



Application–level Fault Tolerance in Groupware Using Web Technologies

A proposed solution

Bachelor Degree Project in Computer Science

30 ECTS

Spring term 2014

Adam Ringhede

Supervisor: Jana Rambusch

Examiner: Mikael Berndtsson

Abstract

This work is about achieving fault tolerance in real-time groupware implemented for the web. These systems are often implemented using an architecture that involves a single point of failure. The problem is the lack of solutions that solve this problem that can easily be applied to implementations. The question is whether such a solution can be created and how it would perform. Some research has previously been conducted on this subject; however, the solutions proposed often have significant weaknesses and lack published results on how reliable the solutions actually are. In this work, a new solution is created and presented, which does not include the weaknesses as the previously suggested solutions. An experiment is conducted involving the implementation of a groupware application using common web technologies with the proposed solution applied and running simulations with fault injection as to measure to what degree the solution is able to tolerate faults in different scenarios. The results show that the solution is able to achieve a higher fault tolerance.

Keywords: fault tolerance, web technology, reliability, groupware, collaborative software

Table of Contents

1	Introduction	1
2	Background	2
2.1	Groupware	2
2.1.1	Real-time groupware	2
2.1.2	Implementations and issues	2
2.2	Fault-tolerance for reliability	3
2.2.1	Failure classification and causes	4
2.2.2	Tolerating faults	4
3	Problem	6
3.1	Solving the reliability problem	6
3.2	Hypothesis	7
4	Method	8
4.1	Milestones	8
4.1.1	Create a solution	9
4.1.2	Implementation of application with solution	9
4.1.3	Create a realistic environment for simulations	9
4.1.4	Create software for simulations	9
4.1.5	Gather and Present Data	10
4.2	Ethics	10
4.2.1	Lack of security in implementation	10
4.2.2	Repeating the study	10
5	Execution	11
5.1	Process	11
5.2	The solution	11
5.2.1	Server cluster	12
5.2.2	Group state	12
5.2.3	Fault-detection, Recovery and Reconnection	13
5.3	Implementation of the fault-tolerance mechanisms	15
5.3.1	Server-to-server communication	15
5.3.2	Detecting faults	15
5.3.3	Emitting, Receiving and Backing up group states	15
5.3.4	Putting it all together	16
5.4	Implementation of the groupware system	17
5.4.1	Group states	17
5.4.2	Server	17
5.4.3	Client	18
5.5	An environment for simulations	18
5.6	Implementation of simulation software	18
5.7	Pilot study	19
6	Evaluation	20
6.1	The study	20
6.2	Analysis	20
6.2.1	Interpretation	20
6.2.2	Hypothesis testing	22
6.2.3	Time to reconnect	23

7	Conclusions	24
7.1	Summary	24
7.2	Discussion	24
7.2.1	Comparisons	24
7.2.2	Application	25
7.2.3	Validity	25
7.2.4	Ethics and societal impact	26
7.3	Future work	26
7.3.1	Short-term perspective	26
7.3.2	Long-term perspective	27
7.3.3	Concluding remarks	27
	References	28

1 Introduction

This work explores and propose a solution to a common problem in groupware implemented for the web. Groupware is a type of software with the purpose to assist groups of people to communicate within the group and to collaborate with each other to achieve a common goal in a shared environment (Ellis et al., 1991). Software that allows users to collaborate with each other at the same time and to be updated in real-time by the actions of other users is referred to as real-time groupware. Implementing specifically reliable real-time groupware for the web, where hardware components are bound to fail eventually, can be problematic. The model often used to implement real-time groupware systems poses a problem of single point of failure. If this critical component faults, then the other components will lose the ability to communicate with each other and solving this issue if very important.

Despite many research efforts in the area of fault-tolerance and reliability in distributed systems, not much work has been done which focuses specifically on the tolerance of server-faults in real-time groupware applications (Nastase et al., 2009). Some of the proposed solutions found in research literature only solve parts of the problem, regarding reliability in real-time groupware, and/or has significant flaws in their designs. For these reasons, there is still a need for further exploration and development of potential solutions.

The purpose of this work is to create a new solution using mechanisms for fault tolerance as to support the reliability of groupware implemented using web technologies. To assess how well this solution actually functions, a simple prototypal groupware system using this solution will be implemented. It will use common technologies related to the web such as JavaScript and Node.js on the backend. Moreover, simulations will be carried out to measure how well this system is able to tolerate faults. Faults will be injected in the system to emulate different scenarios under different circumstances. This is to determine how well it is able to handle simultaneous failures meanwhile the system is experiencing different loads. The system will run on a single physical machine using multiple virtual machines to emulate a larger distributed system.

The execution of the project involves creating a new solution, implementing a groupware application, applying the solution, and implementing other software used to perform the experiment. There are a few problems that need to be solved to create the solution. For instance, all clients in a group need to reconnect to another server after a server fault and data shared by clients within a group need to be replicated in a way that achieves redundancy but also consistency. Only when these are solved can an appropriate solution be implemented and applied to a groupware application using web technologies. A program is to be implemented that performs the simulations using the groupware application running on all servers and on up to 200 simulated clients. Moreover, the program collects a few measures for each simulation; one example of such is the amount of clients able to succesfully re-join their group after a server fault. As to asses whether the chosen method is able to produce useful measures, a pilot study is to be performed.

2 Background

This chapter will first describe what groupware is and how it can be implemented using web technologies. Fault-tolerance and related subject will then be described.

2.1 Groupware

Groupware is a type of software with the purpose of assisting groups of people to communicate within their group and to collaborate with each other to achieve a common goal in a shared environment (Ellis et al., 1991). Such applications include, for instance, shared text editors, some multiplayer games, collaborative drawing tools and instant messaging, some of which can be classified as real-time groupware.

2.1.1 Real-time groupware

Groupware as a category of software encompasses many different kinds of systems in which users are able to collaborate. Software that allows users to collaborate with each other at the same time and in real time be updated about the actions of other users is referred to as real-time groupware (Ellis et al., 1991). This type of groupware is the one used in this work.

Data replication is often used in groupware to create a shared workspace where users at different clients share the same data. Data is sent between the users' clients through a network. Any changes to this data should be replicated on the other clients so that all users can see the same data, which for some systems is a requirement for it to function correctly. Some systems may work without workspaces being completely consistent with each other (Xhafa et al., 2012), while others have complete consistency as a requirement. To conclude, users should be able to interact with a workspace and changes should be reflected at the other users' clients; in addition, a consistent shared workspace should be achieved in some applications.

There are several different types of real-time groupware applications, which require different update frequencies to work properly. Card and board games can usually work well with a low update frequency. The same goes for chat and messaging applications. It may not matter if it takes a few extra seconds for the recipients or other group members to receive the message. Shared workspaces where users are able to interact with common objects as in a shared text editor, to see each others mouse pointer, etc. may require more than 25 updates per second to work well. Some games such as first-person-shooters often send updates at a frequency of 20 updates per second. Videoconferencing applications often require an even higher update frequency with a large amount of data per update (Gutwin et al., 2011).

2.1.2 Implementations and issues

A recent addition to regular web browsers is WebSockets, which makes the web browser an appropriate platform for real-time groupware applications. A considerable amount of research has been conducted and it has determined that WebSockets is the best approach to the implementations of real-time groupware applications. The technology satisfies the high update frequency required by some types of real-time groupware applications (see section 2.1.1). Other techniques used for the implementations of groupware on the web are not able to achieve the required update frequency in a real-world environment (Gutwin et al., 2011). Such techniques will now be described.

Commonly used solutions to achieve service availability do not perform well for real-time groupware with a high update frequency. They typically work by using a cluster of two or

more independent web servers running the same service. A load balancer then distributes requests to these servers. If a server stops working, then the requests will go to another server, which is how reliability and service availability is achieved. The only way for clients to communicate with each other is to store data somewhere where all servers can reach it and having the other clients continuously send requests to the servers to see if there is data waiting. The network latency in a real-world environment would limit the update frequency and some applications would not perform well (Gutwin et al., 2011).

Real-time groupware implementations using WebSockets is usually implemented according to a certain model, which may suffer from a single-point of failure. This is how it is implemented in several experimental studies on WebSockets (e.g. Chen & Xu, 2011) despite the well-known issue of single-point of failure. All clients in a group are connected over a network to a single machine running a server often implemented using Node.js. The only way for the clients to communicate with each other is through this server. Any changes to the shared data objects is sent to the server from the client where the change was made and then sent from the server to all the other clients. If this server stops working, then the clients will no longer be able to communicate with each other and the data making up the shared workspace in the group will become inconsistent since no changes can be propagated to the other clients.

Technology that allows the use of peer-to-peer connections or something similar to it does not have the problem of single point of failure when it comes to inter-group communication; however, it may have problems with inconsistency. When all clients have their own copy of data in a groupware implementation, there is a risk that the data will become inconsistent across clients. This may happen if, while a client propagates a change to other clients, it crashes and the change was only sent out to some of the intended recipients. Xhafa et al. (2012) propose using something they refer to as a “single master” approach to deal with inconsistency problems. It works by having one dedicated client holding the data in the group, which they refer to as the master. The other clients can then send updates to the master, which are propagated to the other clients; meanwhile, the master makes sure that there is one consistent copy of the data. Late joining clients can then read the data from the master instead of having to read it from the other clients. This approach still has the problem of single point of failure because if the master stops working, the data is lost and the system fails.

In conclusion, having the clients connected to a single server is a good solution because it makes it possible to avoid inconsistency problems. Moreover, it is easy to implement using existing technologies and it can attain a high enough update frequency in many demanding real-time groupware applications in a real-world environment according to several research studies.

2.2 Fault-tolerance for reliability

Many researchers define the measure of reliability as the probability to successfully be able to finish a task within a time interval (Xiong, 2009). To achieve high reliability in an environment where components will fault, fault-tolerance techniques need to be applied. The *Federal Standard 1037* (1996) of the United States defines a fault as “[a]n accidental condition that causes a functional unit to fail to perform its required function” and as “[a] defect that causes a reproducible or catastrophic malfunction,” where “[a] malfunction is considered reproducible if it occurs consistently under the same circumstances.” It also

defines fault tolerance as “[t]he extent to which a functional unit will continue to operate at a defined performance level even though one or more of its components are malfunctioning.” This section will describe common failures in systems and cover some of the previous research relevant to this area.

2.2.1 Failure classification and causes

Failures and crashes in a system can be divided into several types (Cristian, 1991), some of which will be described here. *Omission* failures are when the service stops responding, A *timing* failure is relevant for systems with real-time requirements; the response may be correct even though it was late. Moreover, this may affect performance negatively, so it is also a *performance* failure. A crash in a system is a type of omission failure; if the machine stops working, then it will not be able to respond. An amnesia-crash implies that, when or if the computer restarts, it loses everything from before the crash and is not able to continue from where it left off. In pause-crash however, the computer will be in the exact same state as it was in before the crash. A halting-crash prohibits the computer from restarting (Cristian, 1991). The failures and crashes described here are the most relevant to this work.

Failures and crashes can happen for several reasons, and they may have to be handled differently. An uncaught exception in the implementation of the software would terminate the process causing an omission failure. All clients and servers connected to the failing servers would drop their connections. The data stored in the server, such as information related to the group, could potentially be lost unless stored in secondary storage. Physical damage such as fires can cause halting crashes. Still, the majority of failures in distributed systems are software related, and attacks such as malware can cause much damage since one infected node can continue to infect the rest of the nodes in the distributed system over the network (Xiong, 2009).

Node.js, which is a common platform for developing real-time groupware applications for the web, has several weaknesses that may cause faults. Since Node.js runs on a single thread, applications programmed with it may be fragile for two main reasons. Throwing an uncaught error, which usually stems from a programming mistake, will result in a termination of the application. Second, since it is single threaded, processing a large dataset will prevent the application from serving client requests (Ojamaa, 2012). This would cause a timing or performance failure. Node.js is not a very secure or reliable platform in comparison to others such as Java and PHP. Still, it has received much attention from developers and its popularity has not declined despite many of its vulnerabilities (Ojamaa, 2012).

2.2.2 Tolerating faults

Components in distributed systems will fail eventually, which can be disastrous in critical systems and an annoyance in others. The reliability of the server in some system architectures for collaborative software is crucial for the application to continue to function, and is an important issue (Fu et al., 2007). The ability to recover from component-faults in any distributed system is a well-known problem, and it is a major problem in distributed groupware systems (Ellis et al., 1991). Therefore, mechanisms to handle server-faults need to be provided (Fu et al., 2007).

Fault-tolerance can be implemented at different levels, which have different implications. Mechanisms implemented directly in the hardware can mask some failures in the hardware, while other failures in the hardware have to be handled at the operating system level. However, the system is still susceptible to faults due to failures at higher levels.

Implementing fault-tolerance at the application level can handle failures both in the hardware and operating system by using redundant software servers. The service state is replicated among these servers so if a failure happens at one server, the other servers may be able to handle the work of that failing server. Implementing fault tolerance at the application level is most beneficial since it is able to mask most types of failures (Cristian, 1991). A solution for fault-tolerance implemented at the application level also does not require any special/modified hardware and/or operating system, and can potentially be used on different platforms (Fu et.al, 2007). For these reasons, the solution proposed in this work is to be implemented at the application level.

The vulnerabilities in Node.js described in the previous section cannot be masked at the operating system level or in the hardware, and therefore needs to be handled using fault-tolerance mechanisms at the application level. For instance, to recover from a termination of the application could possibly be done by having a process restart the application whenever it terminates. Depending on how the application is implemented, a termination could at best be a pause-crash. It is up to the application programmer to create a solution for this.

According to Nastase et al. (2009), no mature implementation exists for a solution able to tolerate faults in this type of system. The proposed solutions found in research literature only solve parts of the problem and/or has considerable flaws in their designs. For instance, Shim & Prakash (1998) proposed a solution only focusing on the issue of client-faults and according to themselves may lack the ability to recreate cross group communication channels after faults. Nastase, et al. (2009) propose another solution to mask faults, which makes parts of the architecture more dependable, but at large still suffers from single point of failure. Some proposed solutions requires a considerable amount of storage space which is used for storing all intragroup communication in order to recreate the state of a group in case of fault (Shim & Prakash, 1998), whereas another, which could be applied to groupware implementations, require redundant hardware components which are never used unless the primary component fail (Nguyen & Liu, 1998). For instance, a solution proposed in Qin & Sun (2001) assumes an infinite storage capacity at client sites. It might still work for some clients with large storage capacities, but platforms with very limited storage capacities such as current mobile devices may not be able to support the storage requirement of applications using this system design. Many of those proposed in research literature has not been sufficiently tested and measured in a real world environment or in simulations, and often lack enough detail to be implemented by others. All in all, there is a clear need for the development of a more mature solution not suffering from these flaws, which is the motivation for this work.

3 Problem

There are many implementations of groupware using web technologies, yet it does not seem to exist any obvious general-purpose solution for fault-tolerance in groupware implemented according to the model described in 2.1.2.

3.1 Solving the reliability problem

The issue is that groupware applications should be able to continue to function in the presence of component faults in the distributed groupware system (Shim & Prakash, 1998). The problem has existed for decades and a perfect solution that easily can be applied to new or existing implementations of groupware for the web does not seem to exist. Despite many research efforts in the area of fault-tolerance and reliability in distributed systems, not much work has been done which focus specifically on the reliability in case of server-fault in real-time groupware applications (Qin & Sun, 2001); therefore, there is still a need for further exploration and development of potential solutions.

The problem is the absence of a good and tested solution. Existing solutions, as described in 2.2.2, either try to create a more reliable architecture or provide mechanisms for redundancy and reconnection to achieve fault-tolerance. One issue many of them share is that they are not rigorously tested in real environments. Some are unfinished, meaning that they still have issues concerning the ability to tolerate faults (eg. Nastase et al., 2009). On the other hand, some solutions that have the potential to achieve a higher reliability have other weaknesses. For instance, they may require additional hardware components and/or a completely different architecture than what normally would have been used to implement the system (eg. Nguyen & Liu, 1998). Moreover, they rely on logging and synchronization to achieve consistency, which could be very time consuming. Also, the solution proposed by Shim and Prakash (1998), which relies on synchronization, may according themselves be unable to support cross group communication after a fault.

In conclusion, a set of qualities and requirements should be achieved to create a good solution. First, the solution should not be based on an architecture that has any single point of failure since this would only move the problem elsewhere. Second, it should not require major changes to the often-used architecture to create groupware implementations for the web as the model in 2.1.2 describe. Furthermore, it should not rely on logging and synchronization since this could be time consuming and worsen the complexity of implementations. The recovery from faults should be as fast as possible so that the user experience suffers less. Implementing the solution in a real system should be inexpensive, which means that it does not require any additional hardware components that otherwise would not be used.

In addition, the solution should be designed for application level failures, for several reasons. One reason for this is because the platform Node.js is especially susceptible to these types of failures as described in 2.2.1. Failures in the hardware can usually be tolerated using other fault tolerance techniques as described in 2.2.2. Moreover, according to Xiong et al. (2009): “most failures occurred in distributed system can be classified as software problems.” The type of application failure focused on in this work is the type of failure that causes the application to terminate, such as an uncaught exception, division by zero, syntactic errors, etc.

3.2 Hypothesis

The question at issue is whether the solution created later in this work will be able to comply with the presented requirements in section 3.1 while providing good mechanisms to achieve some degree of fault tolerance.

The hypothesis is that a solution fulfilling these requirements, as presented in section 3.1, should be able to achieve a higher reliability than otherwise possible in a realistic environment. This hypothesis would be true if, when a server crashes, at least one group that were hosted by that server is able to continue from where it left off with a consistent workspace across all clients and with a functioning channel for communication.

Null hypothesis, H_0 : There is no significant difference in the fault tolerance F (measured in one subtracted by the percentage of groups unable to reconnect) in the occurrence of server faults after applying the solution s in comparison to not having any mechanisms for fault tolerance, denoted as b .

$$H_0: F(b) \approx F(s)$$

Alternative hypothesis, H_1 : The solution s is able to achieve a higher average degree of fault tolerance.

$$H_1: F(b) < F(s)$$

Measure needed: the degree of fault tolerance the solution is able to achieve on average.

4 Method

The method involves creating a solution that is able to conform to the requirements presented in 3.1. Moreover, the solution need to be measured to determine how well it is able to mask failures by tolerating faults and to test the hypothesizes described in 3.2. As stated in 2.2.2, the solutions for fault-tolerant groupware proposed in research often lack methods for measuring reliability. Therefore, a method needs to be developed which is able to evaluate the solution as good as possible with very limited hardware resources.

Research on fault-tolerance and reliability often use mathematical modeling to evaluate the dependability, fault-tolerance, and reliability for it is less expensive and less time-consuming. However, according to Yang et al. (2009), it requires making many assumptions and may not cover all factors that could have an impact on the execution of the system in a real environment.

Building a realistic environment for simulations to take place is challenging without actually having a real environment at hand. The most reliable way to assess the reliability and other qualities of a complex large-scale system may be to perform a case study. This can be performed by monitoring the system under a long period of time running in the actual target environment and track specific attributes. This is nonetheless often economically infeasible and can possibly be very time consuming. Gupta et al. (2011) propose using an accurate scale model of the target environment by using virtual machines. A scale factor is then used to be able to run multiple nodes on a single physical machine yet it is proportionally equal to the actual target system. This is sufficient for finding failures and recovery problems (Gupta et al., 2011), which is the goal of this method.

The method to be used involves conducting an experiment with an implementation of a system using the solution. To run simulations with fault-injection is the method that is going to be used in this study to evaluate the reliability of the solution. This is a method, which is recommended in Zia et al. (2005), for evaluating the dependability of systems. An environment needs to be created, which should be as realistic as possible, in which the system will be executed and evaluated. This is done by using multiple virtual machines scaled down to be able to emulate a realistic environment, as suggested by Gupta et al. (2011).

The simulations will include different scenarios and circumstances to evaluate how well the solution is able to tolerate faults in different scenarios and circumstances. The scenarios are varying numbers of simultaneous server faults. Circumstances include different loads on the system, such as the number of state updates per second, the amount to which the CPU is used in percentage and the amount of running servers in the cluster. The purpose of this is to achieve a high reliability of measures as to avoid making conclusions based on data gathered from simulations running exclusively in very undemanding or demanding circumstances and scenarios.

To summarize, the goal is to analyze a fault tolerance solution for the purpose of determining if the solution is a viable solution with respect to its ability to tolerate faults from the point of view of the author in the context of web application development.

4.1 Milestones

A few milestones need to be reached in this method, which will be described in the following subsections.

4.1.1 Create a solution

The first step is to develop a fault tolerance solution to the problem of server faults in groupware. It should be inexpensive, easily able to be integrated in existing and new groupware implementations for the web without interfering with the architecture or underlying platform and able to provide mechanisms to recover from faults as fast as possible to effectively mask faults. The result of this step is a specification of the solution from which an implementation can be created.

4.1.2 Implementation of application with solution

An implementation of a system using the solution is needed to carry out simulations using it in a realistic environment. The implementation needs to conform to the specification created in the earlier step. The implementation should also use web technologies relevant to this type of application. The programming language should be JavaScript and run on the platform Node.js. It should also be implemented so that it can be used in a web environment, meaning that clients using browsers should be able to use it over a network and communicate with each other in real-time.

4.1.3 Create a realistic environment for simulations

To evaluate the system in a realistic environment, one needs to be created or emulated. Testing distributed software on many physical machines is a difficult problem according to Gupta et al. (2011), who propose using a smaller number of physical machines and instead use several virtual machines running on each physical machine. Some distributed systems may use thousands of nodes (Gupta et al., 2011); however, this is not necessary for this study. Instead, a single physical machine will be used with a few virtual machines running, as to assess whether this solution is applicable even with very limited hardware resources.

Several servers need to run concurrently to emulate the behavior of a real server cluster. Furthermore, each server needs to run with limited hardware resources. The memory should be limited since the amount of memory allocated to the server may have an impact on how many group clones it is able to hold at one time. The server should also have a network latency that is limited to emulate a real system operating over a network. The amount of client requests and communications between clients could also be limited by the performance of the CPU, which mean that each server in needs to have a limited access to the CPU. When one server goes down, the other servers should not receive a higher amount of hardware resources. To accomplish this, several virtual machines are needed to run servers.

4.1.4 Create software for simulations

The execution of the entire system needs to be simulated with running servers and clients. There need to be a considerably large amount of clients, which connect to the servers to achieve a high statistical power for the sake of conclusion validity. Clients need to update the state of the group so that the consistency requirement in the distributed workspace can be tested.

This solution is supposed to be applied to web applications with real-time requirements and a high update frequency. It also sees it as a requirement that the users' workspaces are consistent for the users to be able to achieve their common goal. Therefore, the simulations need to take this into account to produce realistic behavior.

4.1.5 Gather and Present Data

After each simulation, data needs to be collected from the client. The clients do this by writing to a database to record whether or not they were able to reconnect after a fault and whether or not the state of the group was kept consistent despite the fault and recovery. To be able to carry out an analysis of how well the solution functions, the data recorded in the database need to be presented in a format easily readable by humans by using graphs and/or tables.

4.2 Ethics

There are few ethical issues in this method. The only societal ethical issues are regarding the potential security vulnerabilities in the implemented system created for the experiment. However, the purpose of the implementation is not for it to be used in an unsafe environment. Furthermore, the ability to repeat this study is reviewed in subsection 4.2.2.

4.2.1 Lack of security in implementation

Identifying security vulnerabilities in the solution or implementing the solution with security in mind is not subject to this work so there may be a significant lack of security in the final implementation from the step described in 4.1.2. The implementation created in this work is only a prototype and is as simple as needed to be able to evaluate the proposed solution. The entire implementation or parts of it can be used as a basis for further development. However, since no security measures will be taken, data will be stored on servers unencrypted. Data sent between servers will neither be encrypted. Moreover, servers connect to each other but they do not authenticate themselves, which leads to the possibility that any computer able to connect to the port used by servers to listen for other server connection is able to act as a server, which could be bad since it will be able to receive data used to create the shared workspaces of every group currently in the system. These are just some security vulnerabilities that may exist in the implementation. The platform Node.js, which it is implemented for, is considered to be an insecure platform according to Ojaama and Duuna (2012). A considerable amount of work has to be done to make this implementation secure enough to actually be acceptable to use in a production environment connected to the Internet.

4.2.2 Repeating the study

The method should be repeatable as the solution as a concept is described in high enough detail to be able to create an implementation. The implementation of the system using the study the implementation of the software used to simulate the system is described in chapter 5. Furthermore, all the source code created is included in appendices so that an exact copy of the software used in the study can be used to repeat the study.

However, the hardware used in the experiment may affect the result. This means that the exact same results as presented in this study might not be achievable with different hardware. The hardware specifications and operating system used are presented, but recreating the exact same environment with the exact same configurations might be difficult.

5 Execution

This chapter will describe a proposed solution for fault tolerance in groupware and the implementation a groupware system with applied fault tolerance mechanisms. Moreover, the implementation of tools needed to perform the experiment will also be described. The chapter will conclude with a pilot study to show that the measurements collected during the simulations can be used to analyse how well the implementation of the solution performs.

5.1 Process

The process used in the execution of this study has been iterative in almost all parts. Creating the solution described in section 5.2 involved much iteration of developing architectures and mechanisms, finding problems with them and refining them, or starting over. The creation of the components used to achieve fault tolerance iterated between creating physical and component designs, implementing and testing. Implementing the groupware system described in 5.4 was fairly simple, and did not require a lot of iteration. However, integrating the components for redundancy did involve going back to the previous step to fix bugs and making changes. Some suspicious behaviour could not be discovered until simulations were carried out and analysing the data. The issue here was to determine whether the behaviour stemmed from faulty component design, bad choices made in the solution or bugs in the simulation tools. Overall, the implementation has been carried out using a bottom-up approach; however, verification has been done using a mix of top-down and bottom-up integration testing.

