RZ 3210 (# 93256) 03/06/00
Computer Science/Mathematics 15 pages

Research Report

Prefix-based Parallel Packet Classification

A.P.J. Engbersen, J. van Lunteren

IBM Research
Zurich Research Laboratory
8803 Rüschlikon
Switzerland

Prefix-based Parallel Packet Classification

A.P.J. Engbersen, J. van Lunteren

IBM Research, Zurich Research Laboratory, 8803 Rüschlikon, Switzerland

Abstract

This paper presents a novel scheme for parallel packet classification. The basic idea behind the scheme, which differentiates it from other parallel classification schemes, is that the results of parallel range searches on different packet header segments embed classification rule information in the form of prefixes. The overall classification result can then be determined by applying a conventional longest matching prefix search on these intermediate search results, which are interleaved in a specific order. This provides two key advantages over other methods. First, the storage requirements can be reduced significantly, which is an essential prerequisite for scaling beyond a couple of thousand rules. Second, the assignment of intermediate result values becomes more flexible, which makes it possible to perform dynamic updates for at least a selected subset of the classification rules.

1 Introduction

Significant efforts are being undertaken to transform the current Internet, in which all packets receive the same best-effort service, into a network in which new services such as Integrated Services and Differentiated Services dedicate available network resources differently over packet streams in order to provide certain transmission and delivery guarantees. To enable these types of services routers must be able to perform multi-field packet classification function at wire-speed.

Packet classification is the problem of searching among multiple rules for the one with the highest priority that matches a packet header. A rule is said to match a packet header if all the conditions specified by that rule are met by the actual values of fields in the given packet header. Rules conditions are typically expressed as exact match, prefix match and range match operators on IP source and destination addresses, TCP source and destination port numbers, protocol type and other fields.

Packet forwarding for traditional best-effort service is based on searching a routing table for the longest matching prefix of one field, the IP destination address. This problem is considered to be solved, as can be seen from the large number of publications in the past few years. In contrast, only a few methods for packet classification have been published, which have in common that they are limited to a few thousand classification rules and do not support fast dynamic updates. Packet classification is a much harder problem than conventional routing table search, due to the much larger part of the packet header that forms the input to the classification operation. The fields upon which the classification rules are specified can cover more than one hundred bits, resulting in an enormous input value space. As a consequence, one of the main problems in realizing a packet classification scheme that supports several thousand rules at wire-speed is the storage required to construct a data structure that allows a fast search to be performed on the entire input value space to determine the highest priority rule that matches. In addition, complex relations that occur within this data structure due to overlapping conditions of multiple rules make it difficult to create an efficient data structure that can be updated incrementally.

As it is currently not clear what kind of characteristics the classification rules are expected to have even in the near future, a general packet classification scheme must meet wire-speed performance for worst-case input and rule conditions in order not to risk being outdated by future internet developments.

This paper presents a new scheme for parallel packet classification. Parallel packet classification involves multiple parts of the packet header to be processed in parallel, which can be used to exploit parallelism available in custom hardware implementations in order to obtain high classification performance. This in contrast to classification schemes such as [1] that process various parts of the packet header in a sequential way. The paper is organized in the following way. Section 2 describes the basic elements of a parallel packet classification scheme. Section 3 discusses previous work on parallel classification schemes. Section 4 introduces the novel classification scheme, which is the main contribution of this paper. Section 5 discusses the storage and performance aspects of the scheme. The final Section 6 summarizes the paper.

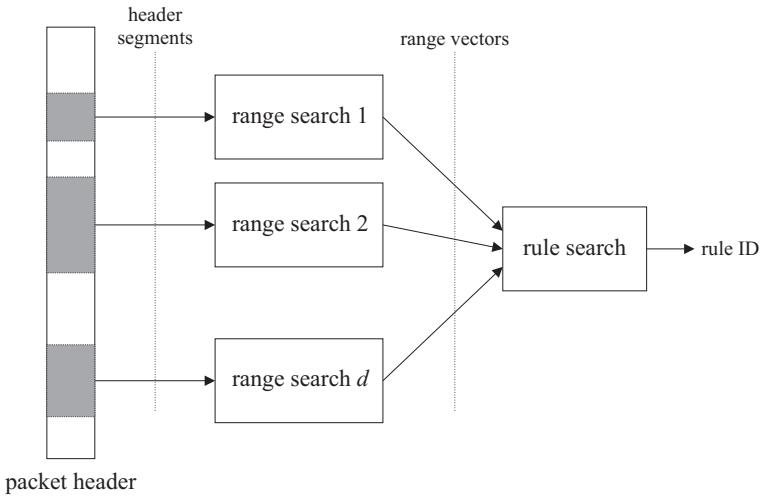

Figure 1: d-way parallel packet classification.

2 Concept of Parallel Packet Classification

Figure 1 illustrates the concept of d-way parallel packet classification in which d distinct segments of the packet header are processed in parallel. Each segment can consist of any not necessarily contiguous part of the packet header, which may cover multiple packet header fields entirely or partially. The concept of parallel packet classification discussed here is partially based on [2] and [3].

Rules are typically specified as operators such as exact match, prefix match and range match, applied on various packet header fields. By 'projecting' the rule specification on (the fields covered by) a header segment, it can be determined for which segment values the rule certainly will not apply, and for which values of the header segment the rule might apply (which is also dependent on the values of the other header segments). The values for which a rule might apply are denoted as a *rule range*, which is not necessarily a contiguous range.

