.10.10pl20.tgz	2002-03-08
			ircu2.10.11	ircu2.10.11.tgz	2002-05-05

Table 14.7: Undernet server versions u2.9 and u2.10

14.4 IRCnet

2.9 VERSIONS: Technically, the versions 2.9 to 2.9.2 were developed while IRCnet was still part of the EFnet. But since the main developer of the 2.9 series left EFnet to continue to maintain the irc2.9 version, these first 2.9 versions are included here as well.

All versions are available from <ftp://ftp.irc.org/pub/pub/irc/server/Old/> (2005-04-27)

2.10 VERSIONS: All versions are available from <ftp://ftp.irc.org/pub/pub/irc/server/> (2005-04-27)

14 List of IRC server source code packages

Version	Archive filename	Release Date
ircu2.10.00	ircu2.10.00.tgz	1997-07-03
ircu2.10.02	ircu2.10.02.tgz	1998-03-05
ircu2.10.03	ircu2.10.03.tgz	1998-04-08
ircu2.10.04	ircu2.10.04.tgz	1998-05-01
ircu2.10.05.9	ircu2.10.05.9.tgz	1999-02-01
ircu2.10.07	ircu2.10.07.tar.gz	1999-11-06
ircu2.10.07.pl6	ircu2.10.07.pl6.tgz	2000-01-00
ircu2.10.07.11	ircu2.10.07.11.tgz	2000-03-08
ircu2.10.08.01	ircu2.10.08.01.tar.gz	2000-05-07
ircu2.10.08.02	ircu2.10.08.02.tar.gz	2000-05-08
ircu2.10.08.03	ircu2.10.08.03.tar.gz	2000-05-09
ircu2.10.10.pl5	ircu2.10.10.pl5.tgz	2000-04-01
ircu2.10.10.pl6	ircu2.10.10.pl6.tgz	2000-04-01
ircu2.10.10	ircu2.10.10.tgz	2000-04-04
ircu2.10.10pl20	ircu2.10.10pl20.tgz	2002-03-08
ircu2.10.11	ircu2.10.11.tgz	2002-05-05

Version	Archive filename	Release Date
irc2.10.0	irc2.10.0.tgz	1998-09-07
irc2.10.0p1	irc2.10.0p1.tgz	1998-09-07
irc2.10.0p3	irc2.10.0p3.tgz	1998-10-09
irc2.10.0p4	irc2.10.0p4.tgz	1998-10-01
irc2.10.0p5	irc2.10.0p5.tgz	1998-10-08
irc2.10.1	irc2.10.1.tgz	1998-11-02
irc2.10.2	irc2.10.2.tar.gz	1999-01-09
irc2.10.3	irc2.10.3.tar.gz	1999-08-03

Table 14.8: EFnet server versions 2.7 and 2.8

Bibliography

- Bärwolff, M., Gehring, R., and Lutterbeck, B. (2005). *Open Source Jahrbuch 2005*. Lehmanns Media.
- Bell, T. W. (2000). Pornography, Privacy, and Digital Self-Help. Public Law and Legal Theory Working Paper 17, University of San Diego, School of Law.
- Benkler, Y. (2000). From Consumers to Users: Shifting the Deeper Structures of Regulation Toward Sustainable Commons and User Access. *Federal Communications Law Journal*, 52(3):561–579.
- Bitnet-Relay (1986). Bitnet Relay User Guide. Corrective Actions – Relay Operator Responsibilities 8/21/86, <http://web.inter.nl.net/users/fred/relay/reluse.html>, 2004-12-16.
- Boehm, B., Port, D., and Sullivan, K. (2001). Value Based Software Engineering, http://www.nitrd.gov/subcommittee/sdp/vanderbilt/position_papers/barry_boehm_value_based_software.pdf, 2005-09-04.
- Brinton, A. (1997). IRC Operators Guide, <http://efnet.org/docs/opersguide>, 2002-12-22.
- Brown, S. (2003). history.txt [Undernet history, 1993-2003], <http://pfft.net/stacy/history.txt>, 2003-02-26.
- Commission-on Global-Governance, . (1995). *Our Global Neighborhood*. United Nations Commission On Global Governance.
- DALnet (2003). So you think you want to be a DALnet IRC OPERator, <http://help.dal.net/dnh/oper.php>, 2003-06-13.
- Ellickson, R. C. (1991). *Order without Law. How Neighbors Settle Disputes*. Cambridge (MA), London (UK): Harvard University Press.
- Freeman, P. and Hart, D. (2004). A Science of Design for Software-Intensive Systems. *Communications of the ACM*, 47(8):19–21.

