Traffic Modelling and Simulation Techniques for Evaluating ACL Implementation

V.Grout, J.N.Davies and J.McGinn

Centre for Applied Internet Research (CAIR)
University of Wales, NEWI, Wrexham, UK
e-mail: {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 know as *Access Control Lists* (or *ACL*s). 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 deny ip 192.168.212.0 0.0.0.255 10.0.0.0 0.255.255.255
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 0.0.255.255 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 210.120.183.0 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 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 102 permit ip 192.168.16.0 0.0.0.255 any
         :        :
         :        :
access-list 102 deny ip any 10.0.0.0 0.255.255.255
         :        :
         :        :
{access-list 102 deny all} {implicit}
```

Access list 2(b)

```
         :        :
         :        :
access-list 102 deny ip any 10.0.0.0 0.255.255.255
         :        :
         :        :
access-list 102 permit ip 192.168.16.0 0.0.0.255 any
         :        :
         :        :
{access-list 102 deny all} {implicit}
```

Figure 2: Dependent rules

3. Modelling 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'. 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 $\underline{F}^* = \{0, 1\}^w$ to be the set of *w flag vectors* ($\{0, 1\}$ w-tuples acting on B^*) valid for the system. For completeness, X^* represents the set of payloads.

3.1. Packets, rules and policies

For a given format, 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 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,

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

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, $t_n = deny$, $SA_n = A^*$, $DA_n = A^*$, $B_n = B^*$ and $\sigma_n \equiv true$.

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 ($(p_k \nabla r_i) \wedge (p_k \nabla r_j)$); that is r_i and r_j are *dependent* if

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

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:

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

where $\Sigma_i \subseteq \underline{F}^*$ is the subset of flag vectors satisfying σ_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:

$$DI = \frac{2}{n(n-1)} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} d_{ij} \tag{4}$$

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 $r_i \hookleftarrow r_j$) if there exists a rule, r_i ($i < j$), in Z, such that all packets matching r_j will be matched by r_i.

$$r_i \hookleftarrow r_j \Leftrightarrow (t_i = t_j) \wedge (SA_i \supseteq SA_j) \wedge (DA_i \supseteq DA_j) \wedge (B_i \supseteq B_j) \wedge (\Sigma_i \supseteq \Sigma_j). \qquad (5)$$

A redundant rule may be removed from the policy without changing its purpose.

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,

$$\begin{aligned} r_i \hookrightarrow r_j \Leftrightarrow (t_i = t_j) \wedge (SA_i \subseteq SA_j) \wedge (DA_i \subseteq DA_j) \wedge (B_i \subseteq B_j) \qquad (6) \\ \wedge (\Sigma_i \subseteq \Sigma_j) \wedge \forall v \rightarrow (i < v < j),\ d_{vj} = 0. \end{aligned}$$

Both forms of redundancy include the case, $r_i = r_j$.

Finally, and in brief, rules, r_α, r_β, r_γ .., 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_α, r_β, r_γ ... 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 φ_k to be the length of packet k (largely dependent upon the payload X_k) and ψ_k to be the time difference between the arrival (or passing) of packets k and $k+1$. ($\psi_0 = 0$ 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.

