Characterising ACLs by Rule Dependency: Effects on Optimisation Effectiveness

Vic Grout, Stuart Cunningham, Rich Picking, John McGinn, John Davies

Centre for Applied Internet Research (CAIR), University of Wales, NEWI, Wrexham, UK

Summary
This paper considers the effects of dependencies between rules in Access Control Lists (ACLs). Dependent rules may not be reordered in an ACL if the policies of the list are to be preserved. This is an obstacle to the optimisation of rule order intended to reduce the time taken matching packets against rules. In this paper, the concept of rule dependency is defined in relation to the problem of minimising processing latency. The concepts of dependence and possible dependence are introduced and the relationship between them considered. Two measures of dependency, the dependency index and the fragmented dependency index are defined and formulated and an upper bound for each is derived. Examples of real-world ACLs are studied and the implications for practical optimisation discussed.

Key words:
Access Control Lists (ACLs), Rule dependencies, Optimisation, Packet latency.

1. Introduction

Access Control Lists (*ACL*s) play a major rôle in the process of passing or blocking traffic through certain regions of a network. They can permit or deny traffic from or to given sources or destinations, or discriminate on the basis of content or other characteristics. In addition, their ability to *filter* network traffic makes ACLs suitable for a wider purpose; any in which there is a need to choose certain traffic, probably as data *packets*, for a given policy. *Network Address Translation* (*NAT*), *traffic shaping*, various aspects of *Internet routing*, and numerous other traffic *policies* all require packets to which the policy is to be applied to be separated from those to which it is not. ACLs may vary considerably in size, structure and purpose but it is not uncommon for each packet to be tested against several ACLs on its passage across a single internet router and many more across a complete autonomous system or domain. It is therefore useful to *optimise* ACLs for efficiency.

An ACL is an ordered list of *rules*. Each rule accepts or rejects a packet based on one or some of its characteristics - its *profile*. Typically, a packet may be considered on the basis of its source and/or destination address or traffic type, although other features, or *flags*, may be relevant

[1][2]. Fig. 1 gives an example of a typical ACL in the syntax of the Cisco *Internetwork Operating System* (*IOS*) [3]. The use of the terms *permit* and *deny* reflect the original role of ACLs in passing or blocking traffic (although their use is now considerably more widespread). Each packet to be tested against an ACL is compared with the first rule, then the second, and so on, until a rule matches its profile. The rule is then permitted or denied accordingly and no more rules are considered. There is taken to be an implicit 'deny all' rule terminating each list to deal with packets not matched by any other rule. ACL optimisation effectively means finding an ordering of its rules that minimises processing time and thus packet latency.

```
(1) access-list 173 permit icmp any any
(2) access-list 173 permit tcp any any established
(3) access-list 173 deny ip RANGE MASK any
(4) access-list 173 deny ip 10.77.23.0 0.255.255.255 any
(5) access-list 173 deny ip 172.16.2.0 0.15.255.255 any
(6) access-list 173 deny ip 192.168.1.0 0.0.255.255 any
(7) access-list 173 deny ip 169.254.1.0 0.0.255.255 any
(8) access-list 173 deny ip 192.168.2.0 0.0.0.255 any
(9) access-list 173 permit tcp any host MAILSERVER eq smtp
(10) access-list 173 permit tcp any host NAMESERVER eq domain
(11) access-list 173 permit udp any eq 53 host NAMESERVER gt 1024
(12) access-list 173 permit tcp host MANAGER host SUN eq telnet
(13) access-list 173 permit tcp host MANAGER host SERIAL0
(14) access-list 173 permit tcp host MANAGER host ETHERNET0
(15) access-list 173 permit udp host MANAGER host SERIAL0 eq snmp
(16) access-list 173 permit tcp any host FTPSERVER eq ftp
(17) access-list 173 permit tcp any eq ftp-data host FTPSERVER
(18) access-list 173 permit tcp any eq ftp-data any gt 1024
(19) access-list 173 permit tcp any host WWWSERVER eq www
(20) access-list 173 permit tcp any host SWWWSERVER eq 443
(21) access-list 173 permit udp EXT-NTPSERVER any eq 123
(22) access-list 173 permit udp any range 6970 7170 any
(23) access-list 173 deny ip any any
```

Fig. 1. A Typical ACL

However, rule *order* can be critical in an ACL. To illustrate this, consider two rules as follows: *rule 1* permits packets with characteristic *A* (source address, for example) and *rule 2* denies packets with characteristic *B* (destination address, say). A packet with a profile matching both characteristics (from *A* to *B* in this case) will match both rules. The rules are *dependent*.

