Traffic Modelling and Simulation Techniques for Evaluating ACL Implementation

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*

Follow this and additional works at: [http://epubs.glyndwr.ac.uk/cair](http://epubs.glyndwr.ac.uk/cair)

Part of the [Computer and Systems Architecture Commons](http://epubs.glyndwr.ac.uk/Computer-and-Systems-Architecture), [Digital Communications and Networking Commons](http://epubs.glyndwr.ac.uk/Digital-Communications-and-Networking), [Hardware Systems Commons](http://epubs.glyndwr.ac.uk/Hardware-Systems), and the [Systems and Communications Commons](http://epubs.glyndwr.ac.uk/Systems-and-Communications)

**Recommended Citation**
Traffic Modelling and Simulation Techniques for Evaluating ACL Implementation

Abstract
This paper presents a modelling and simulation framework for analysing Access Control List (ACL) implementation on Internet devices. It uses the established modelling/simulation techniques of abstraction and simplification to isolate the essential components of the system from peripheral issues. As a case study, the viability of a simple real-time optimisation technique is demonstrated.

Keywords
internet traffic, access control lists, packet classification, ACL optimisation

Disciplines
Computer and Systems Architecture | Digital Communications and Networking | Hardware Systems | Systems and Communications

Comments
This paper was presented at the Seventh International Network Conference (INC 2008), 8-10 July 2008, which was held in Plymouth, UK. It was published by the University of Plymouth and details of the conference are available at http://www.cscan.org
Traffic Modelling and Simulation Techniques for Evaluating ACL Implementation

Vic Grout, John N. Davies and John McGinn

Centre for Applied Internet Research (CAIR)
University of Wales, NEWI, Wrexham, UK
{v.grout|j.n.davies|j.mcginn}@newi.ac.uk

Abstract

This paper presents a modelling and simulation framework for analysing Access Control List (ACL) implementation on Internet devices. It uses the established modelling/simulation techniques of abstraction and simplification to isolate the essential components of the system from peripheral issues. As a case study, the viability of a simple real-time optimisation technique is demonstrated.

Keywords

Internet traffic, Access control lists, Packet classification, ACL optimisation

1. Introduction: Modelling Internet Traffic

Internet traffic flow has complex characteristics, both in scale and structure. Modelling and simulating traffic can, consequently, be troublesome. However, in certain circumstances, not all traffic features are relevant to the simulation and a degree of simplification is appropriate. This paper considers a model of Internet traffic, as sequences of packets, applied to the simulation of the behaviour of Access Control Lists (ACLs) using different implementations. The nature and structure of ACLs are described, together with the packets they process. The essential elements of the simulation are identified and relevant parameters are introduced. An extensive case study with results is given in conclusion.

There has been substantial discussion of the nature of Internet traffic over the years, (Paxson and Floyd, 1995) (Paxson, 1999) for example. One of the few coherent conclusions, albeit it an obvious one, is that Internet traffic is very complex indeed. Analysing traffic on a network over time shows both a level of self-similarity (Leyland et al., 1994) (Rezaul and Grout, 2007) and an effective randomness (Dang et al., 1999) (Jerkins and Wang, 1999) on unpredictable scales. This makes the effective modelling and simulation of network activity extremely difficult in any general form. It is true that some fairly sophisticated network simulators exist, (ns-2, 2007) (cnet, 2007) (Cisco, 2007) for example, but the power of these tools lies primarily in their scale rather than their fine-tuning. Although most network protocols are supported, for example, it is still hard to imitate the subtleties of real network traffic under changing conditions. None are particularly straightforward to use and, to date, successful simulations using such tools have tended to be small (Tan et al., 2006) (Zhu and Roy, 2004). Also, although sources of 'real' network
traffic for experimentation are to be found (Kos et al., 2003) (Abilene, 2007), they are not numerous and, beyond a basic description of form (time, place, traffic type, etc.) do not offer obvious means of identifying traffic characteristics (distribution, randomness, stability, self-similarity, etc.). Results from network simulations might reasonably be expected to vary with these traffic characteristics and, without such knowledge, will have limited value.

However, a completely generic network model is often unnecessary. Depending upon the purpose of the simulation, certain traffic characteristics will be significant; others not. Dispensing with the unnecessary network characteristics can often simplify the traffic model and allow relevant parameters to be used to fine-tune the simulation as required. This paper is concerned with the implementation of Internet traffic filters, otherwise known as Access Control Lists (or ACLs). An ACL processes traffic as a sequence of packets, and is itself a sequence of rules, as described in sections 2 and 3. Modelling the interaction of two such sequences proves to be a comparatively simple process in which most characteristics of the packets and rules may be ignored.

The remainder of the paper is organised as follows: Section 2 introduces the essential role and behaviour of ACLs in simple, practical terms. This intuitive model is then extended and formalised in sections 3 and 4. Section 3 derives an appropriate model for the rules of an ACL processing a sequence of packets before section 4 introduces the relevant traffic characteristics in this environment, which prove to be few. Some parameters then depend on the method of ACL implementation. This is considered generically in section 5 and illustrated through an example in section 6. Section 7 concludes briefly.

2. An Overview of ACL Purpose and Structure

We begin with a very brief description of context. An internetwork (internet) is a ‘network of networks’. (The Internet, with which we are familiar, is conventionally written with a capital.) Key devices, known as routers, switch, or route, communications traffic, usually in the form of discrete packets, between networks. The primary function of a router is to forward each packet to the most suitable device, typically another router, at each step of the journey. However, a vital secondary role is to consider whether a given packet should be passed at all, according to a set of tests, or rules, against which it is matched.

A typical rule, in the syntax of the Cisco Internetwork Operating System (IOS) (Colton, 2002), is

```
access-list 101 deny icmp any 10.0.0.0 0.255.255.255 echo-reply
```

which states that ICMP echo-reply packets from any source to the network 10.0.0.0 are to be blocked at this point. The first part of the rule simply assigns it to access list 101 (and may be ignored when discussing single lists in isolation.).

