Automated allocation and configuration of dual stack IP networks

Wilfried Daniels, Bart Vanbrabant, Danny Hughes, Wouter Joosen

iMinds-DistriNet, KU Leuven, Heverlee, B-3001, Belgium
Email: {firstname.lastname}@cs.kuleuven.be

***Abstract*—The manual configuration and management of a modern network infrastructure is an increasingly complex task. This complexity is caused by factors including heterogeneity, a high degree of change and dependencies between configuration parameters. Due to increasing complexity, manual configuration has become time consuming and error prone. This paper proposes an automatic configuration tool for dual stack IP networks that addresses these issues by using high level abstractions to model the network topology and key parameters. From this high level configuration model, low level configuration files can be generated and deployed. A key parameter specified in the high level model is the network prefix of the entire network. When translating a configuration model to configuration files, IPv4 and IPv6 subnets need to be allocated. We provide an allocation algorithm to do this allocation in the most efficient way. Evaluation of our approach shows that there is a significant increase in efficiency compared to manual configuration.**

I. Introduction

The manual configuration of modern network infrastructures is a very time consuming task. To make matters worse, human mistakes occur frequently. Studies have shown that up to 36% of the errors that cause downtime for ISPs are operator errors [1]. In more than 50% of cases, operator errors took the form of configuration errors. Finding and correcting configuration errors takes a lot of time and resources. In fact, 75% of Time To Repair (TTR) hours are spent searching for and correcting operator configuration errors. The resulting outages can be very costly [2], [3]. In summary, the manual configuration of networks is inefficient due to the complexity of the configuration problem. This complexity arises from a number of sources:

The first source of complexity is *dependencies* between configuration parameters. For example, the configuration of a new primary DNS server requires a chain of reconfigurations as the DHCP server has to be reconfigured to supply hosts in the network with the IP of this new DNS server and the secondary DNS servers will have to be reconfigured to do zone transfers from the new DNS server. The failure to adjust one of these parameters can result in an interruption of the network service.

A second point is the increasing *heterogeneity* of networks. Each device or operating system has its own configuration system and syntax. The network administrators have to become familiar with all of these different systems to configure the entire network. Learning to use these systems is time consuming, and the risk of configuration errors for administrators working with new systems is significantly higher.

Finally, networks are becoming very *dynamic*. The topology and requirements frequently change, causing reconfiguration. These reconfigurations are difficult to support manually. The effort to manually reconfigure a network is approximately linear to the size of the network. For this reason, manual reconfiguration is very inefficient for large networks with a high degree of change.

This paper addresses the aforementioned problems by using a configuration model to specify high level configuration parameters. This model-based approach is implemented in a tool that focuses on configuring and managing a dual stack network infrastructure. In the high level configuration model abstractions of the network components and relations between them are specified. This way dependencies between low level parameters are resolved automatically, and a uniform interface is offered that encapsulates heterogeneity. The proposed model provides a uniform centralized interface to the configuration of the network infrastructure, which increases the efficiency of configuration and management.

The main contributions of this paper are twofold. The first contribution is a model-based approach to configuring a dual stack network infrastructure. By introducing a model containing a high level configuration specification, the complexity of the configuration problem is reduced. The second contribution is the design and implementation of a prefix allocation algorithm. Using this algorithm, a single prefix specified in the high level model is split and allocated to all the subnets in an optimal way.

The remainder of this paper is structured as follows. In section II we identify the requirements of the proposed tool. Section III further explains the design of both the tool and the configuration model. Section IV describes the prefix allocation algorithm. In section V we evaluate the proposed tool. Section VI discusses related work. Finally, we conclude in section VII.

II. Requirements

The proposed automatic configuration tool has to satisfy the following requirements:

- **Efficiency**: The main aim of the tool is to improve the efficiency of configuration by decreasing complexity. This way configuration can happen with less effort and

978-3-901882-50-0

the likelihood of configuration errors is lower. The tool achieves this by using a high level configuration model to specify the configuration.

