1-1-2007

An argument for simple embedded ACL optimisation

Vic Grout
Glyndwr University, v.grout@glyndwr.ac.uk

John N. Davies
Glyndwr University, j.n.davies@glyndwr.ac.uk

John McGinn
Glyndwr University, j.mcginn@glyndwr.ac.uk

An argument for simple embedded ACL optimisation

Abstract
The difficulty of efficiently reordering the rules in an Access Control List is considered and the essential optimisation problem formulated. The complexity of exact and sophisticated heuristics is noted along with their unsuitability for real time implementation embedded in the hardware of the network device. A simple alternative is proposed, in which a very limited rule reordering is considered following the processing of each packet. Simulation results are given from a range of traffic types. The method is shown to achieve savings that make its use worthwhile for lists longer than a given number of rules. This number is dependent on traffic characteristics but generally around 25 for typical network conditions.

Keywords
Access Control Lists, ACLs, Packet latency, Optimisation

Disciplines
Computer Engineering

Comments
Original publication is available at www.sciencedirect.com

Computer Communications, Vol. 30, No. 2, January 2007, pp280-287

AN ARGUMENT FOR SIMPLE EMBEDDED ACL OPTIMISATION

Vic Grout[1], **John Davies** and **John McGinn**
Centre for Applied Internet Research (CAIR)
University of Wales, NEWI, Wrexham, LL11 2AW, UK

Abstract

The difficulty of efficiently reordering the rules in an Access Control List is considered and the essential optimisation problem formulated. The complexity of exact and sophisticated heuristics is noted along with their unsuitability for real time implementation embedded in the hardware of the network device. A simple alternative is proposed, in which a very limited rule reordering is considered following the processing of each packet. Simulation results are given from a range of traffic types. The method is shown to achieve savings that make its use worthwhile for lists longer than a given number of rules. This number is dependent on traffic characteristics but generally around 25 for typical network conditions.

Keywords *Access Control Lists, ACLs, Packet latency, Optimisation*

1. Introduction

Access Control Lists (*ACL*s) are ubiquitous in internetworking. As the name suggests, they play a major role in the process of passing or blocking traffic through sensitive 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. As an extension to these simple security aspects of ACLs, certain traffic may be chosen for tunnelling in a *Virtual Private Network* (*VPN*) for example.

However, this ability to filter network traffic makes ACLs suitable for a wider purpose. They may be applied in any situation in which there is a need to *choose* certain data packets for a given traffic *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 but it is not uncommon for a single packet to be tested against several ACLs on its passage across a single internet router and many more across a complete domain.

This paper considers the delay this process adds to the packet's progress. It discusses optimisation of ACL structure to make the process more efficient and thus reduce the

[1] Corresponding author - Tel. +44 (0)1978 293203 Fax. +44 (0)1978 293168

delay. Optimisation is often a complex process, however. There is no value in an optimisation regime that takes longer to implement than the latency it saves, and this has often blocked attempts to implement ACL optimisation in real time and embedded within the interface (say) hardware of the router in all but the largest models. However, this paper demonstrates that a simple optimisation algorithm *can* be applied in this environment with reductions in packet latency that exceed the time taken for it to run. The essence of the technique is to consider a very limited reordering of the list of rules following the processing of every packet.

