Efficient Multi-field Packet Classification for QoS Purposes

Niklas Borg[1], Emil Svanberg[1] and Olov Schelén[2]

[1]Telia Research AB
SE-977 75 Luleå
Sweden
niklas.g.borg@telia.se,
emil.svanberg@capgemini.se

[2]Computer Science and Electrical Engineering
Luleå University of Technology
SE-971 87 Luleå
Sweden
olov@cdt.luth.se

Abstract

Mechanisms for service differentiation in datagram networks, such as the Internet, rely on packet classification in routers to provide appropriate service. Classification involves matching multiple packet header fields against a possibly large set of filters identifying the different service classes. In this paper, we describe a packet classifier based on tries and binomial trees and we investigate its scaling properties in three QoS scenarios that are likely to occur in the Internet. One scenario is based on Integrated Services and RSVP and the other two are based on Differentiated Services. By performing a series of tests, we characterize the processing and memory requirements for a software implementation of our classifier. Evaluation is done using real data sets taken from two existing high speed networks. Results from the IntServ/RSVP tests on a Pentium 200 MHz show that it takes about 10.5 µs per packet and requires 2,000 KBytes of memory to classify among 11,000 entries. Classification for a virtual leased line service based on DiffServ with the same number of entries takes about 9 µs per packet and uses less than 250 KBytes of memory. With an average packet size of 2,000 bits, our classifier can manage data rates of about 200 Mbps on a 200 MHz Pentium. We conclude that multi-field classification is feasible in software and that high performance classifiers can run on low cost hardware.

1 Introduction

Connectionless datagram networks rely on per-packet classification in routers. Traditional best-effort unicast packet forwarding is done by classifying destination addresses of packets against a set of address prefixes. This is known as single-field classification. The increasing use of the Internet for business and commercial purposes introduces an economic incentive for providing service differentiation. Also, increasing availability and popularity of applications with real-time constraints help drive the evolution of service differentiation.

To support service differentiation between packets, classification must be extended to involve also other header fields. This is known as multi-field classification. There are several application areas for multi-field packet classification (e.g., service differentiation, firewalls, QoS routing, etc.). In this paper we present a QoS classifier and we evaluate it specifically in the context of providing service differentiation in IP networks.

To provide service differentiation, the IETF has standardized Integrated Services (IntServ) [12] and the Resource Reservation protocol (RSVP) [15] and is currently working on a more scalable solution called Differentiated Services (DiffServ) [1][2]. The DiffServ effort was motivated, among other things, by the scaling problems of the IntServ/RSVP model resulting from the need for per-flow state in routers. A basic idea with DiffServ is that complexity is pushed to the edges of the network to keep the network core free from per-flow state and processing.

In this paper, we investigate the cost for classification in the contexts of IntServ/RSVP and Diffserv. For the IntServ/RSVP model we study classification as performed by all RSVP capable routers and end-systems. For the DiffServ model, we focus on classification at DiffServ edges and between DiffServ domains. We explore the classification cost to support path sensitive aggregated commitments such as virtual leased lines (VLLs). We do not focus on DiffServ (DS) code point classification as performed by routers interior to a DiffServ domain [1]. This classification is simpler as it only involves classification against a set of code points and the regular routing lookup. The goals of our study are to assess the scalability of our packet classifier and to compare classification costs for IntServ/RSVP and for DiffServ at edges of administrative domains.

The paper is organized as follows. Section 2 covers related work. Section 3 describes the needs for classification in our selected QoS models. Section 4 describes the classifier implementation and section 5 describes the experiments. In section 6, the results of our tests are presented and finally in section 7, the study is concluded.

2 Related work

Packet classification has been studied by many others [5][6][7][8][9]. Srinivasan et. al. [6] approaches the general problem of multi-field classification, calling it *Layer four switching*. It is not completely clear from reading their paper, but we believe that their measurements are based on a model implementation rather than on real packets arriving on a network interface in a router. Our study differs from theirs in that we present an implementation which is plugged into the networking code of an operating system and our measurements are based on actual network traffic. While their study is general, we have chosen to provide performance metrics for some likely QoS scenarios. The scenarios we have chosen are based on IETF activities [1][15] and resent research in the area [3][4].

A common application for multi-field packet classification is firewalls. For an example of a firewall study see [18]. Efficient single-field classification (e.g., best-effort routing) have been explored in Degermark et. al. [10] and Nilsson et. al. [11].

3 Classification requirements in different QoS models

There are different needs for different QoS models, ranging from single fixed length field classification to variable length multi-field classification. In this paper, we focus on three classification cases; IntServ/RSVP classification and two cases of classification in Diffserv that supports the virtual leased line service [3].

3.1 The Integrated Services / RSVP model

The RSVP specification [15] defines a flow as a combination of the source address, destination address, source port, destination port and protocol id. We call this the RSVP quintuple. RSVP is used to signal QoS requirements in terms of IntServ [12][13] objects to the network. Reservation requests are forwarded and processed upstream the datapath from receiver to sender. An end-to-end reservation is established when a request has been forwarded and admitted all the way to the sender and reservation state has been installed in every RSVP capable router along the path.

