# Agent based auto-configuration of OSPF networks

Visa Holopainen
TKK Helsinki University of Technology
Networking Laboratory
P.O. Box 3000, FI-02015 TKK, Finland
visa.holopainen@netlab.tkk.fi

## Abstract

*This paper introduces an auto-configuration system that is intended to enable people with very basic computer networking skills to set up a router network from open-source software and commodity PC hardware. Our system is composed of three subsystems: Operating System auto-installation subsystem, IP address auto-assignment subsystem, and OSPF auto-configuration subsystem. We have implemented the last one these subsystems, and present its performance test results. We find that the execution time of the subsystem largely depends on Perl's SSH-module.*

## 1. Introduction

As complexity of networks constantly increases, auto-configuration is seen as a necessary step towards disruption-free communications [1]. The rising need for auto-configurable networks has manifested itself into many research papers presenting high-level auto-configuration architectures. However, only few of those papers properly evaluate feasibility of the architectures they present. That is, if the architectures can be implemented into real networks, and under which pre-conditions.

The goal of this paper is to stray from this "high-level research path" by introducing and evaluating a practical auto-configuration system, by which computer networking novices can easily convert a network of commodity PCs into a router network without any manual configuration.

The system is composed of three subsystems: Operating System (OS) auto-installation subsystem, IP address auto-assignment subsystem, and OSPF auto-configuration subsystem. Each of the subsystems works by its own "flooding principle". The OS auto-installation system floods OS from PC to PC, IP address auto-assignment subsystem floods IP configuration from PC to PC, whereas OSPF auto-configuration subsystem floods OSPF protocol configuration.

Routing protocol auto-configuration has received little attention in previous work. Hence we discuss the OSPF auto-configuration subsystem in more detail, and present performance test results regarding that subsystem.

The rest of this paper is organized in the following way: Section 2 presents related work that helps to put this paper into context. Section 3 gives a high-level conceptual overview of our system, whereas Section 4 describes it in detail. Section 5 presents performance test results of our system. Based on test results, Section 6 concludes our work.

## 2 Related work

Lehtihet at al. present a goal-based autonomic configuration and optimization architecture [2]. The architecture introduces self-organizing capabilities into the network, so that the aggregate behavior of autonomic elements (routers) satisfies high-level operational goals defined by network's administrator. The authors focus more on architecture description, and it remains somewhat unclear how the auto-configuration process actually works in the context of their paper. Hereby it is difficult to evaluate the feasibility of their approach.

Bullot and Gaiti describe an architecture of collaborative and autonomic software agents [3]. The agents are embedded inside routers. The role of agents is to share local and situated knowledge, in order to control and optimize the existing control mechanism of the router. Although the authors present simulation results, a working prototype is not described. Also, protocol auto-configuration is not discussed in the paper.

Zeroconf [5] or Zero Configuration Networking is a set of techniques that automatically create a usable LAN network. Zeroconf is especially concerned in making devices like network printers, cameras and PCs interwork on a LAN without manual configuration. Zeroconf does not support router configuration.

Akinlar et al. review IPv4 auto-configuration algorithms for hosts and single-router zeroconf networks, and propose

auto-configuration algorithms for multi-router zeroconf networks [12]. However, no prototype implementation is described in the paper.

Arai et al. propose a novel method for auto-configuration of power-line and coaxial cable modems [8]. Their system does not support routers either.

Fully Automatic Installation (FAI) [4] is an automated installation tool for Debian Linux and other distributions. It can be used to install Linux on a cluster of PCs.

Many papers, for example [6], [7], [9], [13] and [15], discuss IP (v4/v6) router IP address auto-configuration and distribution. However, routing *protocol* auto-configuration is out of their scope.

To summarize the previous work, it is evident that a lot of work has been done in the field of auto-configuration: currently one can install Linux-PCs automatically, configure IPv4 and IPv6 addresses automatically, set up LAN-networks automatically, and configure modems automatically. There are also many high-level architectures and algorithms for auto-configuration of router networks.

However, to the best of our knowledge, there are no practical systems that would enable automatic installation, configuration, and startup of an OSPF-router network. We intend to fill this gap.

## 3 System overview

One possible way to convert a PC network into an OSPF network would be to use portable media, like a CD or an USB stick, and run the required auto-configuration software from that on each of the network's PCs. However, this may be logistically complicated if the PCs are in separate physical locations. Another problem may arise if computer display or keyboard is not available in some of those physical locations.

The conversion would still be relatively easy if every PC in the network could initially be connected from a single management point. This would be possible if a separate management LAN, that would extend to every PC in the network, were available. However, in many cases, establishment of such a LAN will not be possible.

Hence we propose the following solution to the problem: once certain pieces of software have been installed on a "master PC", that will most likely be a laptop computer, a networking novice can take the master PC, plug it into one of the network PCs, and run a script, which converts the PC network into an OSPF-network.

Our solution is based on the idea of incrementally increasing the range of connectivity until all PCs in the network have been configured to run OSPF. Figure 1 illustrates this concept.