2. The Problem

An ACL is an ordered list of *rules*. Each rule accepts or rejects a packet based on one or some of its characteristic(s) - its *profile*. Typically, a packet may be considered on the basis of its source, destination or traffic type, although other features may be relevant [1]. Figure 1 gives an example of a typical ACL in the syntax of the Cisco *Internetwork Operating System* (*IOS*) [2]. The use of the terms *permit* and *deny* reflect the original role of ACLs in passing or blocking traffic.

```
access-list 100 permit icmp any any
access-list 100 permit tcp any any established
access-list 100 deny ip MYIPRANGE1 MYIPREVMASK1 any
access-list 100 deny ip 10.0.0.0 0.255.255.255 any
access-list 100 deny ip 172.16.0.0 0.15.255.255 any
access-list 100 deny ip 192.168.0.0 0.0.255.255 any
access-list 100 deny ip 169.254.0.0 0.0.255.255 any
access-list 100 deny ip 192.0.2.0 0.0.0.255 any
access-list 100 permit tcp any host MAILSERVER eq smtp
access-list 100 permit tcp any host NAMESERVER eq domain
access-list 100 permit udp any host NAMESERVER eq domain
access-list 100 permit udp any eq 53 host NAMESERVER gt 1024
access-list 100 permit tcp host MANAGER host SUN eq telnet
access-list 100 permit tcp host MANAGER host SERIAL0 eq telnet
access-list 100 permit tcp host MANAGER host ETHERNET0 eq telnet
access-list 100 permit udp host MANAGER host SERIAL0 eq snmp
access-list 100 permit tcp any host FTPSERVER eq ftp
access-list 100 permit tcp any eq ftp-data host FTPSERVER
access-list 100 permit tcp any eq ftp-data any gt 1024
access-list 100 permit tcp any host WWWSERVER eq www
access-list 100 permit tcp any host SWWWSERVER eq 443
access-list 100 permit udp EXT-NTPSERVER any eq 123
access-list 100 permit udp any range 6970 7170 any
access-list 100 deny ip any any
```

Figure 1. An Example of an Access Control List (ACL).

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 usually an implicit ‘deny all’ rule

terminating each list to deal with packets not matched by any other rule. A precise treatment of rule and packet formats and profiles is given in [3]. This level of analysis is not required here except in its final formulation of the problem. However it is necessary to note that rule order is critical in an ACL …

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. Consequently, the order of *rule 1 … rule 2* will permit the packet whereas the order *rule 2 … rule 1* will deny 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.

Some rules will take longer to process than others and some are more likely to match packets than others. The difference in processing time comes from the level or extent to which a rule has to examine a packet, and the likelihood of a match, its *hit-rate*, will vary with changing traffic flows. For any given ACL, there may be a better version, with rules in a different order, which performs the same task more efficiently – always remembering that any reordering must preserve the order of dependent rules.

For straightforwardness in what follows, we denote the rule at *position i* in an ACL simply as *rule i*. Then for a given rule i, in an ACL A, define its *latency*, l_i, to be the time taken to match it against a single packet and its hit-rate, h_i, to be the probability that the next packet will match the rule, at its present location in A (and not any rule $j < i$ that precedes it). Then the *cumulative latency* of rule i, λ_i, is given by

$$\lambda_i = \sum_{j=1}^{i} l_i \ , \tag{1}$$

the sum of the latencies of all rules up to and including i. The *expected latency* of A, E_A, the average time taken to process the list, is then

$$E_A = \sum_{i=1}^{n} h_i \lambda_i \ , \tag{2}$$

where there are n rules in A. Define the *dependency matrix*, $D = (d_{ij})$ as $d_{ij} = 1$ if rules i and j are dependent in A and $d_{ij} = 0$ otherwise. Starting from an initial (probably administratively defined) ACL, A_0, the optimisation problem then is then to find the list, A^*, with minimum expected latency, obeying the dependency constraints, that is,

$$E_{A^*} = \min_A E_A \tag{3}$$

subject to the constraint that, for any rules i and j with $d_{ij} = 1$ in A_0, their relative *order* (but not necessarily exact *position*) must be preserved in A^*.

We consider changing traffic profiles as the paper progresses. However, even in its static form, the problem is complex – *NP-complete* in fact [3]. No exact solutions are to be found in reasonable time on any platform.

The efficiency of ACL structure is first considered theoretically in [4] and [5]. Cisco [6] provide the first real attempt at optimisation. Their "Hits Optimizer" records which rules match packets in real time on the router, then their "ACL Optimizer" works offline to reorder the rules in line with dependency constraints. Apart from the obvious limitation of working offline, this system does not discriminate between rules of different latencies. Bukhatwa and Patel [7] demonstrate the savings available from ACL optimisation but ignore both rule dependencies and differing rule latencies. An improved approach [8] gives a simplified method for reordering rules based on latency but still ignores dependencies. Both methods are implemented offline. Cisco [9] introduce "Turbo Access Lists" with rules searched as look-up tables but only on high-end routers and specialist firewalls.