Manuscript received June 5, 2007
Manuscript revised June 20, 2007

Consequently, the order of *rule 1 ... rule 2* will permit the *A* to *B* packet whereas the order *rule 2 ... rule 1* will deny it. In Fig. 1, rules 8 and 9 are dependent: an SMTP packet from the `192.168.2.0` network to the mail-server will match both. It is the intention of the policy, in its given form, that such a packet should be blocked. However, promoting rule 9 above rule 8 would (incorrectly) pass it. Not all rules will be dependent in this way but those that are must have their relative order in the list preserved if the ACL is to retain its intended purpose. Of course, this only applies for rules of opposite types. Several 'permit' rules in a contiguous block, for example, can be freely reordered among themselves. This paper considers the effect of dependent rules on the effectiveness of any optimisation (latency minimisation) process.

2. ACL Structure

Where appropriate in this paper, abbreviations are used as follows: $\exists$, 'there is' or 'there exists'; $\forall$, 'for all' or 'for every'; $\wedge$, 'and'; $\Leftrightarrow$, 'if and only if'; and $\rightarrow$, 'such that'. Using the notation in [4], define A^* to be the set of all *addresses* available within a given system, define B^* to be the set of all *protocols* recognised by the system and define $\underline{F}^* = \{0, 1\}^w$ to be the set of *w flag vectors* ($\{0, 1\}$ *w*-tuples acting on B^*) valid for the system. For completeness only, X^* represents the set of *payloads*.

A *packet*, $p_k = (Sa_k, Da_k, b_k, \underline{f}_k, X_k)$, is defined by its constituents: $Sa_k \in A^*$, the *source* address; $Da_k \in A^*$, the *destination* address; $b_k \in B^*$, the protocol; $\underline{f}_k \in \underline{F}^*$, the flags vector and $X_k \in X^*$, the payload. A *rule*, $r_i = (t_i, SA_i, DA_i, B_i, \sigma_i)$, consists of: a *type*, $t_i \in \{permit, deny\}$, $SA_i \subseteq A^*$: the source *range*, $DA_i \subseteq A^*$: the destination *range*, $B_i \subseteq B^*$: the protocol *range*, and a *flags predicate*, σ_i: $\underline{F}^* \mapsto \{true, false\}$. Only t_i is a required component in all syntaxes. If any other components are absent then $SA_i = A^*$, $DA_i = A^*$, $B_i = B^*$ or $\sigma_i \equiv true$ by default. A *policy*, $Z = [r_1, r_2, ..., r_n]$ is an (ordered) sequence of *n* rules to achieve some purpose. The last rule in any policy implicitly denies *all* traffic; that is, $t_n = deny$, $SA_n = A^*$, $DA_n = A^*$, $B_n = B^*$ and $\sigma_n \equiv true$. A packet, p_k, *matches* a rule, r_i (for which we write $p_k \nabla r_i$), if its addresses and protocols are within the range of the rule and if its flags vector satisfies the rule's flags predicate. That is,

$$p_k \nabla r_i \Leftrightarrow (Sa_k \in SA_i) \wedge (Da_k \in DA_i) \wedge (b_k \in B_i) \wedge \sigma_i(\underline{f}_k), \quad (1)$$

in which case the packet will be permitted or denied according to t_i.