An access list, or Access Control List (ACL), is then a sequence of such rules designed to implement a given objective or set of objectives. ACLs can be used for security purposes, simply to pass or block packets, or as filters for more sophisticated
policies such as traffic shaping, address translation, queuing or encryption (Syngress, 2002). A packet may be matched against several ACLs on a single router and many more on its complete journey from source to destination. Inefficiently implemented ACLs can add significantly to packet delay and even small ACLs will contribute to this latency simply by their aggregation across several routers.

An example of a complete ACL is given in Figure 1. Other than the ACL assignment, a rule may consist of up to five parts: the permit or deny type, the protocol, a source address, destination address and a flag function (as in the echo-reply parameter above) for fine-tuning. Each parameter may be a single value or a range of allowable matches. For example, the any parameter above matches all source addresses whilst the 0.255.255.255 parameter matches destination addresses in the 10.0.0.0 network. The absence of any term, such as an address, a protocol or flag, indicates the rule will match a packet with any such values – provided those fields that are present are matched.

```
access-list 101 permit tcp 192.168.212.0 0.0.0.255 10.0.0.0 0.255.255.255 eq telnet
access-list 101 permit tcp 192.168.212.0 0.0.0.255 10.0.0.0 0.255.255.255 eq ftp
access-list 101 permit tcp 192.168.212.0 0.0.0.255 10.0.0.0 0.255.255.255 eq http
access-list 101 permit icmp any 10.0.0.0 0.255.255.255 administratively-prohibited
access-list 101 permit icmp any 10.0.0.0 0.255.255.255 echo-reply
access-list 101 permit icmp any 10.0.0.0 0.255.255.255 packet-too-big
access-list 101 permit icmp any 10.0.0.0 0.255.255.255 time-exceeded
access-list 101 permit icmp any 10.0.0.0 0.255.255.255 unreachable
access-list 101 permit icmp 172.16.20.0 0.0.255.255
access-list 101 deny icmp any any
access-list 101 permit ip 202.33.42.0 0.0.0.255 any
access-list 101 permit ip 202.33.73.0 0.0.0.255 any
access-list 101 permit ip 202.33.48.0 0.0.0.255 any
access-list 101 permit ip 202.33.75.0 0.0.0.255 any
access-list 101 deny ip 202.33.0.0 0.0.0.255 any
access-list 101 deny tcp 210.120.122.0 0.0.0.255 10.2.2.0 0.255.255.255 eq www
access-list 101 deny tcp 210.120.114.0 0.0.0.255 10.2.2.0 0.255.255.255 eq www
access-list 101 deny tcp 210.120.175.0 0.0.0.255 10.2.2.0 0.255.255.255 eq www
access-list 101 deny tcp 210.120.136.0 0.0.0.255 10.2.2.0 0.255.255.255 eq www
access-list 101 deny tcp 210.120.177.0 0.0.0.255 10.2.2.0 0.255.255.255 eq www
access-list 101 deny tcp any 10.2.2.0 0.255.255.255 eq www
access-list 101 permit tcp any 10.2.2.0 0.255.255.255 eq www
access-list 101 deny tcp any any eq www
access-list 101 permit tcp any any
access-list 101 deny ip 195.10.45.0 0.0.0.255 any
access-list 101 permit ip any any
{access-list 101 deny all} (implicit)
```

Figure 1. A typical Access Control List (ACL)

The interpretation of an ACL is that its rules are considered as being processed in sequential order from the top. That is, each incoming packet is tested against the first rule; if it matches, it is passed or blocked accordingly and no further rules are considered; otherwise it is tested against the second rule, and so on. There is an implicit deny all rule at the end of each ACL to block all packets not otherwise matched.

There are three further points to make in overview. Firstly, this model of an ACL as a sequence of rules, considered in order, is only a question of interpretation: it should not be assumed that the ACL is actually processed sequentially within the
device hardware or software (see section 5). Secondly, taking this interpretation of ACL structure, the order of the rules is crucial: an inherent dependency between rules prohibits arbitrary reordering. For example, in Figure 2, an IP packet from the network 192.168.16.0 to the network 10.0.0.0 will match both rules shown. The packet will be passed in 2(a) but blocked in 2(b). Clearly then, rules may not be reordered if this changes the underlying intention of the policy. Thirdly, not all rules are equally likely to match packets: rules with larger parameter ranges (or indeed absent parameters) may match more packets and rule hit-rate will vary among them. Also, different rules will become more or less significant as traffic (packet) characteristics change so these same hit-rates will be dynamic. These concepts are developed in the next section and used in the case study in section 6.

![Access list 2(a)]

![Access list 2(b)]

**Figure 2. Dependent rules**

### 3. Modelling ACL Structure

Where appropriate in this paper, abbreviations are used as follows: ∃, ‘there is’ or ‘there exists’; ∀, ‘for all’ or ‘for every’; ∧, ‘and’; ⇔, ‘if and only if’; and →, ‘such that’. We also use the terms format and protocol in a precise manner: format refers to the layout of packets and rules in any given system whereas protocol implies a data/traffic type that may be identified within it. Then 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 $F^* = \{0, 1\}^w$ to be the set of $w$ flag vectors ($\{0, 1\}$ $w$-tuples acting on $B^*$) valid for the system. For completeness, $X^*$ represents the set of payloads.

#### 3.1. Packets, rules and policies

For a given format, a packet, $p_k = (s_{ak}, d_{ak}, b_k, f_k, X_k)$, is defined by its constituents: $s_{ak} \in A^*$, the source address; $d_{ak} \in A^*$, the destination address; $b_k \in B^*$, the
protocol; \( f_k \in F^* \), 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, \( t_i \in \{\text{permit, deny}\} \), \( SA_i \subseteq A^* \): the source range, \( DA_i \subseteq A^* \): the destination range, \( B_i \subseteq B^* \): the protocol range, and a flags predicate, \( \sigma_i : F^* \mapsto \{\text{true, false}\} \). 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 \( \sigma_i \equiv \text{true} \) by default.

