Principles of Secure Network Configuration: Towards a Formal Basis for Self-Configuration

Simon N. Foley[1] William Fitzgerald[2] Stefano Bistarelli[4,5] Barry O'Sullivan[1,3]
Mícheál Ó Foghlú[2]

[1] Department of Computer Science, University College Cork, Ireland.
s.foley@cs.ucc.ie
[2] Waterford Institute of Technology, Ireland
(wfitzgerald,mofoghlu)@tssg.org
[3] Cork Constraint Computation Centre, University College Cork, Ireland
b.osullivan@cs.ucc.ie
[4] Dipartimento di Scienze, Università "G. D'Annunzio" di Chieti-Pescara, Italy
[5] Istituto di Informatica e Telematica, CNR, Pisa, Italy
bista@sci.unich.it

Abstract. The challenge for autonomic network management is the provision of future network management systems that have the characteristics of self-management, self-configuration, self-protection and self-healing, in accordance with the high level objectives of the enterprise or human end-user. This paper proposes an abstract model for network configuration that is intended to help understand fundamental underlying issues in self-configuration. We describe the cascade problem in self-configuring networks: when individual network components that are securely configured are connected together (in an apparently secure manner), a configuration cascade can occur resulting in a mis-configured network. This has implications for the design of self-configuring systems and we discuss how a soft constraint-based framework can provide a solution.

1 Introduction

Today's management of the telecommunications and enterprise network infrastructure has become fundamentally more complex than it has been in the past. Network elements have expanded to include functionality to cater for a greater variety of applications and across more layers within the network infrastructure [1]. In order to manage the current complexity of networks, both telecoms and enterprise providers use the support of network management systems (NMS) that provide operational and maintenance capabilities at various levels throughout the network [2]. The process of efficiently deploying a complex system of network services on a complex system of network system nodes and having to manage it thereafter is tedious and error prone. These systems are now becoming overwhelmed by the demands placed on the network by its customers [3].

Current research is investigating new avenues to automate the process of NMS to autonomously handle the labour intensive and error prone configura-

tions. Autonomic computing is one such area that attempts to address the challenges posed on current network management systems. The term *Autonomic Computing* was coined by IBM in 2001 and it is motivated by how the human nervous system can react to its environment. The autonomic human nervous system frees the conscious mind from the burden of having to deal with vital but lower-level functions. Similarly, autonomic computing is intended to free system administrators from complex management and operational tasks [4, 5]. This area revolves around self-management, self-configuration, self-protection and self-healing in accordance with the high level objectives of the enterprise or human end-user [6]. The characteristic of self-(re)configuration plays a large part in ensuring that network devices and its hosted services are configured correctly and, more importantly, in a secure manner.

The contribution of this paper is an abstract model that defines objectives that a self-configuring network entity should uphold. A valid configuration is defined in terms of the quality of the configuration of the underlying components. Using this model we demonstrate a configuration cascade problem, whereby the quality of the security of overall configuration is not just based on the quality of the individual components, but also on how they interoperate. This has practical implications for the design of self-configuring systems. In this paper we are primarily concerned with exploring underlying principles for self-configuration; how these issues relate to the implementation of network management architectures [7–12] is beyond the scope of this paper.

The paper is organized as follows. Section 2 outlines an architecture for autonomic network nodes. In Section 3, a formal model is proposed that is intended to characterise the configuration objectives for these nodes. Section 4 describes a metric that is used to specify how effective a node configuration is at achieving its objectives. Using this metric, Section 5 demonstrates that achieving a valid configuration is more difficult than simply ensuring the validity of the configuration of individual components and their immediate connections.

2 Network Node Configuration

Figure 1 depicts the basic structure of a node. A node has an *ability* to perform certain tasks. For example, a router might have the ability to host multiple types of router specific services/protocols to govern its network segment depending on its role within the network at a given moment in time. Nodes assign *contracts* to the services that it hosts. Each node is capable of hosting different types of services but under the self-managed constraints of the node.

Contracts are advertised to the network about a node's ability to offer a particular type of service hosting. A node can migrate or clone, for redundancy reasons, a service to the advertising node based on the contract requirement meeting the needs of the service. Similarly, a node can also advertise to the network the need to off-load certain services and can do so by advertising a service contract in which each node available to host services can decide if it can provide for the service requirements based on its individual policy.