In Figure 1 a range search is performed on each header segment. The result of a range search operation is called a *range vector*. Based on the d range vectors, it can be determined within which rule ranges the header segment values fall and consequently, the highest priority rule that is applicable to the packet header. This process is called *rule search*. The output of the rule search is a rule identification (ID), which is the packet classification result.

3 Previous Work

Two papers on parallel packet classification schemes have been published, a classification scheme based on bit-parallelism by Lakshman and Stiliadis [2] and a scheme called recursive flow classification (RFC) by Gupta and McKeown [3]. Both schemes employ parallel range searches on multiple distinct header segments as shown in Figure 1. However, none of the schemes specifies the number and location of the segments. As the RFC scheme applies range searches based on table indexing for which the table size grows quadratically with the index size, the segments used by this scheme are likely to be smaller than those of the bit-parallelism scheme, which applies binary search operations based on integer comparators and counters. Both schemes derive for each segment a set of non-overlapping ranges from the rule ranges for

that segment.

The two schemes are in a sense extremes with regard to the rule information that is embedded in the range vectors that result from the parallel range searches, and the way in which this information is processed to determine the classification result. The range vectors of the bit-parallelism scheme include bit flags for all rules, which are ordered according to the rule priorities. Each range vector indicates for all rules whether the segment value is located within the corresponding rule range. As all necessary information is provided by the range vectors, a simple logical AND operation can be used to determine the applicable rule with the highest priority.

The range vectors used by the RFC scheme provide no information regarding the rule ranges in which a segment value is located nor their priorities. Each range vector only consists of a unique range identification. As a result a more complex rule search operation is required, which involves a large amount of precomputed information regarding the highest priority rules that apply for each possible combination of d range identifications. If there are more than two header segments, the rule search operation can be performed in multiple steps, which allows the storage requirement to be minimized using certain rule characteristics.

Storage Requirements

The maximum number of ranges that have to be tested by each range search operation is determined by the boundaries of the rule ranges and has a maximum value of $2n + 1$. This results in a worst-case total number of range vectors equal to $(2n + 1)d$ that have to be stored for d-way parallel classification. The range vectors of the bit-parallelism scheme consist of n bits to support n rules. The range vectors of the RFC scheme, which only provide unique range identifiers within one dimension, consist of $\lceil \log(k) \rceil$ bits for a dimension with k ranges. Consequently the worst-case storage requirements for storing all range vectors scales according to n^2 with the number of rules for the bit-parallelism scheme and according to $n \log(n)$ for the RFC scheme. The latter worst-case storage requirements can also be obtained for the bit-parallelism scheme by storing the range vectors in a different way, however, this will decrease the classification performance [2]. The actual storage required for the range searches depends on the algorithm and corresponding data structure that are used. In order to perform a 'fair' comparison, only the storage of the range vectors is considered here.

The RFC scheme needs to store additional information for the rule search. Since this information basically consists of the classification result for each possible combination of d range vectors resulting from d range searches, the storage required for the rule search could in the worst-case scale according to n^d with the number of rules. However, the RFC scheme involves special heuristics to reduce these storage requirements by exploiting common characteristics observed in existing rule databases. For this reason, it is very difficult to estimate the storage requirements of the RFC scheme for general rule databases.

Classification Performance

Since both schemes could apply the same algorithms to perform the parallel range searches, the difference in classification performance will be related to the rule search operation. The rule search operation of the bit-parallelism scheme consists of a logical AND-operation of d range vectors of n bits. The main factor determining the classification performance is the memory bandwidth, which dictates the speed at which the range vector bits can be read from

memory. Therefore, to support several thousands of rules at wire-speed, the bit-parallelism scheme requires fast memories with wide data paths (e.g., embedded memories). For the same memory technology, the classification performance is limited by the number of rules. By exploiting pipelining, the RFC scheme provides a classification performance that is only dependent on the cycle time of the memories that are used. This allows a very high number of classification operations per second, independent of the number of rules.

Update Performance

Both schemes can only handle updates at a low frequency. For the bit-parallelism scheme this is caused by the obligation to reflect the rule priorities in the bit flags within the range vectors, which might require modification of all range vectors (in the worst-case $2(n+1)d$ vectors) for each rule update. For the RFC scheme, this is caused by the complex data structure used for the rule search, which requires significant precomputation time in order to minimize the storage requirements. Consequently, both schemes should only be applied in environments in which rules have a static nature and no fast dynamic rule updates have to be performed.

4 Prefix-based Classification Scheme

4.1 Concept

This paper presents a novel classification scheme called prefix-based classification. Prefix-based classification involves the same type of parallel range searches on header segments as with the bit-parallelism scheme and the RFC scheme. It is different regarding the range vectors and the corresponding rule search operation.

Prefix-based classification considers ranges to be *related* when they are subsets of at least one rule range. These range relations are expressed within the range vectors in the form of prefixes. This is in contrast to the bit-parallelism scheme in which the range vectors contain rule range information related to individual ranges. The concept of prefix-based classification is illustrated in Figure 2. This figure shows an example of a two-way parallel classification operation using a set of four rules, which are represented as rectangles in a two-dimensional space (Figure 2 shows a representation similar to Figure 2 in [2] but with different rules and range vectors). The two axes correspond to the value spaces of the two header segments that are processed in parallel. For simpler illustration and understanding, all four rules involve contiguous rule ranges in both dimensions. The boundaries of all the rule ranges in both dimensions are projected on the corresponding axes. As a result, two sets of non-overlapping ranges are derived, which are labeled X0 to X8 and Y0 to Y6. These are the ranges that are searched for by the parallel range searches. These ranges are derived in exactly the same way as with the bit-parallelism scheme and the RFC scheme.

