Compiling Policy Descriptions into Reconfigurable Firewall Processors

T.K. Lee, S. Yusuf, W. Luk, M. Sloman, E. Lupu and N. Dulay
Department of Computing, Imperial College,
180 Queen's Gate, London SW7 2BZ, England
(tkl97, sy99, w.luk, m.sloman, e.c.lupu, n.dulay)@doc.ic.ac.uk

Abstract

We describe a framework for capturing firewall requirements as high-level descriptions based on the policy specification language Ponder. The framework provides abstraction from hardware implementation while allowing performance control through constraints. Our hardware compilation strategy for such descriptions involves a rule reduction step to produce a hardware firewall rule representation. Three main methods have also been developed for resource optimisation: partitioning, elimination, and sharing. A case study involving five sets of filter rules indicates that it is possible to reduce 67-80% of hardware resources over techniques based on regular content-addressable memory, and 24-63% over methods based on irregular content-addressable memory.

1 Introduction

A common element of a firewall architecture [14] is an Internet Protocol (IP) packet filter to implement authorisation policies [6]. A packet filter works by checking the content of the IP packet header before deciding if communication is allowed, based on a set of rules. The syntax of the rules [4, 15] is firewall specific. The ordering of the rules within a rule set is significant. A packet is sequentially checked against each rule, starting from the beginning of a rule set, until a match for the conditions specified in a rule is found or the end of the rule set is reached.

Packet filters [1, 9] usually rely on processors running entirely in software. They suffer from increased look-up times as the number of filter rules grows. They therefore have difficulty in keeping up with the current network throughput. With the recent advances in field-programmable gate array (FPGA) technology, custom-developed hardware packet filters [7, 11, 13, 16] that out-perform their software counter parts become possible. However, limitations on the amount of available reconfigurable resources may restrict the number of concurrent matches. Some studies [7, 11, 16] have been conducted to optimise the usage of hardware resources, however, they often do not take into account the redundancy among the firewall rules in a rule set, and have not utilized information other than those offered by the IP packet headers.

Firewall rules are notoriously difficult to maintain. There are several attempts to use high-level languages [1, 4] or graphical user interface [5] for their description. However, with the rapid expansion of the internet and the growing demand of large-scale organisational networks, rule sets comprising 1000 rules are not uncommon. Consequently, the need for appropriate high-level languages for firewall description becomes increasingly important.

Ponder [6] is a language for specifying security and management policies for distributed object systems. Policies can be written as parameterised types, and can have constraints.

We describe a framework to specify high-level firewall rules using Ponder, and to implement such descriptions on reconfigurable hardware. The contributions described in this paper include:

- a method for capturing authorisation policies in a high-level description;
- a rule reduction technique that converts high-level firewall description to hardware firewall rule representation, through partitioning, elimination and sharing;
- a compilation scheme for the framework which involves the rule reduction technique; and
- an evaluation of the effectiveness of the proposed framework based on a number of case studies.

The rest of the paper is organised as follows. Section 2 gives an overview of our design framework. Section 3 discusses the design decision for our high-level firewall description. Section 4 explains our rule reduction technique. Section 5 describes the implementation scheme. Section 6 outlines a compilation scheme for the design framework. Section 7 evaluates our approach through some case studies, while Section 8 provides a summary of current and future work.

Proceedings of the 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'03)
1082-3409/03 $17.00 © 2003 IEEE

2 Framework overview

This section gives an overview of our framework. It outlines the design objectives, and briefly describes each stage in the design and development flow.

Our framework allows us to specify high-level firewall rules and to implement such descriptions on reconfigurable hardware. There are three basic design objectives:

1. To provide a method to simplify the design process and to facilitate the maintenance of a firewall. In particular, to aid the management of authorisation policies for a complex large-scale organisational network; and to express firewall rules for reconfigurable hardware implementation.

2. To separate a design into software and hardware phases; and allowing optimisation to be performed in both phases, for various hardware implementation schemes.

3. To achieve efficient hardware utilization. Emphasis is on overcoming the physical limitations on the size of reconfigurable hardware, by methods such as sharing of hardware functional units and parameterised library blocks.

To achieve these objectives, we have three main phases in the design flow: design phase, compilation phase and hardware implementation phase. Figure 1 shows an overview of our framework for developing reconfigurable-hardware packet filtering firewalls.

At the design phase, the formal requirement of a firewall will be given and additional information that can assist the optimisation of the firewall implementation will be provided. The requirements for a firewall is usually contained in an authorisation policy, which is then transformed into a high-level firewall description. Such a description consists of two parts: a firewall control specification, and the domain hierarchies of the IP addresses and services. Optional information including network topology and the available services together with the firewall description form the input to the next stage of the design flow. At the compilation phase, the firewall description will be converted to a hardware firewall rule representation, which in turn will go through a series of optimisation steps. The result is an optimised representation of a list of hardware firewall rules ready to produce a hardware design in the next stage of the design flow. At the hardware implementation phase, the representation of firewall rules will be converted to a hardware design for specific hardware devices. Hardware specific optimisation techniques can also be used to further optimise the firewall rules and the overall design. Device specific tools are then used to place and route the design