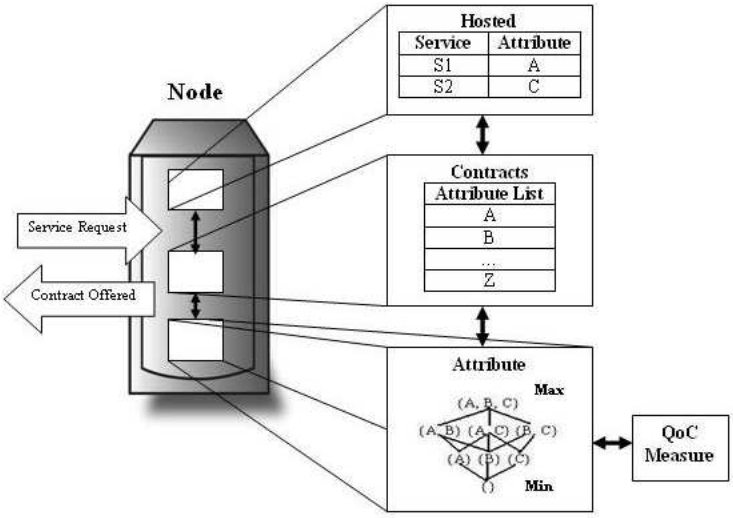

Fig. 1. Architecture of a Network Node

3 A Model of Configuration

A network configuration is defined in terms of a graph of interconnected nodes that host various application-level services. Let *Node* define the set of all possible systems, routers, and so forth that comprise a network. Each node is configured to provide a variety of resources to the services that it hosts. In addition to storage resources such as access to file-systems and databases, hosts may also be configured to offer the use of system-level services/applications, for example, IPv6, SOCKs authentication, SNMP, FTP, and so forth. These are characterised as the *attributes* of a node. Informally, attributes represent anything that a node can offer and/or do.

Let *Attribute* represent the set of all possible attributes of nodes. We use s-expressions [13] to define attributes. This is a Lisp-like notation for naming entity characteristics. Examples of attributes described as s-expressions include:

```
(file "/tmp/foo.bar" ("read" "write"))
(port (1024 1025) ("connect" "listen"))
(app "SSH") (app "Kerberos") (app "POP") (proto "IPv6")
(diskquota "diskA" "100M")
(cpuquota "slice1" "10-MFLOPS") (cpuquota "slice2" "15-MFLOPS")
```

At its most basic level of interpretation, an attribute is not unlike a capability or permission: holding the permission permits a service to access the object in the prescribed way. This interpretation is extended to include anything that a node might make available to services that it hosts, e.g. a disk quota of 100MBytes, or a real-time system that offers a service a 10MFLOPS time-slice.

We assume that the set of attributes forms a partial order under $\leq$. Given $a, b : Attributes$, then $a \leq b$ is interpreted to mean than the attribute b implies a, that is, if a node offers b, then it implicitly offers a, and similarly, if a service holds attribute b then it also holds attribute a. For example, we would expect the following to hold:

`(file"/tmp/foo.bar"("read"))` $\leq$ `(file"/tmp/foo.bar"("read""write"))`

We further assume that this partial order defines a lattice $(Attributes, \leq, \sqcup, \sqcap, \bot, \top)$, with lowest upper bound operator $\sqcup$, greatest lower bound operator $\sqcap$ and unique lowest and unique highest attributes $\bot$ and $\top$, respectively. In principle, this is not an unreasonable assumption, as in the worst-case, a powerset lattice (with ordering defined by subset and lowest upper bound defined by union) of s-expressions can be used to represent the set of attributes. In this way, the lowest upper bound provides a meaning for the combinations of attributes. Figure 2 gives an example of a powerset lattice of attributes.