A *dependency* exists between two rules, r_i and r_j, if they are of opposite type and it is possible that there exists a packet, p_k, that matches both rules ($(p_k \nabla r_i) \wedge (p_k \nabla r_j)$); that is r_i and r_j are *dependent* if

$$(t_i \neq t_j) \wedge \exists p_k \rightarrow (Sa_k \in SA_i \cap SA_j) \wedge (Da_k \in DA_i \cap DA_j) \wedge (b_k \in B_i \cap B_j) \wedge \sigma_i(\underline{f}_k) \wedge \sigma_j(\underline{f}_k). \quad (2)$$

Eliminating the packet, p_k, from this expression, allows a *{0, 1} dependency matrix*, $D = (d_{ij}: 1 \leq i,j \leq n)$, to be defined:

$$d_{ij} \Leftrightarrow (t_i \neq t_j) \wedge (SA_i \cap SA_j \neq \varnothing) \wedge (DA_i \cap DA_j \neq \varnothing) \wedge (B_i \cap B_j \neq \varnothing) \wedge (\Sigma_i \cap \Sigma_j \neq \varnothing), \quad (3)$$

where $\Sigma_i \subseteq \underline{F}^*$ is the subset of flag vectors satisfying σ_i. Two rules, r_i and r_j, are *possibly dependent* if they are of opposite type ($t_i \neq t_j$), giving a *possible dependency matrix*, $P = (p_{ij}: 1 \leq i,j \leq n)$, defined as $p_{ij} \Leftrightarrow (t_i \neq t_j)$. If $d_{ij} = 1$ then the order of rules *i* and *j* must be preserved if the behaviour of the policy is to be maintained. Detecting dependencies and anomalies, particularly in real-time on a production router is not trivial, however [5][6]. If there is any uncertainty then it may be necessary to apply the same restriction when $p_{ij} = 1$.

An *access list*, or simply *list*, *L*, *implements* a policy, $Z = [r_1, r_2, ..., r_n]$, if it is a permutation of the rules of *Z* such that the order of dependencies is preserved. Let $r_i(L)$ be the rule at position *i* in *L*. A special case of a list implementing a policy, *Z*, is the *identity list*, $I_Z = [r_1, r_2, ..., r_n]$, for which $r_i(I_Z) = r_i \ \forall\, i \ (1 \leq i \leq n)$. I_Z is usually the starting point for any ACL optimisation, particularly iterative search techniques.

The *hit-rate*, $h(r_i(L))$, of rule r_i in a list *L*, is the probability that a packet will match r_i in *L*. Hit-rates can be calculated dynamically using counters within the IOS or hardware [7][8]. The *latency*, $\lambda(r_i)$, of a rule r_i is the time taken to (independently) process r_i. This may be calculated from the length of a rule, the nature of the protocols involved or taken from stored tables. In some systems, latencies may be constant for all rules but this is not assumed in this paper. The *cumulative latency*, $\kappa(r_i(L))$, of r_i in a list *L*, is the time taken to process r_i and all rules preceding it in *L*.

$$\kappa(r_i(L)) = \sum_{\varphi=1}^{i} \lambda(r_\varphi(L)). \quad (4)$$

The *expected latency*, *E(L)*, of a list *L*, is then given by

$$E(L) = \sum_{i=1}^{n} h(r_i(L))\kappa(r_i(L)) \qquad (5)$$
$$= \sum_{i=1}^{n} h(r_i(L)) \sum_{\varphi=1}^{i} \lambda(r_i(L)) .$$

Optimising an ACL requires us to find (or approximate) the list, *L*, implementing a policy, *Z*, that minimises *E(L)*, subject to the constraints of the dependency matrices, *D* or *P*. In [4], the problem is shown to be *NP-complete* [9].

P: Permit D: Deny X – No dependencies ? – Possible dependencies

RULE & TYPE	1 P	2 P	3 D	4 D	5 D	6 D	7 D	8 D	9 P	10 P	11 P	12 P	13 P	14 P	15 P	16 P	17 P	18 P	19 P	20 P	21 P	22 P	23 P	24 D
1 P	X		?						X – No dependencies															?
2 P																								
3 D	?		X – No dependencies						? - Possible dependencies															X
4 D																								
5 D																								
6 D																								
7 D																								
8 D																								
9 P	X		? – Possible dependencies						X – No dependencies															?
10 P																								
11 P																								
12 P																								
13 P																								
14 P																								
15 P																								
16 P																								
17 P																								
18 P																								
19 P																								
20 P																								
21 P																								
22 P																								
23 P																								
24 D	?		X						? – Possible dependencies															X