5.2 The solution

There are mainly three problems that need to be solved to create a well functioning solution. Clients should be able to reconnect to another server when the server they used crashes. Since clients are only able to communicate with each other through a single server, every group member has to establish a new connection to the same server as the other members in the group. Second, the load from hosting groups previously hosted by the server that crashed need to be distributed over the remaining servers as to prevent a significant amount of clients from connecting to the same server at the same time. The third problem is that workspaces need to be consistent after the fault. In addition, any data specific to the group from the old server need to be replicated on the new server.

A new solution, which has been created by the author of this work, will be described in this section. It is to be implemented at the application level so that it can handle most types of errors in a uniform manner. An existing framework, which purpose is to add real-time groupware functionalities to web applications, has been used a source for requirements so that it will be appropriate to use in practice. The framework has many similarities to the one proposed for future research in Gutwin et al. (2011) and would be appropriate to use for many different real-time groupware applications. It avoids consistency problems by using a model similar to the single-master approach as described in section 2.1.2. For this reasons and because it is used in a significant amount of existing implementations and research (Chen et al., 2011; Gutwin et al., 2011), it is the model for which the solution in this work is created to support.

The solution created and chosen for this work is beneficial for two reasons in addition to those explained later in this section. First, it is easy to implement, for instance, it does not

require any synchronization mechanics to achieve consistency. The second reason is that it can potentially be used by existing implementations of groupware applications and frameworks as a separate software component or a concurrent process, which is illustrated in upcoming sections. It is a general-purpose solution and does not require making considerable changes to existing software and would therefore be easy and inexpensive to implement.

A client-server architecture using WebSockets as the model described in 2.1.2 is the basis for the solution. A server hosts several groups and each group has a unique *groupID* used to identify each group. Clients can connect to a server to join a specific group by using a *groupID* or send a request to start a new group, which others can later join. The solution does not interfere with how groups are created. For instance, a multiplayer game may be using a matchmaking service on the server, which is able to create groups and put users in groups.

5.2.1 Server cluster

The solution achieves availability and reliability by having redundant software servers. One server is a process, handling client requests and includes mechanisms to achieve fault-tolerance. One computer may run one or more servers at once. Every server is then connected to every other server, so that all servers in the system are aware of each other and able to backup the states of the other servers. There exists one two-directional TCP connection between all servers. When a server starts up, it creates a connection to every other server. The other servers are thereby aware of the new server and can start using it for backup and other services.

Adding more computers to the system should increase the amount of clients or groups it can serve at one time. This is referred to as horizontal scalability (Michael, 2007). Using redundant servers that will never be used unless one server fails is expensive. If the number of clients being served at one time were related to the amount of revenue created, then this would have an impact on the economical feasibility of the system. For these reasons, horizontal scalability was taken into account when creating this solution since it would increase the amount of clients served at one time as well as adding additional computers could increase the reliability and availability of the system.

5.2.2 Group state

State is used as a concept used to describe the current state of a group and the state or current value of the shared data such as documents (Shim & Prakash, 1998). The current state of the group can for instance include information about the members of the group. Some groupware systems refer to the state of the shared data as a document (Xhafa et al., 2012), which would be appropriate for document sharing groupware applications. This solution is however supposed to be appropriate for many types of groupware applications so a more abstract term is selected. Propagating changes of the state is one type of communication. Groupware systems using this type of construct is common and is said to use stateful group communication (Shim & Prakash, 1998). Late joining group members, i.e. those who was not members from the start or was disconnected and need to reconnect, can create an environment consistent with the other members' environments by downloading this state from a server. Therefore, this state has to include all needed information to be able to recreate an environment.

The server hosting the group holds this state as an object or a collection of objects, which should be implemented in a way that decrease the risk of conflicting updates during

concurrent updates and only requires the updated parts of the state to be sent over the network during updates. The data should preferably also be in a format that can be effectively parsed and read on both the client and the server. The solution however does not put any restrictions on how the data is structured, as long as the requirements presented can be achieved.

Group states are replicated across all servers in the clusters, which is key for the recovery mechanism described in 3.2.3 to work. Each server is responsible for broadcasting the state to all the other servers in the cluster. When a change is made to the state, then this change is broadcasted to keep the clones of group states on the other servers up to date.

Stateful group communication, as opposed to stateless, is deliberately chosen for this solution to easily be able to achieve fault tolerance. States are something that easily can be replicated among servers within the cluster and clients are able to recreate the shared environment even when connected to another server after the earlier server crashed. Stateless group communication would have to store logs of all broadcasts made since the group was created and after a fault rebroadcast these to be able to achieve the same consistent state as before the fault (Shim & Prakash, 1998), whereas a state like a snapshot of the shared environment would not require as large storage capacities and can simply be read from the server when needed.

5.2.3 Fault-detection, Recovery and Reconnection

The first step to tolerating faults in this solution is to detect them and then consequently recover from them. Fault-detection should have a low overhead and be fast to work in a system with real-time requirements (Gillen et al., 2007). When a server disconnects from another server, it tries to reestablish the connection. If it is unable to reconnect, it determines that other server has faulted and notifies every other server about it.

Since the process of fault-detection requires some decision making, one server is only monitored by one other to avoid conflicting decisions. Which server monitors which server is decided by using a unique number each server has. For instance, the server with number 2 monitors the server with number 1, which monitors the server with number 0, which then monitors the server with the highest number.

A number system is used in the distributed system where all servers receive a unique number in a varying range. The numbers range from 0 to the number of servers currently in the system minus one. All servers are aware of each other's numbers which is a necessity for decision making in some mechanisms. After the server has started up, it determines its own number either by the server with the highest number and adding one to it, or by counting the number of servers running except itself and use that number. If it is unable to connect to any other server, i.e. it is the only server running, it assigns the number zero to itself. If a server fault occurs, then the server with the highest number replace its own number with the number previously used by the server failing. Consequently, as these numbers change when a servers in the system faults, everyone should have some other server monitoring it.

When it comes to recovery mechanisms in fault-tolerant systems, there are two different techniques; these are *forward error recovery* and *backward error recovery*. This system uses *forward error recovery* which according to Zia et al. (2005): "attempts to construct a coherent, error-free system state by applying corrective actions to the current, erroneous

state.” The recovery mechanism in this proposed solution does this by sharing the work, previously done by the faulting server, across all other error-free servers.

The goal is to make recovery as fast as possible by making the recovery mechanism able to finish as fast as possible by making them able to execute independently of other nodes in the system. If servers have to communicate with each other more than usual during the recovery in order to coordinate the effort in some way, then not only would it increase the network load but could potentially be a huge bottleneck. Designating one server to making the decisions during recovery could decrease the amount of communication needed; however, that would make it a single point of failure and make it an unreliable solution. The recovery mechanism works independently by looking through all cloned group states it has backed up and testing whether the group’s unique id **modulo** the total number of remaining servers equals the server’s own unique number. If it does, then the server starts hosting that group using the information stored in the cloned group state object. This is illustrated in pseudo code in figure 1. The argument *serverId* is a unique identifier of the server that faulted.

```
function recover(serverId)
    numServers := cluster.count()
    for each clone in backup.getGroupClonesByServer(serverId)
        targetHostNumber := clone.id % numServers
        if targetHostNumber = self.number
            self.startHostingGroup(clone)
        else
            backup.addGroupToServer(
                cluster.getByNumber(targetHostNumber).id,
                clone)
    backup.removeGroupsByServer(serverId)
```

Figure 1 Pseudo code describing how the recovery mechanism can be implemented

In the case that another server in the system faults within a short time from the first server-fault, there may not be enough time to receive new versions of the group states from the second faulting server. Therefore old group state clones are reused as depicted after the *else*-statement in figure 1.

Clients use the same fault-detection approach as the server to decide whether or not the server it is connected to has faulted. If it decides that it has faulted, it tries to connect to any other server in the cluster until it finds one it is able to connect to. If this server now hosts the group the client is in, then the client stays connected to this server and the other group members do the same thing. If the server does not host the group, then the server responds to the client with the IP-address of the server hosting the group, which the client can use to connect to the right server.

To conclude, there are three steps to mask the faults. First, the other servers detect the fault, which is followed by the next step is to recover from it. Third, they assist the clients to find the right server that hosts their group. The issue is to see whether or not this is a sufficient solution to the problem of reliability.

5.3 Implementation of the fault-tolerance mechanisms

The implementation of the solution is a separate component written in JavaScript to be run using Node.js. The source code and a class diagram can be found in Appendix A.

5.3.1 Server-to-server communication

One aspect of the solution is the direct communication between servers. This is implemented with client-server relationships between servers. As to avoid confusion, servers will from here on be referred to as nodes when they are discussed in relation to one another. Since there only need to be one duplex connection between two nodes, a component called *NodeConnectionManager* has been implemented, which takes care of all connecting between nodes. When a connection to another node is made, an instance of *NodeSocket* is created. Nodes can then communicate with each other using *NodeSocket* instances by emitting custom events and binding event listeners as functions to events.

Nodes can connect to each other using the method *createConnection* in *NodeConnectionManager* by passing the id of the node it wishes to connect to. This method then uses a local configurations file to look up the other node's IP-address and port used for node-to-node communication. In the case that a connection between the two nodes already exists, it reuses the existing connection, as opposed to creating a new one. If a new connection is created, the event *newConnection* is triggered. This event is also triggered if the node receives a connection from another node. This event is often used by different components to bind functions to certain events on *NodeSocket* instances.

5.3.2 Detecting faults

A component named *FaultDetector* is used to monitor another node. This component only monitors one node at one time. It does this by having the method *monitor* called, which binds a listener to the disconnection event triggered by the *NodeSocket* passed as an argument. When the disconnection event is triggered, it attempts to re-establish the connection to the node. If this is not possible, then it executes a callback function attached as an attribute to the fault detector.

5.3.3 Emitting, Receiving and Backing up group states

The recovery mechanism requires group states from the other nodes to be able to host the groups of the failing node, which are stored using a component called *Backup*. It stores groups within an object being used as a hash map for fast access. Since the node may only receive a part of a state during an update, as described in 5.2.2, there is a method called *changeState*, which changes a part of the state. This method takes a string to describe where in the state to write the change in addition to the updated data.

As to achieve redundancy, the component *GroupStatesEmitter* emits the changes of group states to other nodes by emitting the event named *stateUpdate* to one or all nodes. The backup is then populated by the component *GroupStatesReceiver*. Its only purpose is to listen for state updates on every existing *NodeSocket* instance. As soon as the receiver receives an update, it either creates a new group for the server it received the state from in the backup or, if it already exists, changes the state of the group in the backup.

5.3.4 Putting it all together

The one component that puts it all together is called *RedundancyController*. It instantiates the previously mentioned components, connects to the other nodes, selects another node to monitor, recovers from faults, provides an interface to the groupware implementation using it and sets the value of the nodes' number. This subsection will describe how some of these things have been implemented.

First, it is mentioned in 5.2.3 that each server has a unique number ranging from 0 to the number of nodes in the system minus one. In this implementation, it is referred to as a *Temporary and Unique Node Number* or the abbreviation TUNN. When the *RedundancyController* is instantiated, it creates a connection to every other node in the system. It determines its own TUNN value by the number of connections it creates. It broadcasts its TUNN value whenever it changes to the other nodes so that all nodes are aware of each other's TUNN. Also, when a node fault occurs, several logical statements are tested to find what value the local node should assign to itself. For instance, if the local node has the highest TUNN value before the fault, it will take on the TUNN value of the node where the fault occurred. Some of this logic is also used for determining which other node to monitor.

The *RedundancyController* also responds to server faults by executing a recovery function. Its implementation can be found in appendix A. It is implemented similar to the pseudo code describing the recovery mechanism in figure 1. During the instantiation of the component, a function called a host function is passed as a parameter by the groupware implementation. Recovery can using this function be carried out independently of how groups are hosted in the implementation of the groupware backend. The states of groups formerly hosted by the faulting server that are not to be hosted by the local node are moved in the backup so that they can be reused in the case of a second fault. Not reusing these group state clones could be very demanding on the network after a server fault since they otherwise would have to be reemitted.

It was discovered during testing that the nodes sometimes failed to reuse some groups during recovery. This was because the recovery mechanism is run before the node has received the new TUNN value of the node that before the fault had the highest TUNN. Determining the TUNN value of the node with the highest locally before the recovery solved this. Another option would be to postpone the recovery until the other node has emitted its TUNN. This could, however, postpone the recovery indefinitely in case the node with the highest TUNN crashes before it is able to emit the value.

The component's interface consists of two methods called *update* and *findGroupLocation*, which are to be used by the groupware backend. When a change occurs in the state of a group, this change needs to be propagated to the other nodes for achieving redundancy. The *update*-method should be called with the group's ID and data concerning the update. This is then used by the *GroupStatesEmitter*, which is part of the *RedundancyController*, to emit this update to the other nodes. The second method, *findGroupLocation*, returns the IP address and port clients should connect to of the node hosting a certain group if the group is hosted by any of the other nodes. This method should be called during the reconnection phase after a server fault.

5.4 Implementation of the groupware system

This section describes the main components of a minimal groupware system or framework. It also shows how the fault tolerance mechanisms from 5.3 are applied to the implementation.

5.4.1 Group states

A group state in this implementation is represented by a JSON structure. In its parsed form it is a JavaScript object, which easily can be read and written to since both the backend and frontend is implemented in JavaScript. Moreover, the hierarchical and flexible structure of a JavaScript object makes it possible to create and change parts of the state during runtime without affecting other parts of it during concurrent updates. An alternative would be to instead use XML, which is used in Xhafa et al. (2012) in their implementation of document representation and replication in a collaborative system.

Changing the state is performed by using a specific function so that the change can be propagated to other clients and between servers. This function takes two arguments. The first one being a string to describe where in the state to make the change and the second one should be the thing to add in that place. This string looks like and is inspired by paths in file systems used in operating systems. Figure 2 includes some examples of how this function may be used in different kinds of groupware applications. Since JavaScript objects work like hash maps, reading and writing to them has a time complexity of $O(1)$. A state, however, could include a tree of objects. This means that using a path to change a part of a state would give a time complexity of $O(n)$ where n is the number of steps in a path.

```
changeState("sheets/ExpendituresFall09/B/34/value", 325.50)
changeState("players/JohnDoe/position", { x: 43, y: 5 })
```

Figure 2 Examples of how to change a state

The idea of representing and changing states like this, as well as the implementation of necessary functions, comes from a prototypical groupware framework, created by the author of this work (Ringhede, 2014).

5.4.2 Server

The server acts as the host for groups and holds connections to clients. It is implemented according to the model described in 2.1.2 and is the single point of failure. For the sake of this study, the implementation has been kept minimal and does not contain many of the functionalities one might need from this type of system. Still, clients are able to find groups to join and communicate with each other by updating their shared workspace, or state. Its complete implementation can be found in appendix B.

Clients connected to the server can be added to a group hosted by the server if one is available or join a queue and be added to a new group together with another client as soon as another client asks to join any group. This is just a way to create groups automatically for the purpose of the experiment.

The component RedundancyController described in 5.3.4 is a part of the server. Figure 3 illustrates how the RedundancyController is integrated in the server and how redundancy is created by sending updates the server receives from clients to the RedundancyController, which in turn broadcasts each update to other servers.

```
client.bind('updateState', function (data) {
  client.group.changeState(data.path, data.obj)
  client.group.broadcast('stateChanged', data);
  self.rc.update(client.group.id, data.path, data.obj);
});
```

Figure 3 The integration of the RedundancyController in the server implementation

5.4.3 Client

The client is responsible for the reconnection after a fault. For the purpose of ease of integration in existing solutions, the idea was to perform reconnections using a separate component. However, during the implementation of this, it was discovered that such a component required a lot of configuration to be reusable in many different implementations of clients in groupware implementations on the web. Instead, the reconnection parts are built into the client implementation. If the connection between the client and the server closes, the client tries to reconnect to the same server. If this fails, it connects to a random online server and asks to join the group with the id of the group it was previously a part of. However, if the server is unable to find this group on the server, then the server asks the RedundancyController for the location of the group in the server cluster and emits the IP address and port that the client can use to connect to the correct server. When the client receives this information, it connects to the correct server, joins the group, and triggers an event called *rejoinedGroup*.

5.5 An environment for simulations

An environment has been created using four virtual machines hosted by one physical computer. The computer acting as the host is made up of an ASUS P8Z68-V PRO/GEN3 motherboard, an Intel Core i7-2600K CPU and 16 GB of RAM running Windows 7 Ultimate. The virtual machines are each dedicated 512 MB of RAM, 2GB of secondary storage, 1 processor core and run Debian 7 64-bit. Node.js has been installed on all virtual machines and each machine has their own copy of the server code. Each server has a static IP address and can be accessed within a LAN. Another physical computer is used for clients and is able to communicate with the host computer and the virtual machines it hosts over Wi-Fi.

5.6 Implementation of simulation software

A program executed within a browser runs simulation by creating and simulating clients and controlling the state of servers. This program is able to both start a server on each virtual machine and inject faults by communicating with a running program on each virtual machine. The program on the virtual machines is able, on request, to start a server by spawning a child process. The simulation program can also request this program to inject a fault, which it does by killing the child process using *SIGKILL*. The implementation of the program can be found in appendix C.

Each simulation involves starting a number of servers, creating a number of clients and killing a subset of these servers. 50 clients are created for each server started. For example, if a simulation involves 3 servers, then 150 clients will be created and simulated. However, the clients are not distributed evenly across the servers. The client is implemented to connect to a random server unless otherwise instructed for the purpose of load balancing. Since only 50

clients are created for each server, it is possible that the distribution of clients becomes significantly uneven. However, one might assume that it would be improbable that all servers in a groupware system would have the exact same number of clients connected to them.

Moreover, the simulation program collects data from the simulations. It does this by attaching functions to certain events such as *disconnectedFromGroup*, *rejoiningGroupFailed* and *rejoinedGroup*. The first one is important because it is able to tell which clients were affected by a server fault since there are clients connected to servers during the simulation that will not fault. The last one, which is *rejoinedGroup*, is triggered by those clients able to reconnect to the right server after a fault they were affected by and join the group they were previously in before the fault.

While testing by running simulations, it was discovered that starting several servers at the same time caused all servers to assign the same TUNN value to them selves. However, waiting as little as 100ms between starting each server solved this.

5.7 Pilot study

Data collected by the simulation program described in 5.6 has been used to create the diagram in figure 4. The data is collected from 10 simulations for each scenario. Each column represents the mean percentage of clients affected by a fault that are able to re-join their group after the fault. For instance, the second column from the left represents a scenario in which one out of three servers faulted. The mean of the data collected from simulations in this scenario indicate that 88% of the clients affected by the fault were able to re-join their group.

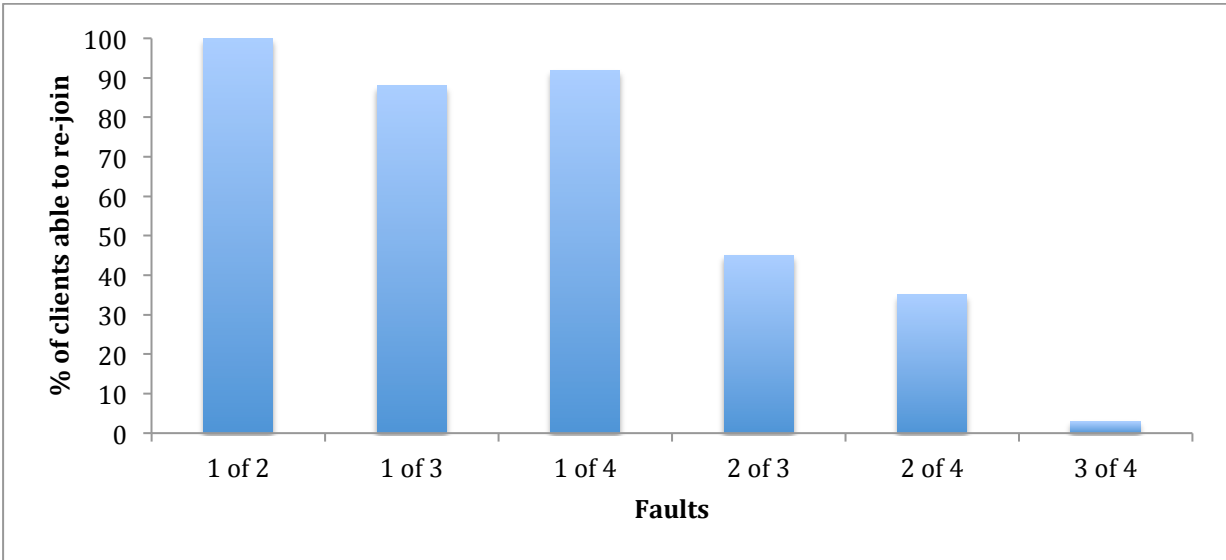


Figure 4 The average percentage of successful re-joins by clients affected by a fault in different scenarios.

In conclusion, more simulations need to be performed under different circumstances as to create more reliable data, from which one can draw conclusions. At this state, it is also unclear if for instance the amount of clients used during simulations negatively affects the results.

6 Evaluation

This chapter will start by describing the conducted study, which is followed by an analysis of the data set coming from the experiment. The chapter concludes with a summary of the conclusions made in the analysis.

6.1 The study

An experiment has been conducted running simulations and data from these have been collected to be able to analyse the implementation of the proposed solution. A total of 2520 simulations have been performed, simulating several different scenarios under different circumstances. These scenarios include varying amounts of simultaneous server faults and the circumstances include the load on servers, the amount of simulated clients and the amount of servers running in the system. Different loads on servers are divided into two cases: low and high, which reach average CPU usages at about 10% and 60% respectively. The scenarios, i.e. the amount of simultaneous server faults, and the varying number of servers in the system result in six different cases being, for example, 1 failing server out of 2, 2 failing servers out of 3, etc. These are abbreviated to 1 of 2, 2 of 3, etc. in this chapter. 400 simulations have been performed for each of these cases with a low and high load. The amount of clients is mostly held constant; however, one test case involve a significantly less amount of clients as to verify whether the amount of clients could have a meaningful affect on the results from the simulations.

The data collected from simulations include several measurements. These are the amount of clients directly affected by the fault, the amount of clients able to join their group on another server after the server fault, the amount of clients that were in a group during the simulation and the amount of clients that were unable to join their group after being redirected to another server. All these measurements for every simulation can found in appendix D.

6.2 Analysis

The analysis in this section will start with the interpretation of data collected from simulations. In addition, it will end with a generalization of the average time needed for a client to successfully reunite with its group.

6.2.1 Interpretation

This section focuses on exploring the distribution of a single calculated variable from the data set. Since this work is about measuring how well fault tolerance is achieved using the solution, the most important measures are the amount of clients actually affected by a server fault and the subset of these that were able to tolerate the fault. Therefore, the variable of concern is the result of the division of the amount of clients able to join their group after a fault on another server over the amount of clients affected by a fault from one simulation. This fraction or percentage will be referred to as the fault tolerance.

Figure 5 shows the difference in results when the servers are experiencing different loads. Simulations with a high load result in a slightly lower fault tolerance. Still, both cases show a similar pattern. For instance, there is a distinct difference in case 1 of 3 and 1 of 4 in comparison to 1 of 2, which exist with both loads. Since the amount of simulated clients rise as the amount of servers in the clusters rises, it was hypothesized that the amount of

simulated clients could be the cause of this difference. A new sample was produced from running simulations with a much lower amount of clients. This was used to create the chart in figure 6, which shows that simulating fewer clients does not have a significant affect on the result.

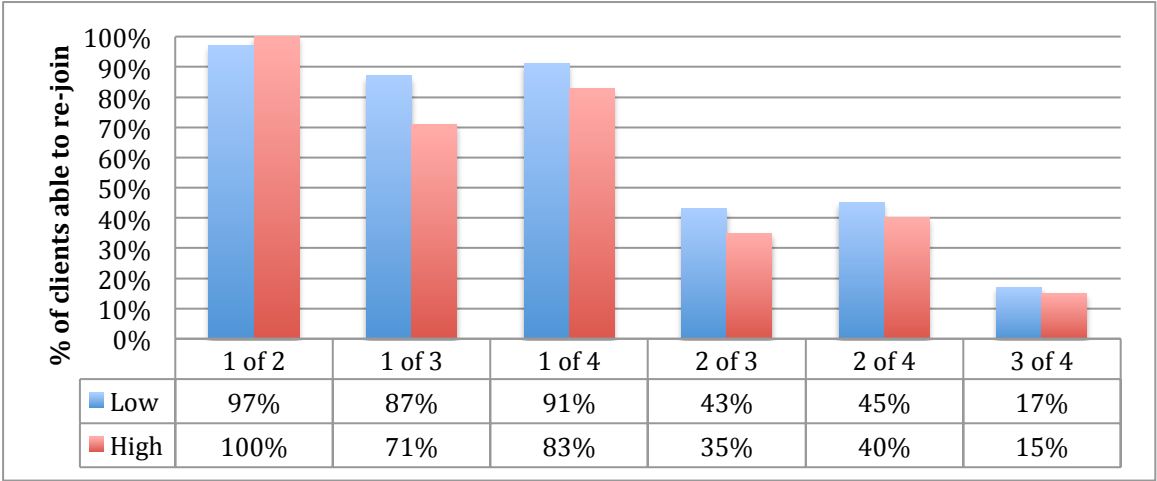


Figure 5 The mean percentage of successful re-joins by clients affected by a fault in different scenarios with low and high load

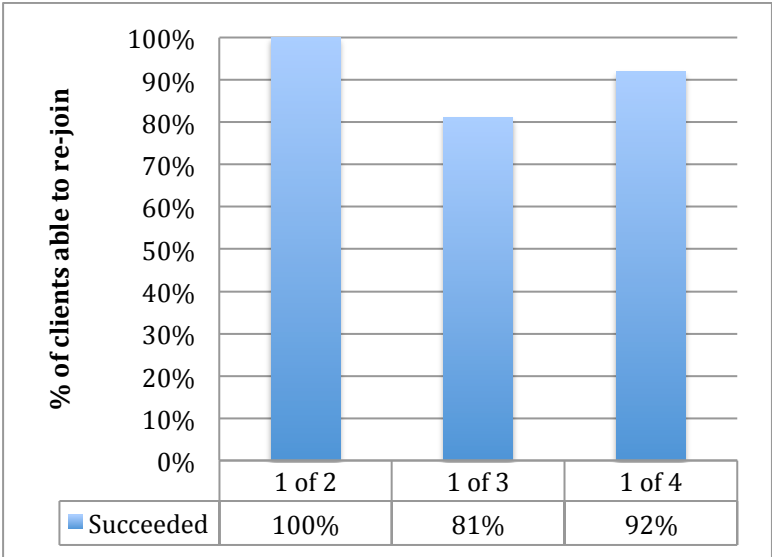


Figure 6 The mean percentage of successful re-joins by clients affected by a fault in different scenarios with few clients per simulation.