In the figure, "master" refers to an auto-configuration master PC. This PC has been configured beforehand to manage the installation and configuration of network. Subsection 4.1 presents details of the master PC. All that is needed in order to convert the PC network into a router network is to plug the master PC to one of the PCs in the network, run a script, and wait.

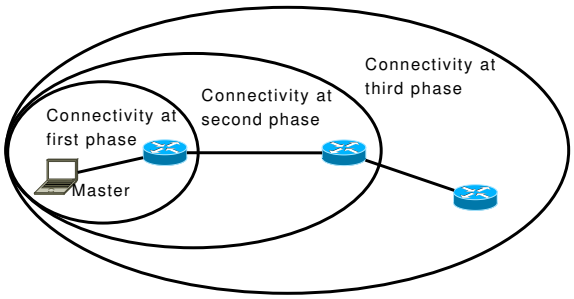


**Figure 1. Connectivity expands in the network as the routing protocol is flooded.**

Our system is composed of the following three subsystems:

- Operating System (OS) **auto-installation** subsystem (presented in subsection 4.2)
- IP address **auto-assignment** subsystem (presented in subsection 4.3)
- OSPF **auto-configuration** subsystem (presented in subsection 4.4)

The OS auto-installation subsystem, as its name suggests, installs Linux operating system to every PC that has been physically connected to the network with an ethernet-cable. The IP address auto-assignment subsystem distributes IP addresses to every PC in the network. Finally the OSPF auto-configuration subsystem takes care of routing software installation, configuration and start-up.

Based on the previous work done within the field it seems clear that the last subsystem (OSPF auto-configuration subsystem) has received least attention. Hence we will describe it in more detail than the other two subsystems.

The benefit of the subsystem separation is that it allows the use of three different levels of control in the network PCs:

- Completely empty PCs in the network (minimum control)
- PCs that have an OS installed but no interface IPs configured (medium control)
- PCs that have OS installed and manually configured interface IPs (maximum control)

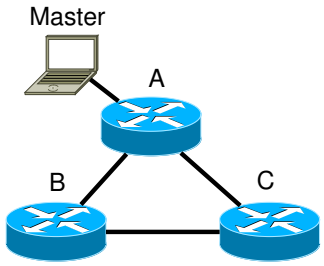


**Figure 2. Simple example network.**

We tested the last one of these scenarios. In other words, we installed OS and entered IP addresses to the network PC manually. We then connected the master PC to one of the network's PCs, and executed the OSPF auto-configuration subsystem.

Throughout the paper we will use an example network that is presented in Figure 2. While describing the system, we assume that nodes A, B and C are empty PCs in the beginning; in other words, nothing has been installed to A, B or C.

# 4 System details

## 4.1 Master PC

Main purpose of the master PC is to initiate and control each of the mentioned subsystems (OS auto-installation, IP address auto-assignment, and OSPF auto-configuration).

Master PC should have Linux operating system installed. We have tested the system with Ubuntu 6.06 distribution. The master should also contain latest version of Perl, which was 5.8.7 during our tests. The master PC should also contain the following pieces of software relating to the subsystems:

- **OS auto-installation**:
  OS mirror, FAI-server, TFTP-server, auto-installation master and agent scripts
- **IP address auto-assignment**:
  Auto-assignment master and agent scripts
- **OSPF auto-configuration**:
  Routing software (Quagga) tarball [10], Auto-configuration master and agent scripts
- Additionally:
  Perl Net::SSH::Expect-, Net::SCP::Expect-, and Net::DHCP-modules and all their perquisite modules

Normally perl modules are installed via the procedure described in [11]. However, this would have been impossible in our system. Hence we downloaded all of the required

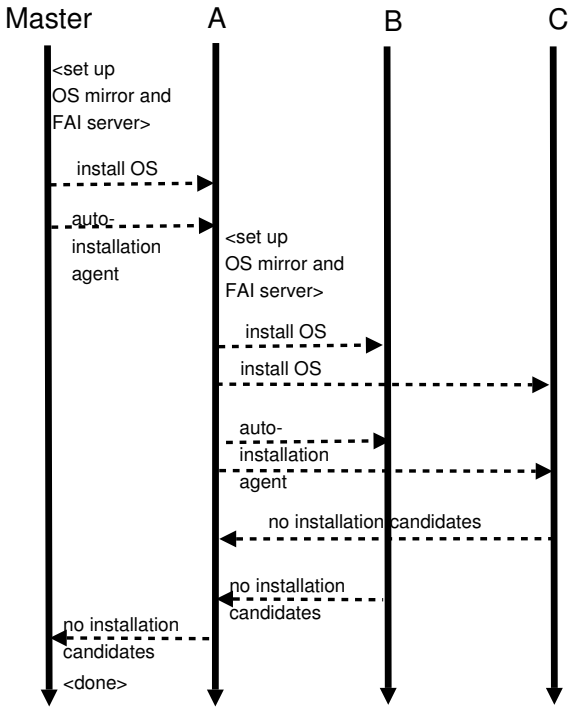


**Figure 3. Flowchart of the OS auto-installation subsystem in the example network.**

modules, and placed them on the master PC to the same directory where the master-scripts are located, under `lib`-subdirectory. We then advised the master and agent scripts to use this newly created library:

```
use lib 'lib/';
```

This way the scripts can use SSH-, SCP-, and DHCP-modules without the need to install anything, and the newly created library can be easily sent to other PCs in tar-format (tarball).