Fig. 2. ACL Dependencies

3. Rule Dependencies

Define the *dependency index* (*DI*) to be the ratio of dependent rule pairs to all rule pairs. Larger numbers of rule dependencies (larger DIs) restrict ACL optimisation by making more potential rule reorderings (e.g. swaps) illegal [10]. For *n* rules, there are n^2 potential dependencies. However, dependencies are not possible between rules of the same type so, for a policy of *x* permits and *y* denys ($n = x + y$), the number of possible dependencies is $n^2 - x^2 - y^2$ with DI bounded above by $(n^2 - x^2 - y^2) / n^2$. Fig. 2 shows these relationships for the ACL in Fig. 1 with $n = 24$, $x = 17$, y = 7 and DI ≤ $(576 - 289 - 49) / 576 = 0.41$. Fig. 3 shows how the limit for DI varies with *x* (and *y*), the minimum value of *0* occurring when *x* (or *y*) = *n* and the maximum value of *0.5* when $x = y = n/2$.

DI provides a measure of the (lack of) freedom to reorder rules in the optimisation process. However, this assumes that all rule swaps (say) are considered within the optimisation algorithm ([4][11][12][13], for example). In the real-world, such an approach would be too complex to be embedded in a router's hardware or software and, typically, only adjacent swaps are considered (Grout et al., 2005). If the search algorithm prohibits swaps between non-adjacent (permit or deny) blocks then a different dependency index is required to be meaningful.

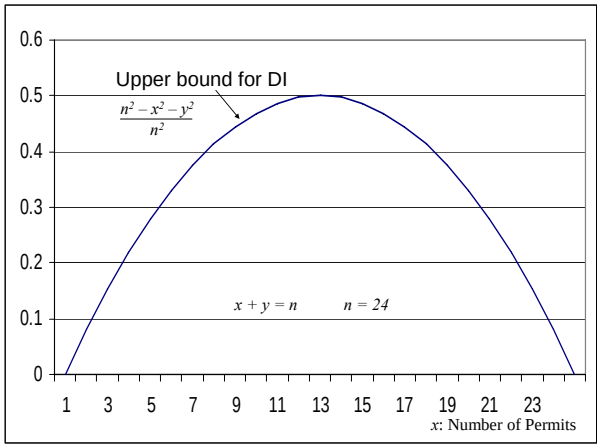

Fig. 3. A Bound for DI

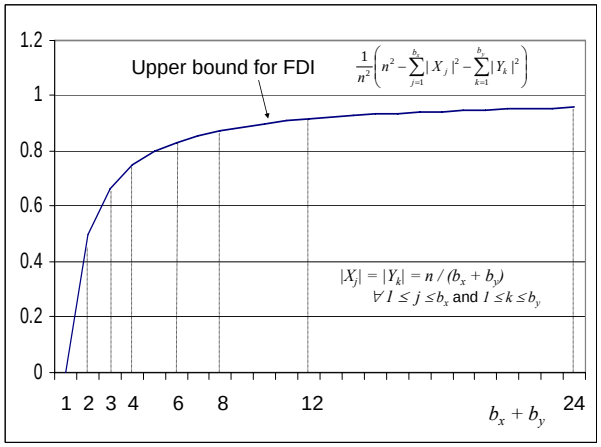

Fig. 4. Typical Bound for the Fragmented Dependency Index (FDI).

To this end, suppose an ACL, *L*, consists of b_x blocks of permits, X_j $(1 \le j \le b_x)$ and b_y blocks of denys, Y_k $(1 \le k \le b_y)$. Then

$$n = x + y = \sum_{j=1}^{b_x} |X_j| + \sum_{k=1}^{b_y} |Y_k| , \qquad (6)$$

where $|B|$ represents the number of rules in block *B*. If swaps are not permitted (or considered) between non-contiguous blocks, then the number of infeasible *or* possibly dependent pairs is increased to

$$n^2 - \sum_{j=1}^{b_x} |X_j|^2 - \sum_{k=1}^{b_y} |Y_k|^2 \qquad (7)$$

(again consider Fig. 2) and the *fragmented dependency index* (*FDI*) bounded above by

$$\frac{1}{n^2}\left(n^2 - \sum_{j=1}^{b_x} |X_j|^2 - \sum_{k=1}^{b_y} |Y_k|^2 \right). \qquad (8)$$

