Crossroads

Karel Kubat
e-tunity

2005

Abstract

Crossroads is a load balance and fail over utility for TCP based services. It is a daemon program running in user space, and features extensive configurability, polling of back ends using 'wakeup calls', detailed status reporting, 'hooks' for special actions when backend calls fail, and much more. Crossroads is service-independent: it is usable for HTTP(S), SSH, SMTP, DNS, etc.

Contents

1 Introduction **3**
1.1 Obtaining Crossroads . . . 3
1.2 Copyright and Disclaimer . . . 3
1.3 Terminology . . . 3

2 Using Crossroads **4**

3 The configuration **5**
3.1 General language elements . . . 5
3.1.1 Empty lines and comments . . . 5
3.1.2 Keywords, numbers, identifiers, generic strings . . . 5
3.2 Service definitions . . . 6
3.3 Backend definitions . . . 8
3.4 How back ends are selected in load balancing . . . 11
3.4.1 Bysize or byduration . . . 11
3.4.2 Averaging size and duration . . . 11
3.4.3 Specifying decays . . . 11
3.4.4 Adjusting the weights . . . 12
3.4.5 Throttling the number of concurrent sessions . . . 13
3.5 Configuration examples . . . 13
3.5.1 A load balancer for three webserver back ends . . . 13
3.5.2 A HTTP forwarder when travelling . . . 16
3.5.3 SSH login with enforced idle logout . . . 17

4 Benchmarking **17**
4.1 Environment . . . 17
4.2 Results . . . 18
4.3 Discussion . . . 18

5 Compiling and Installing **18**
5.1 Prerequisites 18
5.2 Compiling and installing 19
5.3 Configuring crossroads 19
5.4 A boot script 20
5.4.1 SysV Style Startup 20
5.4.2 BSD Style Startup 20

1 Introduction

Crossroads is a daemon that basically accepts TCP connections at preconfigured ports, and given a list of 'back ends' distributes each incoming connection, so that a client process is served. Additionally, crossroads maintains an internal administration of the back end connectivity: if a back end isn't usable, then the client request is handled using another back end. Crossroads will then periodically check whether a previously not usable back end has come to life yet. Also, crossroads can select back ends by estimating the load, so that balancing is achieved.

Using this approach, crossroads serves as load balancer and fail over utility. Crossroads will very likely not be as reliable as hardware based balancers, since it always will require a server to run on. This server, in turn, may become a new Single Point of Failure (SPOS). However, in situations where cost efficiency is an issue, crossroads may be a good choice. Furthermore, crossroads can be deployed in situations where a hardware based balancing already exists and augmenting service reliability is needed. Or, crossroads may be run off a diskless system, which again improves reliability of the underlying hardware.

This document describes how to use crossroads, how to configure it in order to increase the reliability of your systems, and how to compile the program from its sources.

1.1 Obtaining Crossroads

As quick reference, here are some important URL's for Crossroads:

- `http://public.e-tunity.com` is the site that serves Crossroads and other packages. You can browse this at leisure for documentation, sources, and so on.
- `http://public.e-tunity.com/crossroads/crossroads-latest.tar.gz` is the 'latest' distribution archive. Instead of *latest* you can substitute a specific version, such as *0.07*. Too many older versions however aren't stored.
- `http://public.e-tunity.com/crossroads/crossroads.html` is the documentation in HTML format (this text). Substitute *.pdf* for *.html* to get the documentation in PDF format.

1.2 Copyright and Disclaimer

Crossroads is distributed as-is, without assumptions of fitness or usability. You are free to use crossroads to your liking. It's free, and as with everything that's free: there's also no warranty.

You are allowed to make modifications to the source code of crossroads, and you are allowed to (re)distribute crossroads, as long as you include this text, all sources, and if applicable: all your modifications, with each distribution.

While you are allowed to make any and all changes to the sources, I would appreciate hearing about them. If the changes concern new functionality or bugfixes, then I'll include them in a next release, stating full credits.

1.3 Terminology

Throughout this document, the following terms are used: [1]

A client is a process that initiates a network connection to get contact with some service.

A service or **server process** is a central application that accepts network connections from clients and sevices them. Based on its configuration, crossroads will start one or more services that wait for clients to connect.

[1] Many more meanings of the terms will exist – yes, I am aware of that. I'm using the terms here in a very strict sense.

Back ends are locations where crossroads looks in order to service its clients. Crossroads sits 'in between' and does its tricks. Therefore, as far as the back ends are concerned, crossroads behaves like a client. As far as the true client is concerned, crossroads behaves like the service. The communication is however transparent: neither client nor back end are aware of the middle position of crossroads.

A session is a conversation between client and service, where data are transferred to and fro over the network. As far as crossroads is concerned, a session is succesful if connections are succesfully established and no errors on the network level occur. Crossroads isn't aware of service pecularities. E.g., when a webserver answers `HTTP/1.0 500 Server Error` then crossroads will see this as a succesful session, though the user behind a browser may think otherwise.

Listeners are programs that listen to a given TCP port. Based on its configuration, crossroads starts a listener for each service.

Back end selection algorithms are methods by which crossroads determines which back end it will talk to next. Crossroads has a number of built-in algorithms, which are configured per service.