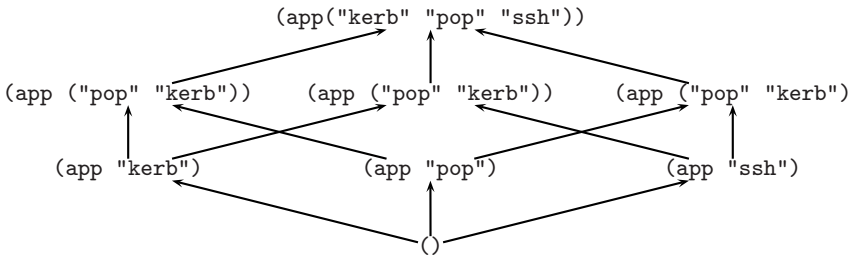

Fig. 2. Sample Powerset Lattice of Attributes

Services. Let *Service* represent the set of all possible services that are to be hosted on network nodes. A dependency relation is defined for services

$$depends : Service \leftrightarrow Service$$

whereby $depends(s_1, s_2)$ means that service s_1 uses service s_2.

In order to operate properly, a service requires a minimum collection of attributes to be available on its hosting node. We define

$$minSvcAttrib : Service \rightarrow Attribute$$

whereby, given service s, then $minSvcAttrib(s)$ defines the minimum attribute required by s. For the sake of simple exposition, we assume that if a service requires attributes $a, b, c, \ldots$, then its minimum requirement is defined by the join of these attributes, that is, $(a \sqcup b \sqcup c \sqcup \ldots)$. For example, based on the attribute lattice in Figure 2, a service s that provides ssh'ified email has requirements $minSvcAttrib(s) =$ `(app("pop" "ssh"))`.

Nodes. Nodes are configured to host services. This is specified as:

$$hosts : Node \rightarrow (Service \rightarrow Attribute)$$

whereby, the services that a node n hosts is specified by $hosts(n)$ and this defines a function that maps these services to their assigned attributes. For example, $hosts(n)(s)$ defines the attribute that is provided by the node n to the service s that it hosts. We require that the attributes offered by a node are sufficient for the service, that is, for all nodes n and services $s \in \mathrm{dom}(hosts(n))$,

$$minSvcAttrib(s) \leq hosts(n)(s)$$

The function from *Service* to *Attribute* is a partial injection: the domain of $hosts(n)$ defines the services hosted and no two hosted services share the same attribute. If two different services require similar node attributes, then we assume that the corresponding attribute s-expressions contain sufficient information to distinguish the attribute as offered to different services. For example, node `n` allows service `s` to listen on port 1024 over IPv6 using a 10/100 Ethernet card.

```
hosts(n)(s)=(svc s (port 1024 "listen")
            (proto "IPv6") (hw "eth 10/100"))
```

In this way, these attributes may be thought of as node *contracts* that have been assigned by the node to the services that it hosts.

A node is limited by the resources, system-level applications, and so forth, that it can potentially offer to any services that it hosts. We define

$$maxAttrib : Node \rightarrow Attribute$$

whereby, $maxAttrib(n)$ is the maximum attribute that node n can offer. This sets an upper bound on the contract/attributes defined by *hosts*. For any node n we require that the combined contracts assigned as $hosts(n)$ do not exceed $maxAttrib(n)$, that is,

$$\sqcup\{s : Service | s \in \mathrm{dom}(hosts(n)) \bullet hosts(n)(s)\} \leq maxAttrib(n)$$

Note that we use a Z-like notation (such as the set specification in comprehension above) to specify requirements. Recall that the function defined by $hosts(n)$ is an injection and, therefore, the join ($\sqcup$) of the set of attributes assigned, can be used to effectively represent a summation of the contracts issued.

For symmetry, we also define the minimum attributes hosted by a node.

$$minAttrib : Node \rightarrow Attribute$$

whereby, $minAttrib(n)$ is the minimum attribute that node n is willing to offer. For example, a multiprocessor cluster might only offer contacts to services with high computational or resource requirements; a node with an SSL hardware accelerator may only be willing to host services that need to use SSL. Thus, we require the following to hold.

$$minAttrib(n) \leq \sqcap\{s : Service | s \in \mathrm{dom}(hosts(n)) \bullet hosts(n)(s)\}$$

If a node is not concerned with enforcing a minimum contract then we define $minAttrib(n) = \bot$.