Al Shaer and Hamed [10] give an improved formulation of the problem for the purposes of detecting rule anomalies. An alternative for the full problem is given by Grout and McGinn [3] along with a simple, but not particularly efficient, method of solution that makes real-time online optimisation possible for moderate numbers of rules. The complexity of the algorithm is around $O(n^3)$ – impractical for large lists. Finally, Grout et al. [11] offer an efficient heuristic. However, at the time of going to print, the paper left some questions unanswered. It is the purpose of this paper to finish the story by completing the analysis, modifying the algorithm slightly and testing and establishing the value of key parameters.

3. An Efficient Solution

Grout et al. [11] note the following:

- In comparing rule order for a list A, the significance of rule hit-rates is only *relative*. It is not necessary for them to be normalised (i.e. summing to *1*) probabilities. This implies that the hit-rate of a newly hit rule, *i,* can increase without changing the hit-rates of the other rules.
- Following an increase in a rule i's hit-rate, the only possible change in rule order (to reduce E_A) is to promote i up the list. The most likely candidate with which to exchange it is rule $i-1$, immediately above it.
- The potential saving in expected latency in swapping rules $i-1$ and i is given by $h_i l_{i-1} - h_{i-1} l_i$ (see the original paper for the full expansion), a simple, local calculation.
- Considering rule promotions continuously in this manner is entirely responsive to dynamically changing traffic patterns.

These observations allow the search process to be simplified considerably. A simple three-part heuristic algorithm for ACL optimisation is then proposed as follows:

```
Step 1: Initialisation
      for i := 1 to n do
         h_i := 1/n

Step 2: On processing a packet matching rule i
      h_i := θh_i;
      if (d_{i-1 i} =0) and h_i l_{i-1} - h_{i-1} l_i > 0 then
         Swap(i-1, i)

Step 3: Renormalisation to prevent overflow
      for i := 1 to n do
         h_i := h_i / H
```

Step 1 sets all rule hit-rates to be equal, normalised probabilites when the ACL is initially configured (or reconfigured). Step 2 is executed after the processing of every packet. The hit-rate of the matched rule is increased by a factor θ, the *promotion coefficient*, and the rule is swapped with its predecessor if the two are independent and the hit-rate/latency trade off is favourable. (Note, for the analysis to follow, that swapping two rules also entails swapping their respective hit-rates.) At certain intervals, Step 3 stops the hit-rates increasing without bound and thus prevents overflow. In the original paper, H is the sum of the individual hit-rates for all rules, $H = \sum_{i=1}^{n} h_i$, thus renormalising values. Steps 1 and 3, of order $O(n)$ are executed infrequently whilst the continuously used Step 2 is a single simple calculation. The algorithm places an exponentially decreasing importance on older rule matches, parameterised by θ.

Grout et al. [11] leave the following questions largely unanswered with respect to this algorithm. What should the value of θ be? How frequently is Step 3 necessary?

Before continuing, this paper revises (streamlines) the above algorithm as follows:

```
Step 1: Initialisation (on configuration/reconfiguration)
      for i := 1 to n do
         h_i := 1

Step 2: Promotion (on a match of rule i)
      h_i := θh_i;
      if (d_{i-1 i} =0) and h_i l_{i-1} > h_{i-1} l_i then
         Swap(i-1, i)

Step 3: Reduction (every p packets)
      for i := 1 to n do
         h_i := h_i / H
```

The simplified initialisation (Step 1) reflects the fact that the hit-rates need not be normalised. Step 2 remains trivial.

p and H are easily determined. The fastest route to overflow is through a stream of packets all matching the same rule. The hit-rate of this rule will increase by a factor θ on each packet and, after a packets, will have a hit rate of θ^a. If M is the largest value permitted in the data range being used, then p can be calculated as $p = \lfloor log_\theta M \rfloor$, a router constant. ($\lfloor x \rfloor$ is the integer part of x.) Taking $H = \max_i h_i$ will reduce the maximum value back to *1* each time Step 3 is executed. It remains to determine the value for θ.