Bibliography

- Frey, B. S. and Eichenberger, R. (2000). A Proposal for a Flexible Europe. Technical report, Institute for Empirical Research in Economics, University of Zurich.
- Friedman, B., Kahn, P. H., and Borning, A. (2003). Value Sensitive Design: Theory and Methods, <http://www.ischool.washington.edu/vsd/vsd-theory-methods-draft-june2003.pdf>, 2003-06-11.
- Frischmann, B. M. (2003). The Prospect of Reconciling Internet and Cyberspace. *Loyola University Chicago Law Journal*, 35:205–234.
- Froomkin, M. (2000). Wrong Turn in Cyberspace: Using ICANN to Route Around the APA and the Constitution, <http://personal.law.miami.edu/froomkin/articles/icann.pdf>, 2002-01-19.
- Gehring, R. (2005). The Institutionalization of Open Source. (*to appear in: Poiesis und Praxis*).
- Gehring, R. and Lutterbeck, B. (2004). *Open Source Jahrbuch 2004*. Lehmanns Media.
- Gilmore, J. (1991). Privacy, Technology, and the Open Society, <http://www.toad.com/gnu/cfp.talk.txt>, 2005-04-19.
- Goldsmith, J. (1998). Regulation of the Internet: Three Persistent Fallacies. *Chicago Kent Law Review*, 73(4):1119–1131.
- Grewlich, K. W. (1999). Conflict and good Governance in "Cyberspace", http://www.mpp-rdg.mpg.de/pdf_dat/grewlich.pdf, 2001-11-26.
- Hadfield, G. K. (2000). Privatizing Commercial Law: Lessons from the Middle and the Digital Ages, <http://ssrn.com/abstract=220252>, 2000-05-10.
- Hardy, T. (1994). The Proper Legal Regime for "Cyberspace". *University of Pittsburgh Law Review*, 55:933–1055.
- IEEE (2004). SWEBOK - Guide to the Software Engineering Body of Knowledge, http://www.swebok.org/ironman/pdf/Swebok_Ironman_June_23_%202004.pdf, 2004-07-15.
- irc faq (1995). IRC Frequently Asked Questions, Version 1.53, <http://ftp.irc.org/irc/docs/FAQ>, 2002-08-22.
- IRClst (1990). Irclst-operlist mailing list April 1990-May 1990 (discussion about irc-2.3), <ftp://metalab.unc.edu/pub/academic/communications/papers/irc/lists/irc-2.3.Z>, 2000-03-24.

- IRClst (1991). Irclst-operlist mailing list March 1990-November 1991, <ftp://metalab.unc.edu/pub/academic/communications/papers/irc/lists/irclst-operlist.Z>, 2000-03-24.
- Johnson, D. R. and Post, D. G. (1996a). And How Shall the Net Be Governed? A Meditation on the Relative Virtues of Decentralized, Emergent Law, <http://www.cli.org/emdraft.html>, 2000-09-19.
- Johnson, D. R. and Post, D. G. (1996b). Law and Borders – The Rise of Law in Cyberspace. *Stanford Law Review*, 48:1367.
- Johnston, D., Handa, S., and Morgan, C. (1997). *Cyberlaw*. Toronto: Stoddart.
- Kalt, C. (2000a). Internet Relay Chat: Architecture (RFC 2810). Request for Comments (RFC) 2810, Internet Engineering Task Force, Network Working Group.
- Kalt, C. (2000b). Internet Relay Chat: Channel Management (RFC 2811). Request for Comments (RFC) 2811, Internet Engineering Task Force, Network Working Group.
- Kalt, C. (2000c). Internet Relay Chat: Client Protocol (RFC 2812). Request for Comments (RFC) 2812, Internet Engineering Task Force, Network Working Group.
- Kalt, C. (2000d). Internet Relay Chat: Server Protocol (RFC 2813). Request for Comments (RFC) 2813, Internet Engineering Task Force, Network Working Group.
- Katyal, N. K. (2003). Digital Architecture as Crime Control. *Yale Law Journal*, 111:2261–2291.
- Kesan, J. P. and Shah, R. C. (2002). Shaping Code. Technical Report 2-18, University of Illinois College of Law.
- Kesan, J. P. and Shah, R. C. (2003a). Incorporating Societal Concerns into Communication Technologies. *IEEE Technology and Society Magazine*, Summer:28–33.
- Kesan, J. P. and Shah, R. C. (2003b). Manipulating the Governance Characteristics of Code. Research Paper 03-18, Illinois Public Law and Legal Theory Research.
- Kesan, J. P. and Shah, R. C. (2004). Deconstructing Code. Research Paper 04-22, University of Illinois College of Law.
- Kzoo and LadyDana (2001). Managing Annoyances on IRC. Version 1.0.0, <http://docs.dal.net/docs/annoy.html>, 2002-12-20.