- **Support for dual stack IP networks**: The IPv4 address space is depleting, despite measures taken in the 90's like CIDR [4] and NAT [5]. The demand for IPv4 address space has only increased the last few years, with the advent of internet enabled mobile devices like smartphones and the many new internet users in developing countries. The only worldwide accepted solution for this problem is the migration to IPv6. A first step in this migration is running networks in a dual stack configuration. Special attention is paid to make sure dual stack networks are supported by the tool as well.
- **Support for critical configuration aspects**: The aim of the proposed tool is to set-up basic network functionality in a network infrastructure, facilitating deployment of more advanced services like mail servers or remote file systems. A minimal set of services is selected which are necessary for a functional network, including configuration of the following aspects:
  - Host configuration protocol: The configuration of DHCPv4 [6], DHCPv6 [7] or SLAAC [8] in all the subnets of the network.
  - DNS Server: The configuration of DNS servers in a master-slave configuration, including the generation of forward and reverse zones containing all hosts.
  - Routing: The configuration of both static and dynamic routing. The dynamic routing protocols supported are OSPFv2 [9] and RIP [10] for IPv4, and OSPFv3 [11] and RIPng [12] for IPv6.
  - Generic servers: A hosts in a subnet can be designated as a server, which will be automatically configured with a static IP and are included in the DNS records. After this basic configuration other services can be configured on top of these generic servers.

III. Design

In order to meet the proposed requirements, several preliminary design decisions were made.

A. IMP framework

The proposed configuration tool is built on the IMP framework [13]. The Integrated Management Platform is a generic framework for system configuration tools that manage an entire infrastructure. In most cases, the configuration parameters of different network components are closely related. These relations are mostly determined by the physical network topology. When a parameter changes, its related configurations should be altered as well. IMP facilitates the resolution and updating of these parameters by providing a framework to develop an object oriented declarative modelling language, in which relations between entities can be modelled at a high abstraction level. This high level configuration model can then be translated to the various low level configuration files on every machine.

Both the definition of the configuration model, containing the definitions of the various entities and their relations, and the translation mechanisms can be isolated in modules. This way tools developed with IMP are modular and extendible.

B. High level configuration model

An example of a high level model is shown in Listing 1. This high level model serves as input for the tool. Based on this model, the tool will generate and deploy the low level configuration files of the various machines in the network.

```
1  net1 = ip::Network(ipversion = "ds", border = ro1,
       routingv4="rip", routingv6="ospf"):
2        ip::ipv6::Range(range="2001:06a8:2900:3828::", mask
           =61)
3        ip::ipv6::DNSConf(conf="dhcpv6")
4        ip::ipv6::HostConf(conf="radv")
5        ip::ipv6::DNS(addr="2001:06a8:2900:3820::1")
6        ip::ipv4::Range(range="172.16.0.0", mask=16)
7        ip::ipv4::DNS(addr="192.168.150.254")
8        ip::DNSDomain(domain="netwglab.cs.kuleuven.be")
9  end
10
11 subnet1 = ip::Subnet(net=net1,name="subnet1",size=200)
12 subnet2 = ip::Subnet(net=net1,name="subnet2",size=200)
13 subnet3 = ip::Subnet(net=net1,name="subnet3",size=200)
14
15 ro1 = ip::Router(name = "router-1",net=net1,os = "ios"):
16       include ip::exportRouterConf
17       ip::RouterInterface(mac="00:b0:8e:0c:84:1c",subnet=
             subnet1,name="FastEthernet1/0")
18       ip::RouterInterface(mac="00:b0:8e:0c:84:1d",subnet=
             subnet2,name="FastEthernet1/1")
19 end
20
21 ro2 = ip::Router(name = "router-2", net = net1 ,os = "
       ubuntu-11.10"):
22       include ip::exportRouterConf
23       ip::RouterInterface(mac="00:0a:5e:3c:6b:e6",subnet=
             subnet1,name="eth0")
24       ip::RouterInterface(mac="00:c0:4f:01:9a:06",subnet=
             subnet3,name="eth1")
25       ip::DNSServer(dnstype="primary")
26 end
```

Listing 1. Example of a high level configuration model

The advantage of specifying the configuration in a high level configuration model is that the complexity of the configuration problem is reduced, consequently reducing the effort and time required to enact configurations. Dependencies between low level parameters can be derived from the model and are automatically kept consistent. Heterogeneity is encapsulated in the high level abstractions, and the configuration model uses a uniform platform independent syntax. Because the configuration is centralized in the configuration model and deployment is automated, the reconfiguration of the network infrastructure can happen efficiently. The system administrators can think on a higher level without being preoccupied by the low level configuration parameters.