Reservations are enforced in routers by classification, queuing and scheduling as defined by the state associated with the RSVP quintuple. All packets, independently of whether there are resources reserved for them, will be classified in the same manner by all routers along the path. The need for efficient multi-field packet classification can be anticipated by imagining a backbone node through which thousands, possibly millions, of end systems communicate.

There are proposals for using RSVP to signal DiffServ edge devices and thereby make these devices mark packets for aggregated DiffServ classification inside the domain. This means that classification in all edge devices along the path remains as complex as before. Alternative solutions for DiffServ provisioning are also suggested. One of these approaches is described in section 3.2.1.

3.2 The Differentiated Services model

One reason for developing the DiffServ model, is the drawback of per flow-state and classification in all routers in the IntServ/RSVP model. DiffServ pushes the need for demanding multi-field classification to network edges by aggregating flows into a few well known classes in the core.

In core routers, DiffServ relies on classification involving one fixed length field only (i.e., the DS codepoints contained in the DS field [2]). Diffserv mechanisms can be used to offer qualitative (relative) services that are defined independently of where traffic is sent. In the IETF, the forwarding functionality for providing qualitative services is known as Assured forwarding [25]. Quantitative (absolute) services, on the other hand, can only be offered if the paths that will be used are known (at least roughly), or if we accept a low utilization. In the IETF, the forwarding functionality for providing quantitative services is known as Expedited forwarding [24]. In the Internet, there are several trust boundaries, (e.g., between a customer connecting to an ISP and between two ISPs with peering agreements). The backbone is constituted from a large number of independently administered DiffServ domains. At each trust boundary, quantitative services require destination sensitive classification for policing purposes.

3.2.1 A Virtual Leased Line Service

In [3] Schelén and Pink describe a simple service for virtual leased lines (VLLs). This service can be implemented in a DiffServ capable network by managing resources in so called *QoS agents* (also known as Bandwidth Brokers) responsible for admission control. For each link-state routing domain there is an agent knowing the network topology and static per-link resources available for the VLL service. Reservation requests include source and destination addresses to allow path-sensitive admission control.

Reservations can be made between specific endpoints or address prefixes, (i.e., CIDR style prefixes). Reservations for a specific destination domain are aggregated in agents as their paths merge towards the destination. Consequently, the virtual leased lines form sink-trees towards the destination domains so that at any QoS agent in the network there is at most one aggregate reservation for each destination domain. Reservation management and aggregation is taken care of by agents without involving the routers.

The VLL service is enforced by classifiers in routers configured by agents. The basic idea is that packets admitted to the VLL service have a well known DiffServ codepoint to provide appropriate DS forwarding. To enforce the service level, agents manage police points at edge routers of their domains. To obtain a reasonable resource utilization and predictable service, policing for virtual leased lines must be able to distinguish between different destination domains. Therefore, policing on the border between two DiffServ domains involves classifying all packets having the VLL codepoint against destination prefixes of the commitments (i.e., aggregated virtual leased lines). At DiffServ edges (i.e., before packets have obtained their codepoints), the classification is slightly different. In addition to classification against destination address prefixes, other header fields are involved (e.g., the source address, source and destination ports and protocol to obtain a VLL service that is available for only a subset of the traffic).

In this paper, we explore classification for VLL services at DiffServ edges and between DiffServ domains. The objective is to find out how the classification cost depends on the number of entries to classify against. For edge classification we include the source and destination addresses and ports. For classification between DiffServ domains we classify against destination address prefixes only. The latter of these cases is similar to the classification needed for routing, although this case is different as only packets with matching codepoints would need to go through prefix matching for QoS purposes (all packets go through prefix matching for routing purposes). Throughout the paper, we label the DiffServ edge case *VLL/Edge* and the DiffServ domain border case *VLL/Border*.

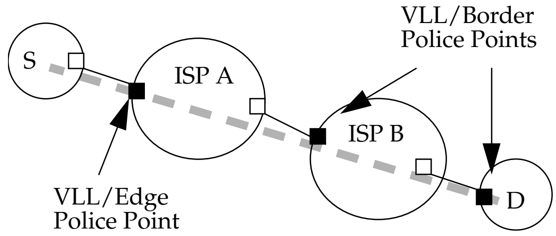

Figure 1: A VLL from site S to site D through two ISPs.

Figure 1 shows a scenario where a VLL has been established from S to D via two interconnected ISPs. Border nodes are shown as squares and police points as filled squares (no links internal to the ISPs are shown).

At the police points, classification is made on destination addresses to check that aggregates of traffic are within admitted rates. At the first police point, packets from S to D are classified, mapped to a certain DS codepoint and policed. Further downstream, classification is made for policing purposes only unless the ISPs have assigned different codepoints for the VLL PHB so that the DS codepoint has to be rewritten.