It should be noted that this approach may not work with all versions of Perl.

## 4.2 OS auto-installation subsystem

A conceptual illustration of the OS installation subsystem logic is presented in Figure 3. The illustration is based on network of Figure 2.

The Network Interface Cards (NICs) of the PCs must support Preboot Execution Environment (PXE) in order for the OS installation subsystem to work. Most modern NICs support this.

In the beginning, the master PC contains both a mirror of the operating system that is going to be installed, as well

as a FAI install server. At this stage PC A is an install client (a host that will be installed using FAI and a whose configuration will be provided by the install server).

PC A is booted via NIC. It gets an IP address from install server (master PC) and boots a Linux kernel, which mounts its root file system via NFS from the install server.After the kernel is loaded, FAI startup script performs automatic OS installation for A, which does not require any interaction.

*Auto-installation master* located at the master PC transfers OS mirror, FAI-server, and an *auto-installation agent*. to A. Once the OS has been installed to A, Auto-installation agent located at A sets up OS mirror and FAI server locally, and executes the same process for B and C. This is done recursively until all PCs have OS installed.

In the end, the auto-installation agents notice that there are no more installation candidates left, and deliver this information to the master. At this point the IP address auto-assignment subsystem may be executed.

### 4.3 IP address auto-assignment subsystem

A simplified illustration of the IP address auto-assignment subsystem logic is presented in Figure 4. The illustration is also based on network of Figure 2.

The *auto-assignment master* located at master PC implements a DHCP-server. A pre-requirement for a DHCP-server to work is a route to IP address `255.255.255.255`. If, for instance, the DHCP-server should give IP addresses to hosts behind NIC `eth0`, the following command needs to be issued at command line:

```
sudo route add -net 255.255.255.255 \
netmask 255.255.255.255 dev eth0
```

In the beginning, all network PCs must contain a Perl-script that checks once in a minute if interfaces have been configured, and if they have not, sends out a DHCPDISCOVER-message out of all network interfaces. This script may be placed to the PCs either manually during their installation or during OS auto-installation subsystem execution.

When the auto-assignment subsystem is run, the auto-assignment master gives an IP address to any PC that asks for it (A in the example case). This is done by standard DHCP-procedure.

When PC A has received IP address, the master can send `/etc/network/interfaces`-file to it. This file describes the IP configuration that the PC should use. Format of the file is following:

```
iface eth0
  inet static
  address 10.0.1.1
```