Bibliography

- Lawrie, M. (2005). Brief history of the #gb channel, <http://uknet.com/gb/gb-lorry.html>, 2005-01-20.
- Lemley, M. A. (1998). The Law and Economics of Internet Norms. *Chicago-Kent Review*, 73:1257–1294.
- Lessig, L. (1998). The New Chicago School. *The Journal of Legal Studies*, XXVII(2):661–691.
- Lessig, L. (1999a). *Code and other Laws of Cyberspace*. Basic Book.
- Lessig, L. (1999b). The Law of the Horse: What Cyberlaw Might Teach. *Harvard Law Review*, 113(2):501–549.
- Lessig, L. (1999c). Open Code and Open Societies: Values of Internet Governance, <http://www.lessig.org/content/articles/works/final.pdf>, 2004-08-24.
- Lessig, L. (2001). *The Future of Ideas. The Fate of the Commons in a Connected World*. New York: Random House.
- MacNeil, M. (1999). Cyberspace Governance: Canadian Reflections, <http://www.admissions.carleton.ca/~mmacneil/118/1999/draft-macneil-cyber.htm>, 2001-02-04.
- McTaggart, C. (1999). Governance Of The Internet's Infrastructure: Network Policy For The Global Public Network. Master's thesis, Faculty of Law, University of Toronto, Canada.
- Mirashi, M. and Brown, S. (2003). The History of the Undernet, <http://www.user-com.undernet.org/documents/uhistory.txt>, 2003-02-26.
- Oberding, J. M. and Norderhaug, T. (1996). A Separate Jurisdiction For Cyberspace? *Journal of Computer Mediated Communication*, 2(1).
- Oikarinen, J. and Reed, D. (1993). Internet Relay Chat Protocol (RFC 1459). Request for Comments (RFC) 1459, Internet Engineering Task Force, Network Working Group.
- Operlist (1992). Operlist mailing list September 1991-February 1992, <ftp://metalab.unc.edu/pub/academic/communications/papers/irc/lists/irclist-operlist.Z>, 2000-03-23.
- Operlist (1993). Operlist mailing list July 1992-February 1993, <ftp://metalab.unc.edu/pub/academic/communications/papers/irc/lists/operlist-archive-12Feb93.Z>, 2000-03-24.

- operlist (1993). Operlist mailing list November 1992-March 1993, <ftp://metalab.unc.edu/pub/academic/communications/papers/irc/lists/operlist-archive-31Mar93.Z>, 2000-03-24.
- Ostrom, E. (1990). *Governing the Commons. The Evolution of Institutions for Collective Action*. Cambridge (UK), New York, Melbourne, Madrid: Cambridge University Press.
- Padlipsky, M. (1982). A Perspective on the ARPANET Reference Model. Request for Comments (RFC) 871, Internet Engineering Task Force.
- Paulsen, V. and Fleckenstein, U. (1997). No Script FAQ: Warum Scripts IRC schaden [german], <http://orgwis.gmd.de/IRC/NoScript.html>, 2003-02-03.
- Perlman, R. (2000). *Interconnections, Second Edition. Bridges, Routers, Switches, and Internetworking Protocols*. Reading (MA): Addison Wesley, 2nd edition edition.
- Pioch, N. (1993). A Short IRC Primer, <http://www.irc.org/docs/primer.txt>, 2002-08-22.
- PJKevin and Dalila (2004). DALnet History. Version 1.1.0, <http://docs.dal.net/docs/history.html>, 2005-01-25.
- PJKevin and LadyDana (2004). NickServ Options. Version 1.1.2, <http://docs.dal.net/docs/nickserv.html>, 2005-01-06.
- PJKevin and Mystro (2004). ChanServ Information. Version 1.1.4, <http://docs.dal.net/docs/chanserv.html>, 2004-12-04.
- PJKevin and quen (2004). Controlling access to your channel. Version 1.1.2, <http://docs.dal.net/docs/csaccess.html>, 2004-08-10.
- Radin, M. J. and Wagner, P. (1999). The Myth of Private Ordering. Rediscovering Legal Realism in Cyberspace, <http://ssrn.com/abstract=162488>, 2001-02-16. *Chicago-Kent Law Review (forthcoming 1999)*.
- Reed, D. P., Saltzer, J. H., and Clark, D. D. (1998). Active Networking and End-To-End Arguments. *IEEE Network*, 12(3):69–71.
- Reid, E. M. (1991). *Electropolis: Communication and Community on Internet Relay Chat*. PhD thesis, University of Melbourne, Department of History.
- Reidenberg, J. R. (1998). Lex Informatica: The Formulation of Information Policy Rules through Technology. *Texas Law Review*, 76(3):533–593.