Furthermore, figure 5 shows that there is a great difference in fault tolerance when the amount of simultaneous faults increases. The difference in average fault tolerance is less than half in scenarios with an additional simultaneous server fault and is true in all three cases. However, it is not clear whether a larger amount of servers could benefit the fault tolerance.

Figure 5 indicates that there is an increase in fault tolerance between 3 and 4 servers with the same number of simultaneous server faults. If this continues to increase with many more servers, then additional simultaneous server faults may have a significantly less negative effect than what they have in the scenarios simulated in this study.

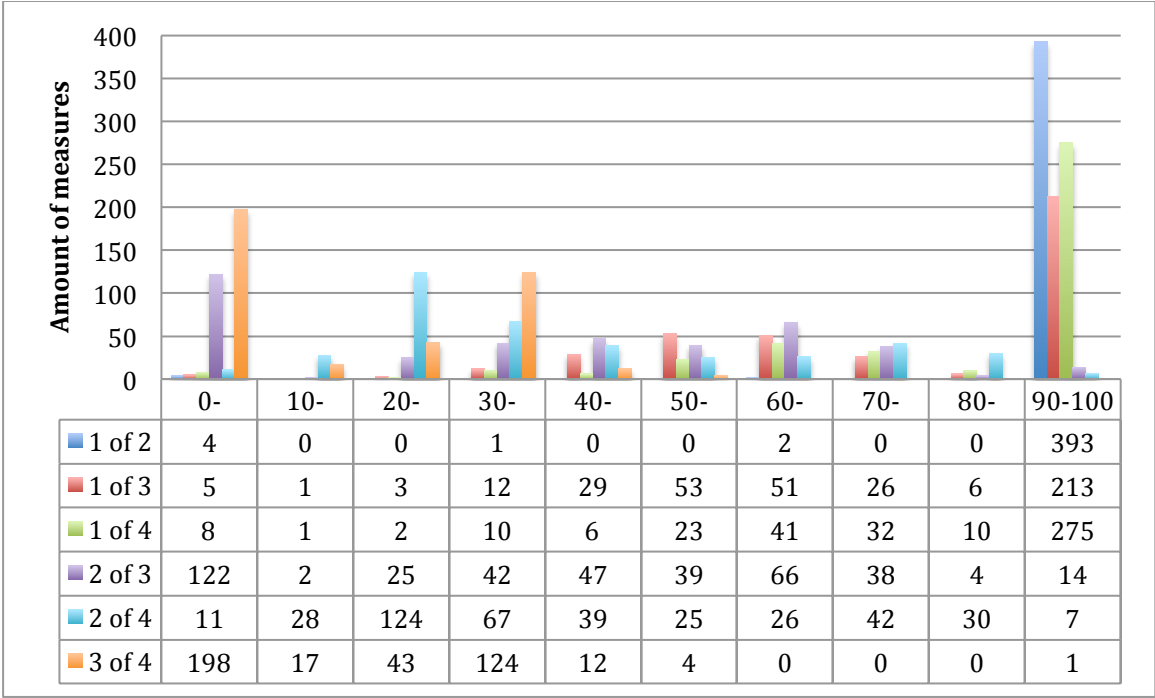


Figure 7 The frequency of simulation results in different scenarios.

Figure 7 shows how the measures from the simulations are distributed and indicates the existence of a few outliers. The simulation results are distributed into intervals ranging from 0 to 9, 10 to 19 and so on. One of the most obvious outlier is the one in 3 of 4 in the interval between 90 and 100. The origin of this one might be that the requests to kill servers, sent over the network, arrived at different points in time, which resulted in less than 3 simultaneous faults and therefore an abnormal measure. If this is the case, then other simulations involving more than one simultaneous fault may show good measures that are incorrect. Some of the measures in the lower intervals in scenarios involving a single server fault may be the result of bugs either in the simulation tools, the groupware implementation or the implementation of the components of the solution. However, these do not significantly impact the interpretation of the data, nor the conclusions. In addition, looking at the table in figure 7, measures tend to centre on more than one point in some scenarios. For example, they centre on about 55 and 100 in 1 of 3, 40 and 100 on 1 of 4, 0 and 60 in 2 of 3, 20 and 70 in 2 of 4, and on 0 and 30 in 3 of 4.

6.2.2 Hypothesis testing

There is no random chance that a client could reconnect to its group without any fault tolerance mechanisms and would have an exact fault tolerance of zero. Therefore, the null hypothesis can be rejected as long as the fault tolerance achieved using the solution is anything but zero. Only about 14% of all simulations resulted in a fault tolerance measure of exactly 0%, 84% did not. The null hypothesis can therefore be rejected. The average fault tolerance in the worst-case scenario, according to figure 5, is 15%, which is greater than what

could be achieved without having any fault tolerance solution; therefore, the alternative hypothesis is supported.

6.2.3 Time to reconnect

This work is mainly centred on achieving a high degree of fault tolerance. However, other research on reliability in real-time collaborative systems (e.g. Qin & Sun, 2001) focuses on performance in regards to the time to recover from faults. Also, one part of the problem was to invent a solution that causes as little stalling as possible between being disconnected due to a server fault and being reconnected to another server and join the same group. This subsection presents equations to illustrate the average amount of connections required and the total average time.

The probability of not connecting to the server hosting the group that the client should belong to on the first attempt is $1 - 1/n$, where n is the number of servers still running after one or more server faults. In addition, there has to be at least one attempt, which results in the equation below.

$$reconnections = 2 - \frac{1}{n}$$

The time to connect to a server and requesting to join a group if the server hosts it is represented by t . The average time for a client to reunite with its group can then be calculated with equation below.

$$time\ to\ join\ group = 2t - \frac{t}{n}$$

In a scenario where 2 out of 4 servers experience a fault, about 40%, on average according to figure 5, of the clients affected will be able to reunite with their group on one of the two remaining servers. If the time to connect to a server and attempt to join a specific group is 60ms, then the average time for the clients to successfully reunite with their group becomes 90ms. However, this does not include the time required to retrieve the state data from the server, which is relative to the network throughput s and the amount of data d that need to be retrieved.

$$time\ to\ recover = 2t - \frac{t}{n} + \frac{d}{s}$$

In conclusion, the amount of client-server connections required by a client to reunite with its group after being affected by a fault will never be more than 2, even in very large server clusters. This may be of concern in future work involving performance comparisons between other solutions and this one in regards the time to recover.

7 Conclusions

This chapter includes a summary of this work from problem definition to results followed by conclusions concerning subjects such as the application of the work in web development, validity, societal impact and future work.

7.1 Summary

The problem lies in the absence of available and inexpensive solutions for achieving fault tolerance in real-time groupware applications using web technologies. The question at issue is therefore if it is possible using existing and widely supported web technologies to create and implement such a solution. Moreover, it should be possible to apply this solution to future implementations of collaborative software applications such as real-time groupware applications and potentially also to existing implementations.

The project has involved creating such a solution, which then has been implemented using JavaScript to run both on the client in web browsers and on servers using Node.js. It has then been integrated into the implementation of a groupware application. The fault tolerance has then been measured by simulating the execution of the groupware application using an emulated environment consisting of two physical machines. One of the physical machines uses multiple virtual machines to run up to 4 servers, while the other one is used to simulate clients. Faults have been injected into the server cluster and data has been collected from the clients regarding their ability to tolerate and mask one or more simultaneous server faults.

The results show that the implementation of the solution is able to tolerate and mask faults to a degree of about 70% to 100% of the clients on average in scenarios involving only one fault. However, two or more simultaneous faults result in a much lower average fault tolerance.

7.2 Discussion

This section discusses comparisons between the results of this work with previous work, its application in web development, the validity of the study, and ethical factors that have not already been discussed in earlier chapters.

7.2.1 Comparisons

The proposed solution can be compared to some of the solutions discussed in 2.2.2. For instance, the solution does not require any additional nodes, nor does it add a new single point of failure as the solution proposed by Nastase et al. (2009). Moreover, it does not require downloading states from other clients in order to achieve a consistent shared state after a fault such as the one proposed by Shim and Prakash (1998). Instead, clients download the state from a centralized server, which achieves a greater performance in regards to the time to recover after a fault according to Qin and Sun (2001).

No measurements have been published from the previously mentioned studies on how well they actually perform when it comes to tolerating faults. Therefore, it is not possible to make comparisons of the results or the conclusions.

7.2.2 Application

This work can be used for two purposes in web development. It can be used for decision-making concerning the architecture and expected reliability in future implementations of new groupware systems for the web and possibly also for the application in existing groupware implementations as a way to improve fault tolerance. Moreover, the implementation created in this work is shown to work and could therefore be used directly in groupware implementations in web development. However, adaptations to the implementation of the solutions may have to be performed and security vulnerabilities may have to be identified and considered.

Considering that reliability is defined as the probability to successfully be able to finish a task within a time interval, one can create an estimate of the reliability achievable of using this solution from the measures presented in chapter 6. The reliability can be calculated by taking one minus the sum of all the probabilities P that each scenario s would occur within a time interval multiplied with one minus the average fault tolerance F achievable in the same scenario multiplied by the probability A that it would affect a client or group. This is illustrated in the equation below.

$$1 - \sum_{i=1}^n P(s_i)(1 - (F(s_i))A(s_i))$$

7.2.3 Validaty

This subsection explores the validity of the study using two validity categories suggested by Cook and Campbell (1986) that are relevant to this study. These include conclusion validity and external validity.

A few things can be discussed regarding the conclusion validity of this study. For instance, is the statistical power high enough? Each scenario has been been simulated 420 times. More simulations could potentially create more accurate average measures. However, the most important aspects of the data may be the pattern, which looks similar even with as few as 20 simulations as shown in figure 6. Also, the amount of collected measures from the experiment is enough to test the hypothesis. Moreover, since the creator of the solution is the same person that conducted the experiment, there may be a credibility issue of fishing for better results. However, all the raw data collected from simulations are included in an appendix and so is all the source code created for the implementations, which could be used to investigate the correctness of the results. Another point regarding the conclusion validity is the reliability of measures. A few outliers have been detected and pointed out in the analysis. However, datasets from simulations of some scenarios are very dispersed as illustrated in figure 7, which makes it difficult to differentiate between anomalies and correct measures. Furthermore, the experimental setting is the same in all simulations; still, some random irrelevancies in the experimental setting may exist since the simulations are carried out over a shared network, which may have caused disturbances that resulted in some anomalies. However, this might have made the environment even more realistic.

The environment in which the simulations are carried out may not be sufficiently realistic, which could question the external validity. However, as stated by Gupta et al. (2011), scaling down a distributed system and running multiple nodes on one physical machine is sufficient for exploring recovery failures. Moreover, the faults are only injected in one way. Still, it has

been concluded, during the execution of the experiment that other failures that may occur on a virtual and physical machine causing amnesia crashes would induce the same behaviour.

7.2.4 Ethics and societal impact

In addition to the ethical aspects discussed in the methods, another one concerning the fault tolerance can be discussed. Even though the proposed solution in the work is a fault tolerance solution, which is designed to be implemented at the application level as to mask almost all types of failures on a server, it is not completely fault tolerant in all scenarios as the analysis in chapter 6 shows. This means that using the solution in practice can be damaging if one expect it to tolerate all faults at all times.

As for societal benefits, many groupware applications implemented for the web could potentially benefit from the application of the proposed solution. The solution is also designed to be cheap to implement and does not require any additional hardware components, which means that small organizations or independent developers with very limited resources could create more reliable groupware applications for the web, even though the solution does not tolerate faults completely in all cases.

7.3 Future work

There are a few things that could be accomplished to improve on this work. For instance, making improvements to the solution and the implementation may have a significant effect on the fault tolerance. Moreover, the solution could be applied in a larger project in a more realistic environment and with more accurate instrumentation as to create more reliable measures. Others could perform work that build on this study by inventing and testing other solutions using a similar method as to more accurately compare aspects of the solutions such as fault tolerance and scalability.

7.3.1 Short-term perspective

Some decisions have been made in the creation of the solution for the implementation to function more reliably in real environment where, for example, network failures may occur. One example on this in the solution is that each server monitors one other server, instead of all servers monitoring every other server as to avoid making different decision on whether or not the server has actually failed. It is very likely that this is what causes the lack of fault tolerance in scenarios where two or more faults occur simultaneously. A hypothesis could then be that the solution should be able to achieve similar fault tolerance despite the number of simultaneous faults if all servers monitor every other server. Making minor changes to the implementation and running simulations and taking measurements using the existing tools created for this study could test this.

Moreover, further improvements could be made to the implementation. This could, for example, involve easier reusability and more flexibility to representations of states. Improvements could also be made to, for instance, the reconnection; at this time, the client gives up after the first attempt or the second attempt if it was redirected. A higher fault tolerance could possibly be achieved by attempting to reconnect again after some time since the recovery mechanism might not be completed on a busy server before the clients tries to reunite with its group.

7.3.2 Long-term perspective

Performing a study using a more realistic environment may create more reliable measures. This environment should include multiple physical machines, a network topology that would be used in practice, and actual clients, instead of simulated ones, running on several physical machines, too. Running simulations with more servers and with more scenarios may be beneficial since it could be used to investigate if the solution is able to scale well in clusters of many more servers than used in this study. In addition, it might be interesting to see whether and how different types of failures in a real production environment affect the result.

Furthermore, the solution created may not be the best solution. Future work could involve inventing other solutions that are able to achieve a higher fault tolerance in the occurrence of multiple simultaneous faults. The recent addition of the technology called WebRTC to browsers, which adds the possibility of peer-to-peer connections (Google, 2013), might be the next step in finding a more reliable solution since clients could be less dependent on servers and, therefore, not be as disturbed by server faults.

There are security vulnerabilities in the implementation of the solution created in for this study. Future work could possibly build on this work by exploring security problems in the solution and documenting these, as to potentially make it possible to create more secure implementations of the solution that more safely could be used in real consumer facing applications.

7.3.3 Concluding remarks

It is clear that the implementation of the solution is able to achieve some degree of fault tolerance. This applies to all simulated scenarios; however, scenarios involving more than one fault lead to a significantly less fault tolerance. It is not possible to conclude whether this is because of how the solution is implemented or because of the design of the solution. The outlier identified from the scenario of 3 of 4, indicate that small delays between faults may result in a higher fault tolerance since the faults are treated as if they are not simultaneous. One can conclude, however, that applying the solution to a groupware implementation increases the reliability. In addition, if there is not more than one fault occurring at the same instant, one can expect the average fault tolerance to be between 70% and 100%.

References

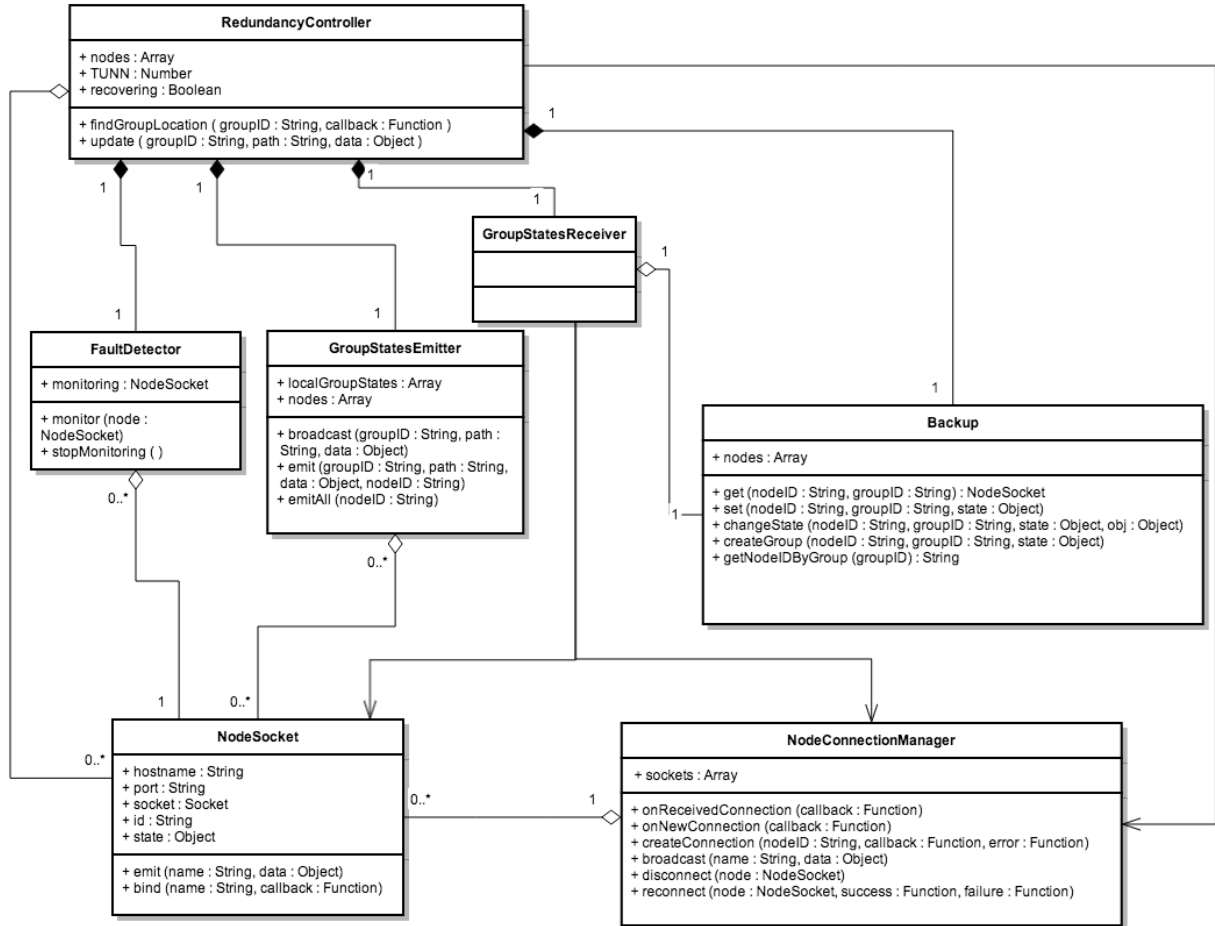
- Chen, B. & Xu, Z. (2011) A framework for browser-based Multiplayer Online Games using WebGL and WebSocket. *2011 International Conference on Multimedia Technology (ICMT)*. pp. 471–474.
- Cook, T.D. & Campbell, D.T. (1986) The causal assumptions of quasi-experimental practice. *Synthese*. 68(1), pp. 141–180
- Cristian, F. (1991) Understanding Fault-tolerant Distributed Systems. *Commun. ACM*. 34 (2), pp. 56–78.
- Ellis, C.A., Gibbs, S.J. & Rein, G. (1991) Groupware: Some Issues and Experiences. *Commun. ACM*. 34 (1), pp. 39–58.
- Federal Standard 1037 (1996) Telecommunications: Glossary of Telecommunication Terms. Available on the Internet: <http://www.its.bldrdoc.gov/fs-1037c.htm> [Accessed February 18, 2014]
- Fu, H., Cai, M., Zhu, W., Huang, Z., Dong, J. (2007) Research on Fault Tolerance in Hybrid P2P-based Collaborative Systems. *11th International Conference on Computer Supported Cooperative Work in Design, 2007. CSCWD 2007*. pp. 30–35.
- Gillen, M., Rohloff, K., Manghwani, P. & Schantz, R. (2007) Scalable, Adaptive, Time-Bounded Node Failure Detection. *10th IEEE High Assurance Systems Engineering Symposium, 2007. HASE '07*. s. 179–186.
- Google (2013) WebRTC. Available on the Internet: <http://www.webrtc.org/> [Accessed December 6, 2013].
- Gupta, D., Vishwanath, K.V., McNett, M., Vahdat, A., Yocum, K., Snoeren, A., Voelker, G. (2011) DieCast: Testing Distributed Systems with an Accurate Scale Model. *ACM Trans. Comput. Syst.* 29 (2), s. 4:1–4:48.
- Gutwin, C.A., Lippold, M. & Graham, T.C.N. (2011) Real-time Groupware in the Browser: Testing the Performance of Web-based Networking. *Proceedings of the ACM 2011 Conference on Computer Supported Cooperative Work. CSCW '11*. New York, NY, USA, ACM. pp. 167–176.
- Nastase, M., Dobre, C., Pop, F. & Cristea, V. (2009) Fault Tolerance Using a Front-End Service for Large Scale Distributed Systems. *2009 11th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*. pp. 229–236.
- Nguyen, D. & Liu, D.-B. (1998) Recovery blocks in real-time distributed systems. *Reliability and Maintainability Symposium, 1998. Proceedings., Annual*. pp. 149–154.
- Ojamaa, A. & Duuna, K. (2012) Assessing the security of Node.js platform. *Internet Technology And Secured Transactions, 2012 International Conference for*. pp. 348–355.
- Qin, X. & Sun, C. (2001) Recovery support for Internet-based real-time collaborative editing systems. *2001 International Conference on Computer Networks and Mobile Computing, 2001. Proceedings*. pp. 181–188.
- Ringhede, A. (2014) WebMF. Available on the Internet: <https://github.com/everse/WebMF> [Accessed April 16, 2014]
- Shim, H.S. & Prakash, A. (1998) Tolerating Client and Communication Failures in Distributed Groupware Systems. *In Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems*. pp. 221–227.
- Khafa, F., Kolici, V., Potlog, A.-D., Spaho, E., Barolli, L., Takizawa, M. (2012) Data Replication in P2P Collaborative Systems. *2012 Seventh International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*. pp. 49–57.
- Xiong, C.-J., Li, Y.-F., Xie, M., Ng, S.-H., Goh, T.-N. (2009) Service Reliability and Availability Analysis of Distributed Software Systems Considering Malware Attack. Dominik Ślęzak, Tai-hoon Kim, Akingbehin Kiumi, Tao Jiang, June Verner, Silvia Abrahao (eds.). *Advances in Software Engineering*. Communications in Computer and Information Science. Springer Berlin Heidelberg. pp. 313–320.

Yang, C., Zhu, Z., Huang, W., Yang, C., Zhang, W. (2009) Application of simulation technology in reliability measure of ad hoc network. *8th International Conference on Reliability, Maintainability and Safety, 2009. ICRMS 2009*. pp. 1137–1140.

Zia, M., Mustafiz, S., Vangheluwe, H. & Kienzle, J. (2005) A Modelling and Simulation Based Approach to Dependable System Design. Lionel Briand & Clay Williams (eds.). *Model Driven Engineering Languages and Systems*. Lecture Notes in Computer Science. Springer Berlin Heidelberg. pp. 217–231.