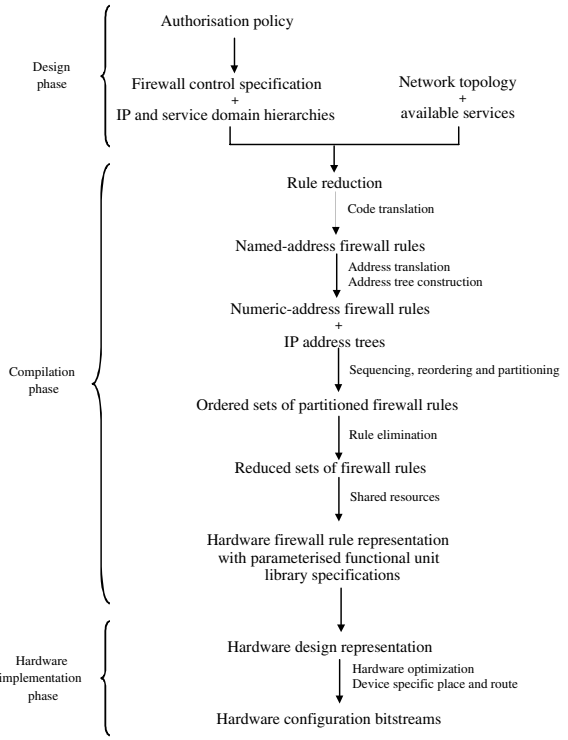


Figure 1: An overview of the design flow in our framework.

and to generate the necessary hardware configuration bitstreams for downloading the design on to hardware.

Our framework employs a two-level optimisation approach. This involves the use of hardware firewall rule representation in the compilation phase, and the use of hardware design representation in the hardware implementation phase. There are two advantages of having an intermediate representation. First, it allows both software and hardware optimisations to be performed, based on different sets of criteria and information available. In particular, it permits using platform-specific optimisations as well as platform-independent techniques. Second, it enables a choice of different hardware implementation schemes based on size, speed, cost or other requirements.

3 High-level firewall description

This section discusses a novel method that we develop to capture an authorisation policy in a high-level firewall description.

To achieve our first design objective (Section 2) of sim-

plified design process, we specify an authorisation policy in a high-level language. We believe such a language should at least have the following properties: simplify the design process, facilitate the maintenance, and allow easy design re-use. In particular, it should

- support abstraction from the hardware implementation, so that changes to the policies will give a minimal or a controllable impact on the hardware; and
- allow the policy administrator, who may have little knowledge of hardware, to specify performance requirements in a high-level description.

Rather than creating a new language, we come up with a high-level firewall rule that uses a subset of the Ponder Authorisation Policies syntax and adopt domain hierarchies [6]. We also provide a compilation scheme to convert such descriptions to a hardware firewall rule representation.

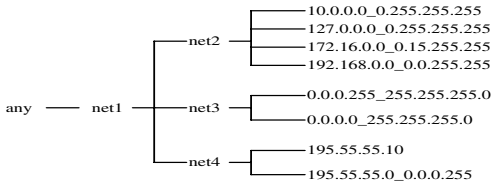
To allow a high level of abstraction and to facilitate maintenance, we separate the control requirement of a firewall rule, with the IP address and the port address, from the conventional syntax of firewall rules. Consequently, our high-level firewall description consists of two parts: a firewall control specification, and the domain hierarchies of the IP addresses and services. In addition we interpret not only a particular address, but also treat address ranges and address masks as objects. Furthermore, both the addresses and ports are included in the domain hierarchy.

It is this design decision that enables our high-level firewall description to become simple to use and easy to maintain. Our method is more compact and human readable than using the conventional syntax of router-based firewall rules. Changes to the control requirement and the objects specified are now independent to each other. Examples of the two domain hierarchies can be found in Figure 2.

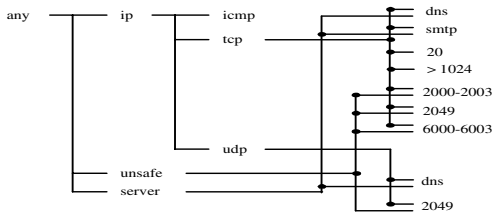
We define policy types, as shown in Figure 3, to specify the *PERMIT* and *DENY* requirements for a firewall. The control requirement can then be linked with the specified objects by instantiating the appropriate policy types with the corresponding domain hierarchies.

Constraints are added as an additional control in the firewall control specification. They can be used for grouping firewall control specification as a hardware partition, determining run-time reconfiguration or hardware software cooperation, or providing hints and criteria for introducing delays, timing requirement, placement requirement, and size requirement for hardware implementation. Figure 4 shows some possible uses of the constraint.

Figure 5 shows an example of a high-level firewall description using our specific approach for describing firewalls. Our firewall description is more abstract and can usually result in more compact description; it permits easy design re-use. The seven lines example shown in Figure 5



(a) An IP address domain hierarchy, which captures the network topology into the domain path structure. The object '10.0.0.0_0.255.255.255' represents the IP address '10.0.0.0' with a mask '0.255.255.255'. Depends on the operator applied, 'net2' can represent all address objects under its branches; and similarly 'any' can represent all objects beneath it, or a 'don't care' condition.



(b) A service domain hierarchy. Any set of services can be grouped together and named. Refer to (a) for interpreting a domain hierarchy.

Figure 2: Examples of IP address and service domain hierarchies.

would have to be described by 16 lines in Cisco firewall rules. If the network topology is more complex or involves more addresses, the differences will be huge. For example, if *net2* in *Figure 2(a)* has 100 extra addresses, then no changes are required for our high-level firewall control specification, but one would have to write another 100 more firewall rules that correspond to these 100 extra addresses.