The DiffServ edge classification needed for VLL service resembles forwarding in traditional best-effort networks. The difference is that the prefixes are expected to be longer for VLLs as it is likely that VLLs are destined for subnets of quite limited size. On the other hand, only packets carrying the VLL codepoint will go through this classification on the border between DiffServ domains.

4 Design of the Classifier

This section presents design choices and data structures used in our packet classifier. The packet classifier is implemented in the programming language C as an integrated part of the operating system NetBSD [16].

4.1 Filter Ordering

Our packet classifier is optimized for QoS classification involving a large number of filters. The filters can consist of multiple fields and allow matching against prefixes or sub-ranges in some of those fields. A problem arise when an incoming packet match more than one filter. There are at least two ways of solving this.

Filters can be assigned costs as in [6] and the problem can be solved by always choosing the filter with the least cost. This method allows great flexibility at the cost of need for filter management. Filter management is needed to, at least, ensure that unique costs are assigned to filters which can match the same packets. Such management potentially requires that the entire data structure needs to be rebuilt. Batching of requests can be used to reduce the rebuild frequency. If applied in operator controlled static environments (e.g., firewalls), it is likely that the gained flexibility is worth the cost of filter management.

We have chosen to introduce a priority order between the fields to unambiguously determine the best match. Assume the priority order; source address, destination address, source port and destination port, and consider a packet matching filters F_x (source and destination addresses) and F_y (source and destination ports). The best match for this packet is F_x since the source address has the highest priority. This method requires no recomputation when inserting or deleting filters which means that there is no need for filter management. Although some flexibility is lost from the priority scheme, we argue that for QoS classification this method is flexible enough.

4.2 The data structure

The filters, consisting of a number of fields and an action, are stored in a data structure. Two important properties of the data structure is low memory consumption and fast lookup times. To optimize on these properties we have chosen to build our structure as a combination of binomial trees [20] and tries [21]. To explain our data structure, a number of example filters are given in figure 2.

Each filter field is stored in a trie. To avoid memory blow up, our tries use path compression. This means that instead of building an ordinary trie, only the nodes that have two children are inserted. In figure 3 we show the source address trie for the filters in figure 2 with and without path compression. The nodes in a path compressed trie need to store their original position in the ordinary trie. For this reason path compression will be most beneficial in sparse tries since in a dense trie almost all nodes have two children.

Filter	Source Addr.	Dest. Addr.	Source Port	Dest. Port
F_1	0100*	0110*	*	*
F_2	0100*	01*	*	*
F_3	0101*	0110*	001	*
F_4	1*	00*	*	*
F_5	1*	*	011	010

Figure 2: Example filters with at the most three fields.

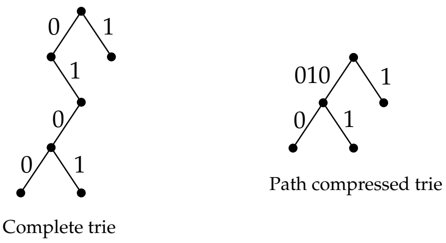

Figure 3: Source address tries for filters F_1 through F_5.

To extend the first trie containing the source address information we expand the structure with one more dimension for each field. A trie that consists of both addresses and both ports will be four dimensional. The new tries will spawn from the parent trie like a binomial tree. The expanded data structure consists of, at the most, as many trees as there are fields that can be included in a filter. Filters that contain the highest prioritized field will be inserted in the first tree. Filters that do not include this field but includes the second highest, will be inserted in the second tree and so on. A schematic figure, with the trie trees marked as nodes, of a four-dimensional data structure where every possible combination of the fields exist, is shown in figure 4.

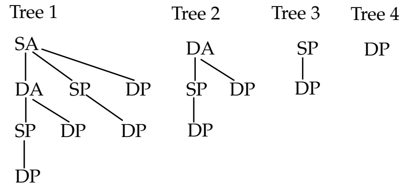

Figure 4: Tries as nodes in a binomial tree.

We use SA as short for source address, DA for destination address, SP for source port and DP for destination port. In the figure all tries that have the same parent are drawn as one node, (e.g., the destination field for filter F_1 will schematically, but not implementation wise, appear in the same trie tree as the destination field for filter F_3).

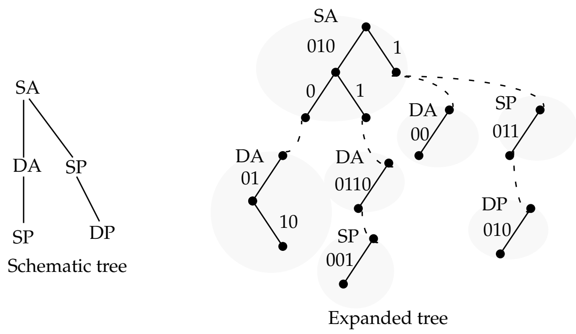

Figure 5: Schematic and expanded tree for F_1 through F_5.