Appendix A - Fault tolerance components

Design



NodeConnectionManager.js

```

var net = require('net'),
    config = require('./config'),
    EventEmitter = require('events').EventEmitter,
    carrier = require('carrier');

function NodeConnectionManager () {
  this.sockets = {};
  this.events = new EventEmitter();

  var self = this;
  var server = net.createServer(function(socket) {
    var nodeSocket = new NodeSocket(socket);
    nodeSocket.once('nodeid', function (data) {
      nodeSocket.setID(data.nodeID);
      nodeSocket.bind('disconnect', function () {
        delete self.sockets[data.nodeID];
      });
    });
  });
}
  
```

```

        });
        self.sockets[data.nodeID] = nodeSocket;

        self.events.emit('receivedConnection', nodeSocket);
        self.events.emit('newConnection', nodeSocket);
    });
});
var port = config.nodes[config.server.nodeID].port;
var host = config.nodes[config.server.nodeID].host;
server.listen(port, function() {
    console.log("Listening on host:port "+host+": "+ port);
});
}
NodeConnectionManager.prototype.onReceivedConnection = function(callback)
{
    this.events.on('receivedConnection', callback);
};
NodeConnectionManager.prototype.onNewConnection = function(callback) {
    this.events.on('newConnection', callback);
};
NodeConnectionManager.prototype.broadcast = function(name, data) {
    for (var nodeID in this.sockets) {
        if(this.sockets[nodeID].state === 0) {
            this.sockets[nodeID].emit(name, data);
        }
    }
};
NodeConnectionManager.prototype.disconnect = function(nodeSocket) {
    if (this.sockets[nodeSocket.id]) {
        try {
            nodeSocket.end();
        } catch (e) {}
        delete this.sockets[nodeSocket.id];
    }
};
NodeConnectionManager.prototype.reconnect = function(nodeSocket, success,
failure) {
    var node = config.nodes[nodeSocket.id];
    if (!node) {
        if (failure) failure(nodeSocket);
        return false;
    }

    var socket = net.connect({host: node.host, port: node.port},
function() {
        nodeSocket.setSocket(socket);
        if (success) success(nodeSocket);
        self.events.emit('newConnection', nodeSocket);
    });
    socket.on('error', function(err) {
        if (err.code === 'ECONNREFUSED') {
            if (failure) failure(nodeSocket);
        }
    });
});
};

```

```

};
NodeConnectionManager.prototype.count = function() {
    var n = 0;
    for (var nodeID in this.sockets) {
        if(this.sockets[nodeID]) n++;
    }
    return n;
};
NodeConnectionManager.prototype.countOpen = function() {
    var n = 0;
    for (var nodeID in this.sockets) {
        if(this.sockets[nodeID] && this.sockets[nodeID].state === 0)
n++;
    }
    return n;
};
NodeConnectionManager.prototype.delete = function(nodeID) {
    delete this.sockets(nodeID);
};

NodeConnectionManager.prototype.createConnection = function(nodeID,
callback, errorCallback) {
    var node = config.nodes[nodeID];
    if(!node) throw "There is no node with the nodeID: " + nodeID;
    if(this.sockets[nodeID] && this.sockets[nodeID].state === 0){
        callback(this.sockets[nodeID]);
    } else {
        var self = this;
        var socket = net.connect({host: node.host, port: node.port},
function() {
            var nodeSocket = new NodeSocket(socket, nodeID);
            nodeSocket.bind('disconnect', function () {
                delete self.sockets[nodeID];
            });
            self.sockets[nodeID] = nodeSocket;
            nodeSocket.emit('nodeid', {
                nodeID: config.server.nodeID
            });
            callback(nodeSocket);
            self.events.emit('newConnection', nodeSocket);
        });
        socket.on('error', function(err) {
            if (err.code === 'ECONNREFUSED') {
                errorCallback('ECONNREFUSED');
                console.log("The connection to " + node.host + ":"
+ node.port + " was refused");
            }
        });
    }
};

function NodeSocket (socket, nodeID) {
    if (!socket) throw "Can not create a NodeSocket without a socket";
    var node = config.nodes[nodeID];

```

```

    this.events = new EventEmitter();
    this.hostname = node && node.host;
    this.port = node && node.port;
    this.socket = socket;
    this.id = nodeID || "noID";
    this.TUNN = -1;
    this.state = 0; // 0 = Open , 1 = Closed
    this.carrier;

    this.setSocket(socket);
}
NodeSocket.prototype.setSocket = function(netSocket) {
    this.socket = netSocket;
    this.socket.setEncoding('utf8');
    this.carrier = carrier.carry(netSocket);

    var self = this;
    this.socket.on('close', function() {
        self.state = 1;
        self.events.emit('disconnect', self);
    });
    this.socket.on('error', function(para) {

    });
    this.carrier.on('line', function(line) {
    var data = JSON.parse(line);
        self.events.emit(data.name, data.data);
    });
};
NodeSocket.prototype.setID = function(id) {
    var node = config.nodes[id];
    this.id = id;
    this.hostname = node && node.host;
    this.port = node && node.port;
};
NodeSocket.prototype.emit = function(name, data) {
    if (!this.socket.writable || this.state !== 0) return;
    if(!name || typeof name !== 'string') throw "Emit requires a name of
type String";

    this.socket.write(JSON.stringify({
        name: name,
        data: data || {}
    }) + "\r\n");
};
NodeSocket.prototype.bind = function(name, callback) {
    this.events.on(name, callback);
};
NodeSocket.prototype.once = function(name, callback) {
    this.events.once(name, callback);
};
};

```

```
module.exports = new NodeConnectionManager();
```

GroupStatesReceiver.js

```
var ncm = require('./NodeConnectionManager'),
    config = require('./config');

function GroupStatesReceiver (backup) {

    ncm.onNewConnection(function (nodeSocket) {
        nodeSocket.bind('stateUpdate', function (data) {
            var groupStateClone = backup.get(data.emitterID,
            data.groupID);
            if (groupStateClone) {
                backup.changeState(data.emitterID, data.groupID,
            data.path, data.data);
            } else {
                if (data.path === null) {
                    backup.createGroup(data.emitterID,
            data.groupID, data.data);
                }
            }
        });
    });
}

module.exports = GroupStatesReceiver;
```

GroupStatesEmitter.js

```
var ncm = require('./NodeConnectionManager'),
    config = require('./config');

function GroupStatesEmitter (nodes, localGroupStates) {
    this.localGroupStates = localGroupStates || [];
    this.nodes = nodes;
}

GroupStatesEmitter.prototype.broadcast = function (groupID, path, data) {
    for (var nodeID in this.nodes) {
        this.emit(groupID, path, data, nodeID);
    }
};

GroupStatesEmitter.prototype.emit = function (groupID, path, data, nodeID)
{
    if (this.nodes[nodeID].state === 0) {
        this.nodes[nodeID].emit('stateUpdate', {
            emitterID: config.server.nodeID,
            groupID: groupID,
            path: path,
            data: data
        });
    }
}
```

```

};
GroupStatesEmitter.prototype.emitAll = function(nodeID) {
    for(var i = 0, l = this.localGroupStates.length; i < l; i++) {
        this.emit( this.localGroupStates[i].id,
                    null,
                    this.localGroupStates[i].state,
                    nodeID );
    }
};

module.exports = GroupStatesEmitter;

```

FaultDetector.js

```

var ncm = require('./NodeConnectionManager'),
    config = require('./config'),
    EventEmitter = require('events').EventEmitter;

function FaultDetector (onFault) {
    this.monitoring;
    this.onFault = onFault || function(){};
    var self = this;
    this._onDisconnect = function (nodeSocket) {
        function success() {
            console.log("Reconnected to node: " + nodeSocket.id)
        }
        function failure () {
            self.onFault(self.monitoring);
        }
        ncm.reconnect(nodeSocket, success, failure);
    };
}

FaultDetector.prototype.monitor = function (nodeSocket) {
    if (!nodeSocket) return;
    console.log("Monitoring " + nodeSocket.id);
    this.stopMonitoring();
    this.monitoring = nodeSocket;
    nodeSocket.bind('disconnect', this._onDisconnect);
};

FaultDetector.prototype.stopMonitoring = function () {
    if (this.monitoring) {
        this.monitoring.events.removeListener('disconnect',
this._onDisconnect);
        this.monitoring = undefined;
    }
};

module.exports = FaultDetector;

```

Backup.js

```

function Backup () {
    this.nodes = {};
}

```

```

Backup.prototype.get = function(nodeID, groupID) {
    if(this.nodes[nodeID]) {
        return this.nodes[nodeID][groupID];
    } else {
        return undefined;
    }
};

Backup.prototype.set = function(nodeID, groupID, state) {
    if(this.nodes[nodeID]) {
        this.nodes[nodeID][groupID] = state;
    } else {
        this.createGroup(nodeID, groupID, state);
    }
};

Backup.prototype.changeState = function(nodeID, groupID, path, obj) {
    var pathSteps = path.split('/');
    var state = this.get(nodeID, groupID);
    var stateObjectReference = state;
    for(var i = 0; i < pathSteps.length; i++){
        if(stateObjectReference[pathSteps[i]] === null ||
stateObjectReference[pathSteps[i]] === undefined){
            stateObjectReference[pathSteps[i]] = {};
        }
        if(pathSteps[i] !== null){
            if(i === 0){
                if(pathSteps.length === 1){
                    stateObjectReference[pathSteps[i]] = obj;
                } else {
                    stateObjectReference = state[pathSteps[i]];
                }
            } else {
                if(i === pathSteps.length-1){
                    stateObjectReference[pathSteps[i]] = obj;
                    break;
                } else {
                    stateObjectReference =
stateObjectReference[pathSteps[i]];
                }
            }
        }
    }
};

Backup.prototype.createGroup = function(nodeID, groupID, state) {
    if (!this.nodes[nodeID]) {
        this.nodes[nodeID] = {};
    }
    this.nodes[nodeID][groupID] = state || {};
};

Backup.prototype.getNodeIDByGroup = function(groupID) {
    for (var nodeID in this.nodes) {
        if (this.nodes[nodeID][groupID]) {

```

```

        return nodeID;
    }
}
return undefined;
};
module.exports = Backup;

```

RedundancyController.js

```

var GroupStatesEmitter = require('./GroupStatesEmitter'),
    GroupStatesReceiver = require('./GroupStatesReceiver'),
    FaultDetector = require('./FaultDetector'),
    EventEmitter = require('events').EventEmitter;
Backup = require('./Backup'),
ncm = require('./NodeConnectionManager'),
config = require('./config');

function RedundancyController (hostFunction, scope, localGroupStates) {
    this.nodes = {};
    this.backup = new Backup();
    this.gsr = new GroupStatesReceiver(this.backup);
    this.gse = new GroupStatesEmitter(this.nodes, localGroupStates);
    this.faultDetector = new FaultDetector();
    this.TUNN = -1;
    this.events = new EventEmitter();
    this.recovering = false;

    var self = this;

    var hashCode = function(s){return
s.split("").reduce(function(a,b){a=((a<<5)-a)+b.charCodeAt(0);return
a&a},0);}
    function recover (serverID, callback) {
        self.recovering = true;
        var numServers = ncm.countOpen() + 1, // Include the local node
            recoveredCount = 0,
            totalCount = 0;

        console.log("Starting recover with the following variables:");
        console.log("    TUNN: " + self.TUNN + "    NumServers: " +
numServers);

        for (var cloneID in self.backup.nodes[serverID]) {
            totalCount++;
            var targetHostNumber = Math.abs(hashCode(cloneID) %
numServers);

            if (targetHostNumber === self.TUNN) {
                hostFunction.call(scope, {
                    state: self.backup.get(serverID, cloneID),
                    id: cloneID
                });
                recoveredCount++;
            } else {
                for (var nodeID in self.nodes) {

```

```

        if (self.nodes[nodeID].TUNN ===
targetHostNumber && self.nodes[nodeID].state === 0) {
            self.backup.createGroup(nodeID,
cloneID, self.backup.get(serverID, cloneID));
            break;
        }
    }
    delete self.backup.nodes[serverID][cloneID]; // Otherwise
Backup.getNodeIDByGroup may return an offline node.
}
    console.log("Recovered " + recoveredCount + "/" + totalCount +
" groups from node " + serverID);
    self.recovering = false;
    self.events.emit('finishedRecovery');
    callback.call(this);
}

ncm.onNewConnection(function (nodeSocket) {
    self.nodes[nodeSocket.id] = nodeSocket;

    nodeSocket.bind('TUNN', function (data) {
        nodeSocket.TUNN = data.TUNN;
        if (data.TUNN >= 0 && self.TUNN > 0 && data.TUNN+1 ===
self.TUNN) {
            self.faultDetector.monitor(nodeSocket);
        }
        if (self.TUNN === 0) {
            var nodeWithHighestTUNN,
                highestTUNN = 0;
            for (var nodeID in self.nodes) {
                if (self.nodes[nodeID].TUNN > highestTUNN) {
                    highestTUNN = self.nodes[nodeID].TUNN;
                    nodeWithHighestTUNN =
self.nodes[nodeID];
                }
            }
            self.faultDetector.monitor(nodeWithHighestTUNN)
        }
    });
    nodeSocket.bind('fault', function (nodeID) {
        self.events.emit('nodeFault', self.nodes[nodeID]);
    });

    self.gse.emitAll(nodeSocket.id);
});

var numConnections = 0,
    numAttempts = 0;

self.events.on('nodeFault', function (nodeSocket) {
    self.recovering = true;
    console.log(nodeSocket.id + " faulted");

```

```

        var highest = true,
            nodeWithHighestTUNN,
            highestTUNN = 0,
            toMonitorIfHighest;
        for (var nodeID in self.nodes) {
            if (self.nodes[nodeID].state === 0 &&
self.nodes[nodeID].TUNN > self.TUNN) {
                highest = false;
            }
            if (self.nodes[nodeID].state === 0 &&
self.nodes[nodeID].id !== nodeSocket.id && self.nodes[nodeID].TUNN >
highestTUNN) {
                highestTUNN = self.nodes[nodeID].TUNN;
                nodeWithHighestTUNN = self.nodes[nodeID];
            }
            if (self.nodes[nodeID].state === 0 &&
self.nodes[nodeID].TUNN === nodeSocket.TUNN-1) {
                toMonitorIfHighest = self.nodes[nodeID];
            }
        }
        if (highest && nodeSocket.TUNN < self.TUNN) {
            self.TUNN = nodeSocket.TUNN;
            if (nodeSocket.TUNN !== 0)
self.faultDetector.monitor(toMonitorIfHighest);
            console.log("Changed local TUNN to: " + self.TUNN);
            ncm.broadcast('TUNN', {
                TUNN: self.TUNN
            });
        } else if (nodeWithHighestTUNN) {
            // Set the new TUNN of the one with the highest before
the fault.
            nodeWithHighestTUNN.TUNN = nodeSocket.TUNN;
        }
        if (self.TUNN === 0 && nodeWithHighestTUNN) {
            // If no node is selected, then the local node is the
only one still up.
            self.faultDetector.monitor(nodeWithHighestTUNN);
        }

        recover(nodeSocket.id, function () {
            ncm.disconnect(nodeSocket);
            delete self.nodes[nodeSocket.id];
        });
    });

    this.faultDetector.onFault = function (nodeSocket) {
        self.events.emit('nodeFault', nodeSocket);

        // Notify the other nodes about the faulting node
        ncm.broadcast('fault', nodeSocket.id);
    };

    // Connect to every known node
    var nodeIDsToConnectTo = [];

```

```

function onConnect (nodeSocket) {
    numConnections++;

    onConnectionAttempt();
    console.log("Connected to node " + nodeSocket.id);
}
function onFinishedConnecting () {
    self.TUNN = numConnections;
    ncm.broadcast('TUNN', {
        TUNN: self.TUNN
    });
    for (var nodeID in self.nodes) {
        if (self.nodes[nodeID].TUNN >= 0 && self.TUNN > 0 &&
self.nodes[nodeID].TUNN+1 === self.TUNN) {
            self.faultDetector.monitor(self.nodes[nodeID]);
        }
    }
    console.log("Local TUNN: " + self.TUNN);
}
function onConnectionAttempt () {
    numAttempts++;
    if (numAttempts === nodeIDsToConnectTo.length) {
        onFinishedConnecting();
    } else {
        ncm.createConnection(nodeIDsToConnectTo[numAttempts],
onConnect, onConnectionAttempt);
    }
}
// Collect nodeIDs
for (var nodeID in config.nodes) {
    if (nodeID !== config.server.nodeID) {
        nodeIDsToConnectTo.push(nodeID);
    }
}
if (nodeIDsToConnectTo.length > 0) {
    ncm.createConnection(nodeIDsToConnectTo[0], onConnect,
onConnectionAttempt);
} else {
    onFinishedConnecting();
}

// Listen for incoming connections
ncm.onReceivedConnection(function (nodeSocket) {
    nodeSocket.emit('TUNN', {
        TUNN: self.TUNN
    });
    console.log("Received connection from " + nodeSocket.id);
});
}
RedundancyController.prototype.update = function(groupID, path, data) {
    this.gse.broadcast(groupID, path, data);
};

```

```

RedundancyController.prototype.findGroupLocation = function(groupID,
callback) {
    var self = this;

    function readyToRespond () {
        var nodeID = self.backup.getNodeIDByGroup(groupID);
        if (nodeID) {
            callback.call(self, {
                host: config.nodes[nodeID].host,
                port: config.nodes[nodeID].clientsPort
            });
        } else {
            // The group is not in backup. It is either hosted on
this server or not at all.
            callback.call(self, false);
        }
    }

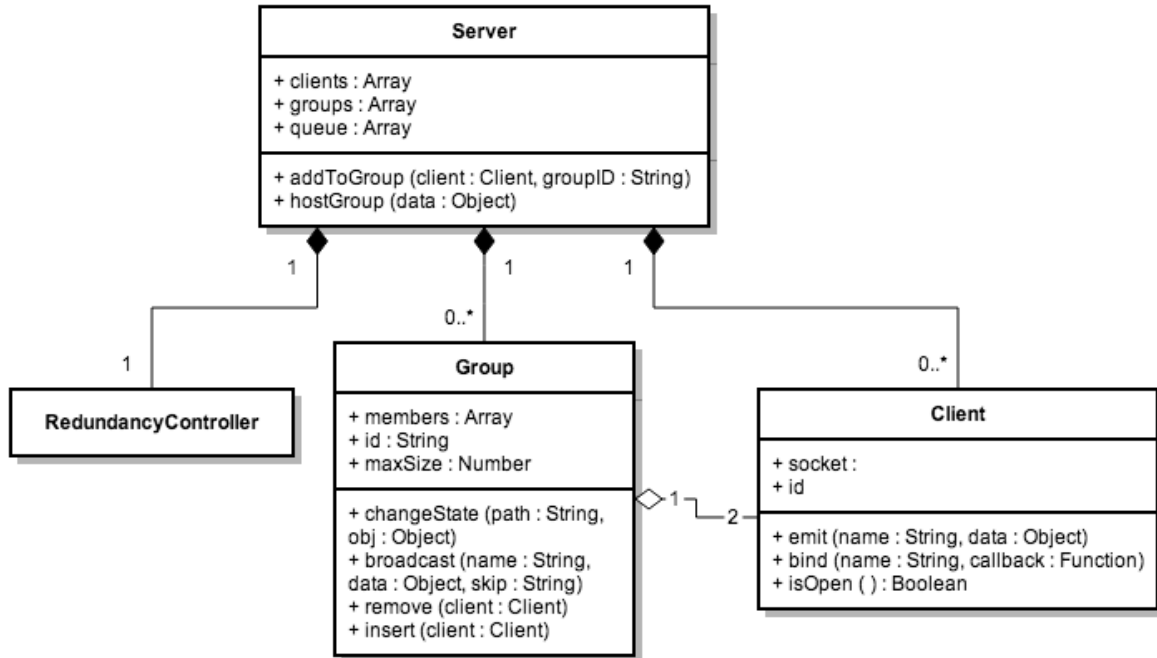
    if (self.recovering) {
        self.events.once('finishedRecovery', readyToRespond);
    } else {
        readyToRespond();
    }
};

module.exports = RedundancyController;

```

Appendix B - Groupware system with fault tolerance

Design



Server.js

```
var net = require('net'),
    config = require('../config'),
    Client = require('./Client'),
    Group = require('./Group'),
    EventEmitter = require('events').EventEmitter,
    WebSocketServer = require('ws').Server,
    RedundancyController = require('../RedundancyController');

function Server () {
  this.clients = {};
  this.numClients = 0;
  this.groups = [];
  this.queue = [];
  this.rc = new RedundancyController(this.hostGroup, this,
  this.groups);

  var self = this;
  var setClientListeners = function (client) {
    client.bind('disconnect', function () {
      if (client.group) {
        client.group.remove(client);
        client.group.broadcast('disconnected', client.id);
        client.group = false;
      }
    });
    // Remove the client from the queue
  }
}
```

```

        for (var i = 0, l = self.queue.length; i < l; i++) {
            if (self.queue[i].id === client.id) {
                self.queue.splice(i, 1);
                console.log("Removed from queue " +
client.id);
                console.log(self.queue);
            }
        }
        // Remove the client from the server
        delete self.clients[client.id];
        self.numClients--;
    });
    client.bind('joinGroup', function (data) {
        if (!data.id || typeof data.id !== 'string') {
            // Join any group
            self.addToGroup(client)
        } else {
            // Join a specific group
            self.addToGroup(client, data.id);
        }
    });
    client.bind('updateState', function (data){
        client.group.changeState(data.path, data.obj)
        client.group.broadcast('stateChanged', data);
        self.rc.update(client.group.id, data.path, data.obj);
    });
    });
    var wss = new WebSocketServer({port:
config.nodes[config.server.nodeID].clientsPort});
    wss.on('connection', function(ws) {
        var client = new Client(ws);
        client.emit('clientID', {
            id: client.id
        });
        self.clients[client.id] = client;
        self.numClients++;
        setClientListeners(client);
    });
}
Server.prototype.hostGroup = function (data) {
    var group = new Group();
    group.id = data.id;
    group.state = data.state;
    //this.rc.update(group.id, null, group.state); // This should
hopefully not be needed since this would be very demanding on the network
    this.groups.push(group);
};
Server.prototype.addToGroup = function (client, groupID) {
    function onFoundGroup (group, client) {
        client.group = group;
        var clientIDs = [];
        for (var i = 0, l = group.members.length; i < l; i++) {
            clientIDs.push(group.members[i].id);
        }
    }

```

```

        client.emit('_foundGroup', {
            state: group.state,
            id: group.id,
            members: clientIDs
        })
    }

    if (!groupID) { // Add to any group
        var foundGroup = false;
        for (var i = 0, l = this.groups.length; i < l; i++) {
            var group = this.groups[i];
            if (group.members.length < group.maxSize) {
                foundGroup = true;
                group.insert(client);
                onFoundGroup(group, client);
                return true;
            }
        }
        if (!foundGroup) {
            if (this.queue.length > 0) {
                // Find another viable client
                var otherClient = false;
                for (var i = 0, l = this.queue.length; i < l; i++)
                {
                    if (this.queue[i] !== client &&
                        this.queue[i].isOpen() && !this.queue[i].group) {
                        otherClient = this.queue[i];
                        this.queue.splice(i, 1); // remove the
other client from the queue

                        // Create a new group
                        var newGroup = new Group();
                        this.rc.update(newGroup.id, null,
newGroup.state);

                        this.groups.push(newGroup);
                        newGroup.insert(client);
                        newGroup.insert(otherClient);
                        onFoundGroup(newGroup, client);
                        onFoundGroup(newGroup, otherClient);
                        break;
                    }
                }
                if (!otherClient) {
                    this.queue.push(client);
                    client.emit('findGroupPutInQueue', {
                        position: this.queue.length
                    });
                }
            }
        }
        else {
            this.queue.push(client);
            client.emit('findGroupPutInQueue', {
                position: this.queue.length
            });
        }
    }
}

```

```

    }
  } else { // Add to a specific group
    for (var i = 0, l = this.groups.length; i < l; i++) {
      if (this.groups[i].id === groupID) {
        if (this.groups[i].members.length <
this.groups[i].maxSize) {
          this.groups[i].insert(client);
          onFoundGroup(this.groups[i], client);
          return true;
        } else {
          client.emit('findGroupErrorFull', {
            groupID: groupID
          });
          return false;
        }
      }
    }
  }
  // The group is not hosted on this server.
  this.rc.findGroupLocation(groupID, function (location) {
    if (location) {
      client.emit('findGroupRedirect', location);
    } else {
      client.emit('findGroupErrorFindGroup', {
        groupID: groupID
      });
    }
  });
});
};

new Server();

```

Group.js

```

var Puid = require('puid');

var puid = new Puid();
function Group () {
  this.id = puid.generate();
  this.state = {};
  this.members = [];
  this.maxSize = 2;
}
Group.prototype.insert = function(client) {
  this.members.push(client);
};
Group.prototype.remove = function(client) {
  for (var i = 0, l = this.members.length; i < l; i++) {
    if (this.members[i].id === client.id) {
      this.members.splice(i,1);
    }
  }
}

```

```

    }
};
Group.prototype.broadcast = function(name, data, skipID) {
    for (var i = 0, l = this.members.length; i < l; i++) {
        if (this.members[i].id !== skipID) {
            this.members[i].emit(name, data);
        }
    }
};

Group.prototype.changeState = function(path, obj){
    var pathSteps = path.split('/');
    var stateObjectReference = this.state;
    for(var i = 0; i < pathSteps.length; i++){
        if(stateObjectReference[pathSteps[i]] === null ||
stateObjectReference[pathSteps[i]] === undefined){
            stateObjectReference[pathSteps[i]] = {};
        }
        if(pathSteps[i] !== null){
            if(i === 0){
                if(pathSteps.length === 1){
                    stateObjectReference[pathSteps[i]] = obj;
                } else {
                    stateObjectReference =
this.state[pathSteps[i]];
                }
            } else {
                if(i === pathSteps.length-1){
                    stateObjectReference[pathSteps[i]] = obj;
                    break;
                } else {
                    stateObjectReference =
stateObjectReference[pathSteps[i]];
                }
            }
        }
    }
};

Group.prototype.getState = function(path){
    if(!path) return this.state;
    var pathSteps = path.split('/');
    var stateObjectReference = this.state;
    for(var i = 0; i < pathSteps.length; i++){
        // Return the last element if the other one does not exist.
        if(stateObjectReference[pathSteps[i]] === null ||
stateObjectReference[pathSteps[i]] === undefined){
            return stateObjectReference;
        }
        if(pathSteps[i] !== null){
            if(i === 0){
                if(pathSteps.length === 1){
                    return stateObjectReference[pathSteps[i]];
                }
            }
        }
    }
};

```

```

        } else {
            stateObjectReference =
this.state[pathSteps[i]];
        }
    } else {
        if(i === pathSteps.length-1){
            return stateObjectReference[pathSteps[i]];
        } else {
            stateObjectReference =
stateObjectReference[pathSteps[i]];
        }
    }
}
};

module.exports = Group;