IV. Prefix allocation algorithm

To further increase the efficiency and achieve a higher level of abstraction, the allocation of prefixes to subnets is automated. In the configuration model, a prefix is specified for the entire network. When translating the configuration model to the various low level configuration files, this prefix has to

be divided over the subnets of the network. This has to be done in the most optimal way, without wasting address space and possibly allowing aggregated routing. The subnet prefixes are allocated based on the network topology and the input network prefix. The network topology consists of a number of interconnected routers and subnets, with one designated border router connecting the network to the Internet. Prefix allocation happens differently for IPv4 and IPv6. The prefix allocation algorithms proposed are roughly inspired on an algorithm to allocate an IPv6 prefix discussed in [14]. The algorithm proposed in [14] is not fit for the prefix allocation required, mainly due to the strict need of a tree topology. Another problem is the assumption that routers should be connected via point-to-point links, and subnets with hosts are only allowed as leaves in the tree topology. Due to these inflexibilities, some of the basic ideas like the use of pools and metrics are reused to design new algorithms for both IPv4 and IPv6 prefix allocation.

A. IPv4

In the case of IPv4, address space is limited. The given network prefix has to be divided as efficient as possible over the subnets. Due to this, no special care is taken to make routing aggregated along the topology. The IPv4 allocation algorithm is displayed in Listing 2. The allocation algorithm uses a global pool which contains all the prefixes of varying length available. All the subnets in the network are assigned a prefix derived from this pool. The needed prefix length of each subnet is determined by their size, which is specified in the input configuration model. This size should always be rounded up to a power of 2. When the pool does not contain a prefix matching the needed size, an attempt is made to split a prefix of a smaller length until the desired length is reached. The residual prefixes are stored in the pool for reuse. For example, when splitting a /8 to get a /11 prefix, a /9, /10 and an extra /11 are generated. These prefixes are put back into the pool. This ensures an efficient allocation which minimizes the use of the address space.

Used variables and functions:

- P: Pool of all available prefixes of varying length.
- M_s: Desired subnet size for subnet s, rounded up to a power of 2.
- *prefix.split()*: Splits a prefix of length l in 2 prefixes of length $l-1$.
- *neededPrefixLength(size)*: $32-log_2(size)$. Calculates the minimum prefix length needed by a subnet based on its size.

B. IPv6

Contrary to the IPv4 prefix allocation, the IPv6 allocation algorithm doesn't have to be as economical with the address space. Also, the prefix length of an IPv6 subnet is always /64 per specification [15]. Because of this the prefix length doesn't have to be calculated based on the size of the subnet. The proposed IPv6 allocation algorithm will try to allocate the subnet prefixes in a way that allows aggregated routing,

```
1  void allocateSubnets() {
2      P.add(Network.getIPv4Prefix());
3      for(s in Subnets) {
4          length = neededPrefixLength(M_s);
5          s.prefix = getPrefix(length);
6      }
7      return;
8  }
9
10 //Returns a prefix of the given length, splitting
       prefixes in the pool when necessary
11 Prefix getPrefix(length) {
12     pref = getAvailablePrefix(length);
13     while(pref.length < length) {
14         (pref,pref2) = pref.split();
15         P.add(pref2);
16     }
17     return pref
18 }
19
20 //Returns a prefix from the pool smaller or equal to the
       given length
21 Prefix getAvailablePrefix(length) {
22     if(length < 0 || length > 32) {
23         printError("No available prefixes!");
24     } else if(!P.containsLength(length)) {
25         return getAvailablePrefix(length - 1);
26     } else {
27         pref = P.getPrefixWithLength(length);
28         P.remove(pref);
29         return pref;
30     }
31 }
```

Listing 2. Pseudocode prefix allocation IPv4 subnets

but still tries to minimize address space usage. The algorithm operates in 3 steps.

1) Step 1 - Dijkstra: In the first step the network graph is reduced to a tree with the border router as root. Because deeper trees will cause a less ideal prefix allocation, a shortest path tree is preferred. Dijkstra's algorithm is used to generate a shortest path tree from the network graph, minimizing the distance between each router and the border router. The output of this step will consist of the connected child routers of each router per interface.