4. Simulation and Results

The intention of this paper is to establish, *in principle*, the viability of a simple, embedded heuristic optimisation technique acting upon rule order in ACLs. This is fortunate because the process of obtaining results from live ACLs is hindered by a number of factors:

- Our initial aim is to be able to *characterise* both ACLs and traffic flows, and the relationship between them. ACLs are characterised by their size, the level of interdependence between rules and the distribution of rule hit-rates. (For some ACLs, hit-rates may be close to uniform and/or largely independent of rule order; in others, some rules – typically those close to the start – may have hit-rates considerably larger than the rest.) Traffic is characterised, in particular, by its stability: a stream of similar packets may match the same rule whereas unstable flows will find different matches throughout an ACL. Our ultimate aim is to investigate how appropriate our proposed method will be for a spectrum of ACL types and traffic flows. This level of parameterisation will be difficult to obtain from production lists and flows. Although there are some sources of 'real' traffic for simulation purposes, ACLs are harder to find: their data is simply not available in sufficient, measurable quantity (at least in their thousands) to enable us to produce consistent results.
- The low-level implementation of packet-matching techniques varies considerably from platform to platform [12]. Although primarily hardware-based, the precise relationship with the (eg, router) operating system is vendor-, and sometimes model-, dependent. Even in hardware alone, the number of steps to perform any operation will depend on the unique architecture and organisation of the platform. Using real ACLs for simulation serves little purpose if their performance varies between models. Using an abstract simulation with parameterisation that can be tailored to any given environment is actually a stronger proposition.
- Experimenting with the proposed algorithm, and testing variations of key parameters (eg, θ) in a working environment is difficult on some platforms, particularly in hardware implementations. Without access to the embedded hardware itself, experimentation is impossible on most production routers. Limited control is possible on some platforms ([13] for example) but can only test a small proportion of cases. The final alternative is a general network simulator. ns-2, for example [14], offers some generic packet-processing functionality. However, such simulations are generally small and, abstracted as they themselves are, no more accurate or 'real' than the processes described below.

Table I. Simulated Results: Rank and Cumulative Latencies (example).

ACL length (*n*): 1 000 rules. Stream length: 4 000 000 packets. $\theta = 1.5$

3 changes in packet flow characteristics.

Dependency index (*DI* – probability of a dependency between any two rules): 0.25
Self-similarity index (*SSI* – probability of each packet belonging to the same stream as the previous one): 0.75

Table shows mean position of matched rule and mean (cumulative) latency since last checkpoint (*), since last traffic variation (") and since start of packet stream (^)