```

Client.js

```

var EventEmitter = require('events').EventEmitter,
    Puid = require('puid'),
    crypto = require('crypto');

var puid = new Puid();
function Client (socket) {
    if (!socket) throw "Can not create a Client without a socket";

    this.events = new EventEmitter();
    this.socket = socket;
    this.id = puid.generate();

    this.setSocket(socket);
}
Client.prototype.setSocket = function(socket) {
    this.socket = socket;

    var self = this;
    this.socket.on('close', function() {
        //self.events.emit('disconnect', self);
    });
    this.socket.on('error', function(e) {
        self.events.emit('error', e);
    });
    this.socket.on('message', function(message){
        // Increase CPU Usage using encryption
        for(var i = 0, l = 20; i<l; i++){ // high = 20, veryhigh = 100
            var cipher = crypto.createCipher('aes-256-cbc',
'dfnjdfopr2f2ccsdf3ayh43w8ofnwevo8v3');
            var c = cipher.update(message, 'utf8', 'hex');
            c += cipher.final('hex');
            var decipher = crypto.createDecipher('aes-256-cbc',
'dfnjdfopr2f2ccsdf3ayh43w8ofnwevo8v3');

```

```

        var d = decipher.update(c, 'hex', 'utf8')
        d += decipher.final('utf8');
    }

    var data = JSON.parse(message);
    self.events.emit(data.name, data.data);

    if (data.name === "updateState") {
        setTimeout(function () {
            self.socket.emit('message', message);
        }, 40);
    }
});

};
Client.prototype.emit = function(name, data) {
    if(!name || typeof name !== 'string') throw "Emit requires a name of
type String";
    if (this.isOpen()) {
        this.socket.send(JSON.stringify({
            name: name,
            data: data || {}
        }));
    }
};
Client.prototype.isOpen = function() {
    return this.socket.readyState === this.socket.OPEN;
};
Client.prototype.bind = function(name, callback) {
    this.events.on(name, callback);
};

module.exports = Client;

```

Client – Client.js

```

(function () {

    function Connection (parameters) {
        this.server;
        this.servers = parameters && parameters.servers || [];
        this.currentGroupSession;
        this.open = false;
        this.inGroup = false;
        this.id;
        this._events = {};

        if (parameters) {
            this.connect({
                host: parameters.host,
                port: parameters.port,
                onSuccess: parameters.onConnect,
                onFailure: parameters.onFailure
            });
        }
    }

```

```

    });
}

var self = this;
this.bind('findGroupRedirect', function (data) {
    self.connect({
        host: data.host,
        port: data.port,
        onSuccess: function () {
            if (self.currentGroupSession) {
                self.joinGroup({
                    groupID:
self.currentGroupSession.id,
                    onFoundGroup: function
(groupSession) {

                        self.trigger('rejoinedGroup', groupSession);
                    }
                });
            }
            self.trigger('reconnected');
        },
        onFailure: function () {
            self.trigger('rejoiningGroupFailed');
        }
    });
});
}
// parameters may include host, port, onSuccess, onFailure
Connection.prototype.connect = function(parameters) {
    var self = this;
    if (!parameters) parameters = {};

    function createConnection (host, port, success, failure) {
        var newServer;
        if (host && port) {
            newServer = new WebSocket("ws://" + host + ":" +
port );
        } else {
            return false;
        }
        newServer.onopen = function () {
            // console.log("Connected to " + host);
            if (self.server) {
                self.disconnect();
            }
            self.server = newServer;
            self.server.host = host;
            self.server.port = port;
            self.open = true;
            self.once('clientID', function (data){
                self.id = data.id;
            });
        }
    }

```

```

        if (parameters.onSuccess)
parameters.onSuccess.call(self);
    };
    newServer.onmessage = function (e) {
        var data = JSON.parse(e.data);
        if (self._events[data.name]) {
            for (var i = 0, l =
self._events[data.name].length; i < l; i++) {
                if (typeof self._events[data.name][i]
=== 'function') {
                    self._events[data.name][i].call(this, data.data);
                }
            }
        }
    };
    newServer.onerror = function () {
        self.open = false;
        if (failure) failure();
    };
    newServer.onclose = function () {
        if (!self.open) return; // Prevent this listener
from executing when the connection is refused
        self.open = false;
        // Reconnect to the current server first
        self.connect({
            host: self.server.host,
            port: self.server.port,
            onSuccess: function () {
                // Rejoin group if currently in one
                if (self.currentGroupSession) {
                    self.joinGroup({
                        groupID:
self.currentGroupSession.id,
                        onFoundGroup: function
(groupSession) {
                            self.trigger('rejoinedGroup', groupSession);
                        }
                    });
                }
                self.trigger('reconnected');
            },
            onFailure: function () {
                // If that does not work, connect to
any server
                self.connect({
                    onSuccess: function () {
                        // Rejoin group if
currently in a group
                        if
(self.currentGroupSession) {
                            self.joinGroup({

```

```

self.currentGroupSession.id,
function (groupSession) {
    self.trigger('rejoinedGroup', groupSession);
}
});

self.trigger('reconnected');
},
onFailure: parameters.onFailure
// Failed to reconnect to any server
});
}
});
self.trigger('disconnect');
if (self.currentGroupSession) {
    self.trigger('disconnectedFromGroup');
}
}
}

if (parameters.host && parameters.port) {
    // Connect to a specific server
    createConnection(parameters.host, parameters.port,
parameters.onSuccess, function () {
    if (parameters.onFailure)
parameters.onFailure.call(self, "Connection refused");
});
} else if (self.servers.length > 0) {
    // Connect to any possible server
    var attemptNum = 0;
    // Select a random server to start with for the sake of
load balancing
    var startIndex = Math.floor(Math.random() *
self.servers.length);
    function connectToAnyServer () {
        serverIndex = (startIndex + attemptNum) %
self.servers.length;
        attemptNum++;
        if (attemptNum <= self.servers.length) {
            createConnection(
                self.servers[serverIndex].host,
                self.servers[serverIndex].port,
                parameters.onSuccess,
                connectToAnyServer);
        } else if (parameters.onFailure) {
            parameters.onFailure.call(self, "Connection
refused to all");

```

```

        }
    }
    connectToAnyServer();
}
return this;
};
Connection.prototype.disconnect = function() {
    if (this.server) {
        // Remove any current connection
        this.server.onclose = function(){};
        this.server.onerror = function(){};
        this.server.close();
    }
    if (this.currentGroupSession) {
        this.currentGroupSession.connected = false;
    }
};
Connection.prototype.setServers = function (servers) {
    this.servers.length = 0;
    for (var i = 0, l = servers.length; i < l; i++) {
        this.servers.push(servers[i]);
    }
    return this;
};
// Parameters may include groupID, onFoundGroup, onFailure,
onPutInQueue
Connection.prototype.joinGroup = function(parameters) {
    if (!parameters) parameters = {};
    this.removeListener('_foundGroup'); // Remove old listeners.

    var self = this;
    self.emit('joinGroup', {
        id: parameters.groupID
    });
    self.once('_foundGroup', function (data) {
        self.inGroup = true;
        self.currentGroupSession = new GroupSession(self);
        self.currentGroupSession.setState(data.state);
        self.currentGroupSession.id = data.id;
        self.currentGroupSession.members = data.members;
        if (parameters.onFoundGroup)
parameters.onFoundGroup.call(self, self.currentGroupSession);
    });
    if (parameters.onFailure) {
        self.bind('findGroupError', function (message) {
            parameters.onFailure.call(self, message);
        });
    }
    if (parameters.onPutInQueue) {
        self.bind('findGroupPutInQueue', function (data) {
            parameters.onPutInQueue.call(self, data);
        });
    }
    return this;
};

```

```

};
Connection.prototype.emit = function (name, data) {
    if (this.server.readyState !== 1) return;
    this.server.send(JSON.stringify({
        name: name,
        data: data || {}
    }));
    return this;
};
Connection.prototype.trigger = function(name, data) {
    if (this._events[name]) {
        for (var i = 0, l = this._events[name].length; i < l;
i++) {
            if (typeof this._events[name][i] === 'function') {
                this._events[name][i].call(this, data);
            }
        }
        return this;
    }
};
Connection.prototype.bind = function (name, fn) {
    if (typeof fn !== 'function') throw "Bind requires a function
as a callback";
    if (!this._events[name]) {
        this._events[name] = [];
    }
    console.log()
    this._events[name].push(fn);
    return this;
};
Connection.prototype.removeListener = function(name, fn) {
    if (fn) {
        var self = this;
        for (var i = 0, l = self._events[name].length; i < l;
i++) {
            if (self._events[name][i] === fn) {
                self._events[name].splice(i, 1);
            }
        }
    } else {
        this._events[name] = [];
    }

    return this;
};
Connection.prototype.once = function (name, fn) {
    var self = this;
    this.bind(name, function (data) {
        self.removeListener(name, fn);
        fn.call(self, data);
    });
    return this;
};

```

```

function State (obj) {
    this._data = obj || {};
}
State.prototype.changeState = function (path, obj) {
    var pathSteps = path.split('/');
    var stateObjectReference = this._data;
    for(var i = 0; i < pathSteps.length; i++){
        if(stateObjectReference[pathSteps[i]] === null ||
stateObjectReference[pathSteps[i]] === undefined){
            stateObjectReference[pathSteps[i]] = {};
        }
        if(pathSteps[i] !== null){
            if(i === 0){
                if(pathSteps.length === 1){
                    stateObjectReference[pathSteps[i]] =
obj;
                } else {
                    stateObjectReference =
this._data[pathSteps[i]];
                }
            } else {
                if(i === pathSteps.length-1){
                    stateObjectReference[pathSteps[i]] =
obj;
                }
                break;
            } else {
                stateObjectReference =
stateObjectReference[pathSteps[i]];
            }
        }
    }
};
State.prototype.get = function () {
    return this._data;
}
function GroupSession (server) {
    this.server = server;
    this.id;
    this.members = [];
    this.state = new State();
    this.connected = true;

    var self = this;
    this.server.bind('stateChanged', function (data) {
        self.state.changeState(data.path, data.obj);
    });
}
GroupSession.prototype.updateState = function(path, obj) {
    this.state.changeState(path, obj);
    this.server.emit('updateState', {
        path: path,
        obj: obj
    });
});

```

```
};
GroupSession.prototype.setState = function(obj) {
    this.state = new State(obj);
};
window.GroupNet = {
    Connection: Connection,
    State: State,
    GroupSession: GroupSession
}
})();
```

Appendix C - Simulations tools

Simulator

```
<html>
  <head>
    <script src="Chart.min.js"></script>
  </head>
  <body>
    <canvas id="myChart" width="600" height="300"></canvas>
    <script src="./Client.js"></script>

    <script type="text/javascript">

      function simulateClientBehavior (group) {
        if (!group.connected) return;
        for(var i = 0; i < 1; i++) {
          setTimeout(function () {

            group.updateState(Math.ceil(15*Math.random()).toString(), {
              "someValue":
Math.ceil(10000*Math.random()).toString()
            });
          }, Math.floor(200 * Math.random()) * i );
        }
      }

      var k = 0,
          serverConnections,
          startedServers,
          servers = [
            { host: "192.168.0.190", port: 7080 },
            { host: "192.168.0.189", port: 7080 },
            { host: "192.168.0.188", port: 7080 },
            { host: "192.168.0.185", port: 7080 }
          ];

      // Simulation data
      var numClientsInGroups = 0,
          numRejoins = 0,
          numFailuresToRejoinGroup = 0,
          simulationsData = [],
          numAffectedByFault = 0;

      var clientConnections = [];
      function spawnClientConnection () {
        k++;
        var num = k;
        var connection = new GroupNet.Connection();
        connection.setServers(servers).connect({
```

```

        onSuccess: function () {
            this.joinGroup({
                onFoundGroup: function (group) {
                    numClientsInGroups++; //
Collect simulation data
                    simulateClientBehavior(group);
                }
            });
        },
        onFailure: function () {
        }
    }).bind('rejoinedGroup', function (group) {
        numRejoins++; // Collect simulation data
    }).bind('rejoiningGroupFailed', function () {
        numFailuresToRejoinGroup++;
    }).bind('disconnectedFromGroup', function () {
        numAffectedByFault++;
    });
    return connection;
}

function spawnClients (num, callback) {
    clientConnections = [];
    for (var i = 0; i < num; i++) {
        setTimeout(function () {
clientConnections.push(spawnClientConnection());
        }, 100*i);
    }
    setTimeout(callback, 100*num + 2500);
}

function connectToServers (callback) {
    serverConnections = [];
    var numConnections = 0;
    for (var i = 0, l = servers.length; i < l; i++) {
        var c = new WebSocket("ws://" +
servers[i].host + ":7086");
        c.onopen = function () {
            numConnections++;
            if (numConnections === servers.length)
{
                callback();
            }
        };
        serverConnections.push(c);
    }
}

function startSomeServers (numServers) {
    console.log("Starting " + numServers + " servers");
    startedServers = [];
    for (var i = 0; i < numServers; i++) {

```

```

        serverConnections[i].send(JSON.stringify({
            name: "startServer",
            data: {}
        }));
        startedServers.push(serverConnections[i]);
    }
}

function killSomeServers (numServers) {
    console.log("Killing " + numServers + " servers");
    for (var i = 0; i < numServers; i++) {
        var selectedServerIndex =
Math.floor(Math.random() * startedServers.length);

        startedServers[selectedServerIndex].send(JSON.stringify({
            name: "stopServer",
            data: {}
        }));
        startedServers.splice(selectedServerIndex,
1);
    }
}

function killAllServers () {
    for (var i = 0, l = clientConnections.length; i <
l; i++) {
        clientConnections[i].disconnect();
    }
    for (var i = 0, l = serverConnections.length; i <
l; i++) {
        serverConnections[i].send(JSON.stringify({
            name: "stopServer",
            data: {}
        }));
    }
}

function startSimulations () {
    connectToServers(function () {
        repeatMultipleSimulations([
            {start: 2, kill: 1},
            {start: 3, kill: 1},
            {start: 3, kill: 2},
            {start: 4, kill: 1},
            {start: 4, kill: 2},
            {start: 4, kill: 3}
        ], 10, function () {
            console.log("Simulations complete");
            console.log(simulationsData);
            localStorage[(new Date()).toString()] =
JSON.stringify(simulationsData);
        });
    });
}

```

```

function repeatMultipleSimulations (descriptions, times,
callback) {
    if (times > 0) {
        runMultipleSimulations(descriptions, function
() {
            times -= 1;
            repeatMultipleSimulations(descriptions,
times, callback);
        });
    } else {
        callback();
    }
}

function runSimulation (numToStart, numToKill, callback)
{
    // INITIATE DATA STRUCTURES TO COLLECT SIMULATION
    DATA
    numClientsInGroups = 0;
    numRejoins = 0;
    numFailuresToRejoinGroup = 0;
    numAffectedByFault = 0;
    killAllServers();
    setTimeout(function(){
        startSomeServers(numToStart);
    }, 500);

    setTimeout(function () {
        spawnClients(50 * numToStart, function () {
            killSomeServers(numToKill);
            setTimeout(function () {
                simulationsData.push(JSON.stringify({
                    numClientsInGroups:
numClientsInGroups,
                    numRejoins: numRejoins,
                    load: "high",
                    numServers: numToStart,
                    numFaults: numToKill,
                    numFailuresToRejoinGroup:
numFailuresToRejoinGroup,
                    numAffectedByFault:
numAffectedByFault
                }));
                killAllServers();
            }, 3500); // The time given to clients
            to try to reconnect and for simulation data to be collected.
            setTimeout(callback, 4600);
        });
    }, 1000);
}

```

```

    }
    function runMultipleSimulations (descriptions, callback,
index) {
        if (index === undefined) index = 0;
        setTimeout(function () {
            var d = descriptions[index];
            runSimulation(d.start, d.kill, function () {
                index++;
                if (index < descriptions.length) {
                    runMultipleSimulations(descriptions, callback, index);
                } else {
                    if (callback) callback();
                }
            });
        }, 3000); // TIME BETWEEN SIMULATIONS
    }
</script>
</body>
</html>

```

Server Control Program

```

var net = require('net'),
    spawn = require('child_process').spawn,
    WebSocketServer = require('ws').Server,
    EventEmitter = require('events').EventEmitter,
    config = require('./config');

(function(){
    var server = false,
        events = new EventEmitter(),
        wss = new WebSocketServer({port:7086});

    wss.on('connection', function (socket) {
        console.log("Received a connection");
        socket.on('message', function (message) {
            var data = JSON.parse(message);
            events.emit(data.name, data.data);
        });
    });

    events.on('startServer', function () {
        events.emit('stopServer');
        setTimeout(function () {
            server = spawn('node', ['./Server/Server.js']);
            server.stdout.on('data', function (data) {
                console.log('server: ' + data);
            });
        });
    });

```

```

        server.stderr.on('data', function (data) {
            console.log('server error: ' + data);
        });
        server.on('error', function(arg) {
            console.log("ERROR");
            console.log(arg);
        });
        console.log("+ Started");
    }, 100 * (config.server.nodeID.charCodeAt(0) - 96));

});

events.on('stopServer', function () {
    if (server) {
        server.kill("SIGKILL");
        server = false;
        console.log("- Stopped");
    } else {
        console.log("- Server is already down")
    }
});

})();

```

Appendix D - Data from simulations

Faults	Cluster Size	Load	Affected	Clients re-joining group	Clients in Groups	Failed after redirect
1	2	low	76	76	98	0
1	3	low	40	19	148	0
2	3	low	112	74	148	38
1	4	low	56	56	198	0
2	4	low	106	34	198	52
3	4	low	156	58	198	62
1	2	low	26	26	98	0
1	3	low	70	70	148	0
2	3	low	72	46	148	26
1	4	low	50	50	198	0
2	4	low	110	54	198	56
3	4	low	138	0	198	84
1	2	low	80	80	98	0
1	3	low	36	33	148	3
2	3	low	106	106	148	0
1	4	low	56	56	198	0
2	4	low	102	76	198	0
3	4	low	142	0	200	98
1	2	low	28	28	100	0
1	3	low	38	34	148	4
2	3	low	84	50	148	34
1	4	low	48	48	200	0
2	4	low	86	32	198	46
3	4	low	138	68	200	48
1	2	low	26	26	98	0
1	3	low	38	36	148	2
2	3	low	78	40	148	38
1	4	low	56	56	198	0
2	4	low	82	20	200	40
3	4	low	146	0	198	92
1	2	low	74	74	100	0
1	3	low	70	70	150	0
2	3	low	120	0	148	84
1	4	low	48	48	198	0
2	4	low	104	20	198	54
3	4	low	148	50	198	98
1	2	low	30	30	100	0
1	3	low	40	20	150	0
2	3	low	106	0	148	44
1	4	low	48	48	200	0
2	4	low	102	26	196	52

3	4	low	140	0	198	71
1	2	low	70	70	100	0
1	3	low	70	70	148	0
2	3	low	114	26	148	40
1	4	low	48	48	200	0
2	4	low	80	56	198	0
3	4	low	150	0	198	67
1	2	low	82	50	98	0
1	3	low	40	35	148	5
2	3	low	106	0	148	66
1	4	low	44	44	198	0
2	4	low	100	26	198	46
3	4	low	146	0	198	73
1	2	low	76	76	100	0
1	3	low	38	18	148	0
2	3	low	112	68	148	36
1	4	low	48	48	200	0
2	4	low	98	29	200	52
3	4	low	142	56	196	86
1	2	few	4	4	20	0
1	3	few	6	6	28	0
2	3	few	14	4	28	10
1	4	few	4	4	38	0
2	4	few	18	4	38	10
3	4	few	28	0	38	14
1	2	few	4	4	20	0
1	3	few	6	3	28	3
2	3	few	20	14	28	6
1	4	few	12	9	40	0
2	4	few	22	8	38	12
3	4	few	30	10	38	20
1	2	few	12	12	18	0
1	3	few	12	12	28	0
2	3	few	22	18	28	4
1	4	few	12	9	38	3
2	4	few	20	5	38	14
3	4	few	32	10	38	22
1	2	few	14	14	18	0
1	3	few	14	14	30	0
2	3	few	16	12	28	0
1	4	few	8	8	38	0
2	4	few	20	6	36	10
3	4	few	26	8	38	18
1	2	few	2	2	20	0
1	3	few	4	2	30	2
2	3	few	8	4	28	0

1	4	few	14	14	38	0
2	4	few	18	6	38	10
3	4	few	28	0	38	28
1	2	few	16	16	18	0
1	3	few	6	6	28	0
2	3	few	20	10	28	10
1	4	few	10	10	38	0
2	4	few	22	6	38	12
3	4	few	30	0	40	24
1	2	few	14	14	18	0
1	3	few	14	14	30	0
2	3	few	22	22	28	0
1	4	few	12	12	40	0
2	4	few	22	7	38	14
3	4	few	28	0	38	22
1	2	few	4	4	20	0
1	3	few	6	4	30	2
2	3	few	22	4	28	18
1	4	few	6	6	38	0
2	4	few	20	10	38	10
3	4	few	30	0	38	18
1	2	few	6	6	18	0
1	3	few	8	3	30	0
2	3	few	22	22	28	0
1	4	few	8	4	38	0
2	4	few	16	2	38	14
3	4	few	28	0	38	22
1	2	few	4	4	18	0
1	3	few	12	12	28	0
2	3	few	20	0	28	12
1	4	few	14	11	38	0
2	4	few	18	4	38	8
3	4	few	30	12	38	18
1	2	few	4	4	18	0
1	3	few	8	8	30	0
2	3	few	24	24	30	0
1	4	few	12	12	36	0
2	4	few	24	6	38	14
3	4	few	32	0	38	20
1	2	few	6	6	18	0
1	3	few	6	3	28	0
2	3	few	12	4	28	8
1	4	few	8	8	36	0
2	4	few	20	0	38	12
3	4	few	32	0	38	20
1	2	few	4	4	18	0

1	3	few	10	5	28	0
2	3	few	16	10	28	6
1	4	few	6	5	36	0
2	4	few	16	14	38	0
3	4	few	28	16	38	8
1	2	few	2	2	18	0
1	3	few	6	2	30	0
2	3	few	22	0	28	20
1	4	few	8	8	36	0
2	4	few	20	18	38	0
3	4	few	30	14	38	16
1	2	few	6	6	20	0
1	3	few	16	16	30	0
2	3	few	22	14	28	8
1	4	few	12	9	38	0
2	4	few	16	15	36	0
3	4	few	26	14	40	12
1	2	few	16	16	18	0
1	3	few	16	16	30	0
2	3	few	12	6	30	6
1	4	few	6	6	38	0
2	4	few	22	5	38	12
3	4	few	28	10	40	18
1	2	few	12	12	18	0
1	3	few	18	18	28	0
2	3	few	18	12	28	6
1	4	few	6	6	36	0
2	4	few	14	12	38	0
3	4	few	32	0	36	22
1	2	few	16	16	20	0
1	3	few	4	3	28	0
2	3	few	10	4	30	6
1	4	few	12	12	40	0
2	4	few	16	10	36	6
3	4	few	32	10	38	22
1	2	few	6	6	20	0
1	3	few	10	10	28	0
2	3	few	14	8	28	6
1	4	few	12	12	36	0
2	4	few	22	6	38	8
3	4	few	22	6	36	16
1	2	few	6	6	20	0
1	3	few	14	14	28	0
2	3	few	14	14	28	0
1	4	few	8	8	38	0
2	4	few	14	11	38	0

3	4	few	28	0	38	22
1	2	high	24	24	100	0
1	3	high	38	18	148	5
2	3	high	104	0	148	68
1	4	high	48	48	198	0
2	4	high	78	56	198	0
3	4	high	142	0	198	77
1	2	high	80	80	98	0
1	3	high	38	16	148	3
2	3	high	118	84	148	34
1	4	high	48	44	200	0
2	4	high	96	72	198	0
3	4	high	156	0	196	94
1	2	high	74	74	100	0
1	3	high	32	16	148	0
2	3	high	104	66	150	38
1	4	high	50	37	198	0
2	4	high	102	41	198	58
3	4	high	140	20	198	32
1	2	high	68	68	98	0
1	3	high	74	74	150	0
2	3	high	76	33	150	18
1	4	high	46	46	198	0
2	4	high	102	27	196	53
3	4	high	154	40	198	71
1	2	high	26	26	100	0
1	3	high	36	21	148	0
2	3	high	106	0	148	76
1	4	high	44	44	200	0
2	4	high	92	30	198	46
3	4	high	160	56	198	104
1	2	high	82	82	100	0
1	3	high	42	22	148	0
2	3	high	76	30	148	22
1	4	high	48	48	198	0
2	4	high	100	28	198	54
3	4	high	150	0	198	96
1	2	high	24	24	100	0
1	3	high	30	18	148	0
2	3	high	78	31	150	21
1	4	high	50	49	200	1
2	4	high	112	41	198	64
3	4	high	144	45	196	81
1	2	high	80	80	100	0
1	3	high	32	22	150	1
2	3	high	74	27	148	28

1	4	high	44	41	198	0
2	4	high	100	25	200	50
3	4	high	150	0	198	102
1	2	high	74	74	100	0
1	3	high	70	70	148	0
2	3	high	76	21	148	33
1	4	high	54	35	198	0
2	4	high	100	34	198	50
3	4	high	156	50	198	106
1	2	high	24	24	98	0
1	3	high	48	32	148	0
2	3	high	70	18	148	33
1	4	high	50	36	198	0
2	4	high	88	17	198	52
3	4	high	146	30	198	68
1	2	high	76	76	100	0
1	3	high	36	36	148	0
2	3	high	110	110	148	0
1	4	high	48	48	198	0
2	4	high	100	74	198	0
3	4	high	148	34	198	59
1	2	high	28	28	98	0
1	3	high	78	43	148	0
2	3	high	106	80	148	26
1	4	high	48	45	196	3
2	4	high	106	62	198	0
3	4	high	154	0	198	100
1	2	high	78	78	100	0
1	3	high	32	32	148	0
2	3	high	110	0	148	86
1	4	high	52	52	196	0
2	4	high	94	27	198	56
3	4	high	148	0	198	81
1	2	high	24	24	100	0
1	3	high	82	56	148	0
2	3	high	82	34	148	32
1	4	high	58	58	198	0
2	4	high	98	20	196	52
3	4	high	152	41	200	94
1	2	high	20	20	100	0
1	3	high	34	21	148	1
2	3	high	108	78	148	30
1	4	high	60	30	198	0
2	4	high	98	23	198	52
3	4	high	156	72	198	60
1	2	high	16	16	98	0