A packet, \( p_k \), matches a rule, \( r_i \) (for which we write \( p_k \mathrel{\triangleright} r_i \)), 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,

\[
p_k \mathrel{\triangleright} r_i \Leftrightarrow (SA_k \in SA_i) \land (DA_k \in DA_i) \land (b_k \in B_i) \land \sigma_i (f_k),
\]

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

A policy, \( Z = \{r_1, r_2, ..., r_n\} \) is an (ordered) sequence of \( n \) rules to achieve some purpose. The last rule implicitly denies all traffic; that is, \( t_n = \text{deny}, SA_n = A^*, DA_n = A^*, B_n = B^* \) and \( \sigma_n \equiv \text{true} \).

### 3.2. Dependencies and redundancies

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 ((\( p_k \mathrel{\triangleright} r_i \) \( \land \) \( p_k \mathrel{\triangleright} r_j \)));

that is \( r_i \) and \( r_j \) are dependent if

\[
(t_i \neq t_j) \land \exists p_k \rightarrow (SA_k \in SA_i \land SA_j) \land (DA_k \in DA_i \land DA_j) \\
\land (b_k \in B_i \land B_j) \land \sigma_f (f_k) \land \sigma_f (f_i).
\]

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:

\[
d_{ij} \Leftrightarrow (t_i \neq t_j) \land (SA_i \lor SA_j \neq \emptyset) \land (DA_i \lor DA_j \neq \emptyset) \\
\land (B_i \lor B_j \neq \emptyset) \land (\Sigma_i \lor \Sigma_j \neq \emptyset),
\]

where \( \Sigma_i \subseteq F^* \) is the subset of flag vectors satisfying \( \sigma_i \).

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. On this basis, the dependency index, a normalised measure of rule interdependency, for a set of \( n \) rules, can be defined as:

\[
DI = \frac{2}{n(n-1)} \sum_{i=1}^{n-1} \sum_{j=i+1}^n d_{ij}
\]

and is used in section 6. \( DI = 0 \) means no dependent rules; \( DI = 1 \) means all rules dependent upon all others. Higher values of \( DI \) constrain rule order more tightly.

A rule, \( r_j \), in a policy, \( Z \), is redundant (written \( r_i \supseteq r_j \)) if there exists a rule, \( r_i \) \( (i < j) \), in \( Z \), such that all packets matching \( r_j \) will be matched by \( r_i \).

\[
r_i \supseteq r_j \Leftrightarrow (t_i = t_j) \land (SA_i \supseteq SA_j) \land (DA_i \supseteq DA_j) \land (B_i \supseteq B_j) \land (\Sigma_i \supseteq \Sigma_j).
\]
A rule, \( r_i \), in a policy, \( Z \), is potentially redundant if there exists a rule, \( r_j \) \((i < j)\), in \( Z \), such that all packets matching \( r_i \) will be matched by \( r_j \). A redundant rule may be removed from the policy without changing its purpose provided that no other rules between \( r_i \) and \( r_j \) are dependent upon \( r_j \); that is,

\[
\begin{align*}
  r_i \rightarrow r_j &\iff (t_i = t_j) \land (SA_i \subseteq SA_j) \land (DA_i \subseteq DA_j) \land (B_i \subseteq B_j) \\
  &\land (\Sigma_i \subseteq \Sigma_j) \land \forall v \to (i < v < j), d_{ij} = 0.
\end{align*}
\]

Both forms of redundancy include the case, \( r_i = r_j \).

Finally, and in brief, rules, \( r_{\alpha} \), \( r_{\beta} \), \( r_{\gamma} \), .., are said to be co-redundant if there can be found a rule, \( r_i \) \((i < \alpha, \beta, \gamma, ...)\), such that \( r_i \) can replace \( r_{\alpha} \), \( r_{\beta} \), \( r_{\gamma} \), .., Equivalent definitions may be derived for co-redundancy with respect to source/destination address and protocol/flags, and for potential co-redundancy.

A useful tutorial approach to the detection and management of redundancies is given in (Qian et al., 2001). (Al Shaer and Hamed, 2004) gives an updated treatment. Although interesting, these concepts are not central to this work. The models discussed in this paper apply whether or not the policy, \( Z = \{r_1, r_2, ..., r_n\} \), contains redundancies.

### 3.3. Hit-rates and latencies

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 \( r_i(I_Z) = r_i \forall i (1 \leq i \leq n) \).

The hit-rate, \( h(r_i(L),T) \), of rule \( r_i \) in a list \( L \) is the probability that a packet from a traffic flow \( T \) will match \( r_i \) in \( L \). Hit-rates can be calculated dynamically using counters within the IOS or hardware (Cisco, 2002) (Cisco, 2003).

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 the implementation of some systems, latencies may be constant for all rules (see section 5.1) but this is not assumed generally in this paper.

### 4. Modelling Traffic Characteristics

Using the definition of a packet from section 3.1, a packet stream, \( P_q = \{p_1, p_2, ..., p_q\} \), is simply a sequence of \( q \) packets. However, there are two other parameters necessary for a complete description of traffic flow: the length of each packet and the inter-arrival time. Define \( \varphi_k \) to be the length of packet \( k \) (largely dependent upon the payload \( X_k \)) and \( \psi_k \) to be the time difference between the arrival (or passing) of packets \( k \) and \( k+1 \). \((\psi_0 = 0 \text{ by default})\). Then, the sequences \( \Phi_q = [\varphi_1, \varphi_2, ..., \varphi_q] \) and \( \Psi_q = [\psi_1, \psi_2, ..., \psi_q] \) describe the distributions of packet lengths and inter-arrival times respectively for a sequence of \( q \) packets. No assumptions are made with regard to the nature of these distributions (Paxson and Floyd, 1995) (Paxson, 1999) (Rezaul, 2007). The vector triplet \( T_q = (P_q, \Phi_q, \Psi_q) \) describes fully the behaviour of
the traffic flow for $q$ packets and, for sufficiently large $q$, we simply refer to the traffic, $T$. This will provide a full model of traffic flow for any purpose.

$$
\begin{array}{c}
\begin{array}{c}
S_{a_k} \quad D_{a_k} \quad b_k \quad f_k \quad X_k \\
\end{array}
\end{array}
$$

Figure 3. A generic model of traffic (packet) flow

However, this completely generic model may be unnecessarily complex for specific purposes. In the case of packets being matched against ACLs, the matching occurs on (or shortly after) the arrival of each packet. The time between the arrival of packets $k$ and $k+1$ is given by $\omega_k = \phi_k + \psi_k$ and, if $\Omega_q = [\omega_1, \omega_2, ..., \omega_q]$ as before, then the pair, $T_q = (P_q, \Omega_q)$ or $T = (P, \Omega)$ adequately describes the traffic flow. Further simplifications are possible depending upon the nature of the ACL implementation and are discussed in the sections that follow.