The binomial structure of our example is shown in figure 5. When an incoming packet is compared to the data structure it starts in the trie with the highest priority. When we find a matching filter in a node, the search continues either further down the trie tree or, if that is not possible, in the following binomial trees starting at that particular node.

The best matching filter is the filter that has the fields with the highest priority, the longest prefix match and contains the most fields. If a best match can not be decided according to the priority among the fields the longest prefix match is considered. If a best match still can not be found the filter comprising the most fields is chosen.

If we have the priority order SA, DA, SP, DP, filter F_4 would give a better match than filter F_5 although it contains fewer fields. This on account of the fact that DA has higher priority than both SP and DP. This assumes of course that both these filters match the incoming packet.

5 Experiments

The goal of the experiments is to investigate memory consumption and processing requirements of our classifier. The processing requirements (i.e., the lookup times) are important since a classifier needs to operate at high data rates without causing too much delay or, at the extreme, packet drops. It is also important to minimize memory consumption by the classifier. Some data structures which have really good bounds on lookup times increase rapidly in terms of memory requirements as the number of filters grows (e.g., a simple trie scheme). To test our classifier we have chosen to implement a realistic test bed instead of building models. We test our classifier using real packet traces under conditions as similar to the three QoS scenarios in section 3 as possible.

5.1 Test Environment

The experiments are performed using two workstations running NetBSD 1.3.2 [16]. The workstations are connected to a dedicated 100 Mbps Ethernet where there is no other traffic besides what we generate. One machine is dedicated to send packets across the network, while the other machine operates as a router and classifies all packets flowing through it. We call these machines `sender` and `classifier` respectively. `Sender` is a 166MHz Intel Pentium [17] and `classifier` is a 200MHz Intel Pentium. A schematic picture of the test equipment is shown in figure 6.

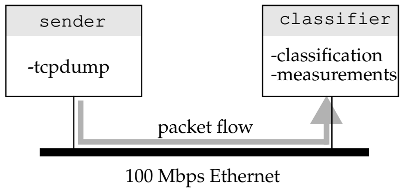

Figure 6: The test equipment.

On `classifier`, we have installed our custom packet classification software described in section 4. On `sender`, a set of test scripts are installed together with a slightly modified version of `tcpdump` [22], which we use to send packets to the network.

5.2 Packet Traces

The packets sent from `sender` to `classifier` are obtained from packet traces taken from two different networks. The first set of traces (T_1), is a collection of traces taken from an experimental network. This network has a 100 Mbps backbone and customers are connected using high bandwidth access solutions such as CATV and ADSL. Most users on this network are residential users with off-the-shelf computers and networking equipment.

The second set of traces (T_2) is taken from a Point Of Presence (POP) within TeliaNet, which is the backbone of a major swedish ISP. The customers connected to the POP include dial up users with modems and ISDN and also corporate customers using higher bandwidth solutions.

Summarizing the packet trace characteristics, T_2 contains more addresses than T_1 and there is a larger number of concurrent flows in the T_2 traces. The T_1 and T_2 traces represent traffic on networks with hundreds of simultaneous users and thousands of simultaneous users respectively. We have chosen to test our classifier with both traces to determine how the distribution of addresses affect the performance of the classifier.

5.3 Filters

The classifier compares fields of incoming packet headers with locally stored filters. Filters are added and deleted via a custom UNIX device. This allows us to construct filters in user space and then copy them to the kernel via this device.

The set of installed filters determines the shape of the tree structure which in turn affects lookup times and memory requirements. The procedure of constructing sets of filters is therefore important to make the tests realistic. As we use packet traces instead of randomly generated traffic, it would not be fair to use pure random filter generation. Rather, we compute filters by browsing the packet trace files and randomly picking flows as filters. In production networks, installed filters are going to have costs associated with them which will yield very few non matching filters. Our filter generation procedure ensures that no filter will be installed that does not match a single packet. If we would include random non-matching filters while keeping the total number of filters fixed, the filter tree would be wider and more shallow, thus increasing the lookup speed.

For the IntServ/RSVP tests we browse the packet traces for unique combinations of the RSVP quintuple and pick filters randomly among those we find. The two types of VLL filters are generated by modifying the IntServ/RSVP filters so that filters contain address prefixes and ports in the VLL/Edge case and destination address prefixes in the VLL/Border case.

The different classification field requirements of the classification cases we described in section 3 are summarized in figure 7. In the figure, *full* means all bits in the field, *prefix* means any number of high order bits in the field, *req* means that the field is required, *req** means that at least one of SA and DA is required and *opt* means that the field is optional.

The lengths of the address prefixes for the VLL cases are randomly picked from the uniform interval 8 to 32 bits. The combinations of ports and addresses in the VLL/Edge case are randomly picked as well. The use of random selection might not be optimal in approximating reality. However, we prefer making that compromise to guessing what combinations of fields would be most common in a future network, and by doing so risk missing important combinations.