Example 1 Table 1 provides sample attribute bindings for nodes based on the attribute lattice defined in Figure 2. Node `Alice` runs sendmail and is configured

n :	`Alice`	`Bob`	`Clare`
$minAttrib(n)$	`()`	`(app "pop")`	`(app "kerb")`
$maxAttrib(n)$	`(app "pop")`	`(app ("pop" "kerb"))`	`(app ("kerb" "ssh"))`

Table 1. Sample Node attribute bindings

to offer, at most, only POP to its hosted services. The minimum attribute offered by `Alice` is `()` ($\bot$) and, therefore, she is also willing to host services that do not need access to any of these resources. Node `Bob` is intended as a dedicated mail node and, therefore, is only willing to host services that intend to use POP (with or without the use of Kerberos). Finally, `Clare` is a host that supports kerberized services that may also require strong authentication services (SSH) for incoming requests that are non-kerberized. $\triangle$

A network configuration is a collection of interoperating nodes that host various services. For the purposes of this paper we consider interoperation simply in terms of links between nodes. We define

$$link : (Node \times Attribute) \leftrightarrow (Node \times Attribute)$$

whereby, $link((n1, a1), (n2, a2))$ means that there is a bi-directional link connecting node $n1$ and node $n2$, that allows a service (on $n1$) assigned attribute $a1$ to interoperate with a service (on $n2$) that is assigned attribute $a2$. We assume that $minAttrib(n1) \leq a1 \leq maxAttrib(n1)$ and $minAttrib(n2) \leq a2 \leq maxAttrib(n2)$. For example, we could have a link from (`Clare, (app kerb)`) to (`Bob, (app (pop kerb))`), reflecting support for a Kerberos-based single sign-on for services between `Clare` and `Bob`.

We define, $link^+$ as the transitive closure of $link$ and it represents all the (assumed traversable) possible paths in the network configuration. Whatever the network configuration may be, we require that services that depend on each other are accessible to each other, that is, for all nodes $n1$ and $n2$, and services $s1 \in \mathrm{dom}(hosts(n1))$ the and $s2 \in \mathrm{dom}(hosts(n2))$ then the following holds.

$$depends(s1, s2) \Rightarrow link^+([n1, hosts(n1)(s1)], [n2, hosts(n2)(s2)])$$

This is a simple interpretation of interoperation that assumes that the existence of a path between nodes implies reachability; it is adequate for our current purposes. Future research will extend the model to consider how the attributes available to hosts may influence reachability along these paths.

4 Quality of Configuration

Different systems can achieve varying degrees of 'quality' in their configurations. For example, a Security Enhanced (SE) Linux system that is configured to support different service proxies in separate protection domains could be considered to be a 'better quality' configuration than a standard Windows operating system-based workstation hosting similar services.

Quality. Let *Quality* define a set of quality measures that are partially ordered according to $\leq$. For example, security evaluation criteria use *assurance levels* to provide comparisons of 'quality' between different systems; the Common Criteria [14] uses evaluation levels $E1 < E2 < \cdots < E6$ to compare different levels of assurance. In this case, the intention is that one can have more confidence in a system with a higher assurance level to enforce its security requirement.

We argue that the *Quality* measure can be used to represent other measures of interest. For example, $lo < med < hi$ might represent levels of quality of service that are achievable by a node. Quality can be regarded as addressing the risk of failure: a high risk of failure requires a high quality configuration, while a low quality configuration is sufficient if there is low risk of failure. We define,

$$rating : Node \rightarrow Quality$$

where, $rating(n)$ gives the quality rating of a node n. In this paper we do not prescribe *how* a quality evaluation of a node might be done, rather, once a measure is available then we are interested in ensuring that its use is consistent and traceable within a configuration.

Quality of Configuration Policy. A quality of configuration policy is defined in terms of the attributes that nodes offer. A firewall node that provides protection domains for mail and news proxies should have high quality: we want to be sure that a failure of one proxy cannot interfere with the other on the system. A switch that routes ADSL and/or Gigabit network traffic should have high quality in its design and configuration: we want to be sure that it will not flood the ADSL side, while ensuring proper speeds for Gigabit-only traffic. In these cases, quality depends on the attributes that the node can offer to the services that it hosts. We define the quality of configuration policy in terms of

$$minQual : Attribute \times Attribute \rightarrow Quality$$

where, $minQual(a, b)$ is the minimum acceptable quality of configuration for a node that offers contracts that range from minimum a to maximum b.