Ranges X4 to X7 and ranges Y2 to Y5 are located within the rule ranges of rule 2 in the respective dimensions. These relations are now embedded in the corresponding range vectors in the form of a common prefix '01' and '1', respectively. In a similar way, the range vectors of ranges X5 and X6, and Y2 and Y3 that are related due to rule 4 have a common prefix equal to '011' and '101', respectively. As Figure 2 shows, not all relations can be expressed using prefixes. For example, rule 1 applies to ranges X2 to X5. As the range vectors of ranges X4 and X5 already have a prefix for the relation due to rule 2, the relation due to rule 1 is

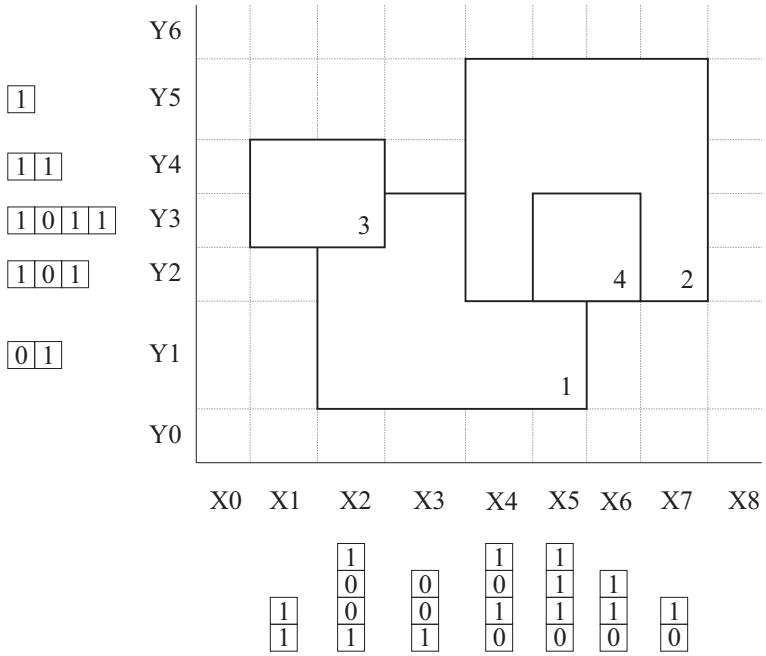

Figure 2: Prefix-based classification.

only coded for the remaining two ranges X2 and X3 by a prefix '100'. The generation of these range vector prefixes will be discussed in Section 4.2.

The range vectors shown in Figure 2 have another special property that will be explained below. Owing to this property, the classification result can be determined by applying a conventional longest matching prefix search operation on the bit-interleaved result of the range vectors that result from the parallel range searches in the X and Y dimensions (differences in range vector sizes are resolved by padding zeros). To each rule corresponds at least one so called *rule prefix*. If a certain rule prefix is found to be the longest matching prefix, then an identification of the corresponding rule is output as the classification result. The rule prefix for rule 2 equals '011' and the rule prefix of rule 4 equals '011011' for the example of Figure 2 (Section 4.3 discusses how these rule prefixes are derived). If the values of the two header segments correspond to a point within the area determined by, for example, range X6 and Y3, then the range searches will result in the range vectors '011' and '1011'. Bit-interleaving these two vectors after padding a zero to the first range vector results in '01101101'. Both rule prefixes of rule 2 and rule 4 are prefixes of this vector. The longest of the two matching rule prefixes (intentionally) corresponds to rule 4, which has the highest priority. This shows a second property of prefix-based classification, namely that when multiple rule prefixes match a range-vector-interleaving-product, then the longest matching rule prefix corresponds to the rule with the highest priority.

There are some variations on the rule search operation mentioned here. Figure 3 will be used to explain the rationale behind these search types. In this figure two range vectors are shown for both the X and Y dimensions that are the result of the parallel range searches on the two header segments. The two range vectors have prefixes '01' and '1', indicating that both segment values are located in the rule range of rule 2 as discussed before. In order to determine that rule 2 applies for these segment values, the first two bits of the X dimension range vector must have been processed and found to be equal to '01' and the first bit of the

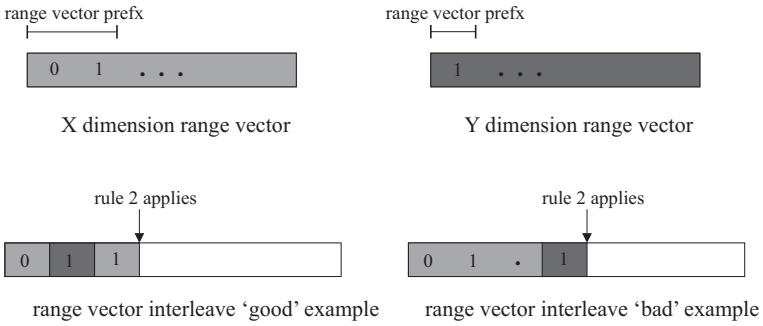

Figure 3: Interleaved processing of range vectors.