4 Rule reduction mechanism

This section explains how the high-level firewall description produced in the previous section is converted to a hardware firewall rule representation. In addition, it describes our new technique for saving hardware resources.

The number of firewall rules that can be simultaneously put into hardware is limited by the configurable resources available. To achieve our second and third design objectives in Section 2, we come up with a hardware firewall rule representation, which is largely implementation independent.

Our two-level optimisation approach allows software optimisation and hardware optimisation [7, 11, 13, 16] techniques to be used simultaneously. Hardware optimi-

```
type auth+ Permit (subject SrcIP, domain SrcService,
                   target DstIP, domain DstService,
                   string UserDefConstraint) {
   action   TCPconnect, UDP;
   when     UserDefConstraint;
}
type auth- Deny (subject SrcIP, domain SrcService,
                 target DstIP, domain DstService,
                 string UserDefConstraint) {
   action   TCPconnect, UDP;
   when     UserDefConstraint;
}
```

Figure 3: Policy types in our framework. *Permit* allows a TCP-connect and UDP packet to pass through if all specified conditions are fulfilled. Similarly, *Deny* does not allow the specified action when the conditions are satisfied.

```
inst auth+ Permit(/any, /any, /any/net, /any/critical, "AlwaysHW");

(a) A constraint that restricts firewall rules to
    hardware implementation; assuming that there is a
    hardware-software partition.


inst auth+ Permit(/any/net1, /any, /any/net1, /any/, "Partition=1");
inst auth+ Permit(/any/net2, /any, /any/net2, /any/, "Partition=1");
inst auth+ Permit(/any/net1, /any, /any/net2, /any/, "Partition=2");
inst auth+ Permit(/any/net2, /any, /any/net1, /any/, "Partition=2");

(b) Constraints that restrict intra-network
    communications to take place on partition 1; while
    inter-network communications can take place on
    partition 2.  The reasons behind this restriction
    may be due to run-time reconfiguration, so that a
    hardware block will be reconfigured in the
    appropriate region; or to restrict the size or
    timing requirement of a partition.


inst auth- Deny(/any, /any, /any/main, /any/game, "Time<17:00");

(c) A constraint that restricts the main servers from
    running game services before the time '17:00'.
    This specification may be implemented as a run-time
    reconfigurable hardware partition.
```

Figure 4: Example uses of constraints in firewall specification.

sations focus on reducing the size for each firewall rule, while software optimisations focus on reducing the number of firewall rules needed. In addition, software optimisation can sometimes enhance the applicability of hardware optimisation.

The rule reduction mechanism consists of five steps described below.

4.1 Code translation

This step involves conversion of high-level firewall description to low-level firewall rule representation. The aim is to generate a representation that can be manipulated for various optimisations before implementing on hardware.

The code translation process employs standard compilation techniques: it has a parsing and a code generation phase. However, the order of the statements listed in a specification does not guarantee the order that they are processed. This is in conflict to the strict ordering requirement

```
inst auth- Deny(/any/net1/net2, /any/ip, /any, /any/ip);
inst auth- Deny(/any, /any/ip, /any/net1/net3, /any/ip);
inst auth- Deny(/any/net1/net4/195.55.55.0_0.0.0.255,/any/ip,/any,/any/ip);
inst auth+ Permit(/any, /any, /any/net1/net4/195.55.55.10, /any/server);
inst auth- Deny(/any, /any, /any, /any/unsafe);
inst auth+ Permit(/any, /any/ip/tcp/20, /any, /any/ip/tcp/>1024);
inst auth+ Permit(/any, /any/icmp, /any, /any/icmp);
```

Figure 5: An example high-level firewall description capturing an authorisation policy for checking incoming packets using the domain hierarchies in *Figure 2*. The first statement instantiates to meet the requirement for denying packets having source IP address *net2*, any destination IP address, any source or destination port address, and with packet type *ip*.

Table 1: An example of expanded firewall rules generated from the specification in *Figure 5* by code translation.

Type	Source IP address	Source port	Destination IP address	Destination port	Action
ip	127.0.0.0/0.255.255.255	*	*	*	deny
ip	10.0.0.0/0.255.255.255	*	*	*	deny
ip	172.16.0.0/0.15.255.255	*	*	*	deny
ip	192.168.0.0/0.0.255.255	*	*	*	deny
ip	*	*	0.0.0.255/255.255.255.0	*	deny
ip	*	*	0.0.0.0/255.255.255.0	*	deny
ip	195.55.55.0/0.0.0.255	*	*	*	deny
tcp	*	*	195.55.55.10	smtp	permit
tcp	*	*	195.55.55.10	dns	permit
udp	*	*	195.55.55.10	dns	permit
tcp	*	*	*	6000-6003	deny
tcp	*	*	*	2000-2003	deny
tcp	*	*	*	2049	deny
udp	*	*	*	2049	deny
tcp	*	20	*	>1024	permit
icmp	*	*	*	*	permit

of firewall rules. To rectify this, we provide an extra preparsing step which is explained in Section 6. Table 1 tabulates the results of converting the example of high-level firewall description in Figure 5.

4.2 Address translation and address tree construction

This step contains two components, name conversion and address construction, which are performed in sequence. There are two aims: first, to produce a representation of firewall rules consisting only numerical values; and second, to generate the critical information for the optimisations in the later steps.

During the first stage, all named identifiers are replaced by their corresponding numeric values. At the second stage, two tree structures are constructed for all the IP addresses contained in the list of firewall rules.