2) Step 2 - Metric propagation: In the second step each router and each interface is assigned a metric denoting the amount of /64 subnets that are below the router or interface in the tree. The metric propagation algorithm is displayed in Listing 3. The algorithm is written recursively and the network tree is traversed depth-first. The output of this step are the metrics of each interface and router and also a set of of router interfaces that are the default gateway interface on their connected subnet.

Used variables and functions:

- M_r: Metric of router r.
- M_i: Metric of interface i.
- C_i: Output of the Dijkstra step. For each interface i, this is a set of connected child routers in the reduced tree.
- G: This set contains all the router interfaces that are the default gateway interface on their subnet.
- *metricCeil(int metr)*: This function rounds its input up to a power of 2.
- *subnet.isTransitSubnet()*: This function checks whether a subnet is a transit subnet of a router. A subnet is a transit

subnet if any router in the topology has child routers in the reduced tree via this subnet.

```
1  //Entry point
2  void propagateMetric() {
3      anounceMetric(border);
4  }
5
6  //Recursive function, calculates the metric for the
       given router
7  int anounceMetric(Router r) {
8      M_r = 0;
9      for(i in r.Interfaces) {
10         M_i = 0;
11         for childRouter in C_i {
12             M_i = M_i + announceMetric(childRouter);
13         }
14         subnet = i.connectedSubnet;
15         if(!subnet.hasGateway() && (!C_i.empty() || !
               subnet.isTransitSubnet()) {
16             M_i = M_i + 1;
17             G.add(i);
18             subnet.addGateway(i);
19         }
20         M_i = metricCeil(M_i);
21         M_r = M_r + M_i;
22     }
23     M_r = metricCeil(M_r);
24     return M_r;
25 }
```

Listing 3. Pseudocode metric propagation IPv6

3) Step 3 - Prefix allocation: In the final step the prefixes are allocated to the subnets top to bottom in the reduced tree. The algorithm is shown in Listing 4. The same pool system is used as in the IPv4 allocation algorithm, except that in this case there is a pool for each router and interface. Each pool contains the prefixes that can be allocated to the subnets below the interface or router it belongs to. Initially, the network prefix is added to the border router pool. Subsequently, in the *assignPrefixRec()* function, the router interfaces of the border router are allocated a prefix from this pool based on their metric. The allocated prefixes are added to the corresponding interface prefix pools. From these pools, the connected subnets are allocated a /64 prefix when the connecting interface is a gateway interface. Also the child routers connected through these interfaces are allocated prefixes and the *assignPrefixRec()* function is called recursively for these child routers. This continues until the leaf subnets are reached and all the subnets have a /64 prefix assigned.

Used variables and functions:

- C_i: Output from step 1.
- M_r: Output from step 2.
- M_i: Output from step 2.
- G: Output from step 2.
- P_r: Prefix pool of router r. Contains all prefixes available for this router.
- P_i: Prefix pool of interface i. Contains all prefixes available for this interface.
- *neededPrefixLength(metric)*: $64 - log_2(metric)$. Calculates the minimum prefix length required according to the given metric.
- *prefix.split()*: Splits a prefix of length l in 2 prefixes of length $l - 1$.

```
1  //Entry point
2  void assignPrefix() {
3      P_border.add(Network.getIPv6Prefix());
4      assignPrefixRec(border);
5  }
6
7  //For a given router, assign prefixes to connected
       subnets and child routers and recurse over children
8  void assignPrefixRec(Router r) {
9      for(i in r.Interfaces) {
10         length = neededPrefixLength(M_i);
11         prefix = getPrefix(P_r,length);
12         P_i.add(prefix);
13         if(G.contains(i)) {
14             i.connectedSubnet.prefix = getPrefix(P_i,64);
15         }
16         for(c in C_i) {
17             length = neededPrefixLength(M_c);
18             prefix = getPrefix(P_i,length);
19             P_c.add(prefix);
20             assignPrefixRec(c);
21         }
22     }
23 }
24
25 //Returns a prefix from the given pool of the given
       length, splitting prefixes when necessary
26 Prefix getPrefix(P_x,length) {
27     pref = getAvailablePrefix(P_x,length);
28     while(pref.length < length) {
29         (pref,pref2) = pref.split();
30         P_x.add(pref2);
31     }
32     return pref
33 }
34
35 //Returns a prefix from the given pool smaller or equal
       to the given length
36 Prefix getAvailablePrefix(P_x,length) {
37     if(length < 0 || length > 64) {
38         printError("No available prefixes!");
39     } else if(!P_x.containsLength(length)) {
40         return getAvailablePrefix(length - 1);
41     } else {
42         pref = P_x.getPrefixWithLength(length);
43         P_x.remove(pref);
44         return pref;
45     }
46 }
```