Y range vector must have been processed and found to be equal to '1'. If the rule search operation is realized as a binary search tree with a search key that is obtained by interleaving the range vectors in a certain fixed order, then the indication that rule 2 is applicable has to be stored at exactly one node in the tree if the range vectors are interleaved such that the first three bits of the interleave result include the first two bits of the X range vector and the first bit of the Y range vector. Figure 3 shows an example of a good interleaving for which this condition holds, and an example of a bad interleaving for which it does not hold. For a search key based on the interleave order of the bad example, the binary search tree has to store an indication that rule 2 is applicable at two nodes, which are reached by search keys starting with '0101' and '0111', respectively, unless the additional X range vector bit, shown as a dot in the bad interleaving example in Figure 3, can only have one value.

The prefix-based classification scheme includes the following three types of rule searches that employ 'good' range vector interleaving as explained above:

- type A: Prefix-based range vectors are applied in only one dimension. The range vectors of the other dimensions consist of fixed-size range identifications. The rule search consists of a conventional longest matching prefix search on a search key, which is the concatenation of the fixed range identifications followed by the prefix-based range vector.

- type B: The rule search is realized as an enhanced longest matching prefix search operation, which determines the interleaving of the unprocessed parts of the range vectors based on those parts that have already been processed and by information stored by in the data structure for the enhanced longest matching prefix search operation.

- type C: The (sizes of the) range vector prefixes are adapted for all rules to match a fixed interleave order. The rule search consists of a conventional longest matching prefix search on a search key that is obtained by interleaving the range vectors according to this fixed interleave order.

Section 4.2 and Section 4.3 discuss how range vectors and rule prefixes can be derived for these three types of rule searches. Section 4.4 describes how this can be done in an incremental way. Although prefix-based classification can be used for multi-way packet classification involving the simultaneous processing of any number of header segments, only examples will be given that involve two dimensions for easy illustration and understanding.

4.2 Range Vectors

A relation between multiple ranges will be expressed by a so called *primitive range* that consists of these ranges. To facilitate the generation of range vectors and rule prefixes, a layered structure of primitive ranges is constructed, which will be denoted as *primitive range hierarchy*.

As the first step in the construction of a primitive range hierarchy, a certain ordering of the rules is derived, which will be called the *rule order*. The rule order can be based on parameters such as rule priority, the 'volume' or 'size' of the d-dimensional 'area' covered by a rule, or the expected lifetime of a rule. A typical rule order involves a rule to come before all rules that have a higher priority and which are subsets of that rule. The rule order will be discussed in more detail in Sections 4.4 and 5. For the example shown in Figure 2 the following rule order will be used: rule 2, rule 4, rule 1 and rule 3.

Based on the selected rule order, a primitive range hierarchy is constructed for each dimension, which has the following properties:

- all primitive ranges at the same hierarchy layer are non-overlapping, and
- primitive ranges at higher layers are a subset of primitive ranges at lower layers.

The construction process is illustrated in Figure 4. The rule ranges of the first rule according to the rule order, rule 2, are taken and placed as primitive ranges at layer 1 (L1) in the primitive range hierarchies in both dimensions. The hierarchy for the Y dimension is shown from left to right at the left side of the Y axis. A primitive range that reflects a range relation due to a certain rule, is (for illustrative purposes) labeled after this rule as shown in Figure 4. Primitive range 2 in the X dimension consists of ranges X4 to X7, and primitive range 2 in the Y dimension consists of ranges Y2 to Y5.

Rule 4 is processed next. The rule ranges of rule 4 are in each dimension a subset of primitive range 2. Therefore, in each hierarchy a primitive range 4 can be created at layer 2 while preserving the hierarchy properties. Next rule 1 is processed. In both dimensions the rule ranges of rule 1 overlap with the primitive ranges that are already in the hierarchy. These overlaps are resolved by splitting rule 1 into rules 1a, 1b, 1c and 1d as shown in Figure 4. The remaining part of rule 1 that is covered by rule 2, which has a higher priority, is discarded because this will not affect the classification result. The corresponding rule ranges of the newly created rules can be placed as primitive ranges on top of the primitive ranges that are already in the hierarchy while preserving the hierarchy property. As within the hierarchy in the Y dimension, primitive range 4 equals the rule range of rule 1a, no new primitive range is created for rule 1a but is instead merged with primitive range 4. The merge operation is reflected by the label (which is used purely for illustrative purposes, as mentioned above) of the primitive range 4/1a as shown in Figure 4. Finally, for similar reasons a split operation is performed on rule 3, and the rule ranges corresponding to the new rules 3a, 3b, 3c and 3d are placed as primitive ranges within the two hierarchies.

In this example, the split and discard operations that are performed to construct the primitive range hierarchies, are performed after the rule order has been determined. However, it is also possible to mix these two steps. This allows rule parts that are split from the same rule, to be placed at different locations in the rule order. This can be used to optimize the storage requirements as will be discussed in Section 5.