Back end states are the statusses of each back end that is known to crossroads. A back end may be available, (temporarily) unavailble or truly down. When a back end is temporarily unavailable, then crossroads will periodically check whether the back end has come to life yet (that is, if configured so).

A spike is a sudden increase in activity, leading to extra load on a given service. When crossroads is in effect, then obviously the spike will also appear at one of the back ends.

Load balancing means that an incoming client request is distributed over not just one back end (which would be the case if you wouldn't be running crossroads). Enabling load balancing is nothing more than duplicating services over more than one back end, and having something (in this case: crossroads) distribute the requests, so that per back end the load doesn't get too high.

Back end usage is measured by crossroads in order to be able to determine back end selection. Crossroads stores information about the number of transferred bytes and about the session duration. These numbers can be used to estimate which back end is the least used – and therefore, presumably, the best candidate for a new request.

Fail over is almost always used when load balancing is in effect. The distributor of client requests (crossroads of course) can also monitor back ends, so that incase a back end is 'down', it is no longer accessed.

Service downtime normally occurs when a service is switched off. Downtime is obviously avoided when fail over is in effect: a back end can be taken out of service in a controlled manner, without any client noticing it.

2 Using Crossroads

Crossroads is started from the commandline, and highly depends on `/etc/crossroads.conf` (the default configuration file). It supports a number of flags (e.g., to overrule the location of the configuration file). The actual usage information is always obtained by typing `crossroads` without any arguments. Crossroads then displays the allowed arguments.

This section shows the basic usage.

- `crossroads start` and `crossroads stop` are typical actions that are run from system startup scripts. The meaning is self-explanatory.
- `crossroads restart` is a combination of the former two. Beware that a restart may cause discontinuity in service; it is just a shorthand for typing the 'stop' and 'start' actions after one another.

- `crossroad status` reports on each running service. Per service, the state of each back end is reported.
- `crossroads tell` *service backend state* is a command line way of telling crossroads that a given back end, of a given service, is in a given state. Normally crossroads maintains state information itself, but by using `crossroads tell`, a back end can be e.g. taken 'off line' for servicing.
- `crossroads services` reports on the configured services. In contrast to `crossroads status`, this option only shows what's configured – not what's up and running. Therefore, `crossroads services` doesn't report on back end states.
- `crossroads sampleconf` shows a sample configuration on screen. A good way of quicky viewing the configuration file syntax, or of getting a start for your own configuration `/etc/crossroads.conf`.

3 The configuration

The configuration that crossroads uses is normally stored in the file `/etc/crossroads.conf`. This location can be overruled using the command line flag `-c`.

This section explains the syntax of the configuration file, and what all settings do.

3.1 General language elements

This section describes the general elements of the crossroads configuration language.

3.1.1 Empty lines and comments

Empty lines are of course allowed in the configuration. Crossroads recognizes three formats of comment:

- C-style, between `/*` and `*/`,
- C++-style, starting with `//` and ending with the end of the text line;
- Shell-style, starting with `#` and ending with the end of the text line.

Simply choose your favorite editor and use the comment that 'looks best'.[2]

3.1.2 Keywords, numbers, identifiers, generic strings

In a configuration file, statements are identified by *keywords*, such as `service`, `verbosity`. These are reserved words.

Many keywords require an *identifier* as the argument. E.g, a service has a unique name, which must start with a letter or underscore, followed by zero or more letters, underscores, or digits. E.g., in the statement `service myservice`, the keyword is `service` and the identifier is `myservice`.

Other keywords require a numeric argument. Crossroads knows only non-negative integer numbers, as in `port 8000`. Here, `port` is the keyword and `8000` is the number.

Yet other keywords require 'generic strings', such as hostname specifications or system commands. Such generic strings contain any characters up to the terminating statement character `;`.

Finally, an argument can be a 'boolean' value. Crossroads knows `true`, `false`, `yes`, `no`, `on`, `off`. The keywords `true`, `yes` and `on` all mean the same and can be used interchangeably; as can the keywords `false`, `no` and `off`.

[2] I favor C or C++ comment. My favorite editor *emacs* can be put in `cmode` and nicely highlight what's comment and what's not. And as a bonus it will auto-indent the configuration!

3.2 Service definitions

Service definitions are blocks in the configuration file that state what is for each service. A service definition starts with `service`, followed by a unique identifier, and by statements in `{` and `}`. For example:

```
// Definition of service 'www':
service www {
    ...
    ... // statements that define the
    ... // service named 'www'
    ...
}
```

The following list shows possible stateements.

Service definition statements define what a service should do. Possible definition statements are shown below. Each statement is terminated by a semicolon (except for the `backend` statement, which has its own block).

The port statement defines to which TCP port a service 'listens'. E.g. `port 8000` says that this service will accept connections on port 8000.

- Syntax: `port` *number* **;**
- There is no default. This is a required setting.

The bindto statement is used in situations where crossroads should only listen to the stated port at a given IP address. E.g., `bindto 127.0.0.1` causes crossroads to 'bind' the service only to the local IP address. Network connections from other hosts won't be serviced. By default, crossroads binds a service to all presently active IP addresses at the invoking host.