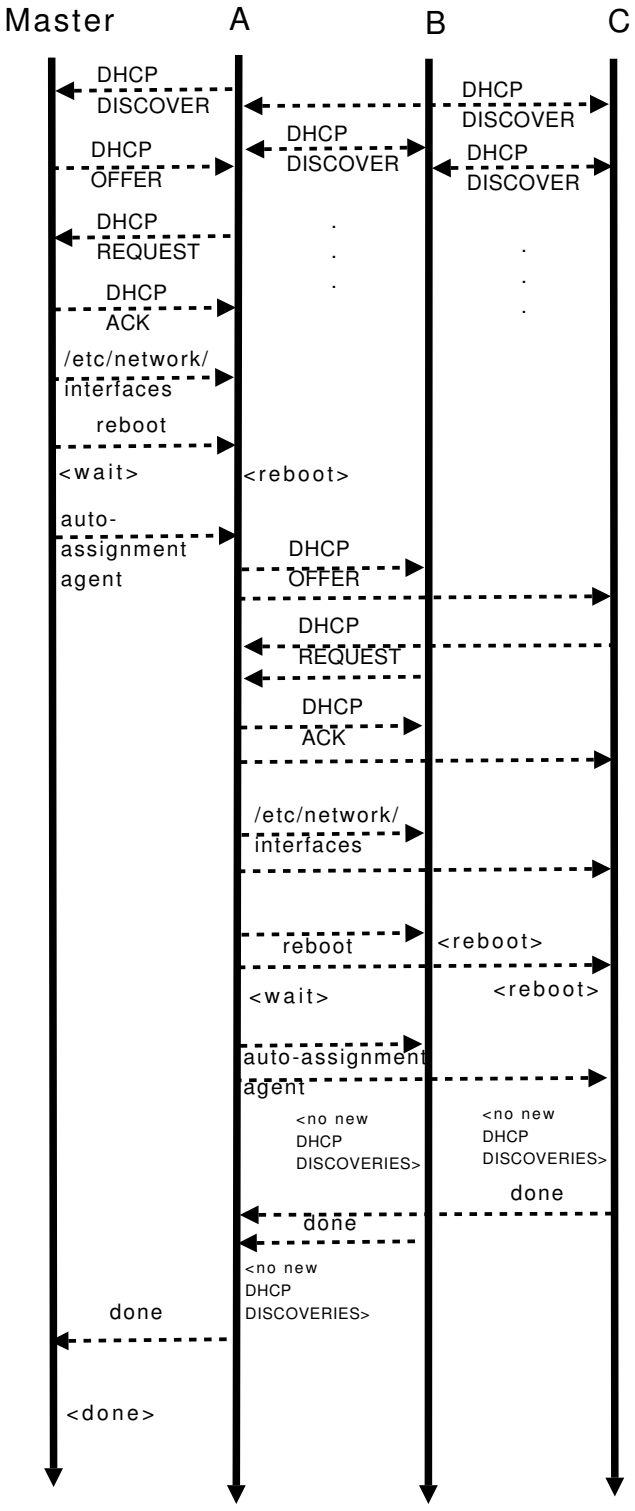


**Figure 4. Flowchart of the IP auto-assignment subsystem in the example network.**

```
  netmask 255.255.255.0

iface eth1
  inet static
  address 10.0.2.1
  netmask 255.255.255.0

...
```

The addresses in the `/etc/network/interfaces`-file can be either user-specified or automatically generated (if the network is behind NAT). Also if the network is not behind NAT, but the routers do not need to be reached from Internet, the router-router interface IPs can be automatically generated, and only user interface IPs entered manually.

Next the master takes SSH-connection to A and tells it to reboot. During the reboot, A reads the IP configuration from `/etc/network/interfaces`-file, and configures interfaces according to it.

When A has rebooted, the master sends an *auto-assignment agent* to it. Master takes an SSH-connection to A and starts the agent. The agent then executes the same process for B and C. This is done recursively until all PCs have IP addresses configured.

It should be noted that it is not possible to run DHCP sever on multiple interfaces simultaneously. Hereby a hack is needed in order for the auto-assignment agent to work. One possibility is to always create the route entry for IP address `255.255.255.255` whenever a DHCPDISCOVER-message is received from some neighbor, and delete it after DHCPACK is sent to that neighbor.

Once the Interfaces' IPs have been configured, the correctness of configuration should naturally be verified (for instance with ping-tool).

## 4.4 OSPF auto-configuration subsystem

At this stage we assume that Linux operating system with SSH- and SCP-daemons has been installed to each of the PCs. This could have been done manually or by using the OS auto-installation subsystem. We also assume that correct IPs have been configured to the PCs' interfaces. Again, this could have been done manually or by using the IP auto-assignment subsystem.

Every PC that is physically connected to the network and has correct IP address configuration when the auto-configuration starts, will be configured to run OSPF-protocol according to a user-defined policy.

The policy is described in a text file at the master PC. The policy file will be used by auto-configuration agents in the auto-configuration process. Most attributes in the policy-file have suitable default values, and the user should not be required to alter them. However, for instance, a default gateway should be entered manually to the policy file, if one is supposed to be configured for the network. The policy file has the following simple format:

```
default-internet-gateway-IP-address="10.0.0.1"

MD5-authentication = "yes"

permit-router-remote-configuration = "no"

load-balancing-for-equal-cost-paths = "no"

...
```

We have not yet implemented many policy features to the system. However, additional features are easy to add. Basically a new feature requires just one if-block to the configuration agent that will be described in subsection 4.4.4.

Our goal is that even a person with very limited computer networking skills will be able to fill in the necessary values to the policy file.

### 4.4.1 OSPF Auto-configuration process

When the OSes have been installed to the PCs, the PC network has been physically connected, and interface IPs configured, the OSPF auto-configuration master is started with the following command:

```
sudo perl autoconfiguration_master.pl
```

From there on, the procedure is fully automatic up to the point when the OSPF network is up and running.

A slightly simplified illustration of the OSPF auto-configuration subsystem logic is presented in Figure 5. Again, the illustration is based on network of Figure 2.

It should be noted that the "Autoconfig package" presented in Figure 5 contains: 1) auto-configuration agent Perl-script, 2) routing software tarball, 3) a file that contains IP addresses of interfaces that have already been configured to run OSPF (the master script maintains this file), and 4) the policy file.

When the auto-configuration master script is started it executes as follows:

- The master probes for neighboring PCs by sending ICMP echo requests to each possible IP in the range of its interfaces (Broadcast pings could not be used for neighbor discovery, since the operating system did not reply to them).
- At this stage the master should receive one neighbor reply (from PC A). The IP that sent this reply will be a candidate.
- The master creates a default route towards A, and uses SCP to send required Perl-library (lib.tar.gz) to A.

- The master sends Autoconfig package to A and starts the agent at A (via SSH).