	IntServ/RSVP	VLL/Edge	VLL/Border
Source addr	full/req	prefix/req*	
Dest. addr	full/req	prefix/req*	prefix/req
Source port	full/req	full/opt	
Dest. port	full/req	full/opt	
Protocol	full/req		

Figure 7: Fields used for classification.

By testing with an increasing number of filters we characterize the scaling properties of the classifier. It is hard to estimate a reasonable number of filters for future networks. A backbone node could potentially have millions of flows (at RSVP quintuple granularity) going through it at any instant in time. However, we expect that in all of our three scenarios only a small fraction of resources would be allocated to reserved traffic. Best-effort packets would still be the bulk of Internet traffic. Based on these factors we choose to test the classifier in the range 1,000 to 11,000 filters with an interval of 1,000 filters.

5.4 Test Runs

We have chosen to run quite large tests with fairly large numbers of packets while we focus on a small set of qualities of the classifier. As stated earlier we are interested in the most important scaling properties which are memory requirements and lookup times. Therefore, by testing under conditions as similar to reality as possible we investigate the properties of the classifier.

The sender sends packets (taken from the trace files) as fast as possible by using a modified version of `tcpdump`. On `classifier` packets are received and processed at the rate they arrive. The packet traces T_1 and T_2 are taken with the `tcpdump` default snaplenght (64 bytes) which means there is very little payload data. The small packet sizes and the fact that packets are sent as fast as possible from `sender` represents very high data rates at `classifier`. In order not to infer large amounts of extra processing on the heavily loaded `classifier`, which could eventually cause packet loss, we measure the time once every 500 packets. For every number of filters we dump 900,000 packets on the network which gives us 1,800 samples.

We ran identical tests for each of our three different scenarios. As described in the previous section, the set of filters is different from one classification case to another. A test run has the following steps:

- Insert the filters on `classifier`.
- Dump 900,000 packets from `sender`.
- Classify and perform measurements on `classifier`.

These steps are repeated eleven times (1,000 through 11,000 filters) for every scenario. The 1,800 samples taken each repetition are written to file on `classifier` and collected for analysis at a later time.

5.5 Discussion

We have made the following choices and assumptions to create a reasonable environment for the test runs.

- *The filter generation procedure*. Filters are generated based on the packet traces. For the VLL cases, we pick random prefix lengths for address filters, using a uniformly distributed prefix length of 8 to 32 bits. This is tougher than actual usage cases for routing would be. Half of all CIDR prefixes on the Internet are 24 bits long, while the rest are distributed between 16 and 24 [23].

- *Source address distribution*. It is likely that the source domains for most DiffServ edge classification will be representable by prefixes ranging from 16 to 24 bits [23]. We have chosen not to limit the source addresses in such a manner. This gives a wider range of source addresses, not necessarily representable by a common prefix, which yields a deeper source address trie. Thus, we claim that our test conditions are tougher than a real usage case would be with respect to this.
- *Priority order*. Throughout the tests we have had the same priority order, see section 4, among the filter fields. We have used the priority order; SA, DA, SP, DP and protocol. We have made no efforts to optimize performance by setting the priority order.
- *Measurement method*. We engineered the measurements to interfere as little as possible with classification itself. The measurements are made using the CPU clock cycle counter in the Pentium processor [17]. We sample once every 500 packets.

6 Results

We gathered the results from the test runs and present them in this section. The plots in figures 8 through 13 are each explained and motivated. Note that the same plots for both the T_1 and T_2 traces are shown in some figures and that they are distinguished by a label in the figure texts.

6.1 Sanity Check

Our tests are run in an environment where many different factors can impact the results. Therefore, we try to isolate some properties of the classifier. Figure 8 shows the measured lookup times as a function of the number of traversed levels in the trie-tree structure for a VLL/Edge test with 1,000 filters. The dots represent samples and the solid line is a least-square best fit of these samples.

This plot indicates that there is a linear relationship between the depth of the lookup and the lookup time. A goodness of fit test shows that the coefficient of determination is 0.94, indicating a fairly good fit. Plots from other runs are similar.

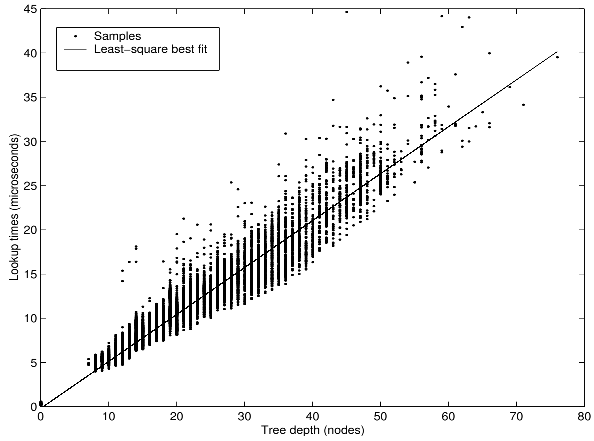

Figure 8: Lookup time as a function of tree depth (T_1).