- Syntax: `bindto` *ip-address* **;**
- where *ip-address* is a numeric IP address, such as `192.168.1.45`, or the keyword `any`
- Default: `any`

Verbosity statements come in two forms: `verbosity on` or `verbosity off`. When 'on', log messages to `/var/log/messages` are generated that show what's going on.[3]

- Syntax: `verbosity` *setting* **;**
- where *setting* is `true`, `yes` or `on` to turn verbosity on; or `false`, `no`, `off` to turn it off.
- Default: `off`.

The dispatch mode controls how crossroads selects a back end from a list of active back ends. The syntax is:

- `dispatchmode roundrobin`: Simply the 'next in line' is chosen. E.g, when 3 back ends are active, then the usage series is 1, 2, 3, 1, 2, 3, and so on.
  Roundrobin dispatching is the default method, when no `dispatchmode` statement occurs.
- `dispatchmode random`: Random selection. Probably only for stress testing.
- `dispatchmode bysize [ over` *sessions* `]`: The next back end is the one that has transferred the least number of bytes. This selection mechanism assumes that the more bytes, the heavier the load.

[3] Actually, the messages go to `syslog(3)`, using facility `LOG_DAEMON` and priority `LOG_INFO`. In most (Linux) cases this will mean: output to `/var/log/messages`.

The modifier `over` *nsessions* is optional. (The square brackets shown above are not part of the statement but indicate optionality.) When given, the size load is computed over the last stated number of sessions. When this modifier is absent, then the load is computed over all sessions since startup.

- `dispatchmode byduration [ over` *sessions* `]`: The next back end is the one that served sessions for the shortest time. This mechanims assumes that the longer the session, the heavier the load.
- `dispatchmode byorder`: The first back end is selected every time, unless it's unavailable. In that case the second is taken, and so on.

The 'right' dispatch mode will depend on the type of service. Given the fact that crossroads doesn't know (and doesn't care) what a stream of network traffic means, you have to choose an appropriate dispatch mode to optimize load balancing. Note that the dispatch mode is totally irrelevant when your only concern is fail over. In most cases, `roundrobin` will do the job just fine.

A reviving interval definition is needed when crossroads determines that a back end is temporarily unavailable. This will happen when:

- The back end cannot be reached (network connection fails);
- The network connection to the back end suddenly dies.

An example of the definition is `revivinginterval 10`. When this reviving interval is given, crossroads will check each 10 seconds whether unavailable back ends have woken up yet. A back end is considered awake when a network connection to that back end can succesfully be established.

- Syntax: `revivinginterval` *number* **;**
- Default: 0, meaning no revivals will occur.

The maximum number of clients is specified using `maxclients`. There is one argument; the number of concurrent established sessions that may be active within one service.

'Throttling' the number of clients is a way of preventing Denial of Service (DOS) attacks. Without a limit, numerous network connections may spawn so many server instances, that the service ultimately breaks down and becomes unavailable.

- Syntax: `maxclient` *number* **;**
- Default: 0, meaning that all client connections will be accepted.

The TCP back log size is a number that controls how many 'waiting' network connections may be queued, before a client simply cannot connect. The syntax is e.g. `backlog 5` to cause crossroads to have 5 waiting connections for 1 active session. The backlog queue shouldn't be too high, or clients will experience timeouts before they can actually connect. The queue shouldn't be too small either, because clients would be simply rejected. Your mileage may vary.

- Syntax: `backlog` *number* **;**
- Default: zero, which takes the operating system's default value for socket back log size.

Reporting based: the shared memory key. Different crossroad invocations must 'know' of each others activity. E.g, `crossroad status` must be able to get to the actual state information of all running services. This is internally implemented through shared memory, which is reserved using a key.

Normally crossroads will supply a shared memory key, based on the service port and bitwise or-ed with a magic number. In situations where this conflicts with existing keys (of other programs, having their own keys), you may supply a chosen value.

The syntax is e.g. `shmkey 123456`. The actual key value doesn't matter much, as long as it's unique and as long as each invocation of crossroads uses it.

- Syntax: `shmkey` *number* **;**
- Default: 0, which means that crossroads will 'guess' its own key, based on TCP port and a magic number.

Session timeouts: Sometimes, clients simply won't close a network connection which leads to unnecessary resource usage. To avoid this, one might state `sessiontimeout 300`. This instructs crossroads to consider a session where nothing has happened for 300 seconds as 'finished'. Crossroads will terminate the connection when this timeout is exceeded.

- Syntax: `sessiontimeout` *number* **;**
- Default: 0, meaning that crossroads will not try to determine timeouts.

3.3 Backend definitions

Inside the service definitions as are described in the previous section, *backend definitions* must also occur. Backend definitions are started by the keyword `backend`, followed by an identifier, and defining statements inside `{` and `}`:

```
service myservice {
    ...
    ... // statements that define the
    ... // service named 'myservice'
    ...

    backend mybackend {
        ...
        ... // statements that define the
        ... // backend named 'mybackend'
        ...
    }
}
```

The backend definition statements are shown below.

Server: Each back end must be identified by the network name (server name) where it is located. For example: `server 10.1.1.23`, or `server web.mydomain.org`.

- Syntax: `server` *servername* **;**
- There is no default. This is a required setting.

Port: Besides the `server` specifier, the `port` definition is also required. It has one argument, the (numeric) port number.

- Syntax: `port` *number* **;**
- There is no default. This is a required setting.