Example 2 Consider the attributes defined by the lattice in Figure 2, and security *Quality* ordering $lo < med < hi$. Suppose that a node is to host services that require attributes ranging from a minimum of $\perp$ to a maximum of `(app "pop" "kerb" "ssh")`. POP services can be relatively easily compromised via

password sniffing. Therefore, we need confidence that if the POP service is compromised then it will be difficult for the attacker to, in turn, compromise the system and obtain, for example, copies of the locally stored Kerberos session keys. In practice this could be achieved using a system that provides protection domains that can sandbox/constrain a service to just the attributes that it requires. In this case, we define

$$minQual(\texttt{(),(app ("pop" "kerb" "ssh"))}) = hi$$

This means that a system that is configured to offer this range of attributes must have a high quality configuration.

We conjecture that nodes providing high quality configuration will be expensive, and therefore, if possible, it is preferable to configure networks from multiple cheaper and lower quality components. One TCB-subset-style strategy is to use nodes that offer only limited capability and may, therefore, be of lower quality. For example, there is lower risk associated with a node thats offers only a POP capability; similarly, there is lower risk associated with a node that offers only SSH. In general, there is a lower risk when nodes/systems are deployed to offer a smaller number of service attributes than when large combinations of attributes are possible. Therefore, we define, $minQual(\texttt{(),(app "kerb")}) = lo$, and so forth.

A node that hosts services requiring only pairs of these attributes is defined to require medium quality configuration; for example,

$$minQual(\texttt{(app "pop"),(app ("pop" "kerb"))}) = med$$
$$minQual(\texttt{(app "kerb"),(app ("kerb" "ssh"))}) = med$$

It follows, that if $rating(\texttt{Alice}) = low$ and $rating(\texttt{Bob}) = rating(\texttt{Clare}) = med$, then the nodes meet the quality of configuration requirement. △

Example 3 Network nodes are configured with support for ADSL, 10/100 Ethernet and 100/1000 Gigabit Ethernet. We define a simple ordering over these attributes as follows:

```
(hw "ADSL") < (hw "eth 10/100") < (hw "eth 100/1000)
```

The node/switch `Eric` is configured to route ADSL and 10/100 traffic and, therefore, the attributes that it offers to a service are constrained as follows.

$$minAttrib(\texttt{Eric}) = \texttt{(hw "ADSL")}$$
$$maxAttrib(\texttt{Eric}) = \texttt{(hw ("ADSL" "eth 10/100"))}$$

The node `George` is configured to route only 10/100 and 100/1000 traffic:

$$minAttrib(\texttt{George}) = \texttt{(hw "eth 10/100")}$$
$$maxAttrib(\texttt{George}) = \texttt{(hw ("eth 10/100" "eth 100/1000"))}$$

A switch configured to route both ADSL and 10/100 traffic A quality switch attempts to avoid flooding the ADSL connections and slowing the 10/100 connections, while a low-quality switch cannot ensure this. Using the same quality ordering as the previous example, we define

$$minQual(\texttt{(hw "ADSL"), (hw ("ADSL" "eth 10/100"))}) = med$$
$$minQual(\texttt{(hw "eth 10/100"), (hw ("10/100" "eth 100/1000"))}) = med$$

If a switch routes only single-speed traffic, then it need not have high quality, as flooding, etc., is not an issue, We define, $minQual(\texttt{(),(hw "eth 10/100")}) = lo$, and so forth. However, if the switch needs to route all combinations of traffic, then a higher quality is required:

$$minQual(\texttt{(), (hw ("ADSL" "eth 10/100" "eth 100/1000"))}) = hi$$

If we regard quality as reflecting the level of risk that a configuration must address, then there is a higher risk of failure when routing messages at multiple speeds versus simply routing at a single speed. △

Quality Configuration. A node n is suitably configured if

$$minQual(minAttrib(n), maxAttrib(n)) \leq rating(n)$$

The above examples rely on a simple quality measure based on a total ordering $lo < med < hi$. We generalize this to a semiring [15]. Intuitively a semiring provides an addition operation on the set of measures. Some background information is provided in Appendix A.