6.2 Lookup Times

Figure 9 shows average lookup times for the three scenarios described in section 3. The plotted data series are averages of the 1800 samples for each number of filters. All classification cases but VLL/Border yield about the same results for both T_1 and T_2. For VLL/Border classification the difference between lookup times for the T_1 and T_2 traces is surprisingly large and hard to explain by looking at the packet trace characteristics. We feel a need to further investigate the address distributions impact on lookup times.

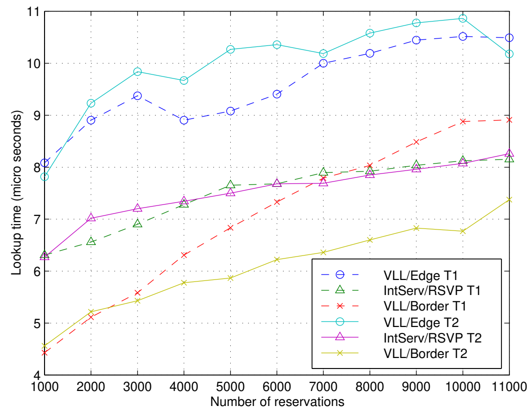

Figure 9: Average lookup times.

Comparing the three classification cases it is clear that the VLL/Edge case is the most demanding. Counting the number of fields to classify against is not enough to explain this fact. The VLL/Edge classification involves fewer fields than IntServ/RSVP classification does. The difference in lookup times is explained by the need to look for longest prefix matches in the VLL/Edge case. When a match has been found in the IntServ/RSVP case, the lookup is completed, while the VLL/Edge case requires that the search is continued until the longest prefix match is found. Note that prefixes of various lengths are used both for SA and DA, contributing to the longer lookup times for this classification case.

Comparing our results to other studies is difficult due to differences in indata, test scenarios, measurement method and hardware. Although the indata and measurement method in [6] differs from ours, the test scenario resembles ours quite well. Their 4 plane grid-of-tries can be compared to our VLL/Edge scenario. In [6] the tests are performed on a 300 Mhz Pentium II resulting in worst case lookup times for their 4 plane grid-of-tries of 3.6 µs when 10,000 filters are stored. Our tests for the VLL/Edge scenario, running on a 200 Mhz Pentium, with the same number of inserted filters results in a measured time of 10.9 µs for our worst set of filters (T_2).

6.2.1 Lookup Time Distribution

The plots in figure 9 hide a lot of relevant information since they are based on average numbers. The imperfections in the plots can be explained by looking at the distribution of the lookup times. Figures 10 and 11 contain histograms and cumulative distributions of lookup times for the VLL/Edge case with 1,000 and 11,000 filters respectively.

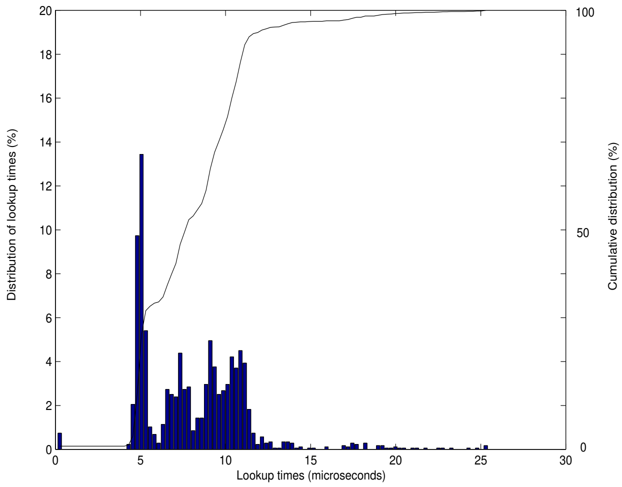

Figure 10: VLL/Edge lookup times with 1,000 filters (T_1).

The distribution of lookup times depends on the installed filters. As shown in figure 8, lookup times vary with the depth of the lookups. Figures 10 and 11 are representative examples of how lookup times increase with the number of filters. For 1,000 filters (figure 10) there is a peak around a lookup time of 5 µs and most of the lookups have times between 7 and 12 µs. The 11,000 filters distribution on the other hand contains about the same amount of lookups in the range 5 to 12 µs but it also has a peak around 25 and is more wide spread than the 1,000 filters distribution.