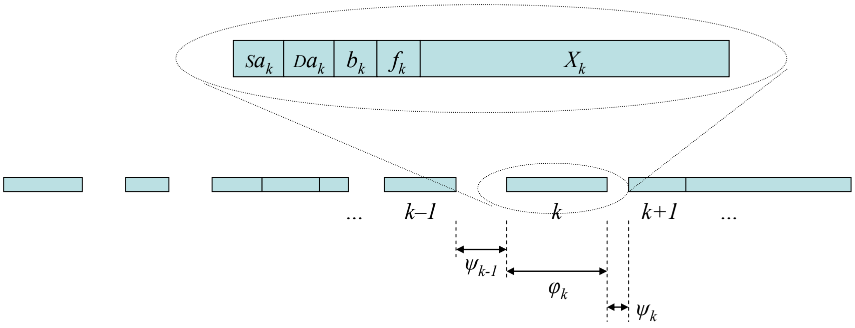

Figure 3: A generic model of traffic (packet) flow

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 $\omega_k = \varphi_k+\psi_k$ 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 and 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 σ_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 $\omega_k = \varphi_k + \psi_k$, there is no instantaneous potential for disruption. In general, for a packet stream of length q, provided

$$qC_{TCAM} \leq \sum_{k=1}^{q} \omega_k \text{ ,} \tag{7}$$

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

$$qC_{TCAM} \leq \sum_{k=1}^{q} \varphi_k \text{ .} \tag{8}$$

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 $O(2^\eta)$ steps, where η 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

$$2^n qC_{TREE} \leq \sum_{k=1}^{q} \varphi_k \tag{9}$$

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

However, in practice, rules are better organised as *tries*. A trie (from 're*trie*val') is essentially a tree with an array of pointers at each node, indicating *subtries*. There is a pointer at each node for each possible value. The bits of each rule are thus stored on the braches 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

$$4qC_{TRIE} \leq \sum_{k=1}^{q} \varphi_k \tag{10}$$

($C_{TRIE} = \frac{1}{4}\lambda(r_i)$). 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*, *κ(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

$$\kappa(r_i(L)) = \sum_{\upsilon=1}^{i} \lambda(r_\upsilon(L)) \cdot \tag{11}$$

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

$$E(L,T) = \sum_{i=1}^{n} h(r_i(L),T)\kappa(r_i(L)) = \sum_{i=1}^{n} h(r_i(L),T)\sum_{\upsilon=1}^{i} \lambda(r_\upsilon(L)) \cdot \tag{12}$$

In a simple sense, for *q* packets, we require that

$$E(L,T) \le \sum_{k=1}^{q} \varphi_k \ , \tag{13}$$

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 (*δ-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 $r_i$)
    $h(r_i)$ := $\theta h(r_i)$;
    if ($d_{i-1\ i}$ =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 *δ-opt* algorithm

The process works by increasing the hit-rate of the currently matched rule (by a factor, θ) 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 (`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 (λ) 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 λ, *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 λ, *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, *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 $d_{ij} = 0$ with probability *1 - DI*. 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.)

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*, ρ_i 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 ρ_i. Subsequent packets match the same rule with probability *SI*, and otherwise match any rule according to the match probabilities, ρ_i. Every *q* packets, the match probabilities, ρ_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 *δ-opt*), $n = 1\ 000$, $DI = 0.25$, $q = 1\ 000\ 000$ and $SI = 0.75$. *4 000 000* packets are generated in total, in four stages with varying profiles. Results are reported every 100 000 packets.

ACL length (*n*): 1 000 rules. Stream length: 4 000 000 packets. θ = 1.5.
3 changes in packet flow characteristics.
Dependency index (*DI* - probability of a dependency between any two rules): 0.25
Similarity index (*SI* - 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

Table 1: Simulated Results: Rank and 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 5.

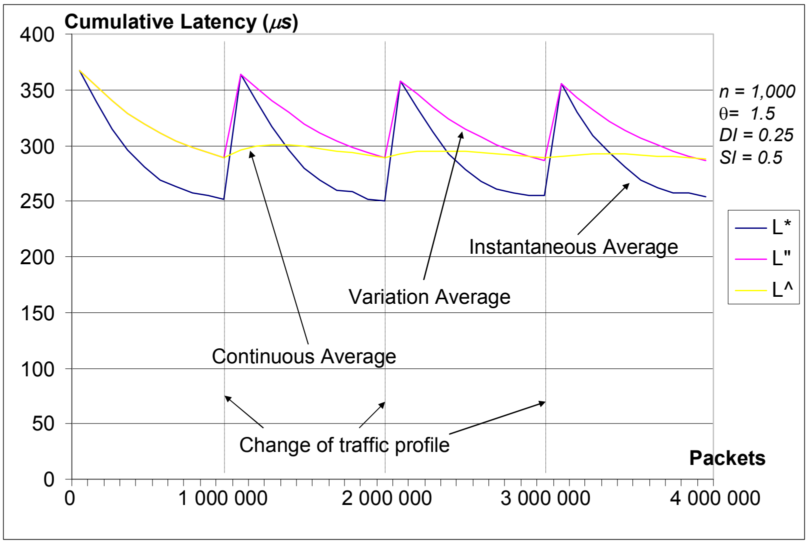

Figure 5: Simulated Results: Cumulative Latencies.