4.3 Sequencing, reordering and partitioning

This step involves locating sequence points and assigning partition boundaries within a rule set. There are two aims: first, to provide a hint for the optimisations in the later steps to avoid violating the original authorisation poli-

cies that a rule set is represented; second, to tailor the number of firewall rules to fit on hardware. This may be due to the reasons such as physical size limit, timing requirement, and run-time reconfiguration.

A sequence point is where changes to the ordering of a rule within a rule set will affect the meaning of the policies being represented. Interchanging the order of any two rules is allowed if and only if there is no sequence point between them. Figure 6 shows an example where a sequence point occurs, as well as an example where there is none.

Partitioning divides a rule set into multiple smaller groups of rules. The size of each group can be specified according to some predefined conditions, such as:

- the amount of available hardware resources for implementing firewall rules;
- the critical path and timing requirement for the resulting hardware circuitries; and
- the use of run-time reconfiguration.

The property of allowing the ordering of rules within a group to be freely interchanged can be useful during hardware implementation:

- first, it enables multiple rule matching to be performed in parallel in hardware without the need for extra circuitry to check or serialize the results;
- second, rearranging the rules and re-partitioning can sometimes change the size of a partition as desired.

The sequencing process takes both the results generated in the previous step as the inputs, which are the list of firewall rules and the IP address trees. It then traverses the IP address trees and looks for conflicting rules. A sequence point is located whenever a conflict is found. In that case, a mark will be put in the list of firewall rules in between the two rules involved to indicate that there is a sequence point. The process continues until both trees are exhausted.

4.4 Rule elimination

This step removes unnecessary firewall rules. The aim is to reduce the total number of firewall rules to be implemented on hardware.

Three types of elimination are performed: i) conflicts due to rules which both allow and deny packets, ii) redundant rule which is a subset of another rule, iii) rules that can not be reached due to rule ordering.

4.5 Rule sharing

This step checks for similarity among the rules and then groups them together if close matches are found. The aim is to share the hardware functional unit among the rules.

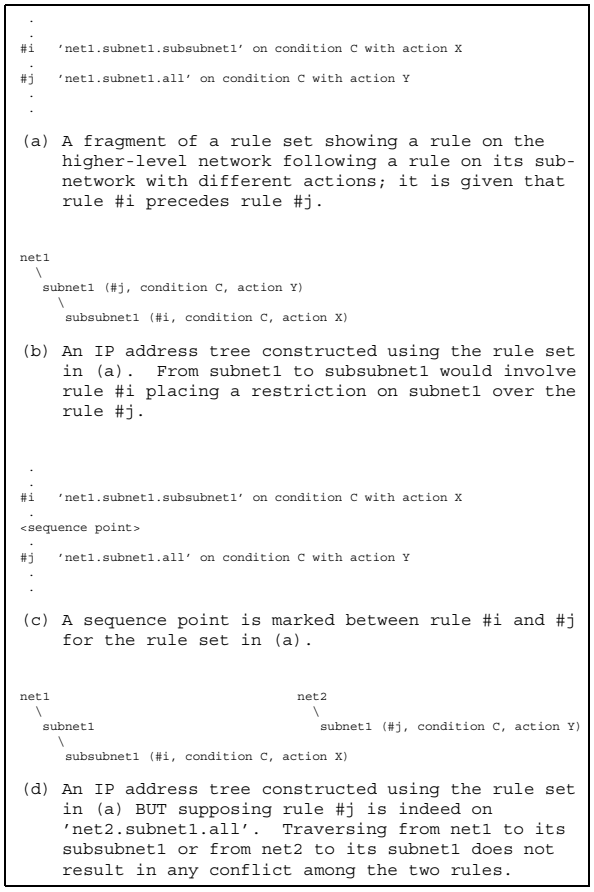


Figure 6: An example of locating and marking sequence point.

There are two types of sharing: field level and bit level.

Field-level sharing

Firewall rules have a number of data fields. Rules having identical values in corresponding data fields are grouped together. In this case, hardware functional units including parameterized variable-bit comparators can be shared. There is no limitation on how many data fields can have different values, within a group of rules. However, the greater the number of fields that can be different, the fewer the number of fields that can share the corresponding hardware among the rules. In addition, the complexity of the pattern matching process for finding the similarity among the rules grows by allowing a higher value for the number of different data field values existed among a group of rules. Figure 7(a) shows an example of two rules that have identical data values in all corresponding fields but the *Type* field. There are two methods of grouping, and their effects and requirement in hardware implementation are different.

Type	Source IP address	Source port	Destination IP address	Destination port	Action
tcp	*	*	195.55.55.10	dns	permit
udp	*	*	195.55.55.10	dns	permit

(a) A fragment of a rule set with two rules which differ in the Type field.

Type	Source IP address	Source port	Destination IP address	Destination port	Action
tcp|udp	*	*	195.55.55.10	dns	permit

(b) Method 1: The rule set in (a) is grouped together to form a single rule, where the Type field becomes a list of alternative values. Hardware comparators except the Type field are now shared.

Type	Source IP address	Source port	Destination IP address	Destination port	Action
tcp	*	*	195.55.55.10	dns	permit
udp	+	+	+	+	+

(c) Method 2: The rule set in (a) is grouped together to share the hardware comparators. A '+' mark in the fields indicates that the values of a particular field of a rule is identical to the corresponding field in the rule precedes it.

Figure 7: An example of field-level sharing. The commonality in the data fields among the rules are exploited. Corresponding data fields having identical values among the rules will share the corresponding hardware.