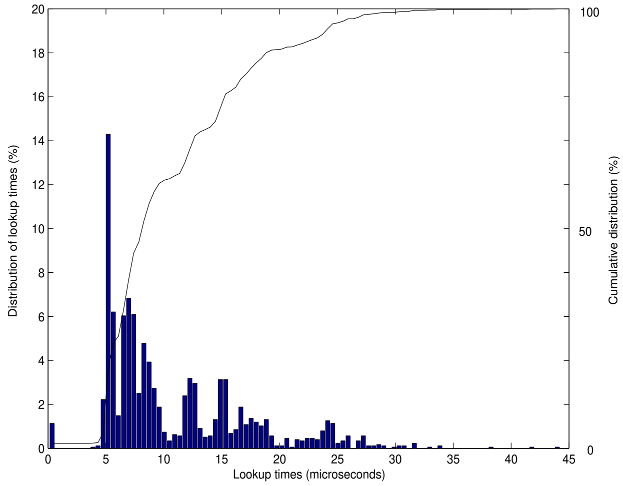

Figure 11: VLL/Edge lookup times 11,000 filters (T_1).

The small peaks around 0 µs in figures 10 and 11 are explained by the small number of packets which are neither TCP nor UDP.

6.3 Memory Consumption

An important scaling property is how the memory consumption of the data structure grows with the number of filters. Memory in routers are limited and fetching data from memory is time consuming.

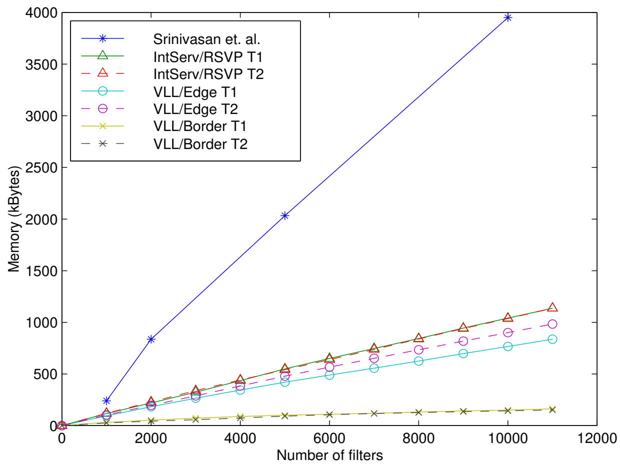

Figure 12: Memory consumption plots.

Figure 12 contains memory plots for our three test scenarios. In addition to our measurements, we have quoted data from [8] (the 4 plane grid-of-tries implementation) and labeled it "Srinivasan et. al.". The growth properties of both our classifier and that of [6] are $O(n)$ where n is the number of filters. However, our classifier implementation requires a lot less memory per filter. Looking at our three scenarios it is obvious that the VLL/Border case with destination address only requires the least memory and the IntServ/RSVP scenario with all fields in the RSVP quintuple the most.

6.4 Insertion times

The time it takes to insert new filters into the classifier is important if the classifier is designed to accept dynamic reservations. Since the insertion time depends on the number of existing filters, we have measured the time it takes to insert one new filter in the range of 0 to 999 existing filters. In figure 13 the insertion times for the three QoS scenarios described in section 3 are shown. Each sample is an average of 250 measurements.

The difference in time for the three scenarios arise from the different number of fields the reservations consist of. As expected the IntServ/RSVP is the most demanding of the three. The maximum average insertion time when 1000 filters are stored is about 15 µs.

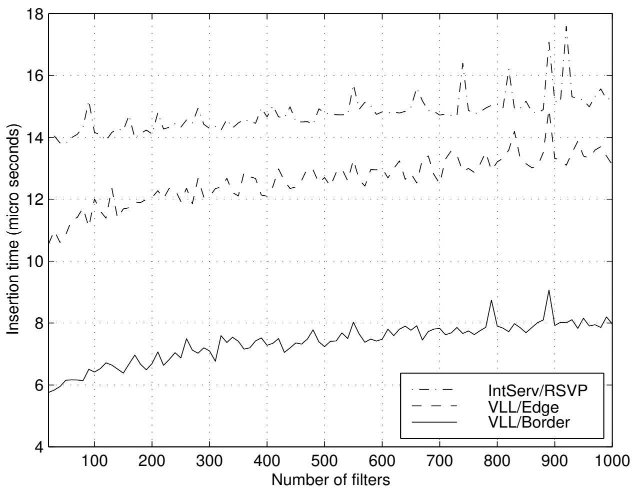

Figure 13: Average insertion times.

7 Conclusions

We have described a data structure and an algorithm based on a combination of tries with path compression and binomial trees. Our implementation has been tested with varying input to compare performance for a set of possible application areas and to verify the implementation. We have focused on three scenarios of packet classification and compared the performance in these cases.

For testing we have chosen to implement a realistic test bed. We have implemented the classifier as a part of the NetBSD kernel and performed packet classification as it would be done in a router, with actual packets arriving on a network interface. We know of no other study that provide results based on measurements in an environment so closely resembling a real network situation. The results presented in [7] and [8] are more general but their classifiers are not tested with real data in a real network environment.

We have tested the scalability of our data structure, verified the implementation, compared performance for a few scenarios and proved that classification itself is not too big an obstacle for new services.

In terms of memory the classifier scales linearly. The lookup time and the insertion time both scales better than linearly with the number of filters. However, without theoretical analyses it is hard to provide strict bounds on the lookup time scaling properties.