Packet flow	Number of Packets	Average Rank* R*	Average Rank" R"	Average Rank^ R^	Average Latency* L*	Average Latency" L"	Average Latency^ L^
(initial)	100000	485.26	485.26	485.26	366.69	366.69	366.69
	200000	448.66	466.96	466.96	338.82	352.76	352.76
	300000	417.56	450.49	450.49	315.14	340.22	340.22
	400000	391.89	435.84	435.84	295.61	329.06	329.06
	500000	372.26	423.12	423.12	280.83	319.42	319.42
	600000	356.86	412.08	412.08	269.20	311.05	311.05
	700000	349.02	403.07	403.07	263.29	304.23	304.23
	800000	340.53	395.25	395.25	256.89	298.31	298.31
	900000	338.29	388.92	388.92	255.16	293.51	293.51
	1000000	333.14	383.35	383.35	251.33	289.30	289.30
(variation)	1100000	487.61	487.61	392.82	364.08	364.08	296.09
	1200000	455.80	471.71	398.07	340.46	352.27	299.79
	1300000	424.65	456.02	400.12	317.41	340.65	301.15
	1400000	396.19	441.06	399.84	296.09	329.51	300.79
	1500000	374.08	427.67	398.12	279.42	319.49	299.36
	1600000	360.43	416.46	395.76	269.12	311.10	297.47
	1700000	348.11	406.70	392.96	260.16	303.82	295.28
	1800000	345.88	399.09	390.35	258.65	298.17	293.24
	1900000	336.54	392.14	387.51	251.78	293.02	291.06
	2000000	334.00	386.33	384.84	249.91	288.71	289.00
(variation)	2100000	480.18	480.18	389.38	358.17	358.17	292.30
	2200000	447.21	463.69	392.01	333.58	345.88	294.17
	2300000	419.02	448.80	393.18	312.50	334.75	294.97
	2400000	391.50	434.48	393.11	292.04	324.07	294.85
	2500000	372.56	422.09	392.29	278.02	314.86	294.17
	2600000	358.98	411.57	391.01	268.09	307.07	293.17
	2700000	348.82	402.61	389.45	260.85	300.46	291.97
	2800000	344.28	395.32	387.83	257.67	295.12	290.75
	2900000	340.32	389.21	386.19	254.85	290.64	289.51
	3000000	339.55	384.24	384.64	254.42	287.02	288.34
(variation)	3100000	476.78	476.78	387.61	355.68	355.68	290.51
	3200000	442.44	459.61	389.33	330.09	342.88	291.75
	3300000	414.21	444.48	390.08	309.26	331.68	292.28
	3400000	393.23	431.67	390.17	293.73	322.19	292.32
	3500000	376.00	420.53	389.77	281.09	313.97	292.00
	3600000	358.76	410.24	388.91	268.47	306.39	291.35
	3700000	350.40	401.69	387.86	262.32	300.09	290.56
	3800000	343.42	394.41	386.70	256.97	294.70	289.68
	3900000	344.01	388.81	385.60	257.34	290.55	288.85
	4000000	339.55	383.88	384.45	254.02	286.90	287.98

Our simulation is based on an in-house numerical model, capable of generating ACLs and traffic flows according to a given parameter set, described as follows. For tested ACLs, the number of rules (n) ranged from *10* to *10 000*. Dependencies between rules were determined using a *dependency index*, *DI*, the probability that any two rules are dependent. Values of *DI* in the range *0* (no dependencies) to *1* (complete dependency) were used. For each rule pair, *(i,j)*, dependencies are randomised as $d_{ij} = 1$ with probability *DI* and *0* otherwise. Rule latencies were uniformly randomised from *0.5*μs to *1.0*μs. Actual values depend on the router hardware of course [14] but it is only *relative*

values that are significant. (Routers that process packets faster will also optimise faster – see the conclusions section that follows.)

For traffic, the simulation is more sophisticated. The traffic simulator generates packets with given probabilities of matching each rule in the list. At intervals, these probabilities may change to reflect shifting traffic patterns. Within a single traffic pattern, however, there is a certain probability that a packet is identical to the previous one – or part of a similar stream - and matches the same rule.

So, at the start of the simulation, a value of the *self-similarity index*, *SSI*, is set. Then a *match probability*, ρ_i is randomised for each rule i and normalised so that $\sum_{i=1}^{n} \rho_i = 1$. The first packet is generated, matching rule i with probability ρ_i. Subsequent packets match the same rule with probability *SSI*, and otherwise match any rule according to the match probabilities, ρ_i. Every q packets, the match probabilities, ρ_i, are re-randomised.

n and *DI* can be set to produce different types of ACL while q and *SSI* vary to reflect different types of traffic. As an example, Table I records simulated output from a test with $\theta = 1.5$, $n = 1\,000$, $DI = 0.25$, $q = 1\,000\,000$ and $SSI = 0.75$. $4\,000\,000$ packets are generated in total, in four stages with varying profiles. Results are reported every 100 000 packets.