For the example in Figs. 1 and 2, $b_x = b_y = 2$, $|X_1| = 2$, $|Y_1| = 6$, $|X_2| = 15$ and $|Y_2| = 1$ giving FDI $\leq (576 - 4 - 36 - 225 - 1) / 576 = 0.54$. In general, FDI is minimised when $b_x = 1$ and $b_y = 0$ ($|X_1| = n$) or $b_x = 0$ and $b_y = 1$ ($|Y_1| = n$) and maximised by alternating single permits and denys ($b_x = b_y = n/2$, $|X_j| = |Y_k| = 1 \ \forall\, 1 \leq j,k \leq b_x,b_y$) giving a bound of $(n^2 - n) / n^2$, which tends to *1* as n increases – the worst case. Fig. 4 illustrates the general bound for equally sized permit/deny blocks, $|X_j| = |Y_k| = n / (b_x + b_y) \ \forall 1 \leq j \leq b_x \wedge 1 \leq k \leq b_y$.

4. Discussion

This section uses the DI and FDI of real-world ACLs to discuss the suitability of simple optimisation techniques. In [10] the following three-part heuristic, called *δ-OPT*, is proposed for simple, embedded minimisation of expected latency:

```
Step 1: Initialisation (following manual
                                ACL configuration)
   for i := 1 to n do
      h_i := 1            \ hit rates equal at start

Step 2: Promotion (on a packet matching rule i)
   h(r_i) := 2h(r_i)      \ increase matched hit-rate
   if (d_{i-1 i}=0) and (h(r_i)λ(r_{i-1})
         > h(r_{i-1})λ(r_i)) then       \ (or p_{i-1 i} = 0)
      Swap(r_{i-1}, r_i)      \ promote if E(L) reduced

Step 3: Reduction (periodically
                                to prevent overflow)
   for i := 1 to n do
      h(r_i) := h(r_i) / max_j h(r_j)
```

Derivation and details are to be found in the original paper. There is some processing cost associated with implementing this algorithm. However, depending upon the nature of the traffic and dependency indices of the rules, this simple optimisation can be shown to be worthwhile (i.e. to reduce overall expected latency) for ACLs above a certain length (number of rules, n). Table 1 summarises these results as the minimum number of rules for the saving in ACL latency to outweigh the latency from the algorithm. S is the *stability* of the traffic flow, essentially a probability that a given packet is similar to the previous one in that it matches the same rule in the ACL, L. *δ-OPT* performs better for more stable traffic. However, only for values of DI approaching *1* is optimisation worthless.

Table 1: Minimum value of n for *δ-OPT* to reduce $E(L)$

	DI =	0.00	0.25	0.50	0.75	1.00
S	0.00	*19*	*21*	*23*	*33*	∞
	0.25	*16*	*19*	*21*	*29*	∞
	0.50	*13*	*15*	*19*	*26*	∞
	0.75	*9*	*10*	*13*	*21*	∞
	1.00	*8*	*9*	*12*	*17*	∞

As an example, on the basis of these results and the calculations from Section 3, *δ-OPT* can be seen to have a positive benefit for the ACL in Fig. 1 for all traffic flows, S. (DI = *0.41* and FDI = *0.54*, $n = 24$ and, from Table 1, taking an index of *0.5*, optimisation will be worthwhile for ACLs larger than *23* rules, even for the worst case, $S = 0$.) This analysis is now applied to a number of real-world ACLs. Table 2 summarises the characteristics of several ACLs taken from a variety of production applications. (No attempt has been made to remove redundancies/inconsistencies, etc. from these ACLs: they are taken directly from source.) ACLs B, C and D are taken from college/university LANs, F, G and H from company networks and A and E from SOHO environments connecting to the Internet via an ISP. ACLs I, J and K are derived from templates for various standard security configurations. In each case, the upper bound is calculated for the two dependency indices. These values are plotted in Fig. 5 for comparison.