1	3	high	38	23	148	0
2	3	high	114	74	148	40
1	4	high	42	36	198	0
2	4	high	102	46	198	56
3	4	high	156	58	198	98
1	2	high	24	24	98	0
1	3	high	42	26	148	1
2	3	high	108	82	150	26
1	4	high	48	48	198	0
2	4	high	94	46	200	48
3	4	high	148	48	198	100
1	2	high	28	28	100	0
1	3	high	46	22	148	0
2	3	high	102	74	148	28
1	4	high	50	25	198	0
2	4	high	96	16	198	42
3	4	high	146	41	198	97
1	2	high	30	30	100	0
1	3	high	28	9	148	7
2	3	high	78	28	148	27
1	4	high	48	48	198	0
2	4	high	94	16	198	25
3	4	high	158	46	198	112
1	2	high	80	80	100	0
1	3	high	36	36	150	0
2	3	high	112	0	148	74
1	4	high	52	52	200	0
2	4	high	100	76	198	0
3	4	high	156	60	198	96
1	2	high	26	26	100	0
1	3	high	46	23	148	0
2	3	high	64	22	148	19
1	4	high	56	36	200	0
2	4	high	102	35	198	56
3	4	high	136	0	198	50
1	2	high	20	20	98	0
1	3	high	78	78	148	0
2	3	high	72	29	148	25
1	4	high	48	48	198	0
2	4	high	102	38	198	48
3	4	high	144	36	198	87
1	2	high	18	18	98	0
1	3	high	76	76	148	0
2	3	high	85	45	150	37
1	4	high	46	46	198	0
2	4	high	92	40	198	52

3	4	high	150	47	200	97
1	2	high	30	30	100	0
1	3	high	38	38	148	0
2	3	high	108	82	150	26
1	4	high	52	36	196	0
2	4	high	96	70	198	0
3	4	high	154	44	198	108
1	2	high	76	76	100	0
1	3	high	34	21	150	1
2	3	high	104	0	150	66
1	4	high	58	58	198	0
2	4	high	116	88	196	0
3	4	high	158	0	198	110
1	2	high	30	30	100	0
1	3	high	32	14	148	0
2	3	high	116	0	148	66
1	4	high	60	60	200	0
2	4	high	92	31	196	44
3	4	high	152	24	196	37
1	2	high	74	74	98	0
1	3	high	28	16	148	7
2	3	high	70	18	150	21
1	4	high	58	58	198	0
2	4	high	90	44	198	46
3	4	high	158	56	198	102
1	2	high	22	22	98	0
1	3	high	48	48	148	0
2	3	high	72	20	150	25
1	4	high	50	50	196	0
2	4	high	112	42	198	58
3	4	high	156	0	198	104
1	2	high	22	22	98	0
1	3	high	44	11	150	0
2	3	high	114	0	148	76
1	4	high	50	50	198	0
2	4	high	100	74	198	0
3	4	high	146	0	198	89
1	2	high	74	74	100	0
1	3	high	76	76	148	0
2	3	high	84	36	148	37
1	4	high	48	48	198	0
2	4	high	98	28	200	43
3	4	high	140	0	198	67
1	2	low	76	76	98	0
1	3	low	78	78	150	0
2	3	low	76	32	148	44

1	4	low	46	46	198	0
2	4	low	100	65	196	0
3	4	low	150	0	198	104
1	2	low	24	24	98	0
1	3	low	28	28	150	0
2	3	low	110	44	148	66
1	4	low	44	34	200	0
2	4	low	100	29	198	44
3	4	low	146	0	196	106
1	2	low	74	74	98	0
1	3	low	42	25	150	0
2	3	low	114	76	150	38
1	4	low	40	40	198	0
2	4	low	102	22	198	52
3	4	low	150	0	198	108
1	2	low	78	78	98	0
1	3	low	30	18	148	0
2	3	low	114	0	148	74
1	4	low	50	50	196	0
2	4	low	100	24	198	54
3	4	low	152	0	198	116
1	2	low	76	76	100	0
1	3	low	40	25	150	0
2	3	low	76	66	148	0
1	4	low	46	46	198	0
2	4	low	90	10	198	50
3	4	low	136	0	198	92
1	2	low	74	74	100	0
1	3	low	46	46	148	0
2	3	low	112	82	148	30
1	4	low	54	54	198	0
2	4	low	96	20	198	52
3	4	low	152	0	200	100
1	2	low	78	78	100	0
1	3	low	38	38	148	0
2	3	low	72	30	148	42
1	4	low	52	26	200	0
2	4	low	104	104	198	0
3	4	low	138	0	196	94
1	2	low	28	28	100	0
1	3	low	72	72	148	0
2	3	low	112	0	148	76
1	4	low	50	50	198	0
2	4	low	96	31	198	50
3	4	low	138	46	198	92
1	2	low	24	24	100	0

1	3	low	44	44	150	0
2	3	low	76	42	150	34
1	4	low	48	48	198	0
2	4	low	110	29	198	71
3	4	low	154	0	196	96
1	2	low	26	26	100	0
1	3	low	70	70	148	0
2	3	low	66	30	148	36
1	4	low	50	50	196	0
2	4	low	94	20	198	46
3	4	low	138	0	198	94
1	2	high	24	24	100	0
1	3	high	32	22	148	4
2	3	high	70	25	148	24
1	4	high	52	52	198	0
2	4	high	106	28	200	52
3	4	high	142	0	200	62
1	2	high	26	26	100	0
1	3	high	34	18	148	0
2	3	high	106	0	148	72
1	4	high	42	42	196	0
2	4	high	98	24	198	60
3	4	high	148	48	198	100
1	2	high	20	20	98	0
1	3	high	44	31	148	1
2	3	high	110	0	150	74
1	4	high	38	35	200	0
2	4	high	98	84	198	0
3	4	high	150	0	200	98
1	2	high	80	80	98	0
1	3	high	62	62	148	0
2	3	high	118	84	148	34
1	4	high	46	35	198	0
2	4	high	86	22	198	50
3	4	high	154	0	198	88
1	2	high	18	18	98	0
1	3	high	80	60	148	0
2	3	high	68	22	150	24
1	4	high	48	48	198	0
2	4	high	110	76	198	0
3	4	high	146	0	198	85
1	2	high	28	28	100	0
1	3	high	74	74	150	0
2	3	high	72	29	148	22
1	4	high	58	56	198	0
2	4	high	98	78	198	0

3	4	high	134	28	196	51
1	2	high	78	78	100	0
1	3	high	44	44	148	0
2	3	high	80	32	148	30
1	4	high	54	53	198	1
2	4	high	112	30	198	50
3	4	high	148	0	198	83
1	2	high	70	70	100	0
1	3	high	70	68	148	2
2	3	high	114	78	150	36
1	4	high	44	43	198	1
2	4	high	96	72	198	0
3	4	high	156	60	198	96
1	2	high	72	72	98	0
1	3	high	32	16	148	0
2	3	high	106	0	148	64
1	4	high	54	54	198	0
2	4	high	112	40	200	56
3	4	high	150	0	198	98
1	2	high	78	78	100	0
1	3	high	42	25	148	1
2	3	high	108	0	148	68
1	4	high	46	46	198	0
2	4	high	92	59	200	0
3	4	high	154	0	200	94
1	2	high	74	74	98	0
1	3	high	80	80	148	0
2	3	high	78	32	148	35
1	4	high	50	50	198	0
2	4	high	86	22	198	48
3	4	high	160	28	198	119
1	2	high	22	22	98	0
1	3	high	40	40	148	0
2	3	high	110	0	150	78
1	4	high	50	20	198	0
2	4	high	100	54	198	46
3	4	high	150	54	198	96
1	2	high	28	28	98	0
1	3	high	40	28	148	0
2	3	high	110	0	148	78
1	4	high	40	32	198	0
2	4	high	102	26	198	56
3	4	high	158	56	198	91
1	2	high	80	80	100	0
1	3	high	74	74	148	0
2	3	high	96	0	150	60

1	4	high	48	48	200	0
2	4	high	112	38	198	50
3	4	high	140	37	198	66
1	2	high	24	24	100	0
1	3	high	46	25	148	0
2	3	high	64	24	148	20
1	4	high	54	35	198	0
2	4	high	88	50	200	0
3	4	high	146	0	198	102
1	2	high	68	68	100	0
1	3	high	40	14	150	0
2	3	high	120	48	150	72
1	4	high	44	44	198	0
2	4	high	88	26	198	48
3	4	high	150	40	198	110
1	2	high	26	26	98	0
1	3	high	82	82	148	0
2	3	high	76	24	148	25
1	4	high	54	54	198	0
2	4	high	98	36	198	42
3	4	high	144	51	198	77
1	2	high	18	18	98	0
1	3	high	30	30	148	0
2	3	high	76	33	148	33
1	4	high	42	42	200	0
2	4	high	100	24	198	48
3	4	high	152	0	198	82
1	2	high	26	26	100	0
1	3	high	36	12	148	0
2	3	high	76	26	148	39
1	4	high	50	50	198	0
2	4	high	98	30	198	52
3	4	high	148	0	198	66
1	2	high	76	76	98	0
1	3	high	44	25	150	0
2	3	high	76	37	148	32
1	4	high	42	40	198	0
2	4	high	112	42	198	54
3	4	high	142	39	200	80
1	2	low	82	82	100	0
1	3	low	38	38	148	0
2	3	low	80	42	148	38
1	4	low	58	32	198	0
2	4	low	86	27	198	38
3	4	low	144	78	198	36
1	2	low	24	24	100	0

1	3	low	36	36	150	0
2	3	low	102	0	148	70
1	4	low	56	36	198	0
2	4	low	102	16	198	74
3	4	low	132	44	198	88
1	2	low	26	26	100	0
1	3	low	46	46	148	0
2	3	low	106	70	148	36
1	4	low	52	52	198	0
2	4	low	82	34	198	48
3	4	low	150	0	198	84
1	2	low	24	24	100	0
1	3	low	34	34	148	0
2	3	low	110	74	148	36
1	4	low	54	40	198	0
2	4	low	106	30	198	44
3	4	low	150	0	200	110
1	2	low	26	26	100	0
1	3	low	42	42	150	0
2	3	low	118	70	150	48
1	4	low	58	58	198	0
2	4	low	92	25	198	54
3	4	low	158	0	198	102
1	2	low	76	76	100	0
1	3	low	40	29	148	0
2	3	low	114	0	148	72
1	4	low	54	54	198	0
2	4	low	94	50	198	44
3	4	low	146	0	198	104
1	2	low	76	76	100	0
1	3	low	36	36	148	0
2	3	low	76	44	148	32
1	4	low	48	35	198	0
2	4	low	106	28	198	54
3	4	low	154	60	198	94
1	2	low	26	26	98	0
1	3	low	26	10	148	0
2	3	low	120	84	148	36
1	4	low	46	46	198	0
2	4	low	106	32	198	52
3	4	low	138	0	196	110
1	2	low	84	84	100	0
1	3	low	30	30	148	0
2	3	low	112	0	148	76
1	4	low	48	48	198	0
2	4	low	88	48	196	0

3	4	low	154	0	196	104
1	2	low	74	74	100	0
1	3	low	28	25	148	3
2	3	low	114	66	150	48
1	4	low	46	46	198	0
2	4	low	98	91	198	0
3	4	low	156	0	200	104
1	2	low	80	80	98	0
1	3	low	72	72	148	0
2	3	low	116	0	148	78
1	4	low	48	48	196	0
2	4	low	84	26	198	42
3	4	low	144	0	198	92
1	2	low	64	64	98	0
1	3	low	34	34	148	0
2	3	low	80	48	148	32
1	4	low	52	52	198	0
2	4	low	98	24	196	52
3	4	low	146	52	200	94
1	2	low	26	26	100	0
1	3	low	30	27	150	3
2	3	low	76	38	148	38
1	4	low	40	40	200	0
2	4	low	98	82	198	0
3	4	low	144	0	198	92
1	2	low	30	30	100	0
1	3	low	44	44	148	0
2	3	low	66	36	148	30
1	4	low	60	60	200	0
2	4	low	96	20	198	58
3	4	low	140	50	198	90
1	2	low	76	76	98	0
1	3	low	32	32	148	0
2	3	low	80	36	148	44
1	4	low	64	64	200	0
2	4	low	106	52	198	54
3	4	low	144	0	198	92
1	2	low	76	76	100	0
1	3	low	32	29	148	3
2	3	low	72	42	148	30
1	4	low	54	54	200	0
2	4	low	92	63	198	0
3	4	low	146	44	198	102
1	2	low	24	24	98	0
1	3	low	44	31	148	0
2	3	low	72	38	148	34

1	4	low	52	52	198	0
2	4	low	96	64	200	0
3	4	low	138	42	198	96
1	2	low	20	20	98	0
1	3	low	76	76	148	0
2	3	low	110	70	148	40
1	4	low	54	54	198	0
2	4	low	102	80	200	0
3	4	low	154	0	196	102
1	2	low	22	22	100	0
1	3	low	22	15	148	0
2	3	low	114	32	148	82
1	4	low	54	52	198	2
2	4	low	94	20	198	46
3	4	low	146	0	198	100
1	2	low	34	34	100	0
1	3	low	76	76	148	0
2	3	low	110	110	148	0
1	4	low	42	42	198	0
2	4	low	88	25	200	50
3	4	low	138	46	198	92
1	2	low	76	76	100	0
1	3	low	36	36	148	0
2	3	low	78	40	148	38
1	4	low	56	56	198	0
2	4	low	92	79	196	0
3	4	low	154	54	198	100
1	2	low	26	26	100	0
1	3	low	38	38	148	0
2	3	low	110	0	148	72
1	4	low	50	50	198	0
2	4	low	104	30	196	50
3	4	low	146	40	198	106
1	2	low	24	24	98	0
1	3	low	90	90	148	0
2	3	low	112	74	148	38
1	4	low	46	46	198	0
2	4	low	102	22	200	64
3	4	low	146	48	198	98
1	2	low	76	76	100	0
1	3	low	66	66	148	0
2	3	low	114	80	150	34
1	4	low	50	33	200	0
2	4	low	104	93	196	0
3	4	low	164	54	200	110
1	2	low	26	26	98	0

1	3	low	40	37	148	3
2	3	low	110	76	148	34
1	4	low	46	23	198	0
2	4	low	110	32	200	52
3	4	low	144	0	198	102
1	2	low	82	82	100	0
1	3	low	38	24	150	0
2	3	low	70	34	148	36
1	4	low	52	52	196	0
2	4	low	112	22	196	60
3	4	low	144	40	198	104
1	2	low	20	20	100	0
1	3	low	70	70	150	0
2	3	low	116	0	148	78
1	4	low	46	37	198	0
2	4	low	98	98	198	0
3	4	low	148	0	198	110
1	2	low	80	80	98	0
1	3	low	34	34	148	0
2	3	low	74	52	148	0
1	4	low	50	50	198	0
2	4	low	96	88	198	0
3	4	low	142	40	198	102
1	2	low	24	24	98	0
1	3	low	36	36	148	0
2	3	low	110	0	150	76
1	4	low	52	52	198	0
2	4	low	102	26	200	54
3	4	low	152	44	198	108
1	2	low	76	76	98	0
1	3	low	44	44	150	0
2	3	low	124	124	148	0
1	4	low	56	56	198	0
2	4	low	94	82	198	0
3	4	low	156	50	198	106
1	2	low	18	18	100	0
1	3	low	84	84	150	0
2	3	low	64	22	148	42
1	4	low	42	42	198	0
2	4	low	98	86	198	0
3	4	low	154	0	198	98
1	2	low	66	66	98	0
1	3	low	28	21	150	0
2	3	low	106	0	148	66
1	4	low	50	50	198	0
2	4	low	110	35	200	54

3	4	low	152	0	198	106
1	2	low	22	22	98	0
1	3	low	34	18	148	0
2	3	low	102	68	148	34
1	4	low	50	50	196	0
2	4	low	114	40	200	60
3	4	low	154	48	198	106
1	2	low	78	78	100	0
1	3	low	72	72	148	0
2	3	low	68	40	148	28
1	4	low	56	56	198	0
2	4	low	106	32	198	58
3	4	low	150	0	200	108
1	2	low	70	70	98	0
1	3	low	36	36	148	0
2	3	low	118	82	148	36
1	4	low	52	52	200	0
2	4	low	98	20	200	72
3	4	low	142	42	198	100
1	2	low	78	78	100	0
1	3	low	40	40	150	0
2	3	low	78	46	150	32
1	4	low	48	47	198	1
2	4	low	100	26	200	46
3	4	low	154	58	198	96
1	2	low	78	78	100	0
1	3	low	36	36	150	0
2	3	low	114	114	148	0
1	4	low	48	48	198	0
2	4	low	94	44	198	50
3	4	low	144	0	198	90
1	2	low	24	24	98	0
1	3	low	80	80	150	0
2	3	low	70	58	150	0
1	4	low	50	37	198	0
2	4	low	102	70	200	0
3	4	low	164	20	198	144
1	2	low	20	20	98	0
1	3	low	68	68	148	0
2	3	low	110	0	148	64
1	4	low	46	46	196	0
2	4	low	102	88	198	0
3	4	low	142	48	198	94
1	2	low	74	74	100	0
1	3	low	68	68	148	0
2	3	low	112	112	148	0

1	4	low	50	48	198	2
2	4	low	92	32	198	44
3	4	low	168	62	198	106
1	2	low	28	28	100	0
1	3	low	32	17	148	0
2	3	low	76	34	148	42
1	4	low	42	42	198	0
2	4	low	98	26	196	52
3	4	low	148	44	200	104
1	2	low	16	16	98	0
1	3	low	72	72	148	0
2	3	low	76	40	148	36
1	4	low	56	56	196	0
2	4	low	96	27	198	50
3	4	low	150	50	198	100
1	2	low	20	20	98	0
1	3	low	44	44	148	0
2	3	low	122	0	148	84
1	4	low	44	44	198	0
2	4	low	102	52	198	50
3	4	low	148	44	200	104
1	2	low	30	30	100	0
1	3	low	40	24	148	0
2	3	low	122	0	148	78
1	4	low	46	29	198	0
2	4	low	106	24	200	60
3	4	low	152	48	198	104
1	2	low	20	20	100	0
1	3	low	66	66	148	0
2	3	low	122	0	148	76
1	4	low	48	48	198	0
2	4	low	94	18	198	46
3	4	low	146	0	198	96
1	2	low	18	18	98	0
1	3	low	36	36	148	0
2	3	low	72	36	148	36
1	4	low	50	50	200	0
2	4	low	96	60	198	0
3	4	low	142	0	196	94
1	2	low	76	76	98	0
1	3	low	26	15	148	0
2	3	low	104	0	148	78
1	4	low	40	40	198	0
2	4	low	102	70	198	0
3	4	low	150	66	200	72
1	2	low	24	24	100	0

1	3	low	38	26	148	0
2	3	low	120	74	148	46
1	4	low	50	50	196	0
2	4	low	94	46	198	48
3	4	low	150	46	200	104
1	2	low	30	30	98	0
1	3	low	74	74	148	0
2	3	low	74	36	148	38
1	4	low	50	50	200	0
2	4	low	94	52	200	42
3	4	low	150	24	198	126
1	2	low	68	68	100	0
1	3	low	34	31	148	3
2	3	low	80	38	150	42
1	4	low	52	52	200	0
2	4	low	102	74	198	0
3	4	low	146	46	198	100
1	2	high	68	68	100	0
1	3	high	46	24	148	0
2	3	high	110	82	148	28
1	4	high	50	30	198	0
2	4	high	100	45	198	40
3	4	high	134	28	198	52
1	2	high	74	74	98	0
1	3	high	84	84	148	0
2	3	high	104	0	148	68
1	4	high	42	42	198	0
2	4	high	104	48	198	0
3	4	high	150	48	198	102
1	2	high	24	24	100	0
1	3	high	76	76	148	0
2	3	high	114	70	148	44
1	4	high	46	32	200	0
2	4	high	100	74	200	0
3	4	high	146	49	198	92
1	2	high	72	72	100	0
1	3	high	42	25	150	0
2	3	high	106	76	148	30
1	4	high	52	52	198	0
2	4	high	96	24	198	46
3	4	high	156	0	196	98
1	2	high	30	30	100	0
1	3	high	88	88	148	0
2	3	high	112	0	148	70
1	4	high	54	30	198	0
2	4	high	106	26	198	58

3	4	high	150	0	198	92
1	2	high	16	16	98	0
1	3	high	34	34	148	0
2	3	high	124	0	150	70
1	4	high	44	44	198	0
2	4	high	100	33	198	54
3	4	high	146	50	198	96
1	2	high	70	70	100	0
1	3	high	42	20	148	0
2	3	high	112	34	148	78
1	4	high	54	52	198	0
2	4	high	94	58	198	0
3	4	high	152	0	196	110
1	2	high	66	66	98	0
1	3	high	38	38	148	0
2	3	high	68	25	148	14
1	4	high	52	32	198	0
2	4	high	100	72	198	0
3	4	high	150	48	198	102
1	2	high	74	74	98	0
1	3	high	40	18	148	0
2	3	high	108	68	150	40
1	4	high	46	28	198	0
2	4	high	96	54	198	42
3	4	high	156	0	198	106
1	2	high	30	30	100	0
1	3	high	34	34	148	0
2	3	high	110	70	148	40
1	4	high	50	50	198	0
2	4	high	106	66	198	0
3	4	high	146	0	198	104
1	2	high	24	24	98	0
1	3	high	36	23	148	0
2	3	high	114	0	148	78
1	4	high	50	46	198	0
2	4	high	90	68	198	0
3	4	high	158	0	200	106
1	2	high	16	16	100	0
1	3	high	46	46	148	0
2	3	high	116	0	148	82
1	4	high	50	37	198	0
2	4	high	104	39	198	54
3	4	high	158	54	198	104
1	2	high	24	24	100	0
1	3	high	34	18	148	0
2	3	high	112	0	150	76

1	4	high	54	36	198	0
2	4	high	94	40	198	54
3	4	high	152	0	200	102
1	2	high	70	70	100	0
1	3	high	42	9	148	0
2	3	high	84	37	148	37
1	4	high	58	33	196	0
2	4	high	90	22	198	56
3	4	high	152	56	198	96
1	2	high	24	24	98	0
1	3	high	40	24	148	2
2	3	high	80	36	150	29
1	4	high	48	35	198	0
2	4	high	96	42	198	54
3	4	high	148	56	198	92
1	2	high	70	70	98	0
1	3	high	40	21	148	1
2	3	high	80	34	150	30
1	4	high	46	38	198	0
2	4	high	96	40	198	56
3	4	high	152	0	198	98
1	2	high	30	30	100	0
1	3	high	36	17	148	0
2	3	high	112	0	150	74
1	4	high	46	46	198	0
2	4	high	104	84	198	0
3	4	high	154	49	198	90
1	2	high	74	74	100	0
1	3	high	80	80	148	0
2	3	high	118	118	148	0
1	4	high	50	30	200	1
2	4	high	96	40	198	41
3	4	high	152	39	200	76
1	2	high	24	24	100	0
1	3	high	42	31	148	0
2	3	high	110	76	150	34
1	4	high	48	48	198	0
2	4	high	100	20	196	58
3	4	high	144	0	196	69
1	2	high	22	22	98	0
1	3	high	70	70	148	0
2	3	high	76	19	148	29
1	4	high	50	29	198	0
2	4	high	94	19	198	34
3	4	high	160	0	198	106
1	2	high	18	18	100	0

1	3	high	34	18	150	3
2	3	high	78	24	148	24
1	4	high	52	52	200	0
2	4	high	96	21	198	37
3	4	high	154	0	198	65
1	2	high	74	74	100	0
1	3	high	38	38	148	0
2	3	high	116	82	150	34
1	4	high	52	52	200	0
2	4	high	98	32	198	40
3	4	high	144	0	198	50
1	2	high	72	72	100	0
1	3	high	40	23	148	3
2	3	high	112	30	150	82
1	4	high	58	58	198	0
2	4	high	92	16	200	26
3	4	high	150	0	198	77
1	2	high	26	26	100	0
1	3	high	90	90	148	0
2	3	high	70	19	150	19
1	4	high	38	37	198	1
2	4	high	108	28	198	78
3	4	high	154	0	198	95
1	2	high	0	0	100	0
1	3	high	72	72	148	0
2	3	high	72	14	148	13
1	4	high	44	44	198	0
2	4	high	96	27	198	38
3	4	high	40	3	178	5
1	2	high	20	20	98	0
1	3	high	48	22	150	0
2	3	high	27	0	97	0
1	4	high	18	0	165	0
2	4	high	106	52	200	0
3	4	high	138	0	198	66
1	2	high	30	30	98	0
1	3	high	32	17	148	0
2	3	high	0	0	42	0
1	4	high	48	25	200	0
2	4	high	98	8	200	26
3	4	high	144	0	198	52
1	2	high	76	76	100	0
1	3	high	36	30	148	0
2	3	high	90	35	150	31
1	4	high	56	56	198	0
2	4	high	108	55	198	0