Listing 4. Pseudocode prefix allocation IPv6

V. Evaluation

The primary requirement of the configuration tool is to increase the efficiency of the configuration problem. In the evaluation of the tool an attempt is made to quantify this increase of efficiency. The tool is evaluated by comparing an automated configuration with the tool to a manual configuration. The configuration is carried out for an existing network topology currently in use at the Department of Computer Science of the KU Leuven shown in Figure 1.

A good measure for the increased efficiency is the comparison of the amount of effort the network administrators have to do in both cases. This effort translates to the amount of lines of configuration that have to be written. The amount of configuration lines contained in the input configuration model is compared to the number of lines in the various output configuration files.

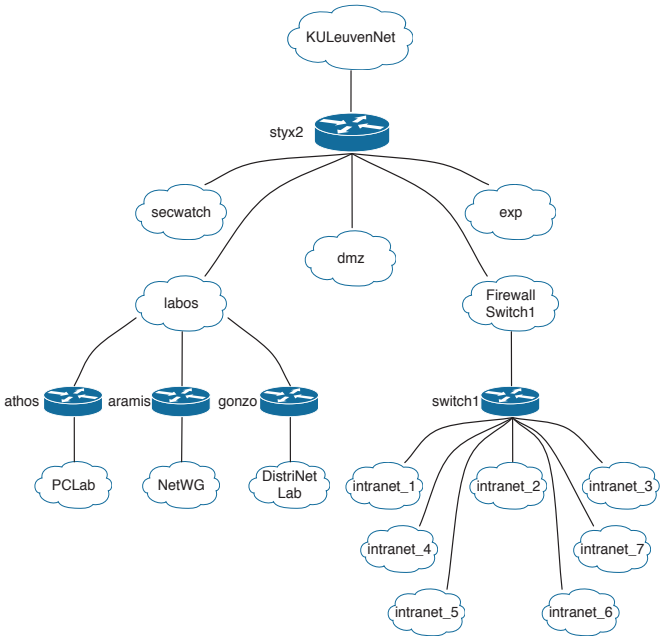

Fig. 1. Network topology Department of Computer Science KU Leuven

To give a more fine grained view of the increase in efficiency, these lines of codes are categorized based on their complexity. An overview of the categories:

- **Addresses**: This category consists of all the lines with an IP address or a netmask. These parameters are very error prone to configure manually. Often dependencies exist between addresses, increasing the complexity.
- **Regular parameters**: This is a residual group containing all the lines that still contribute to the configuration but do not fall under the **Addresses** category. Frequently these lines are switches activating or deactivating a function.
- **Configuration metadata**: This category contains all the lines which do not contribute any useful configuration, but still are required to understand the structure of the configuration file. This category consists of brackets (`{,},(,),[,]`) and keywords.
- **Comments and whitespace**: These are the lines in a configuration file that are completely meaningless configuration wise. The effect of the configuration would be identical without these lines.

The result of the evaluation is shown in Table I. The input configuration model is compared to the output configuration files. When using the configuration model, network administrators have to produce approximately 6.7 times less lines of configuration. This ratio further increases when only the most complex category **Addresses** is compared. In this category, the ratio is roughly 12.8. This shows the positive effects of using a high level configuration model. The amount of complex parameters that have to be produced by the administrators decreases. This is due to the abstractions of the various network elements and their mutual relations in the configuration model. Dependencies between low level parameters are derived from the configuration model, avoiding repetition of these parameters. This way less configuration parameters are needed when expressing the configuration in a high level model.

The compilation time the tool needs to translate the high level model is negligible. Our largest configuration model with 70 managed hosts takes less than 5 seconds to compile.

	Configuration model	Configuration files
Addresses	46	590
Parameters	75	328
Meta	29	81
Comments	37	538
Total	187	1537
Total without comments	150	999

TABLE I
LINES OF CONFIGURATION

VI. Related work

The related work can be divided into automatic configuration of network infrastructures and prefix allocation algorithms.