The last step in the construction of the primitive range hierarchy is the assignment of identifiers (IDs) to each primitive range such that the following conditions hold:

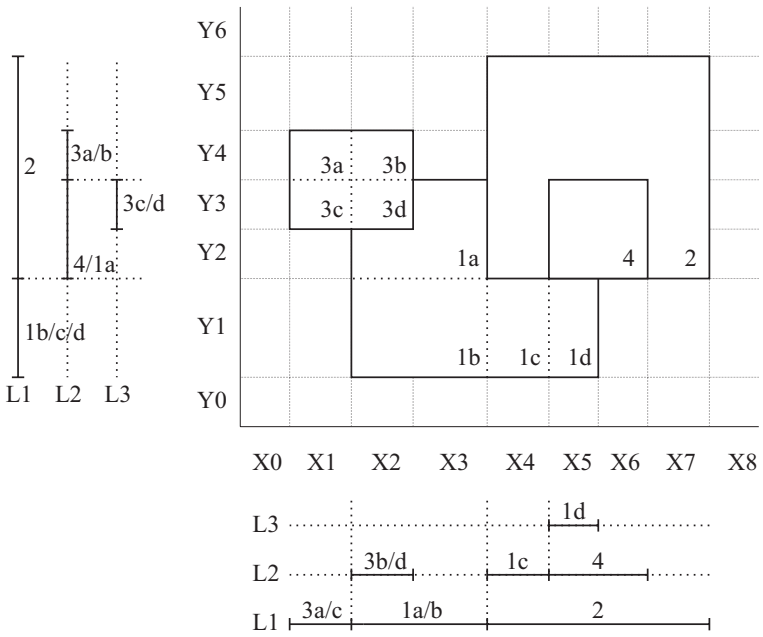

Figure 4: Primitive range hierarchy.

- for all the primitive ranges at layer 1, no ID can be the prefix of another ID,
- for all primitive ranges at layer L that are a subset of the same primitive range at layer $L-1$, no ID can be the prefix of another ID,
- each primitive range ID must contain at least one '1'.

Figure 5 illustrates the IDs assigned to the primitive ranges within the hierarchies in both dimensions. Each set of primitive ranges, as distinguished in the first two items above, are assigned non-zero IDs of at most $\lceil \log(k+1) \rceil$ bits if there are k ranges in a set. In the case that not all combinations for a given ID size are used, then some IDs can be one bit shorter. For example, when there are two primitive ranges in a set (e.g., primitive ranges 1c and 4 at layer 2 in the X dimension), then not all possible non-zero two bit combinations ('01', '10', '11') will be used, and the IDs '01' and '1' can be assigned while preserving the primitive range ID conditions listed before.

The range vectors are obtained directly from the primitive range hierarchy. For each range the range vector is obtained by concatenating the IDs of the primitive ranges of which that range is a subset starting with the primitive range ID at layer 1. Figure 5 also shows the range vectors that correspond to both primitive range hierarchies. The shortest range vector prefix that is embedded in a range vector consists of the ID of the primitive range at layer 1, of which the corresponding range is a subset. The next range vector prefix consists of the same primitive range ID concatenated with the ID of the primitive range at layer 2 of which the given range is a subset, and so on.

The primitive range hierarchy shown in Figure 5 can be used to generate range vectors and rule prefixes (which will be discussed in Section 4.3) for type A and type B rule searches. For type C rule search operations, the primitive range hierarchy has to be adapted to match a fixed range-vector-interleave-order as mentioned in Section 4.1.

Primitive range hierarchy for search type C

An interleave order can be expressed indirectly by indicating for each bit position in the interleave product how many bits of each of d range vectors are already part of the interleave product up to that bit position. For an interleave product consisting of k bits obtained from two range vectors the interleave order can be expressed as the following series:

$$I = \{(c_{x,0}, c_{y,0}), (c_{x,1}, c_{y,1}), (c_{x,2}, c_{y,2}), \ldots, (c_{x,k-1}, c_{y,k-1})\}, \quad (1)$$

where $c_{x,i}$ and $c_{y,i}$ represent the number of bits of the X and Y dimension range vectors, respectively, that are part of the interleave product up to bit position k. For example, a bit-interleave product of the X and Y range vectors can be specified as

$$\{(0,0), (1,0), (1,1), (2,1), (2,2), \ldots, (j,j), (j+1,j), \ldots, (k-1,k-1)\}. \quad (2)$$

The advantage of representing the interleave order in this way is that it directly indicates all the possible bit count combinations that have been processed at certain times by the longest matching prefix rule search operation. In order to fulfill the 'good' interleaving condition expressed in Section 4.1, the lengths of the range vector prefixes for each rule have to match one of the bit count combinations in the interleave order specification, unless the range vector prefixes that are too short to match a combination are not prefixes of longer range vector prefixes and therefore the corresponding (padded) range vectors will always contain zeros at bit positions after these prefixes.

The primitive range hierarchy shown in Figure 4 does not match a bit-interleave order for rule 1a. The two range vector prefixes of this rule are '10' in the X dimension and '101' in the Y dimension, with respective lengths 2 and 3. Since (2,3) is not a valid combination according to Equation (2), and both prefixes are part of longer range vector prefixes (namely prefix '101' of rules 3b and 3d in the X dimension, and prefix '1011' of rules 3c and 3d in the Y dimension), no rule prefix can be created that consists of exactly 5 bits.

There are two ways to construct a primitive range hierarchy for generating range vector prefixes that will fulfill the 'good' interleaving property. One method is to selectively increase the sizes of certain primitive range IDs by padding zeros without changing the range hierarchy. The other method is to build the primitive range hierarchy in a different way such that primitive ranges related to the same rule are always located at the same layer and by making sure that all primitive range IDs at the same layer have the same size.