- *Method 1* uses a list of alternative values to represent data fields that are different, and a single value to represent data fields that are identical among the rules. A number of rules are grouped together to form a single rule. An example is shown in Figure 7(b).
- *Method 2* keeps the same number of rules in a group. However, a mark is used to indicate that a particular field in a rule is having an identical value to the corresponding field in other rules among the group. An example is shown in Figure 7(c).

Bit-level sharing

Bit-wise numeric operation can be used to deduce the redundancy between two or more numeric values of IP or port addresses. This method looks for commonality in bit level, regardless of the numeric values that the data are represented. It matches the binary '1's and '0's at bit-level among the corresponding fields in the rules. The results are two set of match values. One set contains a mask and the bits of the data field that are identical among the rules. The other set contains a mask and a list of bits of the data field that are different among the rules. Two examples can be found in Figure 8.

The sharing process takes the list of firewall rules as input and searches for close matching for the corresponding data fields among the rules. The criteria for the pattern matching process include the number of fields that can have different values among the rules and the number of rules that can be grouped together in a match.

Type	Source IP address	Source port	Destination IP address	Destination port	Action
ip	172.16.0.0/0.15.255.255	*	*	*	deny
ip	192.168.0.0/0.0.255.255	*	*	*	deny

(a) A fragment of a rule set.

```
172.16.0.0/0.15.255.255 = 1010 1100 0001 XXXX XXXX XXXX XXXX XXXX
192.168.0.0/0.0.255.255 = 1100 0000 1010 1000 XXXX XXXX XXXX XXXX

Mask for identical bits = 1001 0011 0100 0000 XXXX XXXX XXXX XXXX
Identical bit           = 1..0 ..00 .0.. .... .... .... .... ....

Mask for different bits = 0110 1100 1011 1111 0000 0000 0000 0000
Difference bit (1)      = .01. 11.. 0.01 XXXX .... .... .... ....
Difference bit (2)      = .10. 00.. 1.10 1000 .... .... .... ....
```

(b) Bit-wise deduction for the rule set in (a).

Type	Source IP address	Source port	Destination IP address	Destination port	Action
tcp	*	*	62.189.241.2	www	permit
tcp	*	*	62.189.241.4	www	permit
tcp	*	*	62.189.241.3	www	permit
tcp	*	*	62.189.241.1	www	permit

(c) A fragment of a rule set.

```
62.189.241.2            = 0011 1110 1011 1101 1111 0001 0000 0010
62.189.241.4            = 0011 1110 1011 1101 1111 0001 0000 0100
62.189.241.3            = 0011 1110 1011 1101 1111 0001 0000 0011
62.189.241.1            = 0011 1110 1011 1101 1111 0001 0000 0001

Mask for identical bits = 1111 1111 1111 1111 1111 1111 1111 1000
Identical bit           = 0011 1110 1011 1101 1111 0001 0000 0...

Mask for different bits = 0000 0000 0000 0000 0000 0000 0000 0111
Difference bit (1)      = .... .... .... .... .... .... .... .010
Difference bit (2)      = .... .... .... .... .... .... .... .100
Difference bit (3)      = .... .... .... .... .... .... .... .011
Difference bit (4)      = .... .... .... .... .... .... .... .001
                          -----------------------------------
                        = .... .... .... .... .... .... .... .XXX
```

(d) Bit-wise deduction for the rule set in (c).

Figure 8: Examples of bit-level sharing. The commonality of the '1's and '0's for a data field among the rules is deduced.

Grouping of rules separated by sequence points can lead to violation to the original authorisation policy that a rule set is represented. In order to avoid such a situation, a rule set is divided into segments where each segment begins and ends with a sequence point (see Section 4.3). The matching process is performed independently on each segment. Reordering can be performed among the rules within the same segment in order to facilitate a grouping.

Technique for implementing the two methods of field-level rule sharing is explained in Section 5.

5 Implementing hardware sharing

This section describes how resource sharing can be achieved on hardware. In particular, it explains the requirements and proposes an implementation scheme for the two different methods of hardware firewall rule representation generated by the rule reduction mechanism.

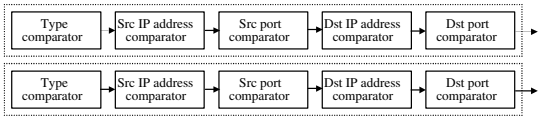
A hardware firewall rule is usually implemented as a set of comparators as in Figure 9(a). Each comparator corresponds to one of the data fields in a firewall rule. Whether the comparators are physically separated or cascaded together is implementation dependent. However, in terms of hardware-resource consumptions, they are basically the same.

Figure 9 shows the differences between a set of rules with and without sharing the resources on hardware. The sharing process does not change the hardware implementation of just a single firewall rule. Indeed, the representation of a group of shared hardware firewall rules is implemented as an overlapping cluster. To achieve a higher saving of hardware resources, achieve a larger overlapping area. This in turn is determined by the number of rules in the sharing group, and the fields that are shared among the rules.

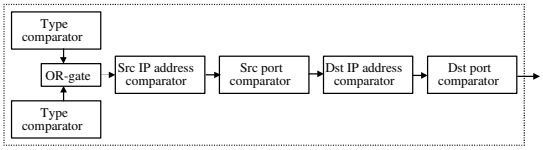
In the following, we provide implementations for the two field-level rule sharing methods described in Section 4.5:

- Method 1: *Siamese Twins*
  Individual fields of the firewall rules having identical data values are simply shared by using the same hardware functional units. Fields that can not be shared have their corresponding parts OR-ed together. An example of which is shown in Figure 9(b). Advantages of this method include simple design, and large savings in hardware designs. However, it produces irregular hardware circuitries that exhibit differences in size and timing behaviour. It poses difficulties when implementing pipelining and regular hardware data structures such as content-addressable memory. On the other hand, it is suitable for implementations that involve irregular hardware data structures. Figure 10 shows the implementation schemes for Siamese Twins.