At this stage the agent is running at PC A. The agent located at A executes as follows:

- The agent checks all configured IPs from the file that was included in the Autoconfig package. Only the master IP should be in the file.
- The agent discovers neighboring PCs by sending ICMP echo requests to each possible *non-configured* IP in the range of its interfaces (the agent does not send request to master).
- The agent should receive two neighbor replies (from B and C). The IPs that replied will be new candidates.
- The agent uses SCP-tool to send required Perl-library (lib.tar.gz) to B and C.
- The agent takes SSH-connection to B and C, decompresses Perl-library, and configures a default route to them via itself; in other words, at B and C the agent essentially says "route add default gateway A". This is done to enable master to communicate with new candidates.
- The agent installs, configures and starts OSPF locally.
- Finally the agent sends its own interface IPs (newly configured) and candidate interface IPs to the master.

At this stage A is the only running OSPF-router in the network. Now the master updates configured IPs, sends Autoconfig package to candidate B, and starts the agent at B.

The newly started agent located at B executes as follows:

- The agent opens the file that contains already-configured IPs, and finds out that A has already been configured, so it will not send Perl-library to A.
- The agent sends Perl-library to C.
- The agent then takes SSH-connection to C, and finds out that the Perl-library has already been decompressed. This means that B should not configure a default route to C, since someone (A in this case) has already done it. This means that B has "0 un-configured neighbors" as is stated in the figure.
- The agent logs out from C and decompresses, installs, configures and starts OSPF locally, and finally sends its newly configured interface IP addresses and neighbor (candidate) interface IP addresses to the master.

Next the master updates configured IPs, sends Autoconfig package to candidate C, and starts the agent at C.

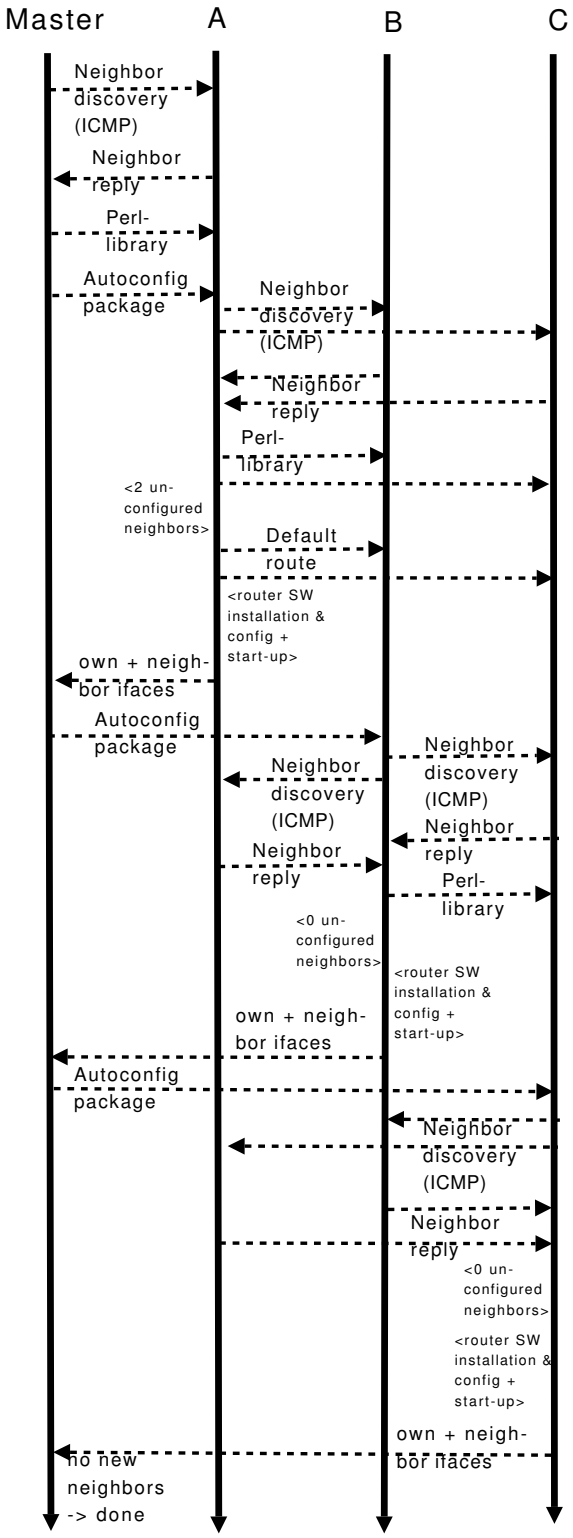


**Figure 5. Flowchart of the OSPF auto-configuration subsystem in the example network.**

The newly started agent located at C executes as follows:

- The agent opens the file that contains already-configured IPs, and finds out that A and B have already been configured, so it will not send Perl-library to them.
- The agent decompresses, installs, configures and starts OSPF locally, and sends its newly configured interface IP addresses to the master (at this stage there were no new candidates).

Now the master notices that there are no candidates left and the process stops. All of the PCs in the network are now running OSPF.