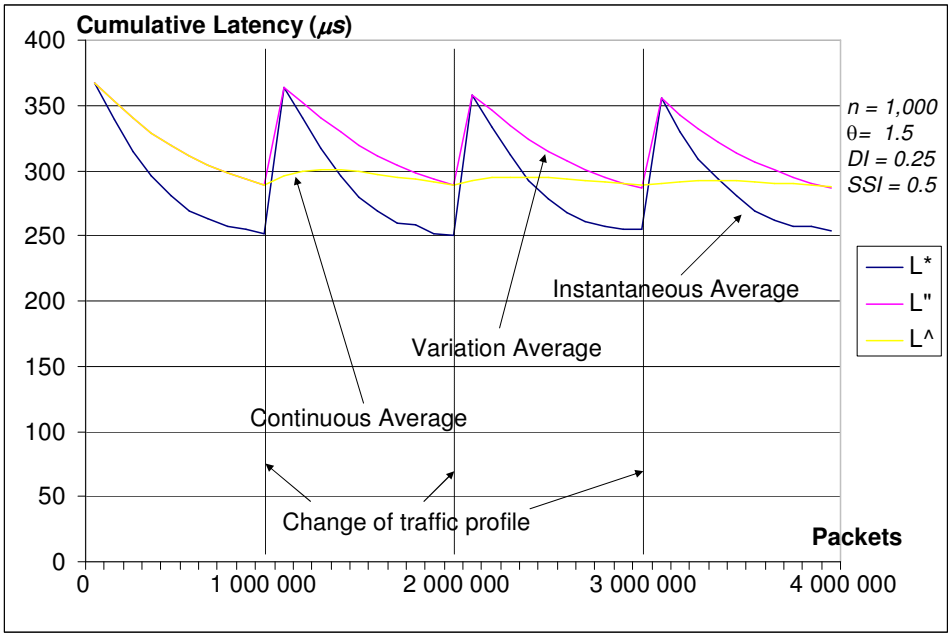

Figure 2. Simulated Results: Cumulative Latencies.

Tabled results are the mean position of the matched rule (*rank*) in the ACL and the mean cumulative latency of this rule. In both cases, three values are given: the mean since the last set of figures (*R* & L**) – the *instantaneous average*, the mean since the last traffic variation (*R" & L"*) – the *variation average*, and the mean of the entire simulation (*R^ &*

L^) – the *continuous average*. The three latency averages, *L**, *L"* and *L^*, are plotted in Figure 2.

The mean rank, *R*, for a *1 000* rule list with no optimisation will be *500* and the mean cumulative latency, *L*, for a latency range of *0.5* to *1.0*, *500 x (1.0 + 0.5) / 2 = 375*. In simulation, optimised averages start at these values and are then progressively lowered as rules with high hit rates are promoted. When traffic profiles change, instantaneous and variation averages become poor again but are gradually improved once more as the ACL adapts to the new characteristics. The continuous average becomes steadier. In this example, *L^* approaches a figure of approximately *287*, an improvement of *23%* on the non-optimised figure.

Table II. Simulated Results: Traffic Parameters and Promotion Coefficient.

ACL length (*n*): 1 000 rules. Stream length: 4 000 000 packets.
DI – Dependency Index. *SSI* – Self-Similarity Index.
Traffic (packet) characteristics change every *q* packets.

Table shows values of percentage improvement in expected latency (*100(L-L^)/L*) for different values of *DI*, *SSI*, *q* and *θ*.

		DI =	0	0.25	0.5	0.75	1
SSI = 0	θ =	1.1	15	14	13	10	0
q = 10		1.5	15	14	13	10	0
		2.0	15	14	13	10	0
		2.5	14	13	12	9	0
		1.5	14	13	12	9	0
SSI = 0.25	θ =	1.1	17	15	13	10	0
q = 1 000		1.5	17	15	13	11	0
		2.0	17	15	14	11	0
		2.5	17	15	14	11	0
		1.5	17	15	13	10	0
SSI = 0.5	θ =	1.1	19	17	15	10	0
q = 50 000		1.5	21	18	15	11	0
		2.0	21	18	15	12	0
		2.5	21	18	15	12	0
		1.5	21	18	15	12	0
SSI = 0.75	θ =	1.1	19	17	15	12	0
q = 1 000 000		1.5	26	23	20	13	0
		2.0	28	27	20	14	0
		2.5	28	27	20	14	0
		1.5	28	27	20	14	0
SSI = 1	θ =	1.1	20	19	16	13	0
no variation		1.5	27	25	20	13	0
		2.0	30	29	22	16	0
		2.5	30	29	22	16	0
		1.5	30	29	22	16	0

Different parameters affect these values as shown in Table II. Results are *proportionally* similar for different *n*. High values of *DI* work against the optimisation process, prohibiting desirable swaps. In the extreme cases, *DI = 1* prevents *any* optimisation whereas *DI = 0* allows rules to move freely. High values of *q* and *SSI* imply greater traffic stability, which improves the optimised values. The effect of *θ* is more subtle.