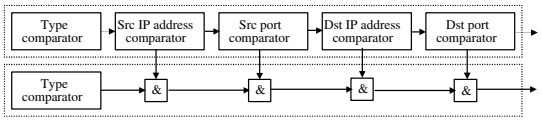
- Method 2: *Propaganda*
  Individual fields of the firewall rules having identical data values are sharing the corresponding hardware functional units through the use of extra 2-input AND gates. It is this relatively little extra cost that compensate for the removal of the much larger cost of the corresponding hardware comparators of the data fields. An example of which is shown in Figure 9(c). The advantage of this method is the regular design and simple to implement. Hardware techniques usually favour regular designs, and hardware cores that are available commercially or in public domains tend to use regular data structures. Therefore, this method is suitable for adoption by current designs with little modifications. Figure 11 shows the implementation schemes for Propaganda.



(a) Two firewall rules without sharing any resources on hardware.



(b) Siamese Twins: two firewall rules sharing the functional units on hardware. In this example, all but the type comparator are shared through the use of an OR-gate.

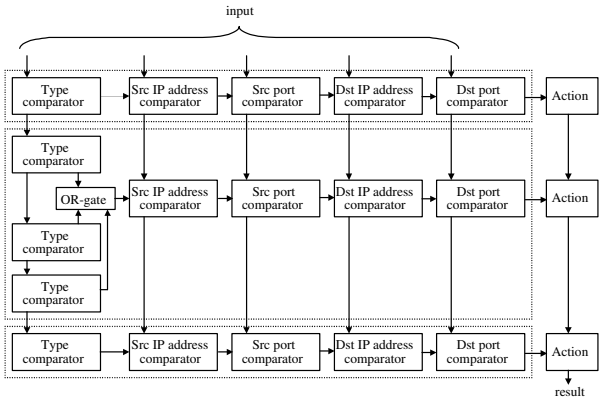


(c) Propaganda: two firewall rules sharing the functional units on hardware. In this example, all but the type comparator are shared through the use of 2-input AND gates.

Figure 9: An example showing the differences of firewall rules with and without sharing the resources on hardware.

Notice that a hardware functional unit is used in a hardware firewall rule only when the corresponding field contains a normal data value. A 'don't care' condition is normally implemented as a by-passing wire and does not require any logic gates at all.

Implementation using our rule reduction technique, and in particular the rule sharing method, requires less hardware for a set of firewall rules. A reduction in hardware consumption with a smaller hardware circuitry will, in general, have less routing and gate delays. Therefore, it is reasonable to expect that designs incorporating rule sharing will have shorter critical paths or at least can meet the same timing requirement as those without using it. In other words, it can maintain the speed performance, if not better.



A pipelined structure showing a group of shared hardware firewall rules with other non-shared rules. Results are obtained from the last stage of the pipeline. Parallel structure will have similar layout, except without the pipeline stages and results are obtained in parallel.

Figure 10: Implementation scheme for Siamese Twins.

6 Compilation scheme

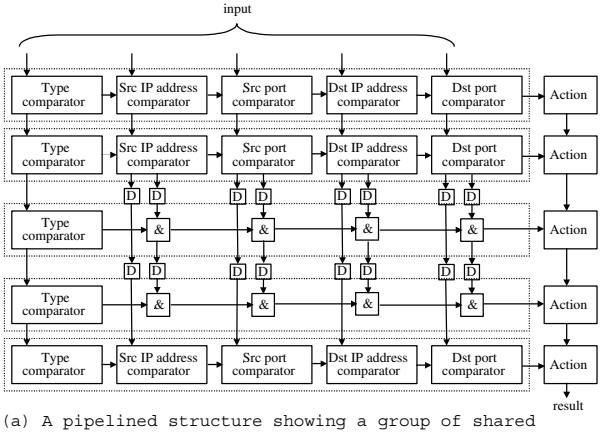
This section outlines a compilation scheme for the framework. In particular, it discusses the issues involved in each stages during the conversion of a high-level firewall description to the desired hardware configuration bitstream. Lastly, we describe how our existing tools are related to the framework.

Figure 12 shows an overview of the compilation steps. There are three stages in the compilation flow:

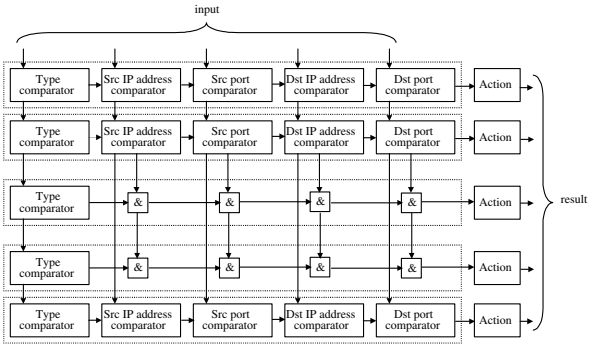
- Parsing
  We use the Ponder Toolkit as the Ponder Parser. However, specifications have no ordering in Ponder and this is in contrast with the firewall rules which require strict ordering. Therefore, we provide a pre-parsing step that put a tag to every statements in the firewall description. This tag is simply a number that gets incremented by one each time after a statement from a firewall description is read. This pre-parsing step allows the ordering to be preserved both during and after the Ponder Parser.