#### 4.4.2 Additional features of OSPF auto-configuration

In addition to what was presented in the example, the auto-configuration system can currently handle broadcast networks (hubs/switches) and stubnets.

A broadcast network that contains more than two routers does not cause any changes to configuration syntax compared to a network that has only two connected routers.

We define stubnet to be a network from which there is no neighbor reply even when its IP is configured to the router. Our system interprets the corresponding interface to be a customer-interface, and hereby it is configured to be passive. This means that the router does not send or accept OSPF packets on that interface, but advertises the interface as a stub link in its router Link State Advertisements (LSAs).

#### 4.4.3 OSPF auto-configuration master

The auto-configuration master is about 500 lines long Perl-script. The script is located on the master PC. The main purpose of the auto-configuration master is to send auto-configuration agents to each of the PCs in the network.

The following is a slightly simplified pseudo-code presentation of the OSPF auto-configuration master.

```
function autoconfiguration_master {

  // initializations
  candidate_IPs = ();
  configured_IPs = ();

  candidate_IPs <- enqueue (discover_IPs());

  // first time there should be 1 candidate
  while(candidate_IPs != empty)
  {
    IP = dequeue (candidate_IPs);
    if (IP exists in configured_IPs)
      next;
    send_configured_IPs_to (IP);
    send_policy_file_to (IP);
    send_routing_software_tarball_to (IP);
    send_configuration_agent_to (IP);
    if (first_time)
      send_perl_library (IP);
    start_configuration_agent_at (IP);
    configured_IPs <- feedback_from_agent;
    candidate_IPs <- feedback_from_agent;
  }
}
```

First the master initializes candidate IPs and configured IPs as empty queues. Then it discovers IP address of the neighboring PC by sending an ICMP echo request to every possible IP in range of its interfaces. The master should receive exactly one response (if the PC that is connected to it has correct IP configuration). The master puts the IP address from which it received the response to queue (enqueue). The main loop of the master is following: First the master extracts an IP address from candidate queue (dequeue). It checks if the IP in question has already been configured. If it has not, the master sends 1) a file containing already-configured IPs, 2) the policy file, 3) the routing software tarball, and 4) the configuration agent to the PC. In the first time the master also sends Perl-library tarball, that contains Perl-modules needed by agent, to the PC. Next the master takes SSH-connection to the PC and starts the agent. In Linux the agent can be started to the background (so that it will not terminate when SSH-connection is terminated) with the following command:

```
sudo screen perl autoconfiguration_agent.pl
```

At the end of loop the master waits for the agent to send newly configured and candidate IP addresses to it.

#### 4.4.4 OSPF auto-configuration agent

The auto-configuration agent is another, also about 500 lines long, Perl-script. The agent is also initially located at the master PC. Whenever an agent is sent to a network PC it takes care of:

- Discovery of neighbor PCs.
- Preparation of neighbor PCs (default route configuration).
- Decompression, installation, configuration, and starting of routing software.
- Informing the master about own (configured) and neighboring IPs.

The agent could also remain running at the PC and monitor user-defined parameters. However, we have not yet implemented this feature.

The following is a slightly simplified pseudo-code presentation of the OSPF auto-configuration agent.

```
function autoconfiguration_agent {

  // initializations
  stubs = ();
  candidate_IPs = ();
  own_IPs = discover_own_IPs();
  interfaces = discover_own_interfaces();
  configured_IPs <- read_from_file();

  for each interface in interfaces
  {
    response = 0;
    for each IP in range of interface
    {
      if IP exists in configured_IPs
        next IP;
      if ( icmp_reply(IP) )
      {
        response = 1;
        candidate_IPs <- IP
        prepare_candidate (IP);
      }
    }
    if ( ! response )
      passive_interfaces <- interface;
  }
  send_to_master (own_IPs);
  send_to_master (candidate_IPs);
  extract_configure_and_install_routing_software
            (own_IPs, passive_interfaces);
  start_routing ();
}
```

First the agent initializes stub networks and candidate IPs as empty lists. Then it discovers own interfaces, own IP addresses, and reads already configured IP addresses from the file that the master sent along with the agent.

For each interface, the agent sends two ICMP echo requests to every *non-configured* IP address within the range of the interface (for instance, there are 253 possible IP addresses in class C networks). It should be noted that this could not be done in threads, since the Perl threads-module caused segmentation faults. It takes about one second to check one IP address, so a class C network would take about 253 seconds to check.

If the agent gets an ICMP echo reply from some IP, that IP is added to candidates. The agent ”prepares” each candidate, in other words, sends Perl-library to the candidate, and installs a default route to the candidate.

If there is no reply from any IP address of an interface, say `eth3`, that interface is configured to be passive (customer interface). In practise this means that the agent will write the following to OSPF configuration file: `passive-interface eth3`. It should be noted that the actual configuration syntax is always platform dependent, so if one wishes to extend our system beyond PC/Quagga environment, platform detection and adaptation needs to be implemented.