5 The Configuration Cascade Problem

The quality of a network configuration is based not only on the quality of the configuration of the individual nodes, but is also based on their interoperation. This is illustrated by the following example.

Example 4 Consider a network composed of the nodes `Bob` and `Clare`, from Example 1. For simplicity, we assume that `Bob` hosts a kerberized mail service that allows limited classes of non-kerberized users to login based only on weak authentication (userid and password transmitted in clear text). Node `Clare` also provides a kerberized mail service and only supports strong authentication of non-kerberized clients; such clients must use SSH. Both of these nodes are deployed with *med* quality of configuration and are therefore acceptable by the configuration policy outlined in Example 2. Assume that the mail services hosted by `Bob` and `Clare` need to interoperate. We examine the quality of this network configuration.

An attacker has the ability to compromise systems that have *med* (or lower) quality of configuration. The attacker uses *dsniff* or *Ethereal* to obtain the email

userid and password of a user of the service on `Bob`. Using a stack-smashing attack on the email service the attacker obtains copies of fresh Kerberos session keys and tickets. These are used to forge an authentic connection to the kerberized service on `Clare` and, in turn, some further vulnerability is exploited on `Clare`, allowing the attacker to access the SSH service.

The *med* quality nodes `Bob` and `Clare` individually meet the quality of configuration policy. By managing to compromise only *med* quality configurations this attacker can masquerade as a user who should be strongly authenticated (via SSH). However, the quality of the configuration policy requires that an attacker must compromise at least a *hi* quality configuration in order to misuse the attributes offered by the system:

$$minQual(\texttt{(),(app "pop" "kerb" "ssh")}) = hi$$

When kerberos services on `Bob` and `Clare` are configured to interoperate a *cascading path* results in a violation of the the quality of configuration policy. △

The traditional *cascade vulnerability problem* [17, 18] is concerned with secure interoperation of compositions of multilevel secure systems that are evaluated to different levels of assurance according to the criteria specified in [17]. The transitivity of the multilevel security policy upheld across all secure systems ensures that their multilevel composition is secure; however, interoperability and data sharing between systems may increase the risk of compromise beyond that accepted by the assurance level.

We argue that cascades occur not just in multilevel secure systems, but in any network in which configuration quality is based on the quality of the individual nodes. The previous example illustrated this for a (non multilevel) security scenario. The next example illustrates this for a quality of service scenario.

Example 5 Consider a network composed of the node/switches `Eric` and `George`, as discussed in Example 3. We have,

$$rating(\texttt{Eric}) = med;\ rating(\texttt{George}) = med$$

and the nodes individually meet the quality of configuration policy.

Suppose that `Eric` and `George` are connected via a 10/100 Ethernet link. The effect of this configuration is that we have a sub-net that routes ADSL, 10/100 and 100/1000 Ethernet traffic. However, a higher quality is required:

$$minQual(\texttt{(), (app ("ADSL" "eth 10/100" "eth 100/1000"))}) = hi$$

The failure of individual *med* quality configurations result in a risk that is supposed to be addressed by a *hi* quality configuration. △

These two examples demonstrate that configuration quality can cascade when linking nodes together in a network. In practice, there are two issues to be addressed. Firstly, *detecting* cascading quality within a network requires evaluating all possible network paths to ensure that the required minimum quality is

achieved over the attributes that are provided by all the nodes along the route. Secondly, if a cascade is discovered, then it is *eliminated* by either breaking appropriate links between nodes or replacing nodes with higher quality configurations until the cascade is eliminated.

The cascade problem has been previously studied in the context of multilevel secure systems. The model presented in this paper has a close relationship with the multilevel security model described in [19], whereby the lattice of multilevel security levels correspond to the lattice of attributes, the assurance levels correspond to the *Quality* c-semiring. Therefore, results on the multilevel cascade problem can be applied to our model. Existing research has considered schemes for detecting cascading multilevel security vulnerabilities and for eliminating them by reconfiguring system interoperation. While the detection of cascade vulnerabilities can be easily achieved [17, 18] in polynomial time, their *optimal* elimination by breaking links is NP-complete [20].