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^* approaches a figure of approximately *287*, an improvement of *23%* on the non-optimised figure.

ACL length (*n*): 1 000 rules. Stream length: 4 000 000 packets.
DI – Dependency Index. *SI* – 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*, *SI*, *q* and *θ*.

		DI =	*0*	*0.25*	*0.5*	*0.75*	*1*
SI = 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
SI = 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
SI = 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
SI = 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
SI = 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

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

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 θ 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 *δ-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.

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 ξ gives an optimisation *trade-off* of

$$T = \frac{4\xi n}{100} - 12 - \frac{2n}{DSIZE}, \tag{14}$$

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

$$\xi > \frac{300}{n} + \frac{50}{DSIZE}. \tag{15}$$

For example, taking $n = 1\,000$ and $DSIZE = 16$, this gives $300 / 1\,000 + 50 / 16 = 3.425$. Table 2 shows that the improvement, ξ, 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 $T = (15 \times 1\,000) / 25 – 12 – 2\,000 / 16 = 463$, 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.

DI = SI = 0.5. θ = 2.

Table shows value of trade-off function, *T = ξn/25 – 12 – 2n/DSIZE*, for different values of *n* and *DSIZE*.

	DSIZE =	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).

Table 3: Optimisation Trade-Off – Saving against Cost

θ = 2. DSIZE = 16

Table shows the value of *n**, the minimum length of list for *T = ξn/25 – 12 – n/8*, to be positive (i.e. for optimisation to be worthwhile) for different values of *DI* and *SI*.

	DI =	0.0	0.25	0.5	0.75	1.0
SI =	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	∞

Table 4: Optimisation Trade-Off – Minimum ACL Length

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 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.

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

ACL	*n*	*DI*	*SI* = 0.00	0.25	0.50	0.75	1.00
A	16	0.47	×	×	×	×	×
B	53	0.47	×	×	✓	✓	✓
C	55	0.30	✓	✓	✓	✓	✓
D	144	0.30	✓	✓	✓	✓	✓
E	19	0.47	×	×	×	×	×
F	93	0.36	✓	✓	✓	✓	✓
G	111	0.39	✓	✓	✓	✓	✓
H	62	0.12	✓	✓	✓	✓	✓
I	172	0.43	✓	✓	✓	✓	✓
J	68	0.40	✓	✓	✓	✓	✓
K	63	0.45	×	✓	✓	✓	✓

Table 5: Real-World Examples

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. *δ-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 (*δ-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 *δ-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.

8. References

Abilene IV Trace Data (2007), http://pma.nlanr.net/Special/ipls4.html, (accessed 20 December 2007).

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.

Cisco (2002) *ACL Optimizer and Hits Optimizer*, Cisco Systems, http://www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/cw2000/fam_prod/acl_mgr/aclm_1_x/1_5/u_guide/ac1js.pdf (accessed 20 January 2007).

Cisco (2003) *ACL Manager*, Cisco Systems, http://www.cisco.com/en/US/partner/products/sw/cscowork/ps402/products_user_guide_book09186a00801f42b9.html (accessed 20 January 2007).

Cisco (2007) *Packet Tracer Version 4.1*, http://netpd.ciscolearning.org/icg/pt, (accessed 20 January 2007).

cnet (2007) "The *cnet* Network Simulator", (University of Western Australia), http://www.csse.uwa.edu.au/cnet/ (accessed 20 December 2007).

Colton, A. (2002), *Cisco IOS for IP Routing*, Rocket Science Press Inc., 2002.

Grout, V., McGinn, J. and Davies, J. (2005), "Reducing Processing Latency in Network Traffic Filters", *Proceedings of the 5th International Network Conference* (*INC 2005*) Samos Island, Greece, 5th-7th July 2005, pp3-10.