5. Methods for Implementing ACLs

Space here permits only a brief overview of ACL implementation and optimisation. See Qian et al. (2001), Al-Shaer and Hamed (2004), Varghese (2005) and Grout et al. (2007a & 2007b) for a fuller treatment. There are essentially three basic approaches to implementation – although hybrids are also possible. Each permits a different form and level of simplification to the ACL/traffic, or rule/packet, model.

5.1. Implementation in TCAMs

A Content Addressable Memory (CAM) is effectively Random Access Memory (RAM) in reverse. Rather than accepting an address and returning the data at that location, a CAM can take an item of data and return the address at which it is to be found. In principle, the operation constitutes a single fetch operation. A Ternary CAM (TCAM) permits wildcard bit matches along with binary ones and zeroes and is consequently ideal for allowing matches within ranges of addresses, protocols, etc. of the form to be found within ACL rules. CAMs and TCAMs can be used for various forms of packet look-up including routing tables as well as ACLs. In a routing table, the longest matching entry is returned; in an ACL, the first. This is the fastest but most expensive form of implementation. Not only is the immensely complex circuitry potentially restrictive; even cooling requirements can be an issue on large platforms (McKeown, 2006).

TCAM implementation is also the most straightforward in terms of modelling. The size of the TCAM will be $n \times m$, where there are $n$ rules and $m$ is the maximum
length of each rule $i$ dependent upon the sizes of the parameters $SA_i$, $DA_i$, $B_i$, and $\sigma_i$ in the format for a given system. The time taken to process any packet $k$ is a constant, $C_{TCAM} (= \lambda(r_i)$ from section 3.3) and, provided this is less than $\omega_k = \phi_k + \psi_k$, there is no instantaneous potential for disruption. In general, for a packet stream of length $q$, provided

$$q C_{TCAM} \leq \sum_{k=1}^{q} \omega_k,$$  \hfill (7)

there will be no net latency. In the worst case, with an unbroken stream of packets, $\psi_k = 0 \Rightarrow \omega_k = \phi_k$, $\forall k$ and this becomes

$$q C_{TCAM} \leq \sum_{k=1}^{q} \phi_k.$$  \hfill (8)

5.2. Implementation as trees or tries

The concept of arranging ACL rules as a searchable tree structure (binary or otherwise) is a fairly obvious one. Assuming a binary tree arrangement, the first matching rule can be found in $O(2^n)$ steps, where $n$ is the length of the packet header in the given format and, although there are some mechanisms for improving this performance in special cases, there will also be non-trivial memory requirements as a result. On this basis, and assuming the worst-case scenario of the previous subsection, we require

$$2^n q C_{TREE} \leq \sum_{k=1}^{q} \phi_k,$$  \hfill (9)

(whether $C_{TREE} = 2^n \lambda(r_i)$) for there to be no net latency for $q$ packets.

However, in practice, rules are better organised as tries. A trie (from ‘retrieval’) is essentially a tree with an array of pointers at each node, indicating subtrees. There is a pointer at each node for each possible value. The bits of each rule are stored on the branches of the trie, not the nodes. Rule look-up can be performed much faster on tries than trees, in a time proportional to the number of header fields in fact. Again, there are storage requirements but this can be restricted to $O(n)$, for the general case, by special techniques involving synergies of hardware and software. The trie equivalent of equation (9) thus becomes

$$4 q C_{TRIE} \leq \sum_{k=1}^{q} \phi_k$$  \hfill (10)

($C_{TRIE} = \frac{1}{4} \lambda(r_i)$). See Varghese (2005) for a comprehensive description of trees and tries applied to packet look-up.

5.3. Implementation as linear lists

The simplest, but generally regarded as least efficient, approach to ACL implementation, is to process the rules sequentially as a linear list, precisely the original interpretation of rule order. Using the definitions of hit-rates and latencies
from section 3.3, define the cumulative latency, \( \kappa(r_i(L)) \), of \( r_i \) in a list \( L \), to be the
time taken to process \( r_i \) and all rules preceding it in \( L \). So

\[
\kappa(r_i(L)) = \sum_{i=1}^{j} \lambda(r_i(L)).
\]  \hspace{1cm} (11)

The expected latency, \( E(L,T) \), of a list \( L \), in traffic \( T \), is then given by

\[
E(L,T) = \sum_{i=1}^{n} h(r_i(L),T) \kappa(r_i(L)) = \sum_{i=1}^{n} h(r_i(L),T) \sum_{j=1}^{i} \lambda(r_j(L)).
\]  \hspace{1cm} (12)

In a simple sense, for \( q \) packets, we require that

\[
E(L,T) \leq \sum_{k=1}^{q} \phi_k.
\]  \hspace{1cm} (13)

as before, to avoid latency. However, the value of this approach is that rules
arranged as a linear list may be reordered to lower the value of \( E(L,T) \), provided
such a rearrangement does not violate any rule dependencies. In general, for a given
traffic flow, \( T \), we require to find (or approximate) the list, \( L \), implementing a policy,
\( Z \), that minimises \( E(L,T) \). Unfortunately, attempting to find such a minimising order
will, of course, itself have some processing cost. In fact it is shown by Grout et al.
(2007a) that the problem is \( NP \)-complete and only heuristics, not exact methods are
viable. However, even for this effort to be worthwhile, the potential reduction in
latency must be large enough to warrant running any optimising algorithm. It
transpires that this potential benefit can be examined effectively and accurately by a
further simplification to our traffic model as discussed in the next section.