- Code generation
  There are two phases at this stage and they correspond to each of the levels in our two-level optimisation approach. We design the code generator which converts the firewall description (see example in Figure 5) to the hardware firewall rule representation (see example in Table 1) at Phase 1. Various choices of hardware design code can be generated at Phase 2.

- Hardware implementation
  Platform and device specific environments as well as place and route tools are used to convert a hardware design into the corresponding hardware configuration bitstreams.



(a) A pipelined structure showing a group of shared hardware firewall rules with other non-shared rules. Results are obtained from the last stage of the pipeline. Notice that the pipeline registers may increase the size of the design.



(b) A parallel structure showing a group of shared hardware firewall rules with other non-shared rules. Results are obtained in parallel.

Figure 11: Implementation schemes for Propaganda.

Our existing development tools can automatically generate hardware designs from representation of firewall rules to either Handel-C or VHDL code. However, they are not yet capable of producing irregular structures as required by our new rule reduction methods. Furthermore, due to the complexity of generating irregular structures to a granularity level as required by the bit-level sharing method, we have no plan to convert the current tools to support this bit-level optimisation. Instead, we decide to modify the tools to support coarse-grain reduction optimisation down to data field level. On the other hand, we plan to support bit-level optimisation through the development of a new tool using other hardware optimisation technique.

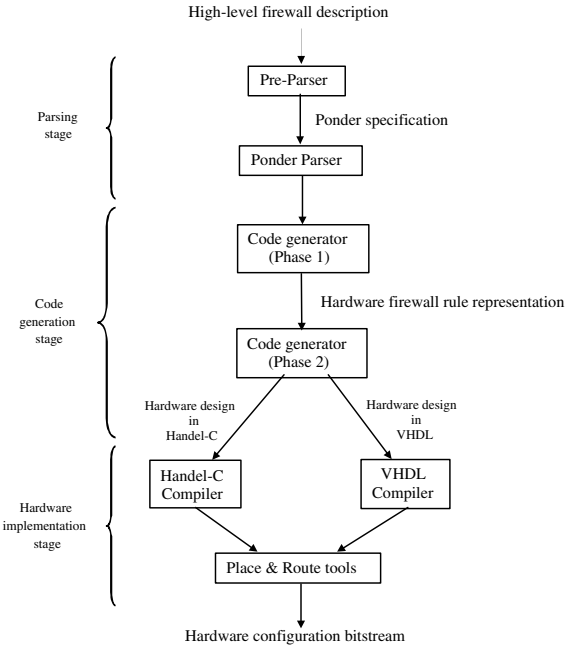


Figure 12: An overview of the compilation steps.

7 Case studies

This section reports some findings on using our new approach for producing packet-filtering firewall on reconfigurable hardware.

We compare the new approach with two other hardware implementation techniques. The reductions in hardware usage are estimated by calculating the area usage of five filter rule sets, on a Virtex XCV1000 FPGA. These rule sets compose of a mixture of both incoming and outgoing traffic controls on general network services, mail and WWW servers. We do not include I/O and related control circuits in the hardware usage estimates. Figure 13 shows the result of the hardware usage of a regular content-addressable memory (CAM) structure [11], an irregular CAM structure [7], and the rule reduction optimisation.

We calculate the amount of hardware resources saved based on the number of look-up tables required. For the new approach using the rule reduction mechanism, we employ our previous hardware architecture [12, 13] for parallel matching structures that does not include any hardware-level of space optimisation. We use only optimisation granularity at data field level.

The estimated result shows that our new approach using the rule reduction optimisation can reduce the hardware usage by 67-80% from the regular CAM implementation,

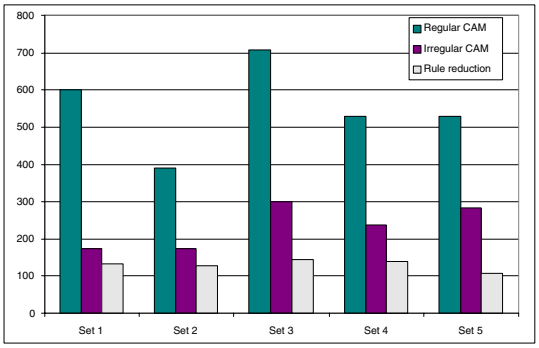


Figure 13: Hardware usage, in terms of number of look-up tables, of five filter rule sets using three different implementation techniques.

and 24-63% from the irregular CAM implementation. It is believed that further improvement can be achieved by exercising hardware-level optimisation such as serialisation.

Although it is possible to produce an irregular pipeline structure, our current tools allocate regular rectangular blocks on hardware for the filter-rule matching. In other words, even though we would have already modified our tools to support this new technique, the hardware space saved would be scattered around as small unused blocks, which are difficult to reuse for additional filter rules.

We also compare how changes to an authorisation policy will reflect on the high-level firewall description approach with the conventional router-based firewall rule method. We estimate the impact by introducing changes on the network topology, and control requirement on IP address and service. The firewall description approach responses to changes by requiring modification on the corresponding domain hierarchies and/or the control specification statement. On the other hand, the conventional rule set responds to changes by requiring the corresponding original firewall rule(s) from an unstructured list being picked up, and then either being modified or replaced by new rules.