3	4	high	154	44	196	89
1	2	high	76	76	98	0
1	3	high	74	74	148	0
2	3	high	76	19	150	21
1	4	high	52	52	196	0
2	4	high	100	24	196	58
3	4	high	148	38	198	59
1	2	high	70	70	98	0
1	3	high	38	38	148	0
2	3	high	114	80	150	34
1	4	high	54	0	198	0
2	4	high	65	0	164	0
3	4	high	51	0	164	0
1	2	high	78	78	100	0
1	3	high	74	74	148	0
2	3	high	108	74	150	34
1	4	high	46	45	198	1
2	4	high	100	30	198	46
3	4	high	146	0	196	59
1	2	high	76	76	98	0
1	3	high	36	17	148	0
2	3	high	116	32	148	84
1	4	high	42	34	198	0
2	4	high	92	17	198	39
3	4	high	148	0	196	100
1	2	high	18	18	98	0
1	3	high	36	36	150	0
2	3	high	66	4	147	2
1	4	high	48	48	198	0
2	4	high	86	36	198	50
3	4	high	154	44	198	110
1	2	high	74	74	98	0
1	3	high	38	38	148	0
2	3	high	108	0	150	68
1	4	high	50	32	198	0
2	4	high	94	30	198	52
3	4	high	136	0	198	31
1	2	high	28	28	100	0
1	3	high	36	22	148	0
2	3	high	60	17	148	19
1	4	high	50	50	200	0
2	4	high	98	25	195	53
3	4	high	0	0	0	0
1	2	high	0	0	0	0
1	2	high	24	24	98	0
1	3	high	28	21	148	0

2	3	high	62	16	148	11
1	4	high	54	54	198	0
2	4	high	96	22	200	30
3	4	high	156	50	200	92
1	2	high	22	22	100	0
1	3	high	36	15	148	0
2	3	high	66	22	150	17
1	4	high	44	44	196	0
2	4	high	96	26	198	50
3	4	high	150	0	200	104
1	2	high	22	22	100	0
1	3	high	76	76	148	0
2	3	high	108	74	148	34
1	4	high	52	39	198	0
2	4	high	96	28	196	46
3	4	high	144	23	198	45
1	2	high	22	22	98	0
1	3	high	40	20	150	0
2	3	high	110	0	150	68
1	4	high	19	0	188	0
2	4	high	89	7	198	19
3	4	high	148	0	198	56
1	2	high	24	24	100	0
1	3	high	38	20	150	4
2	3	high	82	33	148	30
1	4	high	50	29	198	0
2	4	high	96	69	198	0
3	4	high	148	25	198	65
1	2	high	70	70	100	0
1	3	high	36	12	148	7
2	3	high	112	0	148	78
1	4	high	52	52	198	0
2	4	high	98	45	198	38
3	4	high	140	29	198	49
1	2	high	26	26	100	0
1	3	high	82	82	148	0
2	3	high	114	0	150	78
1	4	high	50	34	198	0
2	4	high	88	17	196	39
3	4	high	152	0	198	77
1	2	high	66	66	98	0
1	3	high	36	20	148	2
2	3	high	120	46	148	74
1	4	high	50	36	198	0
2	4	high	92	24	196	42
3	4	high	148	46	198	78

1	2	high	26	26	100	0
1	3	high	72	72	148	0
2	3	high	74	17	150	19
1	4	high	52	52	200	0
2	4	high	100	79	198	0
3	4	high	142	0	196	43
1	2	high	16	16	100	0
1	3	high	82	56	150	0
2	3	high	116	70	150	46
1	4	high	54	15	196	0
2	4	high	84	14	198	0
3	4	high	148	29	200	75
1	2	high	80	80	100	0
1	3	high	72	72	148	0
2	3	high	104	0	148	60
1	4	high	46	46	198	0
2	4	high	88	44	198	44
3	4	high	146	34	198	55
1	2	high	68	68	98	0
1	3	high	44	23	148	0
2	3	high	120	45	148	34
1	4	high	53	52	198	0
2	4	high	108	35	198	51
3	4	high	146	0	198	90
1	2	high	26	26	100	0
1	3	high	42	23	148	0
2	3	high	66	15	148	19
1	4	high	48	48	198	0
2	4	high	94	38	198	44
3	4	high	152	0	198	100
1	2	high	80	80	100	0
1	3	high	82	82	148	0
2	3	high	68	17	148	19
1	4	high	40	32	198	0
2	4	high	102	52	198	50
3	4	high	156	56	198	100
1	2	low	30	30	100	0
1	3	low	70	70	148	0
2	3	low	108	0	148	74
1	4	low	52	52	196	0
2	4	low	104	34	200	70
3	4	low	146	0	200	82
1	2	low	78	78	100	0
1	3	low	40	27	148	0
2	3	low	122	0	150	84
1	4	low	52	52	198	0

2	4	low	104	22	200	54
3	4	low	152	50	200	102
1	2	low	14	14	98	0
1	3	low	80	80	150	0
2	3	low	66	30	148	36
1	4	low	50	50	198	0
2	4	low	102	47	198	0
3	4	low	148	42	198	106
1	2	low	20	20	100	0
1	3	low	36	36	148	0
2	3	low	118	0	148	82
1	4	low	42	42	198	0
2	4	low	98	84	198	0
3	4	low	148	84	200	64
1	2	low	26	26	98	0
1	3	low	72	72	148	0
2	3	low	108	68	148	40
1	4	low	58	58	196	0
2	4	low	108	24	198	62
3	4	low	148	52	200	96
1	2	low	72	72	100	0
1	3	low	66	66	148	0
2	3	low	118	86	148	32
1	4	low	52	52	198	0
2	4	low	108	73	196	0
3	4	low	152	52	198	100
1	2	low	24	24	100	0
1	3	low	36	21	148	0
2	3	low	110	76	148	34
1	4	low	44	44	198	0
2	4	low	100	72	198	0
3	4	low	156	52	198	104
1	2	low	76	76	100	0
1	3	low	76	76	148	0
2	3	low	108	76	148	32
1	4	low	50	50	198	0
2	4	low	198	0	198	0
3	4	low	162	0	198	114
1	2	low	82	82	100	0
1	3	low	70	70	148	0
2	3	low	116	0	148	82
1	4	low	42	42	198	0
2	4	low	94	26	198	42
3	4	low	150	60	198	90
1	2	low	28	28	100	0
1	3	low	30	20	150	0

2	3	low	68	34	148	34
1	4	low	50	50	198	0
2	4	low	104	58	198	46
3	4	low	144	0	198	98
1	2	low	76	76	98	0
1	3	low	38	21	150	0
2	3	low	106	0	148	66
1	4	low	40	40	198	0
2	4	low	102	12	200	58
3	4	low	152	58	196	94
1	2	low	72	72	100	0
1	3	low	38	28	148	0
2	3	low	64	52	148	0
1	4	low	46	46	200	0
2	4	low	96	23	198	56
3	4	low	146	54	200	92
1	2	low	78	78	100	0
1	3	low	42	40	148	2
2	3	low	60	32	148	28
1	4	low	48	48	198	0
2	4	low	108	108	198	0
3	4	low	138	0	198	35
1	2	low	68	68	100	0
1	3	low	46	22	148	0
2	3	low	80	40	148	40
1	4	low	62	46	198	0
2	4	low	104	26	198	52
3	4	low	156	22	198	66
1	2	low	72	72	100	0
1	3	low	78	78	148	0
2	3	low	120	78	150	42
1	4	low	60	60	198	0
2	4	low	96	89	198	0
3	4	low	134	0	198	90
1	2	low	78	78	100	0
1	3	low	46	36	148	0
2	3	low	110	110	150	0
1	4	low	56	56	198	0
2	4	low	110	76	198	0
3	4	low	150	0	198	79
1	2	low	64	64	98	0
1	3	low	86	86	148	0
2	3	low	116	0	148	80
1	4	low	44	44	198	0
2	4	low	94	21	198	50
3	4	low	150	66	198	84

1	2	low	72	72	98	0
1	3	low	68	68	148	0
2	3	low	116	0	148	72
1	4	low	50	37	198	0
2	4	low	92	14	198	48
3	4	low	146	48	198	98
1	2	low	22	22	98	0
1	3	low	32	13	148	0
2	3	low	106	0	148	70
1	4	low	50	47	200	3
2	4	low	104	66	198	26
3	4	low	138	44	198	94
1	2	low	22	22	100	0
1	3	low	80	80	150	0
2	3	low	114	70	148	44
1	4	low	52	52	198	0
2	4	low	90	12	200	54
3	4	low	160	22	196	138
1	2	low	72	72	100	0
1	3	low	76	76	148	0
2	3	low	116	0	148	74
1	4	low	46	29	198	0
2	4	low	110	34	198	48
3	4	low	154	48	198	106
1	2	low	76	76	98	0
1	3	low	78	78	148	0
2	3	low	110	0	148	70
1	4	low	54	53	198	1
2	4	low	112	26	198	58
3	4	low	144	0	198	82
1	2	low	78	78	98	0
1	3	low	76	76	148	0
2	3	low	114	0	148	72
1	4	low	48	48	198	0
2	4	low	102	40	198	48
3	4	low	154	12	200	66
1	2	low	72	72	98	0
1	3	low	38	38	150	0
2	3	low	58	50	148	0
1	4	low	48	48	200	0
2	4	low	94	28	198	46
3	4	low	158	56	196	102
1	2	low	80	80	98	0
1	3	low	88	88	148	0
2	3	low	70	36	148	34
1	4	low	52	52	196	0

2	4	low	110	26	198	56
3	4	low	146	0	198	96
1	2	low	80	80	98	0
1	3	low	44	44	148	0
2	3	low	112	0	148	78
1	4	low	44	44	198	0
2	4	low	98	69	198	0
3	4	low	156	0	198	108
1	2	low	76	76	98	0
1	3	low	84	84	148	0
2	3	low	102	64	148	38
1	4	low	48	29	198	0
2	4	low	84	20	198	48
3	4	low	146	0	198	98
1	2	low	72	72	98	0
1	3	low	34	19	150	0
2	3	low	108	72	148	36
1	4	low	54	54	198	0
2	4	low	98	26	198	54
3	4	low	152	54	198	98
1	2	low	34	34	98	0
1	3	low	72	72	148	0
2	3	low	104	76	148	28
1	4	low	56	56	198	0
2	4	low	94	32	198	50
3	4	low	142	60	198	62
1	2	low	24	24	98	0
1	3	low	42	39	150	3
2	3	low	110	72	148	38
1	4	low	58	58	196	0
2	4	low	102	68	200	0
3	4	low	156	0	198	138
1	2	low	24	24	98	0
1	3	low	82	82	148	0
2	3	low	116	80	150	36
1	4	low	42	26	198	0
2	4	low	96	22	198	52
3	4	low	140	0	198	94
1	2	low	28	28	100	0
1	3	low	38	22	150	0
2	3	low	118	0	150	78
1	4	low	58	58	198	0
2	4	low	102	74	198	0
3	4	low	140	0	198	81
1	2	low	22	22	100	0
1	3	low	36	27	148	0

2	3	low	120	0	148	80
1	4	low	40	40	198	0
2	4	low	96	76	198	0
3	4	low	152	0	198	100
1	2	low	24	24	98	0
1	3	low	32	32	150	0
2	3	low	102	68	148	34
1	4	low	44	44	198	0
2	4	low	102	44	198	58
3	4	low	156	58	198	98
1	2	low	30	30	98	0
1	3	low	34	21	148	0
2	3	low	108	70	150	38
1	4	low	50	50	198	0
2	4	low	96	22	198	48
3	4	low	150	58	196	92
1	2	low	20	20	98	0
1	3	low	48	32	148	0
2	3	low	96	72	148	24
1	4	low	52	52	198	0
2	4	low	96	56	198	0
3	4	low	140	42	198	98
1	2	low	24	24	98	0
1	3	low	38	24	148	0
2	3	low	120	30	150	90
1	4	low	38	38	200	0
2	4	low	96	24	198	46
3	4	low	142	54	198	88
1	2	low	22	22	100	0
1	3	low	78	78	148	0
2	3	low	66	32	148	34
1	4	low	52	52	198	0
2	4	low	102	24	198	58
3	4	low	158	0	200	108
1	2	low	80	80	98	0
1	3	low	30	22	148	0
2	3	low	106	82	148	24
1	4	low	48	48	198	0
2	4	low	94	22	196	40
3	4	low	140	0	198	94
1	2	low	26	26	98	0
1	3	low	28	26	148	2
2	3	low	116	90	148	26
1	4	low	48	48	198	0
2	4	low	96	28	198	40
3	4	low	154	0	198	96

1	2	low	76	76	100	0
1	3	low	72	72	148	0
2	3	low	110	78	148	32
1	4	low	38	38	198	0
2	4	low	108	18	198	52
3	4	low	146	0	198	86
1	2	low	28	28	98	0
1	3	low	80	59	148	0
2	3	low	80	38	150	42
1	4	low	46	46	198	0
2	4	low	102	80	198	0
3	4	low	152	0	198	100
1	2	low	66	66	100	0
1	3	low	34	34	150	0
2	3	low	112	72	150	40
1	4	low	36	36	198	0
2	4	low	102	86	196	0
3	4	low	152	0	198	100
1	2	low	76	76	98	0
1	3	low	30	17	148	0
2	3	low	114	90	150	24
1	4	low	44	30	200	0
2	4	low	106	46	198	60
3	4	low	160	80	196	54
1	2	low	74	74	98	0
1	3	low	44	44	148	0
2	3	low	112	0	148	80
1	4	low	48	48	198	0
2	4	low	106	30	198	50
3	4	low	160	37	196	91
1	2	low	22	22	98	0
1	3	low	34	21	148	0
2	3	low	72	30	148	42
1	4	low	42	23	198	0
2	4	low	100	48	200	52
3	4	low	150	50	198	100
1	2	low	24	24	100	0
1	3	low	40	26	148	0
2	3	low	82	40	148	42
1	4	low	48	30	200	0
2	4	low	100	85	198	0
3	4	low	138	52	198	86
1	2	low	74	74	98	0
1	3	low	82	82	148	0
2	3	low	116	0	148	70
1	4	low	48	29	200	0

2	4	low	86	59	198	0
3	4	low	144	0	200	65
1	2	low	70	70	98	0
1	3	low	76	76	148	0
2	3	low	120	76	148	44
1	4	low	66	66	198	0
2	4	low	102	78	196	0
3	4	low	152	52	196	100
1	2	low	78	47	98	0
1	3	low	40	40	150	0
2	3	low	110	0	148	76
1	4	low	44	37	198	0
2	4	low	92	81	200	0
3	4	low	150	52	198	98
1	2	low	82	82	100	0
1	3	low	38	16	150	0
2	3	low	110	82	148	28
1	4	low	66	64	198	2
2	4	low	108	33	198	56
3	4	low	146	50	200	96
1	2	low	78	78	100	0
1	3	low	76	76	148	0
2	3	low	108	0	150	68
1	4	low	58	58	196	0
2	4	low	102	74	198	0
3	4	low	148	0	200	106
1	2	low	30	30	100	0
1	3	low	26	23	148	3
2	3	low	104	0	148	74
1	4	low	50	50	198	0
2	4	low	102	87	200	0
3	4	low	152	0	198	98
1	2	low	20	20	98	0
1	3	low	62	62	148	0
2	3	low	116	76	148	40
1	4	low	54	34	198	0
2	4	low	92	30	198	42
3	4	low	150	42	198	108
1	2	low	30	30	98	0
1	3	low	84	84	148	0
2	3	low	122	84	148	38
1	4	low	56	56	196	0
2	4	low	92	36	198	38
3	4	low	150	0	200	96
1	2	low	74	74	100	0
1	3	low	34	27	148	0

2	3	low	122	0	148	82
1	4	low	56	55	198	1
2	4	low	104	86	198	0
3	4	low	156	52	198	104
1	2	low	24	24	100	0
1	3	low	76	76	148	0
2	3	low	84	46	150	38
1	4	low	54	53	198	1
2	4	low	102	52	196	50
3	4	low	152	54	200	98
1	2	low	26	26	98	0
1	3	low	46	46	148	0
2	3	low	76	32	148	44
1	4	low	44	34	198	0
2	4	low	94	78	198	0
3	4	low	142	0	196	92
1	2	low	18	18	100	0
1	3	low	46	30	148	0
2	3	low	110	72	148	38
1	4	low	46	46	198	0
2	4	low	96	24	198	46
3	4	low	154	60	198	94
1	2	low	18	18	100	0
1	3	low	44	24	150	0
2	3	low	76	40	150	36
1	4	low	46	46	198	0
2	4	low	92	87	198	0
3	4	low	134	0	198	88
1	2	low	80	80	100	0
1	3	low	78	78	150	0
2	3	low	68	28	148	32
1	4	low	54	30	198	0
2	4	low	100	51	198	0
3	4	low	146	32	200	114
1	2	low	72	22	98	0
1	3	low	40	40	148	0
2	3	low	106	0	150	40
1	4	low	64	64	198	0
2	4	low	104	88	196	0
3	4	low	140	41	196	65
1	2	low	66	66	98	0
1	3	low	76	76	148	0
2	3	low	104	78	148	26
1	4	low	52	52	198	0
2	4	low	104	52	198	52
3	4	low	156	50	196	106

1	2	low	74	74	100	0
1	3	low	42	42	148	0
2	3	low	78	40	148	36
1	4	low	54	33	196	0
2	4	low	98	54	198	44
3	4	low	158	0	198	44
1	2	low	30	30	100	0
1	3	low	68	9	148	0
2	3	low	108	68	148	40
1	4	low	42	16	198	0
2	4	low	102	26	198	54
3	4	low	152	70	198	64
1	2	low	77	0	101	0
1	3	low	30	14	148	0
2	3	low	94	3	149	0
1	4	low	51	44	198	0
2	4	low	140	11	202	0
3	4	low	270	0	198	0
1	2	low	26	0	0	0
1	3	low	75	0	89	0
2	3	low	123	0	86	0
1	4	low	52	3	167	0
2	4	low	49	0	112	0
3	4	low	114	0	76	0
1	2	low	26	0	48	0
1	3	low	85	0	128	0
2	3	low	27	0	47	0
1	4	low	324	0	546	0
2	4	low	978	0	213	0
3	4	low	658	11	198	75
1	2	low	32	32	98	0
1	3	low	239	23	148	0
2	3	low	563	92	148	24
1	4	low	46	44	198	3
2	4	low	82	60	198	0
3	4	low	146	0	198	102
1	2	low	28	28	100	0
1	3	low	74	74	148	0
2	3	low	72	36	148	36
1	4	low	44	35	198	0
2	4	low	98	85	198	0
3	4	low	158	64	198	70
1	2	low	80	80	100	0
1	3	low	64	64	148	0
2	3	low	104	0	148	64
1	4	low	48	48	198	0

2	4	low	92	28	198	48
3	4	low	154	0	198	106
1	2	low	72	72	98	0
1	3	low	38	27	148	0
2	3	low	116	0	148	72
1	4	low	48	48	198	0
2	4	low	102	28	196	46
3	4	low	150	0	198	106
1	2	low	20	20	98	0
1	3	low	40	36	148	4
2	3	low	82	38	148	44
1	4	low	50	50	198	0
2	4	low	104	29	196	60
3	4	low	146	46	198	100
1	2	low	74	74	98	0
1	3	low	70	70	148	0
2	3	low	112	39	150	26
1	4	low	44	44	198	0
2	4	low	92	24	198	44
3	4	low	148	0	198	98
1	2	low	30	30	100	0
1	3	low	36	33	148	3
2	3	low	116	116	148	0
1	4	low	44	44	196	0
2	4	low	110	24	200	48
3	4	low	148	46	198	102
1	2	low	76	76	100	0
1	3	low	34	34	148	0
2	3	low	102	74	148	28
1	4	low	46	46	200	0
2	4	low	90	42	198	48
3	4	low	156	58	198	98
1	2	low	26	26	100	0
1	3	low	72	72	148	0
2	3	low	78	42	148	36
1	4	low	42	39	198	3
2	4	low	98	84	198	0
3	4	low	146	0	198	100
1	2	low	74	74	100	0
1	3	low	32	18	148	0
2	3	low	72	38	150	34
1	4	low	52	32	198	0
2	4	low	102	50	198	52
3	4	low	150	0	198	88
1	2	low	22	22	98	0
1	3	low	42	42	148	0

2	3	low	98	98	148	0
1	4	low	54	54	200	0
2	4	low	90	18	198	46
3	4	low	148	0	200	98
1	2	low	24	24	98	0
1	3	low	74	74	148	0
2	3	low	72	34	148	38
1	4	low	44	44	198	0
2	4	low	94	26	198	44
3	4	low	138	0	196	96
1	2	low	80	80	98	0
1	3	low	42	42	148	0
2	3	low	72	56	148	0
1	4	low	56	56	198	0
2	4	low	88	21	198	42
3	4	low	142	50	198	92
1	2	low	28	28	98	0
1	3	low	74	74	148	0
2	3	low	116	0	148	76
1	4	low	50	50	196	0
2	4	low	106	28	198	54
3	4	low	154	0	198	90
1	2	low	22	22	98	0
1	3	low	84	84	148	0
2	3	low	116	0	150	88
1	4	low	48	48	198	0
2	4	low	94	26	198	40
3	4	low	146	50	196	96
1	2	low	70	70	100	0
1	3	low	32	32	148	0
2	3	low	114	0	150	80
1	4	low	44	32	196	0
2	4	low	96	68	198	0
3	4	low	158	0	200	102
1	2	low	30	30	100	0
1	3	low	46	28	148	0
2	3	low	68	34	148	34
1	4	low	48	32	198	0
2	4	low	88	75	198	0
3	4	low	138	44	196	94
1	2	low	70	70	98	0
1	3	low	28	28	148	0
2	3	low	114	0	148	74
1	4	low	56	38	198	0
2	4	low	98	30	198	42
3	4	low	146	40	198	106

1	2	low	24	24	100	0
1	3	low	30	30	148	0
2	3	low	72	38	148	34
1	4	low	52	52	196	0
2	4	low	100	70	196	0
3	4	low	154	0	198	106
1	2	low	74	74	100	0
1	3	low	72	72	148	0
2	3	low	108	72	148	36
1	4	low	44	21	198	0
2	4	low	122	18	198	34
3	4	low	150	10	198	66
1	2	low	24	24	100	0
1	3	low	78	61	150	0
2	3	low	70	48	150	0
1	4	low	50	50	198	0
2	4	low	98	34	198	44
3	4	low	136	48	198	88
1	2	low	30	30	100	0
1	3	low	40	29	148	0
2	3	low	64	30	148	34
1	4	low	48	24	200	0
2	4	low	108	28	198	56
3	4	low	158	62	198	96
1	2	low	76	76	98	0
1	3	low	40	40	148	0
2	3	low	100	42	148	58
1	4	low	50	50	198	0
2	4	low	102	26	198	50
3	4	low	156	0	198	104
1	2	low	66	66	100	0
1	3	low	38	38	148	0
2	3	low	110	110	148	0
1	4	low	52	52	198	0
2	4	low	104	93	198	0
3	4	low	146	58	198	66
1	2	low	70	70	98	0
1	3	low	68	68	148	0
2	3	low	118	82	148	36
1	4	low	56	56	198	0
2	4	low	98	23	196	63
3	4	low	158	0	200	110
1	2	low	16	16	98	0
1	3	low	36	19	148	0
2	3	low	122	80	148	42
1	4	low	52	52	198	0

2	4	low	92	42	198	50
3	4	low	154	10	198	144
1	2	low	76	76	98	0
1	3	low	36	35	150	1
2	3	low	72	36	148	36
1	4	low	44	44	198	0
2	4	low	104	30	198	58
3	4	low	152	70	198	56
1	2	low	32	32	98	0
1	3	low	48	23	148	0
2	3	low	104	0	150	72
1	4	low	38	38	198	0
2	4	low	100	76	198	0
3	4	low	140	0	198	102
1	2	low	24	24	100	0
1	3	low	40	37	148	3
2	3	low	74	38	150	36
1	4	low	52	31	200	0
2	4	low	110	28	198	56
3	4	low	160	15	200	42
1	2	low	74	74	100	0
1	3	low	72	72	148	0
2	3	low	72	38	148	34
1	4	low	50	50	196	0
2	4	low	98	26	198	40
3	4	low	152	0	196	98
1	2	low	32	32	100	0
1	3	low	38	22	150	0
2	3	low	80	40	148	40
1	4	low	58	58	198	0
2	4	low	92	33	198	48
3	4	low	140	0	198	100
1	2	low	26	26	98	0
1	3	low	84	84	150	0
2	3	low	102	62	148	40
1	4	low	50	50	198	0
2	4	low	100	67	198	0
3	4	low	148	48	198	100
1	2	low	74	74	98	0
1	3	low	46	45	150	1
2	3	low	100	66	148	34
1	4	low	46	31	198	0
2	4	low	106	28	200	56
3	4	low	148	0	200	104
1	2	low	74	74	100	0
1	3	low	74	74	150	0