6. Case Study: Optimising a Linear List with a Simple Algorithm

A number of heuristics for minimising expected latency in a sequentially executed
ACL are given by Grout et al. (2007a). The most efficient algorithm of all (\( \delta \)-opt) is
given (along with its full justification) in Grout et al. (2007b):

**Step 1:** Initialisation (on configuration/reconfiguration)
for \( i := 1 \) to \( n \) do
\( h(r_i) := 1 \)

**Step 2:** Promotion (on a match of rule \( r_i \))
\( h(r_i) := 8h(r_i) \);
if \( (d_{i-1} = 0) \) and \( h(r_i) \lambda(r_{i-1}) > h(r_{i-1}) \lambda(r_i) \) then
Swap\( (r_{i-1}, r_i) \)

**Step 3:** Reduction (every \( DSIZE \) packets)
for \( i := 1 \) to \( n \) do
\( h(r_i) := h(r_i) / \max_{j} h(r_j) \)

**Figure 4. The \( \delta \)-opt algorithm**

The process works by increasing the hit-rate of the currently matched rule (by a
factor, 8) and promoting it one place in the list if the trade-off in expected latency is
positive. (The full calculations are to be found in Grout et al. (2005)) All hit-rates
are assumed equal when the list is originally defined (or redefined) by the network administrator.

This is certainly a very simple and efficient algorithm. The linear \(O(n)\) Step 1 is executed only once, as the list is defined or redefined – an infrequent event. Step 3 (also \(O(n)\)) executes at intervals to prevent buffer overflow (\(\text{DSIZE}\) is the size, in bytes, of the registers holding hit-rates). Only the constant Step 2 executes for each packet. Even so, it is not immediately clear that the latency savings from running such an algorithm will justify its execution time. That this is actually so can be demonstrated through further simulation.

6.1. Packet and Traffic Models

On initial consideration, generating traffic and ACLs for testing appears complicated and difficult. Rules may differ considerably in some, fairly general ACLs, having very diverse address ranges, protocols, flags, etc. and combinations of the same; in other cases, where the role of an ACL is more focused, each rule may be only a slight variant of the others. Traffic also is difficult to predict and model in any generic sense: in principle, packets may be from anywhere, to anywhere, of any type and characteristics. Worse still, is the question of the relationship between traffic and an ACL. Different ACLs and their rules, depending on their purpose, may be expected to match packets with varying degrees of success. For example, the key rules in an ACL used to select traffic, within a local network, for address translation will probably match many packets – precisely those within the range to be translated; however, an ACL acting as a firewall – a safeguard - may have rules defining traffic types or address ranges that rarely pass through it. Generating addresses, protocols and flags for rules and packets to interact in any meaningful, realistic way will be difficult indeed.

Fortunately, simulation at this level is unnecessary for our purposes. A large number of the parameters from sections 3 and 4 can be combined into an essential form that describes the interaction between rules and packets without needing to precisely define their basic form. The key relationship between rule and packet lies not in the detail of addresses, protocols and flags but in the rudimentary issues of how long a rule takes to execute and how likely it is to be hit by a packet – taking into account the key fact that packets in similar streams are likely to match the same rule. Dependencies between rules must also not be overlooked since they prohibit arbitrary rule reordering.

Rule latencies (\(\lambda\)) and the dependency index (\(DI\)) have been defined already. All that is required for what follows is a second value, the similarity index (\(SI\)), describing the probability that any given packet matches the same rule as its predecessor. The advantage of this approach over attempting to generate ‘real’ rules and packets is that values of \(\lambda\), \(DI\) and \(SI\) can be generated comprehensively and exhaustively at will - allowing a complete set of results to be constructed for all types of ACL and traffic. Values of \(\lambda\), \(DI\) and \(SI\) for ‘real’ ACLs and traffic can be calculated simply enough and the corresponding simulation results applied for prediction, etc. The process proceeds as follows.
The simulation is based on an in-house numerical model, capable of generating ACLs and traffic flows according to a given parameter set. For tested ACLs, the number of rules \((n)\) ranged from 10 to 10,000. Values of the dependency index, \(D_I\), 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 \(D_I\) and \(d_{ij} = 0\) with probability \(1 - D_I\). Rule latencies were uniformly randomised from 0.5 \(\mu\)s to 1.0 \(\mu\)s. Actual values depend on the router hardware of course (Varghese, 2005) but it is only relative values that are significant. (Routers that process packets faster will also optimise faster.)

Table 1. Simulated Results: Rank and Cumulative Latencies.