Grout, V., McGinn, J., Davies, J., Picking, R. and Cunningham, S. (2006), "Rule Dependencies in Access Control Lists", *Proceedings of the IADIS International Conference WWW/Internet 2006* (*ICWI 2006*), Murcia, Spain, 5-8 October 2006, pp537-544.

Grout, V., McGinn, J. and Davies, J., (2007a) "Real-Time Optimisation of Access Control Lists for Efficient Internet Packet Filtering", *Journal of Heuristics*, Vol. 13, No. 5, October 2007, pp435-454

Grout, V., Davies, J. and McGinn, J., (2007b) "An Argument for Simple Embedded ACL Optimisation", *Computer Communications*, Vol. 30, No. 2, January 2007, pp280-287.

Jenkins, J.L. and Wang, J.L. (1999), "From the Network Measurement Collection to Traffic Performance Modeling: Challenges and Lessons Learned", *Journal of Brazilian Computer Society*, Vol. 5, No. 3, 1999.

Kos, A., Pustišek, M. and Bešter, J. (2003), "Characteristics of Real Packet Traffic Captured at Different Network Locations", *Proceedings of IEEE Region 8 EUROCON 2003*, Ljubljana, Slovenia, 22-24 September, Vol. 1, pp164-168.

Leland, W.E., Taqqu, M.S., Willinger, W. and Wilson, D.V. (1994), "On the Self-Similar Nature of Ethernet Traffic" (Extended Version), *IEEE/ACM Transactions on Networking*, Feburary 1994, pp1-15.

McKeown, N. (2006), "Internet Routers: Past, Present and Future", *British Computer Society (BCS) 2006 Lovelace Medal Lecture*, http://www.bcs.org/server.php?show=nav.7935 (accessed 20 December 2007).

ns-2 (2007), "The Network Simulator – ns-2", (accessed 20 December 2007) http://nsnam.isi.edu/nsnam/index.php/User_Information

Paxson, V. and Floyd. S. (1995), "Wide Area Traffic: The Failure of Poisson Modeling", *IEEE/ACM Transactions on Networking*, June 1995, pp236-244.

Paxson, V. (1999), "End-to-End Internet Packet Dynamics", *IEEE/ACM Transactions on Networking*, June 1999, Vol.7, No.3, pp277-292.

Qian, J., Hinrichs, S. and Nahrstedt, K. (2001), ACLA: A Framework for Access Control List (ACL) Analysis and Optimization, *Proceedings of the IFIP TC6/TC11 International Conference on Communications and Multimedia Security*, May 21-22, 2001, Darmstadt, Germany.

Rezaul, K.M. and Grout, V. (2007), "Exploring the Reliability and Robustness of HEAF(2) for Quantifying the Intensity of Long-Range Dependent Network Traffic", *International*

Journal of Computer Science and Network Security, Vol. 7, No. 2, February 2007, pp221-229.

Rezaul, K.M. (2007), *Estimating Long-range Dependent Self-similar Network Traffic: Performance Evaluation and Control*, PhD thesis, University of Wales, NEWI, October 2007.

Syngress (2002), *Building Cisco Remote Access Networks*, Syngress Media, 2002.

Tan, L., Zhang, W., Peng, G. and Chen, G. (2006), "Stability of TCP/RED Systems in AQM Routers", *IEEE Transactions on Automatic Control*, Vol. 51, No. 8, August 2006, pp1393-1398.

Trang Dang, D., Sandor, M. and Vidacs, A. (1999), "Investigation of Fractal properties in Data traffic", *Journal on Communications*, 1999, XLIX: pp12-18.

Varghese, G. (2005), *Networking Algorithmics: An Interdisciplinary Approach to Designing Fast Networking Devices*, Morgan Kaufmann, 2005.

Zhu, J. and Roy, S. (2004), "Improving Link Layer Performance on Sarellite Channels with Shadowing via Delayed Two-Copy Selective Repeat ARQ", *IEEE Journal on Selected Areas in Communications*, Vol. 22, No. 3, April 2004, pp472-481.