Comparing the classification cases we find that classification for VLL/Edge is the most demanding out of the three. This is no surprise since the other models have looser requirements in terms of fields for classification. When comparing the three it is important to remind oneself that IntServ/RSVP requires classification in all nodes, also in the core routers where the number of concurrent flows is expected to be much larger.

For the comparison of different address distributions we find that it is hard to say if one scenario is worse than the other. Our tests indicate that the distribution of addresses does not effect the lookup times as much as we had thought, but more research is needed on this subject.

We show that our classifier can classify about 100,000 packets per second among 11,000 filters for our most demanding test case (VLL/Edge) on a 200 MHz Pentium. A packet size of 2,000 bits implies that our software running on a hardware configuration commonly available at below $1,000 can keep up with a 200 Mbps link. With a smaller number of filters (1,000) our software can for the VLL service keep up with a 400 Mbps link.

Finally we conclude that software multi-field classification is certainly feasible even on commonly available hardware. We believe that there is no reason to disregard a service offering solely based on its classification needs and that there will be services with sophisticated classification needs available in a near future.

8 Acknowledgments

We would like to thank Lars-Åke Larzon at Luleå university of Technology for providing code for the modifications of `tcpdump` and the T_1 packet traces. For great efforts in getting us the T_2 packet traces from TeliaNet we thank Urban Hansson at Telia Research in Farsta.

9 References

[1] Blake et. al. (1998): *An Architecture for Differentiated Services*, RFC 2475, Dec. 1998

[2] Nichols et. al. (1998): *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers*, RFC 2474, Dec. 1998

[3] Schelén, Pink (1998): *Resource Reservation Agents in the Internet*. To appear in Proc. NOSSDAV'98, Cambridge, United Kingdom, July 1998

[4] Schelén, Pink (1998): *Resource Sharing in Advance Reservation Agents*, Journal of High Speed Networks, Special issue on Multimedia Networking. Vol. 7, No 3-4, 1998

[5] Lakshman, Stiliadis (1998): *High-Speed Policy-based Packet Forwarding Using Efficient Multi-dimensional Range Matching*, Proc. ACM Sigcomm'98, Sept. 1998

[6] Srinivasan, Varghese, Suri, Waldvogel (1998): *Fast and Scalable Layer Four Switching*, Proc. ACM Sigcomm'98, Sept. 1998

[7] Mogul, Rashid, Accetta (1987): *The Packet Filter: An efficient mechanism for user level network code*, Technical Report 87.2, Digital WRL, 1987

[8] McCanne, Jacobson (1994): *The BSD packet filter: A new architecture for user-level packet capture*, Proc. USENIX Technical Conference, 1994

[9] Yuhara, Bershad, Maeda, Eliot, Moss (1994): *Efficient packet demultiplexing for multiple endpoints and large messages*, Proc. USENIX Technical Conference, 1994

[10] Degermark, Brodnik, Carlsson, Pink (1997): *Small forwarding tables for fast routing lookups*, Proc. ACM Sigcomm'97, Oct. 1997

[11] Nilsson, Karlsson (1998): *Fast Address Look-Up for Internet Routers*, Proc. IEEE Communications Magazine, Jan. 1998

[12] Wroclawski (1997): *Specification of the Controlled-Load Network Element Service*, RFC 2211, Sept. 1997

[13] Shenker, Partridge, Guerin (1997): *Specification of Guaranteed Quality of Service*, RFC 2212, Sept. 1997

[14] Wroclawski (1997): *The Use of RSVP with IETF Integrated Services*, RFC 2210, Sept. 1997

[15] Braden et. al. (1997): *Resource ReSerVation Protocol (RSVP) - Version 1 Functional Specification*, RFC 2205, Sept. 1997

[16] The NetBSD operating system. URL: http://www.netbsd.org

[17] The Intel Pentium processor. URL: http://www.intel.com

[18] Moliter (1995): *An Architecture for Advanced Packet Filtering*, 5th USENIX UNIX Security Symposium, 1995

[19] The ATM Forum Technical Commitee (1997): *LAN Emulation Over ATM Version 2 - LUNI Specification*, Approved ATM Forum specification, Jul. 1997

[20] Cormen, Leiserson and Rivest (1990): *Introduction to Algorithms*, The MIT Press, 1990

[21] Knuth (1973): *The Art of Computer Programming: Sorting and Searching*, Addison-Wesley, 1973

[22] The protocol packet capture and dumper program, `tcpdump`, URL: ftp://ftp.ee.lbl.gov/tcpdump.tar.Z

[23] IPMA Internet Routing Table Statistics, URL: http://www.merit.edu/ipma/routing_table

[24] Jacobsen et. al. (1998): *An Expedited Forwarding PHB*, Internet Draft, Nov. 1998

[25] Heinanen et. al. (1998): *Assured Forwarding PHB Group*, Internet Draft, Nov. 1998