Verbosity: Similar to `service` specifications, a `backend` can have its own verbosity (`on` or `off`). When `on`, traffic to and fro this back end is reported.

- Syntax: `verbosity` *setting* **;**
- where *setting* is `true`, `yes` or `on` to turn verbosity on; or `false`, `no`, `off` to turn it off.
- Default: `off`.

Maxclients: This setting states how many concurrent sessions a back end connection may accept. Note that there is also a `maxclients` statement for the overall service description. The difference is that a `maxclients` statement at the level of a service description avoids too many hits from the outside (DOS prevention). A `maxclients` statement at the level of a back end description makes sure that this particular back end doesn't get overloaded.

- Syntax: `maxclients` *number* **;**
- where *number* is the maximum number of concurrent client connections.
- Default: 0, meaning that there is no limit.

Weight: To influence how backends are selected by size or by duration, a backend can specify its 'weight' in the process. The higher the weight, the less likely a back end will be chosen. The default is 1.

The weighing mechanism only applies for `dispatchmode bysize` or `byduration`. The weight is in fact a multiplier. E.g., if backend A has `weight 2` and backend B has `weight 1`, then backend B will be selected all the time, until its usage parameter is twice as large as the parameter of A.

- Syntax: `weight` *number* **;**

Decay: To make sure that a 'spike' of activity doesn't influence the perceived load of a back end forever, you may specify a certain decay. E.g, the statement `decay 10` makes sure that the load that crossroads computes for this back end (be it in seconds or in bytes) is decreased by 10% each time that **an other** back end is hit.

This means that when a given back end is hit, then its usage data is updated according to the session. However, when a different back end is hit, its usage is decreased by the specified decay.

- Syntax: `decay` *number* **;**
- where *number* is a percentage that decreases the back end usage when other back ends are hit
- Default: 0

Event triggers: As special 'hooks' for actions, two triggers are available: `onfailure` and `onsuccess`. The argument to the triggers is a system command that is executed when a session with the back end either fails or succeeds.

- Syntax: `onfailure` *commandline* **;** and `onsuccess` *commandline* **;**
- There is no default.

Debugging and Performance aids: Incase the traffic between client and backend must be debugged, the statement `trafficlog` *filename* can be issued. This causes the traffic to be dumped in hexadecimal format to the stated filename.

Traffic sent by the client is prefixed by a **C**, traffic sent by the back end is prefixed by a **B**. Below is a sample traffic dump of a browser trying to get a HTML page. The server replies that the page was not modified.

```
C 0000  47 45 54 20 68 74 74 70 3a 2f 2f 77 77 77 2e 63 GET http://www.c
C 0010  73 2e 68 65 6c 73 69 6e 6b 69 2e 66 69 2f 6c 69 s.helsinki.fi/li
C 0020  6e 75 78 2f 6c 69 6e 75 78 2d 6b 65 72 6e 65 6c nux/linux-kernel
C 0030  2f 32 30 30 31 2d 34 37 2f 30 34 31 37 2e 68 74 /2001-47/0417.ht
C 0040  6d 6c 20 48 54 54 50 2f 31 2e 31 0d 0a 43 6f 6e ml HTTP/1.1..Con
C 0050  6e 65 63 74 69 6f 6e 3a 20 63 6c 6f 73 65 0d 0a nection: close..
.
. etcetera
.
B 0000  48 54 54 50 2f 31 2e 30 20 33 30 34 20 4e 6f 74 HTTP/1.0 304 Not
B 0010  20 4d 6f 64 69 66 69 65 64 0d 0a 44 61 74 65 3a  Modified..Date:
B 0020  20 54 75 65 2c 20 31 32 20 4a 75 6c 20 32 30 30  Tue, 12 Jul 200
B 0030  35 20 30 39 3a 34 39 3a 34 37 20 47 4d 54 0d 0a 5 09:49:47 GMT..
B 0040  43 6f 6e 74 65 6e 74 2d 54 79 70 65 3a 20 74 65 Content-Type: te
B 0050  78 74 2f 68 74 6d 6c 3b 20 63 68 61 72 73 65 74 xt/html; charset
.
```

```
. etcetera
.
B 0110  63 6c 6f 73 65 0d 0a 0d 0a                        close....
```

Turning on traffic dumps will *significantly* slow down crossroads.

- Syntax: `trafficlog` *filename* `;`
- There is no default. Without this directive, traffic is not logged.

Besides `trafficlog`, there is also a directive `throughputlog`. This directive also takes one argument, a filename. The file is appended, and the following information is logged:

- The process ID of the crossroads image that serves the TCP connection;
- The time of the request, in seconds and microseconds since start of the run;
- A **C** when the request originated at the client, or **B** when the request originated at the back end;
- The first 100 bytes of the request.