Table 2: Permit/deny block structure for various real-world ACLs with corresponding dependency indices

ACL	n	x	y	b_x	b_y	X_j	Y_k	*DI **	*FDI**
A	*16*	*10*	*6*	*2*	*3*	*6, 4*	*2, 3, 1*	*0.47*	*0.74*
B	*53*	*20*	*33*	*4*	*4*	*10, 7, 2, 1*	*14, 12, 5, 2*	*0.47*	*0.81*
C	*55*	*10*	*45*	*2*	*3*	*5, 5*	*27, 17, 1*	*0.30*	*0.65*
D	*144*	*27*	*117*	*6*	*7*	*4, 7, 6, 6, 3, 1*	*18, 32, 12, 6, 25, 21, 3*	*0.30*	*0.87*
E	*19*	*7*	*12*	*1*	*2*	*7*	*6, 6*	*0.47*	*0.66*
F	*93*	*22*	*71*	*3*	*4*	*13, 8, 1*	*41, 17, 12, 1*	*0.36*	*0.73*
G	*111*	*29*	*82*	*1*	*2*	*29,*	*80, 2*	*0.39*	*0.41*
H	*62*	*4*	*58*	*2*	*3*	*2, 2*	*22, 32, 4*	*0.12*	*0.60*
I	*172*	*54*	*118*	*2*	*3*	*31, 23*	*77,40,1*	*0.43*	*0.70*
J	*68*	*19*	*49*	*4*	*5*	*1, 1,*	*16, 8,*	*0.40*	*0.83*

						15, 2	*12, 10, 3*		
K	*63*	*22*	*41*	*2*	*3*	*18, 14*	*18, 13, 10*	*0.45*	*0.76*

* upper bound

On the basis of the derived dependency indices in Table 2, and the limits given in Table 1, Table 3 summarises the effectiveness of the *δ-OPT* heuristic for each of the ACLs, A, B, …, K. In each case, and separately for each of DI and FDI, the algorithm is marked as worthwhile or otherwise depending on whether its cost in terms of implementation is exceeded by the gain in expected latency.

Table 3 suggests that, at least for the ACLs tested, the choice of DI or FDI bound for assessing the viability of the *δ-OPT* algorithm for different lists may not be as important as might be thought. Only in *3* of the *55* ACL/traffic combinations does it affect the effectiveness of the algorithm. Whether or not this is true generally does not affect this paper's outcomes. The point is that these bounds can be used in this manner to assess algorithmic performance.

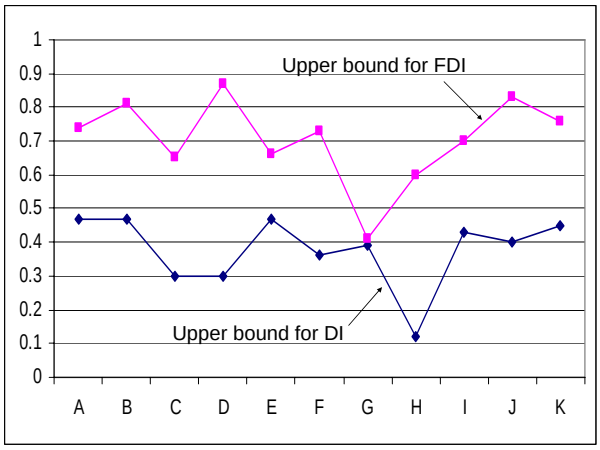

Fig. 5. Comparing Bounds for DI and FDI for real-world ACLs.

Table 3. Effectiveness of *δ-OPT* for real-world ACLs

ACL	S =	0.00	0.25	0.50	0.75	1.00
A	DI* / FDI*	*0 / 0*	*0 / 0*	*0 / 0*	*0 / 0*	*0 / 0*
B		*1 / 0*	*1 / 0*	*1 / 1*	*1 / 1*	*1 / 1*
C		*1 / 1*	*1 / 1*	*1 / 1*	*1 / 1*	*1 / 1*
D		*1 / 1*	*1 / 1*	*1 / 1*	*1 / 1*	*1 / 1*
E		*1 / 0*	*1 / 1*	*1 / 1*	*1 / 1*	*1 / 1*
F		*0 / 0*	*0 / 0*	*0 / 0*	*0 / 0*	*0 / 0*
G		*1 / 1*	*1 / 1*	*1 / 1*	*1 / 1*	*1 / 1*
H		*1 / 1*	*1 / 1*	*1 / 1*	*1 / 1*	*1 / 1*
I		*1 / 1*	*1 / 1*	*1 / 1*	*1 / 1*	*1 / 1*
J		*1 / 0*	*1 / 1*	*1 / 1*	*1 / 1*	*1 / 1*
K		*1 / 1*	*1 / 1*	*1 / 1*	*1 / 1*	*1 / 1*