The primitive range hierarchy in Figure 4 can be adapted to match a bit-interleave order, by padding a zero to the ID of primitive range 1a/1b in the hierarchy in the X dimension which becomes '100'. Only the two range vectors corresponding to ranges X2 and X3 are affected which become '1001' and '100' respectively. These are the range vectors that are shown in Figure 2.

4.3 Rule Prefixes

The generation of rule prefixes for the three types of rule search operations will now be discussed separately. The rules discussed are the rules that remain after the split and discard operations performed for building the primitive range hierarchies.

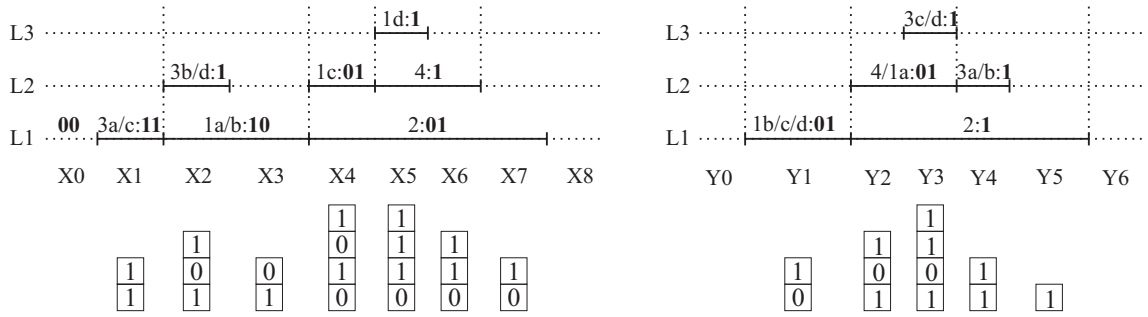

Figure 5: Primitive range IDs for search types A and B.

Rule search type A

This type of rule search is the simplest one. A primitive range hierarchy is constructed in only one dimension. The range vectors in the other dimensions consist of fixed-size range identifications. For example, the range vectors X1 to X7 in Figure 2 could be assigned range vectors '001', '001', '010', to '111' respectively. The rule prefixes are now obtained by determining for each rule the X ranges that are a subset of its X dimension rule range and for which it holds that in these ranges the rule is not completely 'covered' by a higher priority rule. The range identifications of these X ranges are simply concatenated with the range vector prefixes of that rule. For example, in Figure 4, ranges X5 and X6 are subsets of the rule range of rule 4. The rule prefixes for rule 4 are now obtained by concatenating the range identifications of ranges X5 and X6, '101' and '110' with the Y dimension range vector prefix of rule 4, '101' (Figure 5), which results in the rule prefixes '101101' and '110101'. Figure 6 (a) shows the rule prefixes for all the rules. It can be seen that for range X5 the rule prefix of rule 2, '1011' is a prefix of the rule prefix of rule 4, '101101'.

Rule search type B

For a rule search of type B, a primitive range hierarchy is constructed in each dimension. The rule prefix for each rule consists of the corresponding range vector prefixes. For example, the rule prefixes for rule 2, rule 4 and rule 3c can be written as:

X(01) Y(1) → rule 2
X(011) Y(101) → rule 4
X(11) Y(1011) → rule 3c

The rule prefix of rule 2 indicates that the first two bits of the X range vector must equal '01' and the first bit of the Y range vector must equal '1'. Only then will rule 2 be applicable. The rule prefix for rule 4, specifies that the first three X range vector bits must equal '011' and the two first Y range vector bits must equal '101'. In order to realize an efficient search operation in which each bit is analyzed only once, the rule prefixes can be rearranged in the following way, simply by changing the interleaving of the various range vector prefixes from different dimensions:

X(0) X(1) Y(1) → rule 2
X(0) X(1) Y(1) X(1) Y(01) → rule 4
X(1) X(1) Y(1011) → rule 3c

With this representation, the rule prefix of rule 2 can be regarded as a prefix of the rule prefix of rule 4. The way the primitive range hierarchies are constructed guarantees that it is always possible to organize the rule prefixes such that an efficient search operation can be realized in

00111	→	rule 3 (3a)	01101	→	rule 1 (1b)	101101	→	rule 4
0011011	→	rule 3 (3c)	011101	→	rule 1 (1a)	1101	→	rule 2
01001	→	rule 1 (1b)	10001	→	rule 1 (1c)	110101	→	rule 4
01011	→	rule 3 (3b)	1001	→	rule 2	1111	→	rule 2
010101	→	rule 1 (1a)	10101	→	rule 1 (1d)			
0101011	→	rule 3 (3d)	1011	→	rule 2			

(a) Type A rule prefixes

X(0) X(1) Y(0) Y(1) X(0) X(1)	→	rule 1 (1c)
X(0) X(1) Y(0) Y(1) X(1) X(1)	→	rule 1 (1d)
X(0) X(1) Y(1)	→	rule 2
X(0) X(1) Y(1) X(1) Y(01)	→	rule 4
X(1) X(0) Y(0) Y(1)	→	rule 1 (1b)
X(1) X(0) Y(1) Y(0) Y(1)	→	rule 1 (1a)
X(1) X(0) Y(1) Y(0) Y(1) X(1) Y(1)	→	rule 3 (3d)
X(1) X(0) Y(1) Y(1) X(1)	→	rule 3 (3b)
X(1) X(1) Y(1) Y(0) Y(11)	→	rule 3 (3c)
X(1) X(1) Y(1) Y(1)	→	rule 3 (3a)

(b) Type B rule prefixes

0011001	→	rule 1 (1c)
0011101	→	rule 1 (1d)
011	→	rule 2
011011	→	rule 4
1001	→	rule 1 (1b)
110001	→	rule 1 (1a)
11000111	→	rule 3 (3d)
1101001	→	rule 3 (3b)
111000111	→	rule 3 (3c)
1111	→	rule 3 (3a)

(c) Type c rule prefixes

Figure 6: Rule prefixes.

which each bit is processed only once. Figure 6 (b) shows the rule prefixes for all rules.