High values make rule promotion faster, which works well for self-similar, stable traffic but can lead to repetitive, unnecessary swaps for continuously changing, or oscillating, traffic patterns. A balance is necessary: a value around $\theta = 2$ maximises the improvement in expected latency in most cases.

5. Analysis

Routers vary considerably in their operation, particularly in terms of implementation in hardware. The following is, by necessity, generic and, to some extent, imprecise. However, it gives an appropriate indication of the *relative* worth of dynamic optimisation. We discuss an *operation* simply as a unit of calculation or assignment, probably performed in hardware on the appropriate interface. (However, the same argument would apply in relative terms if these operations were to be a part of the operating system software.)

With a B-bit data type/range (e.g. register size) and $\theta = 2$, we have $p = \lfloor log_2 2^B \rfloor = B$. For any given ACL manual configuration (or reconfiguration), Step 1 of the algorithm is executed once and can be taken as part of the configuration, Step 2, every processed packet and Step 3 every B packets. Step 2 consists of an assignment, two calculations, two comparisons and a conjunction (possibly) followed by a swap of six assignments – three for the rules and three for their hit-rates - twelve operations in all. Step 3 has two loops of size n, one to establish the maximum value and the other to reduce each value. The mean complexity (of Step 3) each packet is then $2n / B$ and, in total, $12 + 2n / B$ for Steps 2 & 3 combined.

Table III. Optimisation Trade-Off – Saving against Cost

$DI = SSI = 0.5$. $\theta = 2$.

Table shows value of trade-off function, $T = \varphi n/25 - 12 - 2n/B$, for different values of n and B.

	B =	8	16	32	64
n =	10	-8.50	-7.25	-6.63	-6.31
	30	-1.50	2.25	4.13	5.06
	100	23.00	35.50	41.75	44.88
	300	93.00	130.50	149.25	158.63
	1 000	338.00	463.00	525.50	556.75
	3 000	1038.00	1413.00	1600.50	1694.25
	n^* =	34.28	25.26	22.32	21.10

n^* is the minimum length of list for T to be positive (i.e. for optimisation to be worthwhile).

Matching a packet against a rule consists of at least one operation (permit or deny) followed by between *1* and *5* comparisons (Figure 1). Taking a mean of $1 + 3 = 4$ operations per rule and a percentage saving for an optimised list of φ gives an optimisation *trade-off* of

$$T = \frac{4\varphi n}{100} - 12 - \frac{2n}{B}, \tag{4}$$

which will be positive (i.e. worthwhile) when

$$\varphi > \frac{300}{n} + \frac{50}{B}. \tag{5}$$

For example, taking $n = 1\,000$ and $B = 16$, this gives $300 / 1\,000 + 50 / 16 = 3.425$. Table II shows that the improvement, φ, exceeds this for all values other than $DI = 1$ and is therefore worthwhile. Alternatively, taking $\theta = 2$ and $DI = SSI = 0.5$ gives an improvement of $\varphi = 15$ and a trade-off of $T = (15 \times 1\,000) / 25 - 12 - 2\,000 / 16 = 463$, a positive benefit. Table III extends this calculation across a range of values of n and B and, for each B, shows the key value of n^*, the size of ACL for which optimisation is profitable. Table IV fixes B at 16 and calculates n^* for various values of DI and SSI.

Table IV. Optimisation Trade-Off – Minimum ACL Length

$\theta = 2$.

Table shows the value of n^*, the minimum length of list for $T = \varphi n/25 - 12 - n/8$, to be positive (i.e. for optimisation to be worthwhile) for different values of DI and SSI.