It is generally easy to locate an object from a hierarchy than spotting it from an unstructured list. Moreover, changes on a single specification statement can affect all corresponding objects; while changing a list of unrelated rules requires all relevant rules to be changed at the same time.

However, the effect of a change of the firewall description may not immediately affect the hardware implementation. It is because any changes on the control specification statements or the domain hierarchies will affect the overall data redundancy that can be deduced in the resulting rule set. Therefore, the impact may not be directly proportional to the degree of changes introduced. A big change can have

no effect at all, if it was being picked up and eliminated by the rule reduction step. A small change can have profound effect, if it introduces new partitioning and grouping in the hardware firewall rule representation.

On the other hand, the conventional rule set method 'mirrors' the pattern on hardware. A change on one firewall rule affects only the circuitry on the pre-defined corresponding part on the hardware.

8 Summary

We have presented a design flow for developing hardware packet filters. It employs a two-level optimisation approach that allows software and hardware optimisations to proceed independently.

We have described a method of capturing an authorisation policy in a high-level firewall description. It is based on a policy specification language using domain hierarchies; and supports constraint that specifies restrictions for hardware implementation.

We have explained a hardware optimisation technique and have outlined a compilation scheme for the design framework. A case study shows that hardware reduction of 67-80% and 24-63% is possible, over regular and irregular content-addressable memory implementations respectively.

Current and future work includes using constraints to facilitate run-time reconfiguration [10] and hardware software co-operation [3]. Exploration of various hardware-level optimisation techniques, such as methods based on binary decision diagram [16] and content-addressable memory [7], is under investigation: the former is capable of producing a compact representation of filter rules, while the latter is capable of fast database search on irregular structures. The extension of our framework to cover implementations on reconfigurable platforms [2] and other applications such as network intrusion detection [8] are also of interest.

Acknowledgements. The support of UK Engineering and Physical Sciences Research Council (Grant number GR/R 31409, GR/R 55931 and GR/N 66599), Celoxica Limited and Xilinx, Inc. is gratefully acknowledged.

References

[1] A. Begel, S. McCanne, S.L. Graham, "BPF+: Exploiting Global Data-flow Optimisation in a Generalized Packet Filter Architecture", in *Proc. SIGCOMM*, Computer Communication Review, 29(4), 1999, pp. 123-134.

[2] P. Bellows et. al., "GRIP: A Reconfigurable Architecture for Host-Based Gigabit-Rate Packet Processing", in *Proc. IEEE Symp. on Field-Programmable Custom Computing Machines*, IEEE Computer Society Press, 2002.

[3] G. Brebner, "Single-chip Gigabit Mixed-version IP Router on Virtex-II Pro", in *Proc. IEEE Symp. on Field-Programmable Custom Computing Machines*, IEEE Computer Society Press, 2002.

[4] Cisco Systems Inc., *Cisco PIX Firewall Command Reference*, http://www.cisco.com/.

[5] Cisco Systems Inc., *Cisco Secure Policy Manager Policy Configuration Guide*, 2001. http://www.cisco.com/.

[6] N. Damianou, N. Dulay, E. Lupu and M Sloman, "The Ponder Policy Specification Language", *in Proc. Workshop on Policies for Distributed Systems and Networks*, LNCS 1995, Springer, 2001, pp. 18-39.

[7] J. Ditmar, K. Torkelsson and A. Jantsch, "A Dynamically Reconfigurable FPGA-based Content Addressable Memory for Internet Protocol Characterization", *Field Programmable Logic and Applications*, LNCS 1896, Springer, 2000.

[8] R. Franklin, D. Carver and B.L. Hutchings, "Assisting Network Intrusion Detection with Reconfigurable Hardware", in *Proc. IEEE Symp. on Field-Programmable Custom Computing Machines*, IEEE Computer Society Press, 2002.

[9] P. Gupta and N. McKeown, "Packet Classification on Multiple Fields", in *Proc. SIGCOMM*, Computer Communication Review, 29(4), 1999, pp 147-160.

[10] J.R. Hess et. al., "Implementation and Evaluation of a Prototype Reconfigurable Router", in *Proc. IEEE Symp. on Field-Programmable Custom Computing Machines*, IEEE Computer Society Press, 1999.

[11] P.B. James-Roxby and D.J. Downs, "An Efficient Content-addressable Memory Implementation Using Dynamic Routing", in *Proc. IEEE Symp. on Field-Programmable Custom Computing Machines*, IEEE Computer Society Press, 2001.

[12] T.K. Lee, S. Yusuf, W. Luk, M. Sloman, E. Lupu and N. Dulay, "Development Framework for Firewall Processors", in *Proc. IEEE International Conference on Field-Programmable Technology*, 2002.

[13] W. Luk, S. Yusuf and R. Nagarajan, "Incremental Development of Hardware Packet Filters", in *Proc. International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, CSREA Press, 2001, pp. 115-118.

[14] J.T. McHenry, P.W. Dowd, "An FPGA-Based Coprocessor for ATM Firewalls" in *Proc. IEEE Symp. on Field-Programmable Custom Computing Machines*, IEEE Computer Society Press, 1997.

[15] R. Russel, *Linux IPCHAINS-HOWTO*, http://www.linuxdoc.org/HOWTO/IPCHAINSHOWTO.html.

[16] R. Sinnappan and S. Hazelhurst, "A Reconfigurable Approach to Packet Filtering", *Field Programmable Logic and Applications*, LNCS 2147, Springer, 2001.