As an example, consider the following (the lines are shortened for brevity and prefixed by line numbers for clarity):

```
1 0000594 0.000001 C GET http://public.e-tunity.com/index.html...
2 0000594 0.173713 B HTTP/1.0 200 OK..Date: Fri, 18 Nov 2005 0...
3 0000594 0.278125 B  width="100" bgcolor="#e0e0e0" valign="to...
4 0000595 0.000001 C GET http://public.e-tunity.com/css/style/...
5 0000594 0.944339 B /a></td>..  </tr>.</table>.</td><td class...
6 0000594 0.946356 B smallboxdownl">Download</td>..  <td class...
7 0000594 0.961102 B td><td class="smallboxodd" valign="top"><...
8 0000595 0.698215 B HTTP/1.0 304 Not Modified..Date: Fri, 18 ...
```

This tells us that:

- Line 1: PID 594 served a request that originated at the client. The corresponding time is (almost) 0 seconds, so this is really the start of the run.
- Line 2: A back end replied 0.17 seconds later, and 0.28 seconds later, it was still replying (this is the third line, again a **B**-type transmission).
- Line 4: PID 595 served a request that originated at the client. Again, the corresponding time is (almost) 0 seconds, since this is the first conversation part of this session.
- Lines 5 to 7: This is the continuation of line 2. Line 7 is the last line of the **B** series (not visible from the example, but trust me, it is), so that we may conclude that it took the back end 0.96 seconds to serve the file `index.html` requested in line 1.
- Line 8: This is the answer to the client's request of line 4 (you can tell by the process ID number). So the back end took 0.68 seconds to confirm that the stylesheet requested in line 4 wasn't modified.

It is also worth while remembering that the start time of a **C** request is the time that crossroads sees the activity. Any latency between the true client and crossroads is obviously not included. This is illustrated by the below simple ASCII art:

```
client ---->---->---->--->*crossroads ====>====>====>
                                                    \
                                                  back end
                                                    /
client ----<----<----<---< crossroads ====<====<====<
```

This simple picture shows a typical HTTP request that originates at a client, travels to crossroads, and is relayed via the back end. The **C** entry in a throughput log is the time when crossroads sees the request, indicated by an asterisk. The **B** entries are the times that it takes the back end to answer, indicated by === style lines. Therefore, the true roundtrip time will be longer than the number of seconds that are logged in the throughput log: the latency between client and crossroads isn't included in that measurement.

Summarizing, the throughput times of a client-back end connection can be analyzed using the directive `throughputlog`. In a real-world analysis, you'd probably want to write up a script to analyze the output and to compute round trip times. Such scripts are not (yet) included in Crossroads.

- Syntax: `throughputlog` *filename* **;**
- There is no default. Without this directive, the throughput is not logged.

3.4 How back ends are selected in load balancing

In order to tune your load balancing, you'll need to understand how crossroads computes usage, how weighing works, and so on. In this section we'll focus on `dispatchmode bysize` and `dispatchmode byduration` only: the other dispatching types are self-explanatory.

3.4.1 Bysize or byduration

As stated before, crossroads doesn't know 'what a service does' and how to judge whether a given back end is very busy or not. You must therefore give the right hints:

- In general, a service which is CPU bound, will be more busy when it takes longer to process a request. The dispatch mode `byduration` is appropriate here.
- In contrast, a service which is filesystem bound, will be more busy when more data are transferred. The dispatch mode `bysize` is apppropriate.
- The dispatch mode `byduration` can also be used when network latency is an issue. E.g., if your balancer has back ends that are geograpically distributed, then `byduration` would be a good way to select well available back ends.
- Furthermore it is noteworthy that `dispatchmode byduration` is not usable for interactive processes such as SSH logins. Idle time of a login adds to the duration, while causing (almost) no load. Mode `byduration` should only be used for automated processes that don't wait for user interaction (e.g., SOAP calls and other HTTP requests).

3.4.2 Averaging size and duration

The configuration statement `dispatchmode bysize` or `byduration` allows an optional modifier `over` *number*, where the stated number represents a session count. When this modifier is present, then crossroads will use a moving average over the last *n* sessions to compute duration and size figures.

In the real world you'll always want this modifier. E.g., consider two back ends that are running for years now, and one of them is suddenly overloaded and very busy. When the `over` modifier is absent, then the sudden load will hardly show up in the usage figures – it will flatten out due to the large usage figures already stored in the years of service.

In contrast, when e.g. `over 3` is in effect, then a sudden load does show up – because it highly contributes to the average of three sessions.

3.4.3 Specifying decays

Decays are also only relevant when crossroads computes the 'next best back end' by size (bytes) or duration (seconds). E.g., imagine two back ends A and B, both averaged over say 3 sessions.

Now when back end A is suddenly hit by a large request (a 'spike'), its average would go up accordingly. But the back end would never again be used, unless B also received a similar spike, because A's 'usage data' would forever be larger than B's data.

For that reason, you should in real situations probably always specify a decay, so that the backend selection algorithm recovers from spikes. The below configuration illustrates this:

```
/* Definition of the service */
service soap {
    /* Local TCP port */
    port 8080;

    /* We'll select back ends by the processing
     * duration
     */
    dispatchmode byduration over 3;

    /* First back end: */
    backend A {
        /* Back end IP address and port */
        server 10.1.1.1;
        port 8080;
        /* When this back end is NOT hit because
         * the other one is less busy, then the
         * usage parameters decay 10% per session
         */
        decay 10;
    }

    /* Second back end: */
    backend B {
        server 10.1.1.2;
        port 8080;
        decay 10;
    }
}
```

3.4.4 Adjusting the weights

The back end modifier `weight` is useful in situations where your back ends differ in respect to performance. E.g,. your back ends may be geographically distributed, and you know that a given back end is difficult to reach and often experiences network lag.