2	3	low	68	40	148	28
1	4	low	52	52	198	0
2	4	low	110	90	198	0
3	4	low	156	0	200	98
1	2	low	72	72	100	0
1	3	low	72	72	148	0
2	3	low	72	40	148	32
1	4	low	54	35	198	0
2	4	low	98	88	198	0
3	4	low	154	42	198	112
1	2	low	72	72	100	0
1	3	low	76	76	148	0
2	3	low	120	0	148	72
1	4	low	52	52	196	0
2	4	low	96	28	198	44
3	4	low	150	40	198	110
1	2	low	32	32	100	0
1	3	low	76	76	148	0
2	3	low	108	0	148	74
1	4	low	54	54	198	0
2	4	low	96	26	198	52
3	4	low	144	52	198	92
1	2	low	76	76	98	0
1	3	low	80	80	148	0
2	3	low	104	68	148	36
1	4	low	48	48	198	0
2	4	low	90	28	198	44
3	4	low	136	44	196	92
1	2	low	20	20	98	0
1	3	low	70	70	148	0
2	3	low	114	80	148	34
1	4	low	46	34	198	0
2	4	low	92	20	196	44
3	4	low	156	0	198	96
1	2	low	76	76	98	0
1	3	low	40	26	148	0
2	3	low	108	74	150	34
1	4	low	54	54	198	0
2	4	low	108	54	198	54
3	4	low	154	52	198	102
1	2	low	76	76	100	0
1	3	low	86	86	148	0
2	3	low	108	0	150	78
1	4	low	46	46	198	0
2	4	low	100	26	200	54
3	4	low	152	0	198	92

1	2	low	70	70	100	0
1	3	low	78	78	148	0
2	3	low	118	0	148	78
1	4	low	42	42	198	0
2	4	low	96	28	196	46
3	4	low	142	0	200	86
1	2	low	26	26	98	0
1	3	low	78	78	148	0
2	3	low	110	40	148	70
1	4	low	46	33	198	0
2	4	low	94	80	198	0
3	4	low	152	0	200	104
1	2	low	24	24	98	0
1	3	low	30	30	150	0
2	3	low	116	46	148	70
1	4	low	42	38	198	4
2	4	low	98	72	198	0
3	4	low	150	66	196	84
1	2	low	22	22	98	0
1	3	low	72	72	148	0
2	3	low	78	38	148	40
1	4	low	38	38	198	0
2	4	low	104	22	198	56
3	4	low	150	52	198	98
1	2	low	78	78	98	0
1	3	low	44	42	150	2
2	3	low	80	60	148	0
1	4	low	52	52	198	0
2	4	low	96	36	198	38
3	4	low	144	42	200	102
1	2	low	72	72	98	0
1	3	low	64	64	148	0
2	3	low	116	0	148	74
1	4	low	52	52	198	0
2	4	low	110	52	198	58
3	4	low	134	52	198	82
1	2	low	72	72	98	0
1	3	low	84	84	148	0
2	3	low	110	0	148	76
1	4	low	44	44	198	0
2	4	low	92	50	200	0
3	4	low	152	46	198	106
1	2	low	24	24	98	0
1	3	low	34	34	150	0
2	3	low	78	58	148	0
1	4	low	50	50	198	0

2	4	low	102	32	200	54
3	4	low	152	50	198	102
1	2	low	26	26	100	0
1	3	low	78	78	148	0
2	3	low	72	40	148	32
1	4	low	48	47	198	1
2	4	low	96	35	198	44
3	4	low	146	12	198	134
1	2	low	30	30	98	0
1	3	low	68	68	150	0
2	3	low	104	72	148	32
1	4	low	50	50	198	0
2	4	low	102	14	198	76
3	4	low	150	58	198	92
1	2	low	78	78	100	0
1	3	low	78	78	150	0
2	3	low	74	36	150	38
1	4	low	50	50	198	0
2	4	low	102	32	200	46
3	4	low	140	0	196	84
1	2	low	28	28	98	0
1	3	low	80	80	150	0
2	3	low	116	0	148	84
1	4	low	54	54	198	0
2	4	low	102	33	198	48
3	4	low	164	58	198	106
1	2	low	76	76	98	0
1	3	low	38	38	150	0
2	3	low	68	38	148	30
1	4	low	52	38	198	0
2	4	low	102	90	200	0
3	4	low	154	0	198	98
1	2	low	74	74	100	0
1	3	low	44	27	150	0
2	3	low	112	86	148	26
1	4	low	54	54	198	0
2	4	low	102	83	198	0
3	4	low	160	0	198	96
1	2	low	74	74	98	0
1	3	low	76	76	150	0
2	3	low	116	76	148	40
1	4	low	34	26	198	0
2	4	low	102	81	200	0
3	4	low	140	0	198	100
1	2	low	24	24	100	0
1	3	low	34	34	148	0

2	3	low	68	26	148	42
1	4	low	42	42	198	0
2	4	low	98	25	198	41
3	4	low	132	26	198	106
1	2	low	22	22	98	0
1	3	low	38	27	148	0
2	3	low	108	0	148	64
1	4	low	52	52	200	0
2	4	low	106	90	200	0
3	4	low	150	84	198	50
1	2	low	66	66	100	0
1	3	low	72	72	148	0
2	3	low	86	52	150	34
1	4	low	46	31	198	0
2	4	low	98	46	198	52
3	4	low	142	0	198	130
1	2	low	22	22	98	0
1	3	low	36	14	148	0
2	3	low	72	38	148	34
1	4	low	44	43	198	1
2	4	low	98	60	198	0
3	4	low	140	56	198	84
1	2	low	20	20	100	0
1	3	low	78	78	148	0
2	3	low	72	30	148	42
1	4	low	44	44	198	0
2	4	low	94	40	200	54
3	4	low	140	54	198	62
1	2	low	28	28	98	0
1	3	low	84	84	148	0
2	3	low	112	80	148	32
1	4	low	46	26	198	0
2	4	low	106	26	200	52
3	4	low	146	46	198	100
1	2	high	76	76	100	0
1	3	high	70	48	148	0
2	3	high	120	0	150	80
1	4	high	56	55	198	1
2	4	high	114	76	200	0
3	4	high	162	0	198	100
1	2	high	26	26	100	0
1	3	high	42	20	150	0
2	3	high	116	80	148	36
1	4	high	48	30	198	0
2	4	high	104	48	198	56
3	4	high	142	0	198	65

1	2	high	76	76	100	0
1	3	high	38	20	148	2
2	3	high	72	29	148	24
1	4	high	54	54	198	0
2	4	high	96	40	198	46
3	4	high	160	0	200	106
1	2	high	28	28	98	0
1	3	high	30	21	148	1
2	3	high	110	0	148	66
1	4	high	30	30	196	0
2	4	high	92	72	198	0
3	4	high	146	0	200	73
1	2	high	22	22	98	0
1	3	high	42	27	148	1
2	3	high	120	0	148	90
1	4	high	38	37	196	1
2	4	high	114	78	198	0
3	4	high	142	0	198	78
1	2	high	24	24	98	0
1	3	high	80	48	150	0
2	3	high	68	20	148	22
1	4	high	62	38	198	0
2	4	high	102	20	198	56
3	4	high	150	48	198	102
1	2	high	72	72	100	0
1	3	high	66	66	148	0
2	3	high	116	32	148	84
1	4	high	40	40	198	0
2	4	high	104	22	198	54
3	4	high	146	0	198	88
1	2	high	24	24	98	0
1	3	high	28	17	148	3
2	3	high	72	16	148	24
1	4	high	50	35	200	0
2	4	high	98	30	200	44
3	4	high	154	52	198	102
1	2	high	68	68	100	0
1	3	high	40	24	148	0
2	3	high	108	66	148	42
1	4	high	45	0	200	0
2	4	high	93	0	184	0
3	4	high	113	0	194	14
1	2	high	68	68	98	0
1	3	high	44	25	148	1
2	3	high	112	74	148	38
1	4	high	54	28	196	0

2	4	high	98	68	198	0
3	4	high	162	56	198	106
1	2	high	70	70	98	0
1	3	high	40	25	148	0
2	3	high	102	0	148	68
1	4	high	46	30	198	0
2	4	high	112	90	198	0
3	4	high	144	47	198	94
1	2	high	24	24	98	0
1	3	high	40	13	148	4
2	3	high	35	0	140	0
1	4	high	52	50	198	2
2	4	high	102	22	200	52
3	4	high	144	46	198	85
1	2	high	28	28	100	0
1	3	high	72	72	148	0
2	3	high	120	76	150	44
1	4	high	56	42	198	0
2	4	high	94	37	198	50
3	4	high	144	52	196	92
1	2	high	20	20	98	0
1	3	high	38	38	148	0
2	3	high	80	34	148	34
1	4	high	42	39	198	0
2	4	high	94	58	198	0
3	4	high	142	0	198	101
1	2	high	74	74	100	0
1	3	high	34	34	148	0
2	3	high	80	33	150	28
1	4	high	50	14	198	0
2	4	high	98	56	198	38
3	4	high	148	0	198	100
1	2	high	70	70	100	0
1	3	high	72	72	148	0
2	3	high	116	84	148	32
1	4	high	48	45	198	0
2	4	high	104	22	198	50
3	4	high	150	52	198	98
1	2	high	78	78	98	0
1	3	high	80	80	148	0
2	3	high	110	110	148	0
1	4	high	46	46	200	0
2	4	high	92	32	196	46
3	4	high	148	0	198	94
1	2	high	20	20	98	0
1	3	high	74	74	148	0

2	3	high	104	74	148	30
1	4	high	54	32	200	1
2	4	high	100	41	198	50
3	4	high	146	50	198	96
1	2	high	72	72	98	0
1	3	high	62	62	148	0
2	3	high	86	38	148	48
1	4	high	54	32	198	0
2	4	high	66	0	192	16
3	4	high	4	0	200	2
1	2	high	74	74	98	0
1	3	high	4	0	130	0
2	3	high	122	84	148	38
1	4	high	50	50	198	0
2	4	high	100	75	198	0
3	4	high	154	54	200	100
1	2	high	18	18	98	0
1	3	high	36	16	150	2
2	3	high	104	66	148	38
1	4	high	34	34	196	0
2	4	high	100	30	198	54
3	4	high	140	28	198	61
1	2	high	72	72	100	0
1	3	high	32	22	148	3
2	3	high	114	0	148	76
1	4	high	54	26	198	0
2	4	high	92	28	198	42
3	4	high	82	0	160	12
1	2	high	26	26	98	0
1	3	high	78	78	150	0
2	3	high	74	15	147	18
1	4	high	46	46	198	0
2	4	high	94	28	198	40
3	4	high	134	31	198	43
1	2	high	18	18	98	0
1	3	high	30	13	150	5
2	3	high	76	24	148	30
1	4	high	54	54	200	0
2	4	high	100	72	198	0
3	4	high	154	50	198	104
1	2	high	72	72	100	0
1	3	high	42	24	148	1
2	3	high	110	0	148	66
1	4	high	54	53	198	1
2	4	high	96	74	198	0
3	4	high	140	0	200	59

1	2	high	70	70	100	0
1	3	high	42	18	148	3
2	3	high	70	18	148	30
1	4	high	58	42	198	0
2	4	high	86	48	198	0
3	4	high	152	50	198	102
1	2	high	68	68	100	0
1	3	high	38	19	150	2
2	3	high	118	78	150	40
1	4	high	56	23	198	0
2	4	high	94	31	198	48
3	4	high	140	0	198	60
1	2	high	20	20	100	0
1	3	high	36	10	147	0
2	3	high	108	0	148	72
1	4	high	30	0	196	0
2	4	high	94	37	196	44
3	4	high	144	0	198	62
1	2	high	26	26	100	0
1	3	high	32	20	148	1
2	3	high	78	25	150	33
1	4	high	40	40	198	0
2	4	high	86	58	198	0
3	4	high	156	0	198	70
1	2	high	80	80	100	0
1	3	high	9	0	144	0
2	3	high	0	0	102	0
1	4	high	36	0	200	0
2	4	high	160	3	207	13
3	4	high	335	16	196	86
1	2	high	74	74	100	0
1	3	high	62	29	151	0
2	3	high	74	28	150	24
1	4	high	50	50	198	0
2	4	high	110	20	198	58
3	4	high	148	0	198	110
1	2	high	78	78	100	0
1	3	high	36	18	148	0
2	3	high	68	22	148	22
1	4	high	54	46	198	0
2	4	high	98	30	198	48
3	4	high	146	19	200	66
1	2	high	72	72	98	0
1	3	high	38	28	150	0
2	3	high	74	17	148	26
1	4	high	48	34	198	0

2	4	high	94	34	198	46
3	4	high	154	0	198	96
1	2	high	34	34	98	0
1	3	high	36	23	148	3
2	3	high	74	29	148	23
1	4	high	48	29	200	0
2	4	high	100	42	198	44
3	4	high	138	27	196	66
1	2	high	80	80	98	0
1	3	high	38	26	148	0
2	3	high	72	28	148	23
1	4	high	52	52	198	0
2	4	high	106	27	198	64
3	4	high	158	0	198	110
1	2	high	22	22	98	0
1	3	high	36	19	150	0
2	3	high	106	68	148	38
1	4	high	9	1	195	0
2	4	high	92	35	196	44
3	4	high	146	38	198	96
1	2	high	28	28	100	0
1	3	high	36	17	148	2
2	3	high	110	0	150	74
1	4	high	60	60	196	0
2	4	high	100	28	198	47
3	4	high	152	0	198	87
1	2	high	76	76	100	0
1	3	high	40	20	148	0
2	3	high	106	0	148	74
1	4	high	46	46	200	0
2	4	high	92	16	198	44
3	4	high	156	26	198	49
1	2	high	80	80	98	0
1	3	high	38	14	148	0
2	3	high	112	76	148	36
1	4	high	48	41	198	1
2	4	high	102	76	198	0
3	4	high	142	28	200	76
1	2	high	76	76	100	0
1	3	high	42	28	148	0
2	3	high	106	0	148	76
1	4	high	52	38	200	0
2	4	high	96	76	198	0
3	4	high	138	34	196	67
1	2	high	72	72	98	0
1	3	high	46	46	148	0

2	3	high	114	80	148	34
1	4	high	36	36	198	0
2	4	high	102	39	198	44
3	4	high	144	0	198	70
1	2	high	74	74	98	0
1	3	high	32	23	150	2
2	3	high	104	0	148	74
1	4	high	46	46	200	0
2	4	high	108	24	198	44
3	4	high	148	40	200	88
1	2	high	82	82	98	0
1	3	high	80	80	148	0
2	3	high	74	34	150	20
1	4	high	44	44	198	0
2	4	high	102	28	198	42
3	4	high	152	50	198	102
1	2	high	22	22	100	0
1	3	high	26	26	148	0
2	3	high	118	78	148	40
1	4	high	52	28	198	0
2	4	high	88	23	198	48
3	4	high	150	48	198	102
1	2	high	20	20	100	0
1	3	high	76	76	148	0
2	3	high	106	72	148	34
1	4	high	56	39	196	0
2	4	high	90	36	196	44
3	4	high	150	0	198	96
1	2	high	26	26	100	0
1	3	high	44	28	148	4
2	3	high	112	0	150	72
1	4	high	54	54	200	0
2	4	high	90	52	198	38
3	4	high	156	0	200	108
1	2	high	76	76	98	0
1	3	high	30	30	150	0
2	3	high	70	26	150	20
1	4	high	60	45	198	0
2	4	high	96	26	198	54
3	4	high	146	0	198	96
1	2	high	26	26	98	0
1	3	high	80	50	148	0
2	3	high	110	0	148	70
1	4	high	44	44	198	0
2	4	high	92	28	198	50
3	4	high	158	0	198	106

1	2	high	28	28	100	0
1	3	high	38	20	148	4
2	3	high	106	0	148	78
1	4	high	48	48	198	0
2	4	high	96	12	198	47
3	4	high	148	0	198	90
1	2	high	76	76	100	0
1	3	high	42	31	148	0
2	3	high	82	35	148	39
1	4	high	36	26	196	0
2	4	high	96	68	198	0
3	4	high	150	0	198	94
1	2	high	74	74	98	0
1	3	high	46	24	148	0
2	3	high	114	0	148	78
1	4	high	50	29	198	0
2	4	high	94	12	198	46
3	4	high	152	58	198	94
1	2	high	26	26	98	0
1	3	high	32	15	148	0
2	3	high	114	0	148	80
1	4	high	52	52	196	0
2	4	high	96	26	198	44
3	4	high	150	0	198	112
1	2	high	80	80	100	0
1	3	high	42	30	148	0
2	3	high	122	0	148	84
1	4	high	52	36	198	0
2	4	high	112	34	198	56
3	4	high	136	0	198	53
1	2	high	74	74	98	0
1	3	high	32	23	150	0
2	3	high	110	64	150	46
1	4	high	48	32	198	0
2	4	high	88	66	198	0
3	4	high	148	0	198	100
1	2	high	74	74	100	0
1	3	high	40	20	148	2
2	3	high	108	66	148	42
1	4	high	62	62	200	0
2	4	high	104	24	198	48
3	4	high	148	47	198	98
1	2	high	26	26	98	0
1	3	high	40	24	148	0
2	3	high	80	39	148	37
1	4	high	50	50	196	0

2	4	high	98	10	198	55
3	4	high	154	0	198	108
1	2	high	20	20	98	0
1	3	high	64	44	148	0
2	3	high	106	72	150	34
1	4	high	50	50	198	0
2	4	high	100	42	198	58
3	4	high	148	52	200	96
1	2	high	26	26	98	0
1	3	high	32	10	150	0
2	3	high	108	70	148	38
1	4	high	48	48	198	0
2	4	high	100	71	196	0
3	4	high	117	1	200	0
1	2	high	26	0	96	0
1	3	high	78	78	148	0
2	3	high	116	0	148	82
1	4	high	50	47	198	0
2	4	high	94	22	198	56
3	4	high	146	0	198	83
1	2	high	26	26	100	0
1	3	high	36	36	150	0
2	3	high	108	0	148	78
1	4	high	46	46	198	0
2	4	high	98	20	198	48
3	4	high	154	0	200	96
1	2	high	74	74	100	0
1	3	high	40	17	148	0
2	3	high	112	0	148	76
1	4	high	54	31	198	0
2	4	high	88	8	196	48
3	4	high	154	50	200	104
1	2	high	22	22	98	0
1	3	high	38	38	150	0
2	3	high	72	31	148	21
1	4	high	42	28	198	0
2	4	high	96	24	198	42
3	4	high	140	0	198	69
1	2	high	26	26	98	0
1	3	high	28	28	148	0
2	3	high	80	31	148	34
1	4	high	40	40	196	0
2	4	high	100	26	200	46
3	4	high	148	46	198	79
1	2	high	28	28	100	0
1	3	high	82	82	148	0

2	3	high	102	0	148	70
1	4	high	60	60	196	0
2	4	high	96	74	196	0
3	4	high	136	0	200	57
1	2	high	28	28	100	0
1	3	high	38	21	148	0
2	3	high	78	34	148	28
1	4	high	52	32	198	0
2	4	high	98	38	196	52
3	4	high	144	48	198	86
1	2	high	70	70	98	0
1	3	high	36	19	148	0
2	3	high	112	0	148	72
1	4	high	40	28	198	0
2	4	high	96	42	198	0
3	4	high	148	58	198	90
1	2	high	26	26	98	0
1	3	high	30	17	148	0
2	3	high	64	25	148	17
1	4	high	50	35	198	0
2	4	high	106	32	198	48
3	4	high	148	0	200	92
1	2	high	82	82	100	0
1	3	high	78	78	148	0
2	3	high	116	0	150	80
1	4	high	46	46	196	0
2	4	high	96	33	198	52
3	4	high	158	66	200	92
1	2	high	76	76	100	0
1	3	high	76	76	148	0
2	3	high	110	42	148	68
1	4	high	48	48	198	0
2	4	high	98	27	198	50
3	4	high	154	56	200	98
1	2	high	26	26	98	0
1	3	high	38	16	150	0
2	3	high	72	21	148	31
1	4	high	54	26	198	0
2	4	high	94	82	196	0
3	4	high	154	0	198	100
1	2	high	26	26	98	0
1	3	high	38	32	148	1
2	3	high	114	0	148	80
1	4	high	56	56	198	0
2	4	high	106	39	198	52
3	4	high	152	52	198	100

1	2	high	28	28	100	0
1	3	high	36	27	150	0
2	3	high	116	0	148	84
1	4	high	42	42	200	0
2	4	high	100	68	196	0
3	4	high	152	56	196	96
1	2	high	22	22	100	0
1	3	high	26	18	148	0
2	3	high	100	0	148	66
1	4	high	50	50	198	0
2	4	high	94	17	198	58
3	4	high	142	0	200	70
1	2	high	26	26	100	0
1	3	high	38	17	150	0
2	3	high	72	26	148	26
1	4	high	50	31	198	0
2	4	high	114	38	200	54
3	4	high	150	0	198	98
1	2	high	76	76	100	0
1	3	high	74	48	148	0
2	3	high	118	0	150	80
1	4	high	54	54	198	0
2	4	high	108	46	198	62
3	4	high	148	50	198	98
1	2	high	32	32	100	0
1	3	high	72	72	148	0
2	3	high	104	78	148	26
1	4	high	44	44	198	0
2	4	high	94	20	198	43
3	4	high	150	48	200	102
1	2	high	28	28	98	0
1	3	high	72	72	148	0
2	3	high	114	0	150	82
1	4	high	52	30	196	0
2	4	high	100	22	198	47
3	4	high	136	0	196	59
1	2	high	76	76	98	0
1	3	high	44	23	148	0
2	3	high	110	70	148	40
1	4	high	52	52	198	0
2	4	high	100	34	198	52
3	4	high	160	58	198	102
1	2	high	76	76	100	0
1	3	high	42	24	150	0
2	3	high	80	22	148	29
1	4	high	50	50	198	0

2	4	high	100	28	198	46
3	4	high	152	0	198	87
1	2	high	72	72	100	0
1	3	high	30	20	148	4
2	3	high	104	0	150	70
1	4	high	44	44	198	0
2	4	high	104	26	198	48
3	4	high	152	0	200	102
1	2	high	76	76	100	0
1	3	high	76	76	148	0
2	3	high	76	26	148	30
1	4	high	42	42	198	0
2	4	high	100	25	200	56
3	4	high	156	50	198	106
1	2	high	24	24	100	0
1	3	high	70	70	148	0
2	3	high	72	22	150	29
1	4	high	56	56	198	0
2	4	high	90	28	198	36
3	4	high	154	0	198	104
1	2	high	74	74	98	0
1	3	high	40	15	148	0
2	3	high	102	0	148	68
1	4	high	36	36	198	0
2	4	high	102	12	199	0
3	4	high	134	16	196	61
1	2	high	70	70	100	0
1	3	high	32	16	148	3
2	3	high	126	88	148	38
1	4	high	40	40	198	0
2	4	high	86	70	196	0
3	4	high	142	35	198	84
1	2	high	22	22	100	0
1	3	high	36	36	148	0
2	3	high	80	24	150	20
1	4	high	56	56	198	0
2	4	high	96	13	196	34
3	4	high	136	18	198	28
1	2	high	76	76	100	0
1	3	high	38	15	150	3
2	3	high	78	30	148	33
1	4	high	42	42	198	0
2	4	high	82	44	196	38
3	4	high	138	33	196	57
1	2	high	82	82	98	0
1	3	high	68	32	148	0

2	3	high	98	0	148	70
1	4	high	58	57	198	1
2	4	high	96	24	198	52
3	4	high	154	46	196	108
1	2	high	70	70	100	0
1	3	high	32	32	148	0
2	3	high	68	21	148	26
1	4	high	54	34	198	0
2	4	high	96	50	198	0
3	4	high	144	45	198	59
1	2	high	72	72	100	0
1	3	high	38	15	148	3
2	3	high	96	0	148	68
1	4	high	40	29	196	1
2	4	high	96	34	198	46
3	4	high	152	46	198	106
1	2	high	20	20	98	0
1	3	high	72	72	148	0
2	3	high	114	0	148	70
1	4	high	56	41	198	0
2	4	high	102	51	198	50
3	4	high	150	0	198	108
1	2	high	28	28	100	0
1	3	high	40	40	148	0
2	3	high	78	27	148	26
1	4	high	52	23	198	0
2	4	high	102	41	198	50
3	4	high	152	12	198	140
1	2	high	70	70	98	0
1	3	high	64	64	148	0
2	3	high	112	84	150	28
1	4	high	52	33	198	0
2	4	high	106	72	200	0
3	4	high	156	52	198	104
1	2	high	20	20	98	0
1	3	high	82	56	148	0
2	3	high	74	25	148	26
1	4	high	48	46	198	0
2	4	high	90	62	198	0
3	4	high	150	0	198	98
1	2	high	18	18	98	0
1	3	high	38	16	150	0
2	3	high	112	0	148	72
1	4	high	52	52	198	0
2	4	high	94	68	198	0
3	4	high	150	50	198	100

1	2	high	72	72	98	0
1	3	high	80	80	148	0
2	3	high	66	27	150	0
1	4	high	46	24	198	0
2	4	high	92	24	198	43
3	4	high	146	0	200	91
1	2	high	70	70	98	0
1	3	high	84	84	148	0
2	3	high	116	84	148	32
1	4	high	50	49	196	1
2	4	high	100	17	198	54
3	4	high	134	0	198	60
1	2	high	30	30	98	0
1	3	high	50	33	148	0
2	3	high	110	0	148	76
1	4	high	56	44	198	0
2	4	high	92	28	198	31
3	4	high	154	52	198	102
1	2	high	22	22	100	0
1	3	high	78	49	148	0
2	3	high	76	27	148	27
1	4	high	46	46	198	0
2	4	high	90	22	197	49
3	4	high	154	0	198	98
1	2	high	70	70	100	0
1	3	high	38	38	148	0
2	3	high	106	0	148	72
1	4	high	46	46	198	0
2	4	high	114	78	198	0
3	4	high	146	0	198	88
1	2	high	82	82	98	0
1	3	high	38	27	150	2
2	3	high	84	32	148	34
1	4	high	50	50	200	0
2	4	high	102	84	198	0
3	4	high	140	0	198	47