<table>
<thead>
<tr>
<th>Packet flow</th>
<th>Number of Packets</th>
<th>Average</th>
<th>Average</th>
<th>Average</th>
<th>Average</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Rank*</td>
<td>Rank*</td>
<td>Rank*</td>
<td>Latency*</td>
<td>Latency*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(R^*)</td>
<td>(R^*)</td>
<td>(R^*)</td>
<td>(L^*)</td>
<td>(L^*)</td>
</tr>
<tr>
<td>(initial)</td>
<td>100000</td>
<td>485.26</td>
<td>485.26</td>
<td>485.26</td>
<td>366.69</td>
<td>366.69</td>
</tr>
<tr>
<td></td>
<td>200000</td>
<td>486.66</td>
<td>486.96</td>
<td>486.96</td>
<td>358.82</td>
<td>352.76</td>
</tr>
<tr>
<td></td>
<td>300000</td>
<td>471.56</td>
<td>450.49</td>
<td>450.49</td>
<td>315.14</td>
<td>340.22</td>
</tr>
<tr>
<td></td>
<td>400000</td>
<td>391.89</td>
<td>435.84</td>
<td>435.84</td>
<td>295.61</td>
<td>329.06</td>
</tr>
<tr>
<td></td>
<td>500000</td>
<td>372.26</td>
<td>423.12</td>
<td>423.12</td>
<td>280.83</td>
<td>319.42</td>
</tr>
<tr>
<td></td>
<td>600000</td>
<td>356.86</td>
<td>412.08</td>
<td>412.08</td>
<td>269.20</td>
<td>304.22</td>
</tr>
<tr>
<td></td>
<td>700000</td>
<td>349.02</td>
<td>403.07</td>
<td>403.07</td>
<td>263.29</td>
<td>301.15</td>
</tr>
<tr>
<td></td>
<td>800000</td>
<td>340.53</td>
<td>395.25</td>
<td>395.25</td>
<td>256.89</td>
<td>298.31</td>
</tr>
<tr>
<td></td>
<td>900000</td>
<td>338.29</td>
<td>388.92</td>
<td>388.92</td>
<td>255.16</td>
<td>293.51</td>
</tr>
<tr>
<td></td>
<td>1000000</td>
<td>333.14</td>
<td>383.35</td>
<td>383.35</td>
<td>251.33</td>
<td>289.30</td>
</tr>
<tr>
<td>(variation)</td>
<td>1100000</td>
<td>487.61</td>
<td>487.61</td>
<td>392.82</td>
<td>364.08</td>
<td>364.08</td>
</tr>
<tr>
<td></td>
<td>1200000</td>
<td>455.80</td>
<td>471.71</td>
<td>398.07</td>
<td>340.46</td>
<td>345.88</td>
</tr>
<tr>
<td></td>
<td>1300000</td>
<td>424.65</td>
<td>456.02</td>
<td>400.12</td>
<td>317.41</td>
<td>334.75</td>
</tr>
<tr>
<td></td>
<td>1400000</td>
<td>396.19</td>
<td>441.06</td>
<td>399.84</td>
<td>296.09</td>
<td>329.51</td>
</tr>
<tr>
<td></td>
<td>1500000</td>
<td>374.08</td>
<td>427.67</td>
<td>398.12</td>
<td>279.42</td>
<td>319.49</td>
</tr>
<tr>
<td></td>
<td>1600000</td>
<td>360.43</td>
<td>416.46</td>
<td>395.76</td>
<td>269.12</td>
<td>311.10</td>
</tr>
<tr>
<td></td>
<td>1700000</td>
<td>348.11</td>
<td>406.70</td>
<td>392.96</td>
<td>259.84</td>
<td>301.15</td>
</tr>
<tr>
<td></td>
<td>1800000</td>
<td>345.88</td>
<td>399.09</td>
<td>395.35</td>
<td>258.65</td>
<td>299.17</td>
</tr>
<tr>
<td></td>
<td>1900000</td>
<td>336.54</td>
<td>392.14</td>
<td>387.51</td>
<td>251.78</td>
<td>293.02</td>
</tr>
<tr>
<td></td>
<td>2000000</td>
<td>334.00</td>
<td>386.33</td>
<td>384.84</td>
<td>249.91</td>
<td>288.71</td>
</tr>
<tr>
<td>(variation)</td>
<td>2100000</td>
<td>480.18</td>
<td>480.18</td>
<td>389.38</td>
<td>358.17</td>
<td>358.17</td>
</tr>
<tr>
<td></td>
<td>2200000</td>
<td>480.18</td>
<td>480.18</td>
<td>389.38</td>
<td>358.17</td>
<td>358.17</td>
</tr>
<tr>
<td></td>
<td>2300000</td>
<td>480.18</td>
<td>480.18</td>
<td>389.38</td>
<td>358.17</td>
<td>358.17</td>
</tr>
<tr>
<td></td>
<td>2400000</td>
<td>480.18</td>
<td>480.18</td>
<td>389.38</td>
<td>358.17</td>
<td>358.17</td>
</tr>
<tr>
<td></td>
<td>2500000</td>
<td>480.18</td>
<td>480.18</td>
<td>389.38</td>
<td>358.17</td>
<td>358.17</td>
</tr>
<tr>
<td></td>
<td>2600000</td>
<td>480.18</td>
<td>480.18</td>
<td>389.38</td>
<td>358.17</td>
<td>358.17</td>
</tr>
<tr>
<td></td>
<td>2700000</td>
<td>480.18</td>
<td>480.18</td>
<td>389.38</td>
<td>358.17</td>
<td>358.17</td>
</tr>
<tr>
<td></td>
<td>2800000</td>
<td>480.18</td>
<td>480.18</td>
<td>389.38</td>
<td>358.17</td>
<td>358.17</td>
</tr>
<tr>
<td></td>
<td>2900000</td>
<td>480.18</td>
<td>480.18</td>
<td>389.38</td>
<td>358.17</td>
<td>358.17</td>
</tr>
<tr>
<td></td>
<td>3000000</td>
<td>480.18</td>
<td>480.18</td>
<td>389.38</td>
<td>358.17</td>
<td>358.17</td>
</tr>
<tr>
<td></td>
<td>3100000</td>
<td>480.18</td>
<td>480.18</td>
<td>389.38</td>
<td>358.17</td>
<td>358.17</td>
</tr>
<tr>
<td></td>
<td>3200000</td>
<td>480.18</td>
<td>480.18</td>
<td>389.38</td>
<td>358.17</td>
<td>358.17</td>
</tr>
<tr>
<td></td>
<td>3300000</td>
<td>480.18</td>
<td>480.18</td>
<td>389.38</td>
<td>358.17</td>
<td>358.17</td>
</tr>
<tr>
<td></td>
<td>3400000</td>
<td>480.18</td>
<td>480.18</td>
<td>389.38</td>
<td>358.17</td>
<td>358.17</td>
</tr>
<tr>
<td></td>
<td>3500000</td>
<td>480.18</td>
<td>480.18</td>
<td>389.38</td>
<td>358.17</td>
<td>358.17</td>
</tr>
<tr>
<td></td>
<td>3600000</td>
<td>480.18</td>
<td>480.18</td>
<td>389.38</td>
<td>358.17</td>
<td>358.17</td>
</tr>
<tr>
<td></td>
<td>3700000</td>
<td>480.18</td>
<td>480.18</td>
<td>389.38</td>
<td>358.17</td>
<td>358.17</td>
</tr>
<tr>
<td></td>
<td>3800000</td>
<td>480.18</td>
<td>480.18</td>
<td>389.38</td>
<td>358.17</td>
<td>358.17</td>
</tr>
<tr>
<td></td>
<td>3900000</td>
<td>480.18</td>
<td>480.18</td>
<td>389.38</td>
<td>358.17</td>
<td>358.17</td>
</tr>
<tr>
<td></td>
<td>4000000</td>
<td>480.18</td>
<td>480.18</td>
<td>389.38</td>
<td>358.17</td>
<td>358.17</td>
</tr>
</tbody>
</table>
For traffic, the simulation is only slightly 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 similarity index, SI, is set. Then a match probability, \( \rho \), is randomised for each rule \( r_i \) and normalised so that \( \sum_{i=1}^{n} \rho_i = 1 \).