Or you may have one primary back end, a system with a fast CPU and enough memory, and a small fall-back back end, with a slow CPU and short on memory. In that case you know in advance that the second back end should be used only rarely.

In such cases you will know in advance that the best performing back ends should be selected the most often. Here's where the `weight` statement comes in: you can simply increase the weight of the back ends with the least performance, so that they are selected less frequently.

E.g., consider the following configuration:

```
service soap {
    port 8080;
    dispatchmode byduration over 3;
    backend A {
        server 10.1.1.1;
        port 8080;
```

```
        decay 20;
    }
    backend B {
        server 10.1.1.2;
        port 8080;
        weight 2;
        decay 10;
    }
    backend C {
        server 10.1.1.3;
        port 8080;
        weight 4;
        decay 5;
    }
}
```

This will cause crossroads to select back ends by the processing time, averaging over the last three sessions. However, backend B will kick in only when its usage is half of the usage of A (back end B is probably only half as fast as A). Backend C will kick in only when its usage is a quarter of the usage of A, which is half of the usage of B (back end C is probably very weak, and just a fall-back system incase both A and B crash). Note also that A's usage data decay much faster than B's and C's.

3.4.5 Throttling the number of concurrent sessions

If you suspect that your service may occasionally receive 'spikes' of activity[4], then it might be a good idea to protect your service by specifying a maximum number of concurrent sessions. This protection can be specified on two levels:

On the service level a statement like `maxclients 100;` states that the service as a whole will never service more than 100 concurrent sessions. This means that all your back ends will be protected from being overloaded.

On the back end level a statement like `maxclients 10;` states that this particular back end will never have more than 10 concurrent sessions; regardless of the overall setting on the service level. This means that this particular back end will be protected from being overloaded (regardless of what other back ends may experience).

Using the `maxclients` statement, combined with a back end selection algorithm, allows very fine granularity. The `maxclients` statement on the back end level is like a hand brake: even when you specify a back end algorithm that would protect a given back end from being used too much, a situation may occur where that back end is about to be hit. A `maxclients` statement on the level of that back may then protect it.

3.5 Configuration examples

As a general hint, use `crossroads sampleconf` to view the most up-to-date examples of configurations. The description below shows a few examples too.

3.5.1 A load balancer for three webserver back ends

The following configuration example binds crossroads to port 80 of the current server, and distributes the load over three back ends. This configuration shows most of the possible settings.

[4] which you should always assume

```
service www {

    /* Port on which we'll listen in this service: required. */
    port 8000;

    /* What IP address should this service listen? Default is 'any'.
     * Alternatively you can state an explicit IP address, such as
     * 127.0.0.1; that would bind the service only to 'localhost'. */
    bindto any;

    /* Verbose reporting or not. Default is off. */
    verbosity on;

    /* Dispatching mode, or: How to select a back end for an incoming
     * request. Possible values:
     *   roundrobin: just the next back end in line
     *   random: like roundrobin, but at random to make things more
     *           confusing. Probably only good for testing.
     *   bysize: The backend that transferred the least nr of bytes
     *           is the next in line. As a modifier you can say e.g.
     *           bysize over 10, meaning that the 10 last sessions will
     *           be used to compute the transfer size, instead of all
     *           transfers.
     *   byduration: The backend that was active for the shortest time
     *           is the next in line. As a modifier you can say e.g.
     *           byduration of 10 to compute over the last 10 sessions.
     *   byorder: The first available back end is always taken.
     */
    dispatchmode byduration over 5;

    /* Interval at which we'll check whether a temporarily unavailable
     * backend has woken up.
     */
    revivinginterval 5;

    /* TCP backlog of connections. Default is 0 (no backlog, one
     * connection may be active).
     */
    backlog 5;

    /* For status reporting: a shared memory key. Default is the same
     * as the port number, OR-ed by a magic number.
     */
    shmkey 8000;

    /* This controls when crossroads should consider a session as
     * finished even when the TCP sockets weren't closed. This is to
     * avoid hanging connections that don't do anything. NOTE THAT when
     * crossroads cuts off a session due to timeout exceed, this is
     * not marked as a failure, but as a success. Default is 0: no timeout.
     */
    sessiontimeout 300;

    /* The max number of allowed client sessions. When present, connections
```

```
 * won't be accepted if the max is about to be exceeded. When
 * absent, all connections will be accepted, which might be misused
 * for a DOS attack.
 */
maxclients 300;

/* Now let's define a couple of back ends. Number 1: */
backend www_backend_1 {
    /* The server and its port, the minimum configuration. */
    server httpserver1;
    port 9010;
    /* The 'decay' of usage data of this back end. Only relevant
     * when the whole service has 'dispatchmode bysize' or
     * 'byduration'. The number is a percentage by which the usage
     * parameter is decreased upon each session of an other back
     * end.
     */
    decay 10;

    /* To see what's happening in /var/log/messages: */
    verbosity on;
}

/* The second one: */
backend www_backend_2 {
    /* Server and port */
    server httpserver2;
    port 9011;

    /* Verbosity of reporting when this back end is active */
    verbosity on;

    /* Decay */
    decay 10;

    /* Event triggers for system commands upon succesful activation
     * and upon failure.
     */
    onsuccess echo 'success on backend 2' | mail root;
    onfailure echo 'failure on backend 2' | mail root;
}

/* And yet another one.. this time we will dump the traffic
 * to a trace file. Furthermore we don't want more than 10 concurrent
 * sessions here. Note that there's also a total maxclients for the
 * whole service.
 */
backend www_backend_3 {
    server httpserver3;
    verbosity on;
    port 9000;
    verbosity on;
    decay 10;
    trafficlog /tmp/backend.3.log;
```

```
        maxclients 10;
    }
}
```

3.5.2 A HTTP forwarder when travelling

As another example, here's my `crossroads.conf` that I use on my Linux laptop. The problem that I face is that I need many HTTP proxy configurations (at home, at customers' sites and so on) but I'm too lazy to reconfigure browsers all the time.