Bibliography

- Richter, R. and Furubotn, E. G. (1999). *Neue Institutionenökonomik (Institutions and Economic Theory [german])*. Tübingen: J.C.B. Mohr.
- Riedel (2001). EFnet Oper Guide, <http://ftp.ircdhelp.org/helpdocs/hybrid7/operguide.txt>, 2003-01-13.
- Rollo, T. (1992). A Description of the DCC Protocol, <ftp://ircdhelp.org/helpdocs/misc/DCC.doc>, 2004-01-13.
- Rose, H. and Ian (1999). Early IRC history, <http://www.the-project.org/history.html>, 1999-12-17.
- Rosenoer, J. (1997). *CyberLaw. The Law of the Internet*. New York etc.: Springer.
- Saltzer, J., Reed, D., and Clark, D. (1984). End-to-End Arguments in System Design. *ACM Transactions on Computer Systems*, 2(4):277–288.
- Stenberg, D. (1998). History of IRC (Internet Relay Chat), <http://www.fts.frontec.se/dast/irc/history.html>, 2002-06-14.
- Tanenbaum, A. (1989). *Computer Networks*. Englewood Cliffs (NJ): Prentice-Hall, 2nd edition.
- Undernet-CService (1997). Channel Service Frequently Asked Questions, [http://cservice.undernet.org/Channel service/faq.html](http://cservice.undernet.org/Channel%20service/faq.html), 2000-05-20.
- Undernet-CService (1998). The Undernet Channel Service - Guidelines, <http://cservice.undernet.org/docs/guidelines.html>, 2003-06-04.
- Undernet-CService (1999). X and W Commands List, <http://cservice.undernet.org/docs/xwcoms.html>, 2000-05-20.
- Undernet-CService (2002). Channel Service Acceptable Use Policy, <http://cservice.undernet.org/live/regproc/aup.php>, 2005-04-20.
- Undernet-CService (2003). CService OpSchool: Username and Channel Registrations, <http://cservice.undernet.org/docs/opschool1.undernet.txt>, 2003-04-16.
- Undernet-User-Committee (1996). Interview with Jarkko Oikarinen, <http://www.user-com.undernet.org/promotions/jarkko.php>, 2005-04-24.
- Undernet-User-Committee (1997a). CTCP and DCC Protocol Questions, <http://www.undernet.org/user-com/documents/ctcpinfo.txt>, 2003-06-04.

- Undernet-User-Committee (1997b). Interview with Carlo Wood, <http://www.user-com.undernet.org/promotions/carlo.php>, 2005-04-24.
- Undernet-User-Committee (2001). Undernet IRCop FAQ (for Non-IRCops), <http://user-com.undernet.org/documents/operfaq.txt>, 2003-06-04.
- Valauskas (1996). Lex Networkia: Understanding the Internet Community. *First Monday*, (4).
- van Schewick, B. (2004). *Architecture and Innovation. The Role of the End-to-End Arguments in the Original Internet*. PhD thesis, Technical University, Berlin.
- Wu, T. (1999). Application v. Internet – An Introduction to Application-Centered Internet Analysis. *Virginia Law Review*, 85(6):1163–1204.