The c-semiring representation also provides a convenient way to measure aggregate quality across collections of nodes that make up a network. The minimum quality of a series of nodes along a network path is given by the combination (under the c-semiring) of the quality ratings of the individual nodes. In this case, the weighted c-semiring $\mathcal{S}_{weight} = \langle \mathcal{R}, min, +, +\infty, 0 \rangle$ provides the appropriate measure. Given a series of possible paths that facilitate the interaction of x and y attributes then the effective quality of the path is the shortest path from x to y. There is a cascade vulnerability if the value calculated for this shortest path is more than $minQual(x, y)$. Practical techniques for calculating shortest paths across weighted constraint networks are considered in [15].

In suggesting the use of the weighted c-semiring $\mathcal{S}_{weight} = \langle \mathcal{R}, min, +, +\infty, 0 \rangle$ as one example of a quality measure, we are assuming that the risk of failure of one entity is independent of the risk of failure of any other entity. This is quite a restrictive assumption; however, there are examples where this kind of measure is useful. For example, in practice, the more firewalls/subnets that have to be traversed to directly access a node, then the more 'secure' the node is considered to be. The notion of *security distance* is defined in [21] as the minimum number of servers and/or firewalls that an attacker on the Internet must compromise to obtain direct access to some protected service. We argue that security distance in this case is equivalent to using a weighted c-semiring with each system having an equal rating of '1'. This can be generalized within our model to the *weighted security distance*, whereby, a weight is associated with each server and/or firewall to indicate the amount of effort that is required to compromise that component.

An alternative measure to using the weighted c-semiring is to interpret the probabilistic c-semiring $\mathcal{S}_{prob} = \langle \{x | \in [0, 1]\}, max, \times, 0, 1 \rangle$ [15] in terms of aggregation of risk along a path, which is calculated as combination (multiplication) of probabilities. As systems fail along a cascading path, then overall, there is increasing risk to subsequent systems failing.

6 Conclusion

In this paper we describe an abstract model of configuration that can be used to guide requirements for self-configuring nodes. The model takes a metric-based approach in that it allows networks to be built from nodes of varying quality.

A consequence of this model is that connecting correctly configured nodes together may result in a mis-configured system due to a cascading effect of their interoperation. These results have implications for the design of architectures that support self-configuring/autonomic nodes. When an autonomic node is introduced to a correctly configured network, it is not sufficient for the node to be just correctly configured; how the node interoperates with its neighbours can have a cascading impact on the overall quality of configuration. Therefore, each node must maintain and share sufficient information about the possible paths within the network configuration in order to able to determine whether it is safe to establish new connections. Doing this in an *optimal* way (for example, to maximise connectivity) may not be feasible since the cascade problem is, in general, hard.

Acknowledgements

We would like to thank the anonymous referees for their useful comments and feedback on the paper. This work has received partial support from the following: Enterprise Ireland (SC/2003/007); the MIUR Italian Project PRIN (n.2005-015491); Science Foundation Ireland (04/IN3/I404C and 00/PI.1/C075).

References

1. Drogseth, D., Hultquist, S., Nudler, J.: Network Performance Management: Three key technology challenges. Special Report, www.statseeker.com (2004)
2. Magrath, S., Chiang, F., Markovits, S., Braun, R., Cuervo, F.: Autonomics in Telecommunications Service Activation. First International Workshop on Autonomic Communication for Evolvable Next Generation Networks (2005)
3. Konstantinou, A., Florissi, D., Yemini, Y.: Towards Self-Configuring Networks. DARPA Active Networks Conference and Exposition (DANCE'02) (2002)
4. Ganek, A.G., Corbi, T.A.: The dawning of the autonomic computing era. IBM SYSTEMS JOURNAL, VOL 42, NO 1 (2003)
5. Horn, P.: Autonomic Computing: IBM's Perspective on the State of Information Technology. www.research.ibm.com/autonomic/manifesto (2001)
6. Kephart, J.O., Chess, D.M.: The Vision of Autonomic Computing. IEEE Computer Society (2003)
7. Balasubramaniam S., Barrett K., Strassner J., Donnelly W., van der Meer S.: Bio-inspired Policy Based Management (bioPBM) for Autonomic Communication Systems. 7th IEEE workshop on Policies for Distributed Systems and Networks, (2006)
8. TMF: TMF 053: The NGOSS Technology Neutral Architecture, (2005)
9. IBM: Policy Management for Autonomic Computing. IBM T.J. Watson Research Centre, (2005)