Here's how it used to be before crossroads:

- At home, I would surf through a squid proxy on my local machine. The browser proxy setting is then `http://localhost:3128`.
- Sometimes I start up an SSH tunnel to our offices. The tunnel has a local port 3129, and connects to a squid proxy on our e-tunity server. Hence, the browser proxy is then `http://localhost:3129`.
- At a customer's location I need the proxy `http://10.120.34.113:8080`, because they have configured it so.
- And in yet other instances, I use a HTTP diagnostic tool *charles* that sits between browser and website and shows me what's happening. I run charles on my own machine and it listens to port 8888, behaving like a proxy. The browser configuration for the proxy is then `http://localhost:8888`.

Here's how it works with a crossroads configuration:

- I have configured my browsers to use `http://localhost:8080` as the proxy. For all situations.
- I use the following crossroads configuration, and let crossroads figure out which proxy backend works, and which doesn't. Note two particularities:
  - The statement `dispatchmode byorder`. This makes sure that once crossroads determines which backend works, it will stick to it. This usage of crossroads doesn't need to balance over more than one back end.
  - the statement `bindto 127.0.0.1` makes sure that requests from other interfaces than loopback won't get serviced.

```
service HttpProxy {
    port 8080;
    bindto 127.0.0.1;
    verbosity on;
    dispatchmode byorder;
    revivinginterval 15;

    backend Charles {
        server 127.0.0.1;
        port 8888;
        verbosity on;
    }

    backend CustomerProxy {
        server 10.120.34.113;
        port 8080;
        verbosity on;
    }
```

```
        backend SshTunnel {
            server 127.0.0.1;
            port 3129;
        }

        backend LocalSquid {
            server 127.0.0.1;
            port 3128;
            verbosity on;
        }
    }
```

As a final note, the commandline argument `tell` can be used to influence crossroad's own detection mechanism of back end availability detection. E.g., if in the above example the back ends `SshTunnel` and `LocalSquid` are both active, then `crossroads tell httpproxy sshtunnel down` will 'take down' the back end `SshTunnel` – and will automatically cause crossroads to switch to `LocalSquid`.

3.5.3 SSH login with enforced idle logout

The following example shows how crossroads 'throttles' SSH logins. Connections are accepted on port 22 (the normal SSH port) and forwards these to the actual SSH daemon which is running on port 2222.

Note the usage of the `sessiontimeout` directive. This makes sure that users are logged out after 10 minutes of inactivity. Note also the `maxclients` setting, this makes sure that no more than 10 concurrent logins occur.

```
service Ssh {
    port 22;
    backlog 5;
    maxclients 10;
    sessiontimeout 600;
    backend TrueSshDaemon {
        server 127.0.0.1;
        port 2222;
    }
}
```

4 Benchmarking

As a small benchmark, this section shows how crossroads affects the transmitting of HTML data when used as an intermediate 'station' through which all data travels.

4.1 Environment

The benchmark was run on a system where the following was varied:

1. A website was recursively spidered through a local squid proxy. The spidering was repeated 10 times, the total was recorded.
2. Crossroads was placed in front of the squid proxy, and the website was again recursively spidered. Again, the spidering was repeated 10 times and the total was recorded.

The crossroads configuration of the second alternative is shown below:

```
service HttpProxy {
    port 8080;
    verbosity on;
    backend LocalSquid {
        server 127.0.0.1;
        port 3128;
        verbosity on;
    }
}
```

4.2 Results

The results of this test are that crossroads causes a negligible delay, if it is statistically relevant at all. Without crossroads, the timing results are:

```
real 0m8.146s
user 0m0.130s
sys 0m0.253s
```

When using crossroads as a middle station, the results are:

```
real 0m9.481s
user 0m0.141s
sys 0m0.230s
```

4.3 Discussion

The above shown results are quite favorable to crossroads. However, one should know that situations will exist where crossroads leans towards the 'worst case' scenario, causing up to 50% delay.

E.g., imagine a test where a `wget` command retrieves a HTML document from an Apache server on `localhost`. Now we have (almost) no overhead due to network throttling, hostname lookups and so on. When this test would be run either with or without crossroads in between, then theoretically, crossroads would cause a much larger delay, because it has to read from the server, and then write the same information to `wget`. Each read/write occurs twice when crossroads sits in between.

This worst case scenario will however (fortunately) occur only very seldom in the real world:

- Normally network issues, such as the above mentioned host name lookups or throughput restrictions, will add significantly to the duration of a request. The 'twice as many' read/writes caused by crossroads are then relatively irrelevant.
- Normally a significant amount of time will be spent in a back end, due to processing (e.g., when calling a servlet on a back end). Again, this processing time will weigh much heavier than the multiple read/writes.