The agent sends own (configured) IP addresses and candidate IP addresses to master and finally extracts, installs, configures, and starts routing software that the master sent.

The agent writes configuration files for the routing software based on the policy file that the master sent. For example, if the agent notices that one of the interfaces on the PC it is running on has the IP address of default gateway specified in policy file, the agent writes the following to configuration file: `default-information originate`.

Each ethernet-interface that the agent detects (using `ifconfig`-tool) will cause the agent to write the following line to configuration file: `network X.Y.Z.V/N area 0`. Here `X.Y.Z.V/N` is the IP network configured for the interface.

The following list presents two examples of policy→configuration conversions that the agent may perform:

- **policy**:
  `permit-router-remote-configuration = "no"`
  **configuration**:
  ```
  access-list acl1 permit 127.0.0.1/32
  access-list acl1 deny any
  line vty
  access-class acl1
  ```

- **policy**:
  `MD5-authentication = "yes"`
  **configuration**:
  under each `interface <ifname>` line:
  ```
  ip ospf authentication
  message-digest
  ip ospf message-digest-key 1 md5
  <KEY>
  ```
  under `router ospf` line:
  ```
  area <AREA> authentication
  message-digest
  ```

The first example policy states that routers should not allow configuration connections coming from the network (only local configuration should be allowed). The corresponding configuration commands create an access list that allows connections coming from loopback IP address, and denies all other connections. Finally the access list must be taken into use (last two configuration lines).

The second example policy states that routers should use MD-5 authentication. The corresponding configuration is even more complicated than in the first example. The configuration file must explicitly enable MD-5 on every interface and create a key for the interfaces. Finally the authentication must be taken into use (last configuration line).

More information about Quagga configuration can be found at [14].

## 5 Tests of the OSPF auto-configuration subsystem

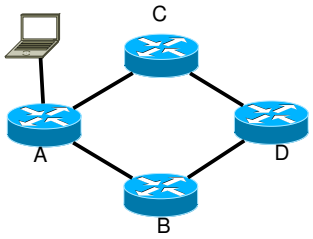


**Figure 6. The first test network.**

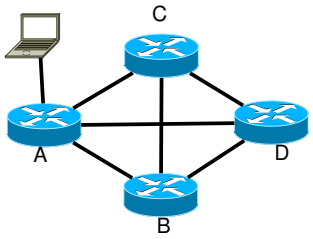


**Figure 7. The second test network.**

In our testbed network, each of the PCs contained 256 MBs of RAM and 1300 MHz CPUs, as well as 3com 3c905 NICs. The master PC had 516 MBs of RAM and 1600 MHz CPU. We installed Ubuntu 6.06 operating system to each of the PCs beforehand. This is a very easy task and can be completed by computer novices. We also installed SSH- and SCP-daemons to each of the PCs. This can be done from the console with command:

```
sudo apt-get install openssh-server
```

We used Quagga routing software [10] version 0.98.6 in our system. We downloaded Quagga tarball to the same directory where the master (and agent) script is located. We also downloaded all required Perl-libraries to the same directory under `lib`-subdirectory, and created a tarball of this library with the following command.

```
tar -cvzf lib.tar.gz lib/
```

Our two test networks are presented in Figures 6 and 7.

We used the auto-configuration system to install, configure, and start OSPF routing in both networks. The tests done in the networks are referred to as ”Test 1” and ”Test 2”. Results from both tests are presented in the table 1.

The routers were configured in the same order (A, B, C, D) in both tests. This is because the interfaces are always processed in the same order, and the A→D interface became after A→B and A→C -interfaces (so the last candidate that auto-configuration master put to queue was D in both tests).

**Table 1. Test results.**

| | Test 1 | Test 2 |
|---|---|---|
| Test started | 0 sec | 0 sec |
| Sent agent to A | 291 sec | 292 sec |
| A is running OSPF | 1799 sec | 2401 sec |
| Sent agent to B | 1978 sec | 2579 sec |
| B is running OSPF | 2969 sec | 3844 sec |
| Sent agent to C | 3147 sec | 4022 sec |
| C is running OSPF | 3788 sec | 4663 sec |
| Sent agent to D | 3967 sec | 4842 sec |
| D is running OSPF | 4298 sec | 5172 sec |

It can be seen that the installation, configurations and startup of the four-node network takes about 1,2 hours in the first test, and about 1,4 hours in the second test. It usually takes about 10 minutes for an agent to decompress the Perl-library sent to its neighbor and to install default route to it over SSH. For example, in Test 2 it took 1720 seconds for the agent located at PC A to finish all three SSH-connections, which is roughly three times the 10-minute period.