* *1*: worthwhile *0*: not worthwhile

5. Conclusions

We deal initially with the limitations of this work. Firstly, no attempt has been made to tighten the (upper) bounds on DI and FDI. It is unlikely to be possible to achieve this formally and to compare rules in a pairwise manner is far from trivial individually and is extremely complex for an entire ACL [5][6]. An empirical study of the relationship between actual DI and FDI values and their theoretical bounds for real-world ACLs is beyond a paper of this length but is left open as an avenue for future research. Secondly, comparative results are only given for the *δ-OPT* heuristic. This is partly because this is the only ACL optimisation process sufficiently efficient to be embedded in router hardware [10] and partly because only for *δ-OPT* are the limit values in Table 1 available. However, extending the analysis to other forms of optimisation ([4][7][8][11][12][14], for example), whilst not providing *efficient* solutions, may serve to aid the analysis of the relationship between DI and FDI and their bounds and their different behaviour for ACLs with varying (e.g. block) structures. Thirdly, while the significance of different traffic characteristics is recognised (by the stability factor, *S*), this cannot be pursued to the fullest extent here.

There are a number of satisfactory outcomes, however. Firstly, the matching of packets and rules and the optimisation of rule order within ACLs is formalised to enable the relationship between ACL structure and rule dependency to be analysed. The optimisation objectives of minimising expected latency are hindered by excessive dependence between rules and may render certain ACLs, or types of ACLs inappropriate for optimisation. This can be measured, in principle, by the DI and FDI dependency indices and, in practice, approximated by their bounds. A simple formula is given for each bound that can be calculated easily for any ACL. A number of tests on real world ACLs then demonstrate how these bounds, in conjunction with empirical testing and simulation [10], show how ACLs may be classified conveniently as appropriate or inappropriate for optimisation.

References

[1] Cisco, 2000. *Access Control Lists*, Cisco Systems, USA, (http://www.cisco.com/univercd/cc/td/doc/product/software/ios113ed/113ed_cr/secur_c/scprt3/scacls.htm).

[2] Sedayao, J., 2001. *Cisco IOS Access Lists*, O'Reilly, USA.

[3] JANet, 2005. *JANET-CERT Example Router Configuration*, (http://www.ja.net/CERT/JANET-CERT/prevention/template.html).

[4] Grout, V. and McGinn, J., 2005. Optimisation of Policy-Based Routing Using Access Control Lists, *IFIP/IEEE Symposium on Integrated Network Management*, Nice, France, 16th-19th May 2005 (full version available at http://www.newi.ac.uk/groutv/papers/acls.pdf).

[5] Hari, B., Suri, S. and Parulkar, G., 2000. Detecting and Resolving Packet Filter Conflicts, *Proceedings of the 19th Joint Conference of the IEEE Computer and Communications Societies (INFOCOM00)*, pp1203-1212.

[6] Al-Shaer, E. and Hamed, H., 2004. Modeling and Management of Firewall Policies, *IEEE Transactions on Network and Service Management*, Vol. 1-1, April 2004.

[7] Cisco, 2002. *ACL Optimizer and Hits Optimizer*, Cisco Systems, USA,(www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/cw2000/fam_prod/acl_mgr/aclm_1_x/1_5/u_guide/ac1js.pdf).

[8] Cisco, 2003 *ACL Manager*, Cisco Systems, USA, (http://www.cisco.com/en/US/partner/products/sw/cscowork/ps402/products_user_guide_book09186a00801f42b9.html).

[9] Garey, M.R. and Johnson, D.S., 1979. *Computers and Intractability: A guide to the theory of NP-completeness*, W.H. Freeman, New York.

[10] Grout, V., Davies, J. and McGinn, J., 2006. An Argument for Simple Embedded ACL Optimisation, *Computer Communications* Vol. 30, No. 2, January 2007, pp280-277.