A. Automatic configuration tools

There are several automatic configuration tools commercially available that partially solve the problems described in the introduction. Examples of state of the art configuration management tools often used in production systems are Cfengine [16], Chef [17] and Puppet [18]. These configuration management tools are used by large dot-com companies like Facebook, eBay, Twitter and Oracle [19], [20], [21].

In contrast to the proposed configuration tool, these tools do not work at a high abstraction level and only facilitate the distribution and management of configuration. Administrators will still have to specify all the low level parameters of each configuration file, resulting in a costly inefficient configuration.

In addition to commercially available tools, some work has been done in research regarding model based configuration particularly for heterogeneous systems. Narain et al. [22] propose a solution for the network configuration problem by using an object oriented configuration model based on first order logic. A solver is used to generate the low level parameters using model finding. This solution differs from our approach by specifying configuration in first order logic. This complicates configuration specification; the administrator has to be familiar with first order logic concepts like inference, quantification and implication. Furthermore, when predicates are written in an inefficient but correct way, a solution may not be found.

Elbadawi et al. [23] present a framework that translates a high level configuration specification to low level parameters of each network component. The tool builds on top of NETCONF, an IETF standardized XML based RPC protocol for network management. The tool coordinates the configuration of interdependent network devices it manages via NETCONF by supplying a global NETCONF interface itself.

B. Prefix allocation algorithms

Atallah et al. [24] propose an algorithm for IPv4 prefix allocation and prove the optimality of the algorithm mathematically. The key difference with the proposed algorithms is the lack of support for aggregated routing.

Bhatia et al. [25] propose a distributed algorithm allocating IPv6 prefixes to MANET (Mobile Ad-hoc Network) nodes using ICMPv6. While a lot of effort is done to recuperate prefixes from dropped out or leaving nodes, no special care is taken to allocate prefixes in an aggregated way. Also, our allocation algorithm guarantees prefixes and routing happens along the shortest path tree with the border router as root. Whereas, in this approach interconnections between nodes are made randomly and a shortest path route to the border router is not guaranteed.

A more advanced dynamic prefix allocation algorithm for MANETs is proposed by Jelger et al. [26]. Introducing the concept of *prefix continuity*, they can guarantee prefixes are allocated in an aggregated way. In addition, path length is taken into account.

Kong et al. [27] developed a dynamic IPv6 prefix allocation algorithm inspired by memory management techniques using a paging mechanism. Bookkeeping guarantees subnet prefixes are split optimally and registers when subnets are added or removed. Route aggregation is not taken into account.

VII. Conclusion

In this paper, an automatic configuration tool for dual stack IP networks is proposed. The primary requirement of this tool is to improve the efficiency of network configuration.

The proposed tool reduces this complexity by using a high level configuration model. From the evaluation, it is clear that the tool significantly increases efficiency. In particular, the amount of complex error prone parameters is reduced.

In the future, the tool will be expanded to include other configuration aspects. As the tool is built on top of the IMP framework [13], support for aspects can be isolated in modules. These can be network related (e.g. firewall configuration), or application oriented (e.g. distributed file systems, web servers). By including these aspects in the high level configuration model, the same increase in efficiency is achieved for these aspects. Furthermore, dependencies between aspects are automatically derived and configured. For example, when adding a web server to the configuration model, the corresponding firewall rules will be automatically adjusted to allow traffic to and from the web server. By adding support for more aspects, the configuration tool can evolve towards a modular integrated configuration platform.

Acknowledgements

This research is partially funded by the Research Fund KU Leuven.

References

[1] D. L. Oppenheimer, A. Ganapathi, and D. A. Patterson, "Why do internet services fail, and what can be done about it?" in *Proceedings of 4th USENIX Symposium on Internet Technologies and Systems (USITS '03)*, 2003.

[2] D. A. Patterson, "A simple way to estimate the cost of downtime," in *Proceedings of the 16th USENIX conference on System administration (LISA '02)*, 2002, pp. 185–188.

[3] Z. Kerravala, "As the value of enterprise networks escalates, so does the need for configuration management," *The Yankee Group*, Jan. 2004.