5 Compiling and Installing

5.1 Prerequisites

The creation of crossroads requires:

- Standard Unix tools, such as `sed`, `awk`, `Perl` (5.00 or better);
- A POSIX-compliant C compiler;
- The grammar generation tools `bison` and `flex`;

- Support for SYSV IPC, networking and so on.

Basically a Linux or Apple MacOSX box will do nicely once you make sure that `bison` and `flex` are installed. To compile and install crossroads, follow these steps.

5.2 Compiling and installing

- Obtain the source distribution. It can be found on `http://public.e-tunity.com`. The distribution comes as an archive `crossroads-`*X.YY*`.tar.gz`, where *X.YY* is a version number.
- Unpack the archive in a sources directory using `tar xzf crossroads-`*X.YY*`.tar.gz`. The contents spill into a subdirectory `crossroads/`.
- Change-dir into the directory.
- Next, edit `etc/Makefile.def` and verify that all compilation settings are to your likings. The settings are explained in the file. **Note that** the default distribution of `Makefile.def` is suited for Linux or Apple MacOSX systems. On other Unices, or on non-Unix systems, you must particularly pay attention to `SET_PROC_TITLE_BY...`. When in doubt, comment out all `SET_PROC_TITLE...` settings. Crossroads will work nevertheless, but it won't show nice titles in `ps` listings.
- Now crossroads is ready for compilation. Do a `make local` followed by `make install`. The latter step may have to be done by the user `root` if the `BINDIR` setting of `etc/Makefile.def` points to a root-owned directory.
- The documentation doesn't install in this process. If you want to install the documentation, then proceed as follows:
  - Optionally, `cp doc/crossroads.html` *htmldirectory/*; where *htmldirectory* is the destination directory for your HTML manuals;
  - Optionally, `cp doc/crossroads.pdf` *pdfdirectory/*; where *pdfdirectory* is the destination directory for your PDF manuals;
  - Optionally, `cp doc/crossroads.man` *manualdirectory*`/man1/crossroads.1`, where *manualdirectory* is e.g. `/usr/man`, `/usr/share`, `/usr/local/man`, `/usr/local/share`. Any possibility is valid, as long as `manualdirectory` has a subdirectory `man1/`;
  - If your manual page system supports compressed manual pages, then you can save some space with `gzip -c < doc/crossroads.man >` *manualdirectory*`/man1/crossroads.1.gz`.

5.3 Configuring crossroads

Now that the binary is available on your system, you need to create a suitable `/etc/crossroads.conf`. Use this manual or the output of `crossroads samplconf` to get started.

Once you have the configuration ready, start crossroads with `crossroads start`. Test the availability of your services and back ends. Monitor how crossroads is doing with:

- In one terminal, run the script:

```
while [ 1 ] ; do
    tput clear
    crossroads status
    sleep 3
done
```

  Please note that depending on your system you might need `sleep 3s`, i.e., with an `s` appended.

- In another terminal, run:

```
while [ 1 ] ; do
    tput clear
    ps ax | grep crossroads | grep -v grep
    sleep 3
done
```

- In yet another terminal, run `tail -f /var/log/messages`.

Now thoroughly test the availability of your back ends through crossroads. The status display will show an updated view of which back ends are selected and how busy they are. The process list will show which crossroads daemons are running. Finally, the tailing of `/var/log/messages` shows what's going on – especially if you have `verbosity true` statements in the configuration.

5.4 A boot script

Finally, you may want to create a boot-time startup script. The exact procedure depends on the used Unix flavor.

5.4.1 SysV Style Startup

On SysV style systems, there's a startup script directory `/etc/init.d` where bootscripts for all utilities are located. You may have the `chkconfig` utility to automate the task of inserting scripts into the boot sequence, but otherwise the steps will resemble the following.

- Create a script `crossroads` in `/etc/init.d` similar to the following:

  ```
  #!/bin/sh
  /usr/local/bin/crossroads -v $@
  ```

  The stated directory `/usr/local/bin` must correspond with the installation path. The flag `-v` causes the startup to be more 'verbose'. However, once daemonized, the verbosity is controlled by the appropriate statements in the configuration.

- Determine your 'runlevel': usually 3 when your system is running in text-mode only, or 5 when you are a graphical interface. If your runlevel is 3, then:

  ```
  root> cd /etc/rc.d/rc3.d
  root> ln -s /etc/init.d/crossroads S99crossroads
  root> ln -s /etc/init.d/crossroads K99crossroads
  ```

  This creates startup (`S*`) and stop (`K*`) links that will be run when the system enters or leaves a given runlevel.

  If your runlevel is 5, then the right `cd` command is to `/etc/rc.d/rc5.d`. Alternatively, you can create the symlinks in both runlevel directories.

5.4.2 BSD Style Startup

On BSD style systems, daemons are booted directly from `/etc/rc` and related scripts. Incase you have a file `/etc/rc.local`, edit it, and add the statement:

```
/usr/local/bin/crossroads start
```

If your BSD system lacks `/etc/rc.local`, then you may need to start Crossroads from `/etc/rc`. Your mileage may vary.