[11] Bukhatwa, F. and Patel, A., 2003. Effects of Ordered Access Lists in Firewalls, *Proceedings of IADIS WWW/Internet International Conference (W3I 2003)*, Algarve, Portugal, 5th-8th November 2003, pp257-264.

[12] Bukhatwa, F., 2004. High Cost Elimination Method for Best Class Permutation in Access Lists, *Proceedings of IADIS WWW/Internet International Conference (W3I 2004)*, Madrid, Spain, 6th-9th October 2004, pp287-294.

[13] Grout, V., McGinn, J. and Davies, J., 2005. Reducing Processing Latency in Network Packet Filters, *Proceedings of the 5th International Network Conference* (*INC 2005*), Samos, Greece, pp. 3-10.

[14] Cisco, 2004. *Turbo Access Control Lists*, Cisco Systems, USA,(http://www.cisco.com/univercd/cc/td/doc/product/software/ios120/120newft/120limit/120s/120s6/turboacl.htm).

Vic Grout was awarded the BSc (Hons) degree in Mathematics and Computing from the University of Exeter (UK) in 1984 and the PhD degree in Communication Engineering from Plymouth Polytechnic (UK) in 1988.

He has worked in senior positions in both academia and industry for twenty years and has published and presented over 120 research papers. He is currently a Reader in Computer Science at the University of Wales NEWI, Wrexham in the UK, where he leads the Centre for Applied Internet Research (CAIR). His research interests and those of his research students span several areas of computational mathematics, particularly the application of heuristic principles to large-scale problems in network design and management.

Dr. Grout is a Chartered Engineer, Chartered Scientist, Chartered Mathematician and Chartered IT Professional, a Member of the ACM, IEEE Computer and Communications Societies, a senior member of the IEEE and a Fellow of the British Computer Society (BCS) and Institution of Engineering and Technology (IET). He chairs the biennial international conference series on Internet Technologies and Applications (ITA 05 and ITA 07).

Stuart Cunningham was awarded the BSc degree in Computer Networks in 2001, and in 2003 was awarded the MSc Multimedia Communications degree with Distinction, both from the University of Paisley (UK).

Stuart is a Member of the British Computer Society (BCS), the Institute of Engineering & Technology (IET) and a member of the MPEG Music Notation Standards working group.

Stuart's research interests are in the areas of computer networks, audio compression, and computer music. He is a current PhD student at the University of Wales, studying under the supervision of Dr. Vic Grout.

Rich Picking has a BSc(Hons) degree in Computing and Operational Research from Leeds Polytechnic (UK, 1986), an MSc in Control Engineering and Information Technology (University of Sheffield, UK, 1987) and a PhD in Interactive Multimedia Interface Design from Loughborough University (UK) in 1996. Dr. Picking has

authored and edited numerous papers, proceedings and guides in various fields of Internet Technologies and Applications. He currently a Principal Lecturer in Computing in the School of Computing and Communications Technology within the Faculty of Business, Science and Technology at the University of Wales NEWI, Wrexham in the UK.

John McGinn was awarded the BSc(Hons) degree in Multimedia Computing by the University of Wales in 2000 and is currently working towards the PhD degree as a Research Fellow in the Centre for Applied Internet Research (CAIR) at the University of Wales, NEWI (UK).

John's research interests include network protocols and standards and distributed collaboration and visualisation. He has published and presented a number of technical papers on topics from information visualisation to ACL optimisation. He is a member of the British Computer Society (BCS) and the Institution of Engineering and Technology (IET).

John Davies has a BSc(Hons) in Control Engineering from the University of Salford, UK (1973). He has worked for British Nuclear Fuels, Sension, the University of London Computer Centre and Daresbury Laboratories (UK) as a Project Manager, Chief Engineer, Senior Lecturer and Higher Scientific Officer respectively. He is currently a Senior Lecturer in Computing at the University of Wales, NEWI (UK) completing a PhD in network traffic prediction.

John has research interests in various aspects of network measurement, simulation and management and has published a number of technical papers on network routing, traffic congestion and optimisation. He is a member of the Institution of Engineering and Technology (IET)