Rule search type C

For this type of rule search, the primitive range hierarchies have been created such that they match a certain interleave order. As a result, the rule prefixes for each rule can now be created by interleaving the corresponding range vector prefixes according to that interleave order. The primitive range hierarchies shown in Figure 5 can be matched to a bit-interleave order by modifying primitive range ID 1a/b to '100' as described in Section 4.2. The range vector prefixes for rule 3a (which do not require modification) equal '11' in both dimensions. The rule prefix for rule 3a is obtained by bit-interleaving these two range vector prefixes, which results in '1111'. Figure 6 (c) shows the rule prefixes for all other rules obtained in a similar way.

4.4 Incremental Updates

Information related to each rule is stored within the range vectors of the ranges that are within the rule ranges of that rule, and in the form of one or multiple rule prefixes. The following two properties of prefix-based classification make it possible to limit the number of modifications that have to be made to the range search and rule search data structures, for updating rule related information. This allows to perform fast incremental rule updates at least for selected sets of rules.

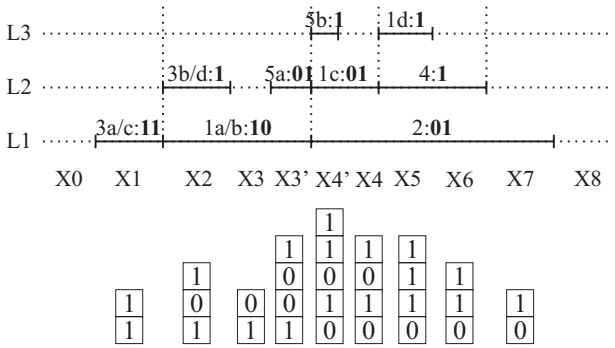

Figure 7: Incremental update.

Addition of a rule requires splitting of at most two ranges into two new ranges, in each dimension. Figure 7 shows how the X dimension rule range of a new rule 5, is added as two primitive ranges 5a and 5b on top of the primitive range hierarchy of Figure 5. This new rule range covers the second part of range X3, denoted as X3', and the first part of range X4, denoted as X4'. As can be seen from Figure 7, the two new primitive ranges 5a and 5b can be assigned unique IDs, without having to modify any other primitive range ID. This is guaranteed for every addition of a new primitive range due to the way the primitive range IDs are generated. The range vectors for the new ranges X3' and X4' embed all the prefixes that were already contained in the original ranges X3 and X4, and in addition a new prefix for the new rule. As a consequence, only rule prefixes for rule 5 have to be added, and all other rule prefixes remain valid. This is a powerful feature of prefix-based classification.

The number of rule prefixes that are contained in the rule search data structure for one rule, equals the number of rule parts into which that rule has been split during the construction of the primitive range hierarchies as described in Section 4.2. The number of rule prefixes is likely to be smaller if the rule is placed earlier within the rule order and, therefore, updates for this rule will require less modifications to the rule search data structure. Of course, it is only possible to place a limited set of rules early in the rule order.

5 Storage Requirements and Performance

Storage requirements

The range search and rule search data structures can be efficiently stored by exploiting their prefix characteristics as will be discussed in this section.

The number of primitive ranges in each primitive range hierarchy equals the number of ranges in the corresponding dimension, due to the similarity between the way in which the ranges are obtained by projection of the rule ranges, and the way in which the range hierarchies are constructed. Consequently, the worst-case number of primitive ranges for one dimension equals $2n + 1$. The IDs of the primitive ranges have to be unique only within certain sets of primitive ranges as described in Section 4.2. Therefore, for search types A and B, each

primitive range ID only requires a maximum of $\lceil \log(2n+1) \rceil$ bits.

Minimum storage requirements for storing the range vectors can be obtained when the range search operations stepwise search for primitive ranges according to the layering within the primitive range hierarchy. For example, a stepwise range search operation according to the primitive range hierarchy shown in Figure 5 for a segment value that is located in primitive range X5, would determine that the segment value is located within primitive range 2 at layer 1 in the first search step, that the value is located within primitive range 4 at layer 2 in the second search step, and that the value is located within primitive range 1 at layer 3 in the final search step. The range vector, that results from the range search, can now be build by concatenating the IDs of the primitive ranges that are found in the successive search steps. In this way, each primitive range ID only needs to be stored once in the data structure. The worst-case total storage requirements for storing all range vectors equals the worst-case total storage requirements for storing all primitive range IDs, and is less than $(2n+1)d \times \lceil \log(2n+1) \rceil$ bits (the actual storage requirements are likely to be much less than the maximum, since the maximum is achieved when all primitive ranges are located at layer 1). This shows that the worst-case storage requirements for storing the range vectors scales better than $n \log(n)$ with the number of rules.