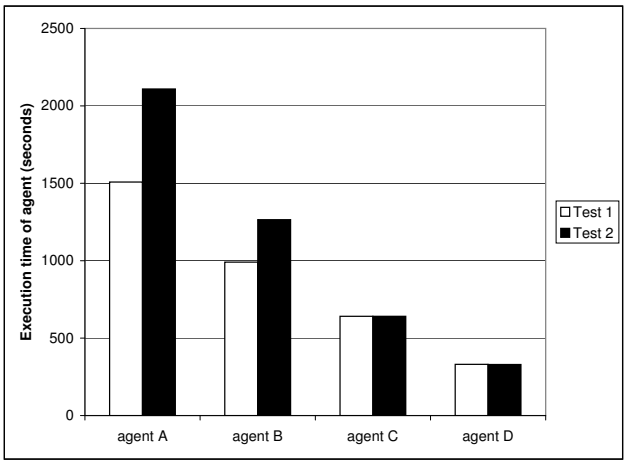


**Figure 8. Execution times of agents.**

The same observation can be made from Figure 8, which presents execution times of agents that were located in PCs A, B, C and D. Based on the figure, it is clear that agent execution time mostly depends on the number of neighbors that the agents needs to connect via SSH. For example, in test 2, agent located at A needs to take three SSH-connections (to B, C and D), agent located at B two (to C and D), agent located at C one (to D), and agent located at D zero.

It should be noted that in our system the master checks if an agent has completed at 300 second intervals. This means that the execution times presented in Figure 8 are estimates.

The actual execution times of agents may be several minutes shorter. An estimate of agent's minimum execution time is the time it takes for an agent to extract, install, configure and start the routing software. This time is about 150 seconds on the PCs we used.

## 6 Conclusions

Auto-configuration is seen as a necessary step towards disruption-free communications. However, most published research papers in the field describe only high-level architectures, and give superficial (if any) descriptions of prototype implementations.

In this work we have developed a practical auto-configuration system that converts back-to-back connected commodity PCs into an OSPF-network. The system is composed of three subsystems: Operating System auto-installation subsystem, IP address auto-assignment subsystem, and OSPF auto-configuration subsystem.

We have implemented the last subsystem and tested its performance. The results show that the subsystem's execution time mostly depends on the total number of SSH-connections the auto-configuration agents need to take during the auto-configuration process.

During the implementation we encountered many problems. For example, we noticed that the Perl Net::SCP::Expect-module may cause race conditions. For instance, if an agent located at a PC, say A, tries to send a file to another PC, say B, that already has an agent running, then the script located at A dies. This is why the configuration is currently done sequentially.

Another problem we encountered was that SSH-connections taken from a script using the Net::SSH::Expect-package are very slow. For instance, it takes about 10 minutes for an agent located at PC A to send Perl-libraries and configure default route on B. However, this does not stop our system from working.

Our main future research topics are performance improvement of the auto-configuration subsystem and implementation of the other two subsystems.

## References


[1] A. G. Ganek, T. A. Corbi: The dawning of the autonomic computing era. IBM Systems Journal, Volume 42, Number 1, 2003.

[2] E. Lehtihet, H. Derbel, N. Agoulmine, Y. Ghamri-Doudane, S. van der Meer: Initial Approach Toward Self-configuration and Self-optimization in IP Networks. Management of Multimedia Networks and Services, Volume 3754/2005, October 2005.

[3] T. Bullot, D. Gaiti: Towards autonomic networking and self-configurating routers. Network Control and Engineering for Qos, Security and Mobility, V, Volume 213/2006, Pages: 127-142, January 2007.

[4] FAI-tool: `http://www.informatik.uni-koeln.de/fai/`, 2007.

[5] Zero Configuration Networking, `http://www.zeroconf.org`, 2007.

[6] A. Dimitrelis, A. Williams: Autoconfiguration of routers using a link state routing protocol. Internet Draft, October 2002.

[7] D. Braun, S. Mukherjee, C. Akinlar: Zero configuration networking. United States Patent 7002924, Filed in February 2001.

[8] D. Arai, K. Yoshihara, A. Idoue, H. Horiuchi: Server Support Approach to Zero Configuration of Power Line Communication Modems and Coaxial Cable Modems. Proc. APNOMS 2007, Pages 92-101, October 2007.

[9] S. Thomson, T. Narten: IPv6 Stateless Address Autoconfiguration. RFC 2462, December 1998.

[10] Quagga routing software: `http://www.quagga.net`, 2007.

[11] Perl module installation: `http://www.cpan.org/modules/INSTALL.html`, 2007.

[12] C. Akinlar, A. Udaya Shankar: IPv4 Auto-Configuration of Multi-router Zeroconf Networks with Unique Subnets. Networking - ICN 2005, Volume 3421/2005, April 2005.

[13] R. Oguz Altug, C. Akinlar: Unique Subnet Autoconfiguration in IPv6 Networks. IPOM 2006: 108-119.

[14] G. W. Schmied: Integrated Cisco and UNIX Network Architectures. Cisco press, 2004.

[15] G. Chelius, E. Fleury, L. Toutain: No administration protocol (NAP) for IPv6 router auto-configuration. Advanced Information Networking and Applications, Volume: 2, Pages: 801- 806, March 2005.