10. Durham D., et al: The COPS (Common Open Policy Service) Protocol. RFC 2748, (2000)
11. Westerinen A., Strassner J.: Common Information Model (CIM) Core Model. DSP0111, version 2.4, (2000)
12. Parker J.: FCAPS, TMN, ITIL: Three Key Ingerdients to Effictive IT Management. OpenWater Solutions, (2005)
13. Rivest, R.L.: S-expressions. Technical report, Network Working Group (1997) Internet Draft: http://theory.lcs.mit.edu/ rivest/sexp.txt.
14. Common Criteria Project: Common criteria for information technology security evaluation version 2.1. Technical report, US NIST (1999)
15. Bistarelli, S.: Semirings for Soft Constraint Solving and Programming. Volume LNCS 2962. Springer (2004)
16. Bistarelli, S., Montanari, U., Rossi, F.: Semiring-based Constraint Solving and Optimization. JACM **44**(2) (1997) 201–236
17. TNI: Trusted computer system evaluation criteria: Trusted Network Interpretation. Technical report, National Computer Security Center (1987) Red Book.
18. Millen, J., Schwartz, M.: The cascading problem for interconnected networks. In: 4th Aerospace Computer Security Applications Conference, IEEE CS Press (1988)
19. Foley, S.N., Bistaelli, S., O'Sullivan, B., Herbert, J., Swart, G.: Multilevel security and the quality of protection. In: Proceedings of First Workshop on Quality of Protection, Como, Italy, Springer Advances in Information Security, vol 23, 2006
20. Horton, R., et al.: The cascade vulnerability problem. Journal of Computer Security **2**(4) (1993) 279–290
21. Swart, G., Aziz, B., Foley, S., Herbert, J.: Trading off security in a service oriented architecture. In: 19th Annual IFIP WG 11.3 Working Conference on Data and Applications Security, (2005)

A Appendix: Soft Constraints and c-semirings

A semiring is a tuple $\langle \mathcal{S}, +, \times, \mathbf{0}, \mathbf{1} \rangle$ such that: $\mathcal{S}$ is a set and $\mathbf{0}, \mathbf{1} \in \mathcal{S}$; $+$ is commutative, associative and $\mathbf{0}$ is its unit element; $\times$ is associative, distributes over $+$, $\mathbf{1}$ is its unit element and $\mathbf{0}$ is its absorbing element. A c-semiring is a semiring $\langle \mathcal{S}, +, \times, \mathbf{0}, \mathbf{1} \rangle$ such that: $+$ is idempotent, $\mathbf{1}$ is its absorbing element and $\times$ is commutative.

Let us consider the relation $\leq_S$ over $\mathcal{S}$ such that $a \leq_S b$ iff $a + b = b$. Then it is possible to prove that (see [16]): $\leq_S$ is a partial order; $+$ and $\times$ are monotone on $\leq_S$; $\mathbf{0}$ is its minimum and $\mathbf{1}$ its maximum. Informally, the relation $\leq_S$ gives us a way to compare semiring values and constraints. In fact, when we have $a \leq_S b$, we will say that b *is better than* a. In the following, when the semiring will be clear from the context, $a \leq_S b$ will be often indicated by $a \leq b$.

The classical Constraint Satisfaction Problem (CSP) is a Soft CSP (SCSP) where the chosen c-semiring is: $S_{CSP} = \langle \{false, true\}, \vee, \wedge, false, true \rangle$. Fuzzy CSPs (FCSP) can instead be modelled in the SCSP framework by choosing the c-semiring $S_{FCSP} = \langle [0, 1], max, min, 0, 1 \rangle$. Many other soft CSPs (probabilistic, weighted, ...) can be modelled by using a suitable semiring structure ($S_{prob} = \langle [0, 1], max, \times, 0, 1 \rangle$, $S_{weight} = \langle \mathcal{R}, min, +, +\infty, 0 \rangle$, ...).