The storage requirements of the rule search data structure, is dependent on the longest matching prefix algorithm that is used. To have a storage parameter that is independent from the selected search algorithm, only the storage requirements of the rule prefixes will be regarded here. The number of rule prefixes equals the number of rules that remain after the split and discard operations during the construction of the primitive range hierarchies, which is directly dependent on the applied rule order. This can be understood from the example shown in Figure 4. If rule 4 was put in the rule order before rule 2, then each of the rule ranges of rule 2 had to be split into three primitive ranges, corresponding to splitting rule 2 into $3 \times 3 = 9$ rule parts from which the center part could discarded since it would be completely covered by rule 4. The number of rules that remain after the split operation has an upperbound equal to $(2n+1)^2$. The number of rule prefixes can be minimized by applying a rule order that matches the rule properties. The storage requirements can further be reduced by exploiting the fact that the rule prefixes have common prefixes, which are the IDs of the primitive ranges at the lower layers. A binary search tree is an example of a data structure in which common prefixes of search table entries are stored only once. This type of search structures allows very efficient storage of the rule prefixes.

We are currently investigating which rule orders and search methods have to be applied on real rule databases in order to obtain minimum storage requirements, and which also allow fast construction times of the classification data structures. An (analytical) analysis of this problem, which includes the characterization of rule properties, is beyond the scope of this paper but will be the topic of a follow-up paper. Figure 9 shows some simulation results for prefix-based classification according to type B, for a small rule set consisting of one hundred randomly generated rules for two 32-bit header segments (e.g., IP source and destination addresses) that have considerable overlaps. The rule order is based on the sizes of the rectangles formed by the rule ranges. Only a small set of rules is used which allows to visualize the overlap characteristics in Figure 8. Better ways of expressing overlap and other rule characteristics and how these can be used to optimize the storage requirements of the classification scheme for real rule based involving multiple thousands of rules, will be discussed in the follow-up paper. Although just an example, Figure 9 shows that the storage requirements of the range vectors represented by the sums of the primitive range ID lengths,

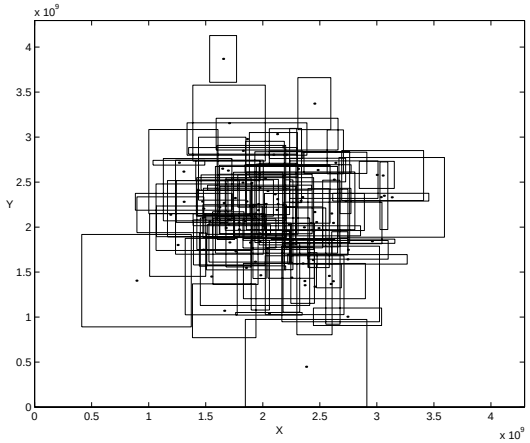

Figure 8: One hundred randomly generated rules.

number of rules, n	100
$2n+1$	201
$(2n+1)\log(2n+1)$	1538
$(2n+1)^2$	40402
number of X ranges	201
number of Y ranges	200
sum of X prim.range ID lengths	539 bits
sum of Y prim.range ID lengths	573 bits
number of rule prefixes	7784
number of binary tree nodes for rule search	23521

Figure 9: Data structure parameters.

are clearly below the worst-case value $(2n+1)d \times \log(2n+1)$ bits. The number of rule prefixes is also much below the worst-case value $(2n+1)^2$. The rule search could be implemented using a binary search tree consisting of 23521 nodes. If each node would contain one bit to select the range vector to be processed, seven bits for a possible rule identifier, and two 32-bit pointers to child nodes, then the search would require $23521 \times (8+32)$ bits $= 117605$ bytes. This can be reduced through an intelligent organization of the tree, in which less bits are needed for the two pointers in each node.

Classification and build performance

Several longest matching prefix algorithms have been published that allow pipelined operation. The rule search is intended to be based on one such algorithm. Consequently, the classification performance will only be determined by the cycle time of the memories used, and is independent of the number of rules. The total time to build the classification data structures from scratch for a given set of rules, is very dependent on the complexity of the rule order and related split and discard operations, and can not be expressed for a general case. The follow-up paper will discuss this issue as well.

6 Summary

This paper has presented a novel scheme for parallel packet classification in which rule information is embedded in the form of prefixes within intermediate parallel search results, and a conventional longest matching prefix search can be used to determine the classification result. The data structures required by the scheme are generated using so called primitive range hierarchies, which also can be done incrementally to allow dynamic rule updates. The use of prefixes has the potential for significant reductions in storage requirements, which would allow multiple thousands of classification rules to be supported at wire-speed.

References

[1] V. Srinivasan, G. Varghese, S. Suri and M. Waldvogel, "Fast and Scalable Layer4 Switching", Proceedings of the ACM SIGCOMM'98, pp. 191-202, 1998.

[2] T.V. Lakshman and D. Stiliadis, "High-Speed Policy-based Packet Forwarding Using Efficient Multi-dimensional Range Matching", Proceedings of the ACM SIGCOMM'98, pp. 203-214, 1998.

[3] P. Gupta and N. McKeown, "Packet Classification on Multiple Fields", Proceedings of the ACM SIGCOMM'99, pp. 147-160, 1999.