[4] V. Fuller and T. Li, "Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan," RFC 4632 (Best Current Practice), Internet Engineering Task Force, August 2006.

[5] P. Srisuresh and K. Egevang, "Traditional IP Network Address Translator (Traditional NAT)," RFC 3022 (Informational), Internet Engineering Task Force, January 2001.

[6] R. Droms, "Dynamic Host Configuration Protocol," RFC 2131 (Draft Standard), Internet Engineering Task Force, Mar. 1997, updated by RFCs 3396, 4361, 5494.

[7] R. Droms, J. Bound, B. Volz, T. Lemon, C. Perkins, and M. Carney, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)," RFC 3315 (Proposed Standard), Internet Engineering Task Force, July 2003, updated by RFCs 4361, 5494.

[8] S. Thomson, T. Narten, and T. Jinmei, "IPv6 Stateless Address Autoconfiguration," RFC 4862 (Draft Standard), Internet Engineering Task Force, September 2007.

[9] J. Moy, "OSPF Version 2," RFC 2328 (Standard), Internet Engineering Task Force, April 1998.

[10] G. Malkin, "RIP Version 2," RFC 2453 (Standard), Internet Engineering Task Force, November 1998, updated by RFC 4822.

[11] R. Coltun, D. Ferguson, J. Moy, and A. Lindem, "OSPF for IPv6," RFC 5340 (Proposed Standard), Internet Engineering Task Force, July 2008.

[12] G. Malkin and R. Minnear, "RIPng for IPv6," RFC 2080 (Proposed Standard), Internet Engineering Task Force, January 1997.

[13] B. Vanbrabant and W. Joosen, "A framework for integrated management tools," in *IFIP/IEEE International Symposium on Integrated Network Management (IM '13)*, 2013.

[14] F. Beck, O. Festor, I. Chrisment, and R. E. Droms, "Automated and secure IPv6 configuration in enterprise networks," in *International Conference on Network and Service Management (CNSM '10)*, 2010, pp. 64–71.

[15] R. Hinden and S. Deering, "IP Version 6 Addressing Architecture," RFC 4291 (Draft Standard), Internet Engineering Task Force, Feb. 2006, updated by RFCs 5952, 6052.

[16] "CFEngine Distributed Configuration Management," URL: http://cfengine.com, accessed on Jan. 2, 2013.

[17] "Chef - Opscode," URL: http://www.opscode.com/chef/, accessed on Jan. 2, 2013.

[18] "Puppet Labs - Puppet Open Source," URL: http://puppetlabs.com/puppet/puppet-open-source, accessed on Jan. 2, 2013.

[19] "CFEngine Distributed Configuration Management - Customers," URL: http://cfengine.com/use_cases, accessed on Jan. 2, 2013.

[20] "Users of Puppet - Puppet Labs," URL: http://puppetlabs.com/customers/companies, accessed on Jan. 2, 2013.

[21] "Customers - Chef Opscode," URL: http://www.opscode.com/customers, accessed on Jan. 2, 2013.

[22] S. Narain, "Network Configuration Management via Model Finding," in *Proceedings of the 19th conference on Large Installation System Administration (LISA '05)*, 2005, pp. 155–168.

[23] K. Elbadawi and J. Yu, "Improving Network Services Configuration Management," in *Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN '11)*, Jul. 2011, pp. 1–6.

[24] M. J. Atallah and D. E. Comer, "Algorithms for variable length subnet address assignment," *IEEE Transactions on Computers*, vol. 47, no. 6, pp. 693–699, Jun. 1998.

[25] A. Bhatia and S. Singh, "A Distributed Prefix Allocation Scheme for Subordinate MANET," in *IEEE Asia-Pacific Services Computing Conference (APSCC '08)*, Dec. 2008, pp. 920–925.

[26] C. Jelger and T. Noel, "Prefix Continuity and Global Address Autoconfiguration in IPv6 Ad Hoc Networks," in *Challenges in Ad Hoc Networking, OCP Science*, 2006, vol. 2, no. 2, pp. 311–320.

[27] R. Kong, J. Feng, and H. Zhou, "A Conservative Prefix Delegation Policy for Nested Mobile Networks Based on Paging Mechanism," in *International Conference on Computational Intelligence and Software Engineering (CISE '10)*, Dec. 2010, pp. 1–6.