The first packet is generated, matching rule \( r_i \) with probability \( \rho_i \). Subsequent packets match the same rule with probability SI, and otherwise match any rule according to the match probabilities, \( \rho_i \). Every \( q \) packets, the match probabilities, \( \rho_i \), are re-randomised.

6.2. Results

\( n \) and DI can be set to produce different types of ACL while \( q \) and SI vary to reflect different types of traffic. As an example, Table 1 records simulated output from a test with \( \theta = 1.5 \) (from \( \delta \)-opt), \( n = 1000 \), \( DI = 0.25 \), \( q = 1000000 \) and \( SI = 0.75 \). 4000000 packets are generated in total, in four stages with varying profiles. Results are reported every 100000 packets.

Figure 5. 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 \( (R^\wedge & L^\wedge) \) – the continuous average. The three latency averages, \( L^*, L'' \) and \( L^\wedge \), are plotted in Figure 5.
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, $500 \times (1.0 + 0.5) / 2 = 375$. 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 over time. In this example, $L^\wedge$ approaches a figure of approximately 287, an improvement of 23% on the non-optimised figure.

**Table 2. Simulated Results: Traffic Parameters and Promotion Coefficient.**

<table>
<thead>
<tr>
<th>$\text{ACL length (n): 1,000 rules. Stream length: 4,000,000 packets.}$</th>
<th>$\text{DI} - \text{Dependency Index. SI - Similarity Index.}$</th>
<th>$\text{Traffic (packet) characteristics change every q packets.}$</th>
<th>$\text{Table shows values of percentage improvement in expected latency}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>$100(L-L^\wedge)/L$ for different values of $DI$, $SI$, $q$ and $\theta$.</td>
</tr>
<tr>
<td></td>
<td>$DI = 0$</td>
<td>$0.25$</td>
<td>$0.5$</td>
</tr>
<tr>
<td>$SI = 0$</td>
<td>$\theta = 1.1$</td>
<td>15</td>
<td>14</td>
</tr>
<tr>
<td>$q = 10$</td>
<td>1.5</td>
<td>15</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>2.0</td>
<td>15</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>2.5</td>
<td>14</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>1.5</td>
<td>14</td>
<td>13</td>
</tr>
<tr>
<td>$SI = 0.25$</td>
<td>$\theta = 1.1$</td>
<td>17</td>
<td>15</td>
</tr>
<tr>
<td>$q = 1,000$</td>
<td>1.5</td>
<td>17</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>2.0</td>
<td>17</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>2.5</td>
<td>17</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>1.5</td>
<td>17</td>
<td>15</td>
</tr>
<tr>
<td>$SI = 0.5$</td>
<td>$\theta = 1.1$</td>
<td>19</td>
<td>17</td>
</tr>
<tr>
<td>$q = 50,000$</td>
<td>1.5</td>
<td>21</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>2.0</td>
<td>21</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>2.5</td>
<td>21</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>1.5</td>
<td>21</td>
<td>18</td>
</tr>
<tr>
<td>$SI = 0.75$</td>
<td>$\theta = 1.1$</td>
<td>19</td>
<td>17</td>
</tr>
<tr>
<td>$q = 1,000,000$</td>
<td>1.5</td>
<td>26</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>2.0</td>
<td>28</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>2.5</td>
<td>28</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>1.5</td>
<td>28</td>
<td>27</td>
</tr>
<tr>
<td>$SI = 1$</td>
<td>$\theta = 1.1$</td>
<td>20</td>
<td>19</td>
</tr>
<tr>
<td>no variation</td>
<td>1.5</td>
<td>27</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>2.0</td>
<td>30</td>
<td>29</td>
</tr>
<tr>
<td></td>
<td>2.5</td>
<td>30</td>
<td>29</td>
</tr>
<tr>
<td></td>
<td>1.5</td>
<td>30</td>
<td>29</td>
</tr>
</tbody>
</table>

Different parameters affect these values as shown in Table 2. 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 $SI$ imply greater traffic stability, which improves the optimised values. The effect of $\theta$ is more subtle. High values make rule promotion faster, which works well for similar, stable traffic but can lead to repetitive, unnecessary swaps for continuously changing, or oscillating, traffic patterns. A balance is necessary, with a
value around $\theta = 2$ appearing to maximise the improvement in expected latency in most cases.

### 6.3. Analysis

Routers vary considerably in their operation, particularly in terms of functional 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.)

For any given ACL manual configuration (or reconfiguration), Step 1 of the $\delta$-opt algorithm is executed once and can be taken as part of the configuration, Step 2, every processed packet, and Step 3, every $DSIZE$ 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 $2n / DSIZE$ and, in total, $12 + 2n / DSIZE$ for Steps 2 & 3 combined.

**Table 3. Optimisation Trade-Off – Saving against Cost**

<table>
<thead>
<tr>
<th>$DSIZE$</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n = 10$</td>
<td>-8.50</td>
<td>-7.25</td>
<td>-6.63</td>
<td>-6.31</td>
</tr>
<tr>
<td>30</td>
<td>-1.50</td>
<td>2.25</td>
<td>4.13</td>
<td>5.06</td>
</tr>
<tr>
<td>100</td>
<td>23.00</td>
<td>35.50</td>
<td>41.75</td>
<td>44.88</td>
</tr>
<tr>
<td>300</td>
<td>93.00</td>
<td>130.50</td>
<td>149.25</td>
<td>158.63</td>
</tr>
<tr>
<td>1 000</td>
<td>338.00</td>
<td>463.00</td>
<td>525.50</td>
<td>556.75</td>
</tr>
<tr>
<td>3 000</td>
<td>1038.00</td>
<td>1413.00</td>
<td>1600.50</td>
<td>1694.25</td>
</tr>
</tbody>
</table>