	DI =	0.0	0.25	0.5	0.75	1.0
SSI =	0	25.26	27.59	30.37	43.64	∞
	0.25	21.62	25.26	27.59	38.09	∞
	0.5	16.78	20.17	25.26	33.80	∞
	0.75	12.06	12.57	17.78	27.59	∞
	1	11.16	11.59	15.89	23.30	∞

6. Conclusions

No amount of traffic modelling can substitute entirely for testing on production routers. However, our simulations are extensive and, within themselves, give consistent results.

The major obstacle to successful (worthwhile) optimisation is highly interdependent rules in an ACL. If no or few rules are permitted to be reordered then it is impossible or difficult to find equivalent lists with lower expected latencies. However, this is rarely the case in practical ACLs. The typical ACL in Figure 1, for example, has large blocks of separate 'permit' and 'deny' blocks with no dependencies within them. A worst-case figure for a practical ACL is likely to be $DI \approx 0.5$, giving good results (Tables II & IV).

Table II suggests $\theta = 2$ as an appropriate (and, in fact, convenient) value for the promotion coefficient. The number of packets between hit-rate reductions (Step 3) is then B, the size (number of bits) of the register being used to store them. The final version of the three part algorithm then becomes:

```
Step 1: Initialisation (on configuration/reconfiguration)
      for i := 1 to n do
         h_i := 1

Step 2: Promotion (on a match of rule i)
      h_i := 2h_i;
      if (d_{i-1 i} =0) and h_i l_{i-1} > h_{i-1} l_i then
         Swap(i-1, i)

Step 3: Reduction (every B packets)
      for i := 1 to n do
         h_i := h_i / max_j h_j
```

Depending on the stability and self-similarity of the traffic (*q* and *SSI*) and the frequency of hit-rate reduction (*B*), optimisation becomes worthwhile for ACLs above a certain length (*n**) (Tables III & IV). For realistic dependencies, this figure ranges between about *10* and *30*. It will be trivial to separate those lists to which optimisation is to be applied from those to which it is not. Of course, it is precisely for longer ACLs that optimisation will yield the best results.

Real-time, online, embedded ACL optimisation on operational routers, as proposed, is possible and worthwhile. It is now recommended that it be put into practice as an integral part of the router hardware for practical testing.

7. References

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

[3] Grout, V. and McGinn, J., 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)

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

[5] Stoica, I., Route Lookup and Packet Classification, *Technical Report No. CS 268*, Department of Electrical Engineering and Computer Science, University of California, Berkeley, USA, 2001.

[6] *ACL Optimizer and Hits Optimizer*, Cisco Systems, USA, (http://www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/cw2000/fam_prod/acl_mgr/aclm_1_x/1_6/u_guide/ac1js.htm)

[7] Bukhatwa, F. and Patel, A., Effects of Ordered Access Lists in Firewalls, *Proceedings of IADIS WWW/Internet International Conference (W3I 2004)*, Algarve, Portugal, 5th-8th November 2003, pp257-264.

[8] Bukhatwa, F., High Cost Elimination Method for Best Class Permutation in Access Lists, *Proceedings of IADIS WWW/Internet International Conference (W3I 2003)*, Madrid, Spain, 6th-9th October 2004, pp287-294.

[9] *Turbo Access Control Lists*, Cisco Systems, USA, (`http://www.cisco.com/univercd/cc/td/doc/product/software/ios121/121newft/121t/121t5/dttacl.htm`)

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

[11] Grout, V., McGinn, J. and Davies, J., Reducing Processing Latency in Network Packet Filters, *Proceedings of the Fifth International Network Conference* (*INC 2005*), Samos, Greece, July 2005, pp3-10.

[12] Varghese, G., *Networking Algorithmics: An interdisciplinary approach to designing fast networking devices*, Morgan Kaufmann, 2005.

[14] *The Network Simulator – ns-2*, (`http://www.isi.edu/nsnam/ns/`)

[15] Suehring, S. and Ziegler, R., *Linux Firewalls (3rd edition)*, Novell Press, 2005.

Biographies

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 100 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, Scientist and Mathematician and a Fellow of the British Computer Society (BCS). He chairs the biennial international conference series on Internet Technologies and Applications (ITA).

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)

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).