$n^*$ 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 $l + s = 4$ operations per rule and a percentage saving for an optimised list of $\xi$ gives an optimisation trade-off of

$$T = \frac{4\xi n}{100} - 12 - \frac{2n}{DSIZE},$$

which will be positive (i.e. worthwhile) when
For example, taking \( n = 1000 \) and \( DSIZE = 16 \), this gives \( 300 / 1000 + 50 / 16 = 3.425 \). Table 2 shows that the improvement, \( \xi \), exceeds this for all values other than \( DI = 1 \) and is therefore worthwhile. Alternatively, taking \( \theta = 2 \) and \( DI = SI = 0.5 \) gives an improvement of \( \xi = 15 \) and a trade-off of \( T = (15 \times 1000) / 25 - 12 - 2000 / 16 = 463 \), a positive benefit. Table 3 extends this calculation across a range of values of \( n \) and \( DSIZE \) and, for each \( DSIZE \), shows the key value of \( n^* \), the size of ACL for which optimisation is profitable. Table 4 fixes \( DSIZE \) at 16 and calculates \( n^* \) for various values of \( DI \) and \( SI \).

Table 4. Optimisation Trade-Off – Minimum ACL Length

<table>
<thead>
<tr>
<th>( DI )</th>
<th>0.0</th>
<th>0.25</th>
<th>0.5</th>
<th>0.75</th>
<th>1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>( SI ) = 0</td>
<td>25.26</td>
<td>27.59</td>
<td>30.37</td>
<td>43.64</td>
<td>( \infty )</td>
</tr>
<tr>
<td>0.25</td>
<td>21.62</td>
<td>25.26</td>
<td>27.59</td>
<td>38.09</td>
<td>( \infty )</td>
</tr>
<tr>
<td>0.5</td>
<td>16.78</td>
<td>20.17</td>
<td>25.26</td>
<td>33.80</td>
<td>( \infty )</td>
</tr>
<tr>
<td>0.75</td>
<td>12.06</td>
<td>12.57</td>
<td>17.78</td>
<td>27.59</td>
<td>( \infty )</td>
</tr>
<tr>
<td>1</td>
<td>11.16</td>
<td>11.59</td>
<td>15.89</td>
<td>23.30</td>
<td>( \infty )</td>
</tr>
</tbody>
</table>

6.4. Discussion

No amount of traffic modelling can substitute entirely for testing on production routers. However, these 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 2 & 4).

Table 2 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 \( DSIZE \), the size (number of bits) of the register being used to store them. (Step 3 is performed to prevent register overflow. 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 of \( \theta = 2 \) on each packet and, after \( a \) packets, will have a hit rate of \( 2^a \). The maximum safe number of packets between successive executions of Step 3 is then \( \log_2 2^{DSIZE} = DSIZE \).)
Depending on the stability and similarity of the traffic \((q\text{ and } SI)\) and the size of registers used to store hit-rates \((DSIZE)\), optimisation becomes worthwhile for ACLs above a certain length \(n^*\) (Tables 3 & 4). For realistic dependencies, this figure ranges between about 10 and 30. (Note also that this analysis assumes the worst-case scenario, from section 5, of packets arriving as an unbroken stream.) It is then trivial to separate those lists to which optimisation is to be applied from those to which it is not (Grout et al., 2006). Of course, it is precisely for longer ACLs that optimisation will yield the best results.

**Table 5. Real-World Examples**

For 11 real-world ACLs, the table shows the cases where \(\delta\)-opt is worthwhile (✓) or not (✗) for different levels of traffic similarity \((SI)\). \(\theta = 2\). \(DSIZE = 16\).

<table>
<thead>
<tr>
<th>ACL</th>
<th>n</th>
<th>DI</th>
<th>SI = 0.00</th>
<th>0.25</th>
<th>0.50</th>
<th>0.75</th>
<th>1.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>16</td>
<td>0.47</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>B</td>
<td>53</td>
<td>0.47</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>C</td>
<td>55</td>
<td>0.30</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>D</td>
<td>144</td>
<td>0.30</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>E</td>
<td>19</td>
<td>0.47</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>F</td>
<td>93</td>
<td>0.36</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>G</td>
<td>111</td>
<td>0.39</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>H</td>
<td>62</td>
<td>0.12</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>I</td>
<td>172</td>
<td>0.43</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>J</td>
<td>68</td>
<td>0.40</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>K</td>
<td>63</td>
<td>0.45</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 5 summarises the characteristics of several ACLs taken from a variety of production applications. (No attempt has been made to remove redundancies or 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 Small Office/Home Office (SOHO) environments connecting to the Internet via an ISP. ACLs I, J and K are derived from templates for various standard security configurations. \(\delta\)-opt is seen to be effective in the majority of real-world cases.

**7. Conclusions**

This case study justifies the use of simple heuristic optimisation \((\delta\text{-opt})\) applied to ACLs implemented as linear lists. It is shown that the savings in latency outweigh the cost of execution time in the majority of cases. Equally significantly, for any given ACL, operating within traffic with known characteristics, a simple calculation based on rule latencies, dependencies and stability can determine whether \(\delta\text{-opt}\) optimisation will be beneficial for that ACL. (If traffic stability cannot be determined, a worst-case scenario can be assumed.)

However, the paper, as a whole, also re-establishes a more well-known general principle. Although traffic and ACL modelling and simulation, in their most general form, may be complex and difficult, an analysis of the relevant parameters, in a
particular situation or application, may offer a level of simplification without losing accuracy of representation or the essential behaviour of the underlying system.

This notion is illustrated here by considering the essential traffic parameters that apply when analysing various implementations of ACLs. Both ACL rules and traffic packets have a fairly complex structure and the relationship between them is more complex still. However, each of the standard ACL implementations offers its own form of simplification to the general model and the case study, for sequentially processed lists, actually takes advantage of the relationship between lists and traffic by discarding those parameters not directly involved by it. The result is a streamlined, but still appropriate, model, capable of yielding efficient and powerful results.

References


