Budapesti Műszaki Egyetem
Távközlési és Médiainformatikai Tanszék

Load Balancing Algorithms in Multipath Networks

Pro Progressio alapítvány számára készítette:
Németh Krisztián
egyetemi tanársegéd

Budapest
2011. december

Abstract

This report examines different traffic splitting strategies for multipath IP networks. The question is whether today's widely deployed algorithms perform well in all situations. We have implemented in a simulator and analyzed different well-known algorithms. Throughout our study several kinds of traffic traces have been used: real ones captured at backbones as well as at access networks, and specially derived, hypothetical traces. Based on the results we argue that the answer to the leading question is fairly negative: there can be several realistic scenarios, where more sophisticated methods seem to be required.

Table of Contents

Problem statement

Definition of load balancing, multipath routing

In IP networks the main task of a network operator is to transfer the traffic offered by its costumers through its network. The art of mapping the traffic to the network, i.e., selecting which paths to use for which part of the traffic, is called Traffic Engineering (TE for short). In some TE methods there is only one route in the network between a data source (sender) and the destination (receiver). Other TE algorithms, however, allow for “load balancing”, which means that the traffic between a sender and a receiver can be distributed over several (partly disjoint) routes.

Load balancing thus enables the operator to send a higher volume of traffic through its network between an ingress and an egress point, by using the parallel paths. It also improves fault tolerance, as a link or node failure may only affect part of the traffic, not the total volume. Moreover, the affected part might be redirected to one of the other, undamaged paths.

In IP networks, finding the path for a data packet is called “routing”, and when several parallel paths can be used, it is called “multipath routing”. Multipath routing is thus an implementation of the load balancing scheme in IP routing.

Problems with multipath routing

Although multipath routing has got several benefits, if it is implemented on a per packet basis, multiple problems can arise [RFC2991]. Among these problems perhaps the most important one is the possibility of packet reordering, which can easily occur if using different paths between a source – destination pair causes different delay times during the transmission. In this case directing the first packet to the slower path and the next one to the faster can result in the situation when the second packet arrives first, if the difference of the path delays is smaller than the difference of the packet sending times. This packet reordering confuses the majority of the TCP versions: after receiving more than two out-of-order packets, they enter a so-called fast-retransmit mode, in which the late packets are unnecessarily retransmitted. This retransmission causes excessive bandwidth usage, excess delay in the data transfer and lower overall throughput.

Packet reordering is not the only problem, though. IP networks utilize an algorithm called path MTU (Maximum Transmission Unit) discovery, which is used to determine the maximal length of an IP packet that can be sent through a series of lower-layer links without fragmentation. As the maximum supported MTU is a property of a link between two IP nodes (including its endpoints), if packets are taking different routes between a source – destination pair, the path MTU discovery algorithm may fail to provide a correct MTU.

The third possible problem is related to network debugging. Frequently used tools, like ping and traceroute rely on the fact that consequent packets will take the same route, and may provide incorrect results if this is not the case.

Finally, another, perhaps less important negative aspect of using multiple paths for a single flow is the possibility of increased jitter, which again stems from the different delays of different routes.

Possible solutions

Flow-based traffic splitting

The classic solution to the problems sketched above is to pin one flow to one path. The question is, what is a considered to be a “flow” in this regard? Different papers and different products use different definitions for this, but usually a flow means a series of IP packets with the following three IP header fields being identical: “source address”, “destination address” and “protocol number”. Moreover, if the applied transport layer protocol is UDP or TCP, the “source port” and “destination port” fields of the TCP/UDP header should be identical as well.

Handling the flows as one unit as described above allows utilizing multiple paths between a source – destination par, while eliminating the out-of-order packet delivery problem. This is because although the total in-order packet delivery is not guaranteed this way, for every single TCP or UDP session the packets are kept in order. In other words each TCP or UDP endpoint will see the incoming packets in order, thanks to that fact that each flow is bound to a single path.

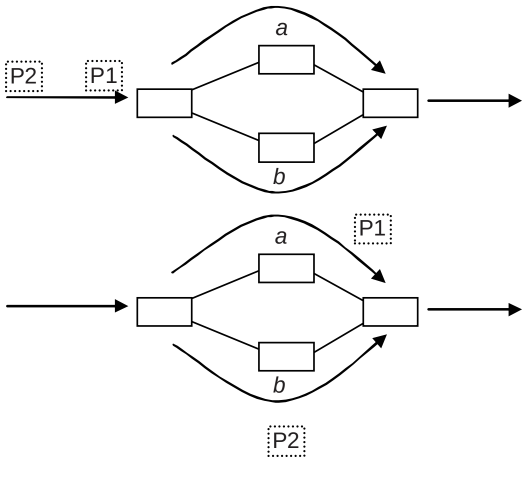
Unfortunately this method is not without drawbacks, either. The per-packet assignment to the paths is very flexible in term of equally sharing the traffic between the outgoing paths, but it might shuffle the packet order. The per-flow approach guarantees in-order delivery, but on the other hand it might be problematic to achieve an equal load on the paths. In the case of a large number of relatively little flows (meaning flows with small bandwidth, i.e. few packets per time unit) we expect no problem of traffic splitting. However, if we have a small number of large flows, then we might run into trouble assigning them the paths in a way that results in equal size of traffic on them. As an extreme example, it is clearly impossible to fairly distribute four equally large flows between three outgoing paths.

Flowlet-based traffic splitting

As a solution to this, Kandula et al. had an interesting proposal [Kandula07]. Their idea is relatively simple. Basically each flow is bound to a given output, just as in the previous case. Here, however, the routers measure for each flow the difference between the arrival times of the consecutive packets of the flow. If this difference is larger than the delay difference of two given output paths, then the flow can be reassigned form one path to the other without the risk of the latter packet overtaking the former one.

Formally, if d_a and d_b are the delays of *path a* and *path b*, and t_1 and t_2 are the arrival times of packets *P1* and *P2*, respectively, then the flow can be reassigned from *path a* to *path b*, before sending out packet *P2*, if $d_a - d_b > t_2 - t_1$.

Such a situation is shown in Figure 1 below.



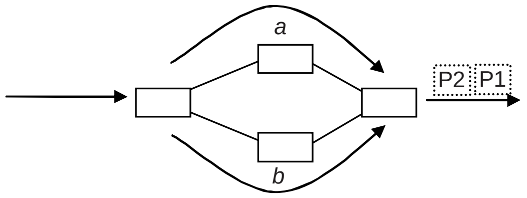


Figure 1. Path change without packet reordering

Looking from the ingress router's point of view, a flow is a series of packets with similar IP headers and different arrival time differences between the consecutive packets. Some packets arrive close to each other, some have larger gaps in between. We call a "flowlet" a set of consecutive packets of the flow, where the time differences between the neighbouring packets are all smaller then a given *Δt* value, but the time difference before the first packet and the time difference after the last one are both larger than *Δt*. Thus a flow can be seen as a series of flowlets. This principle is explained on Figure 2.

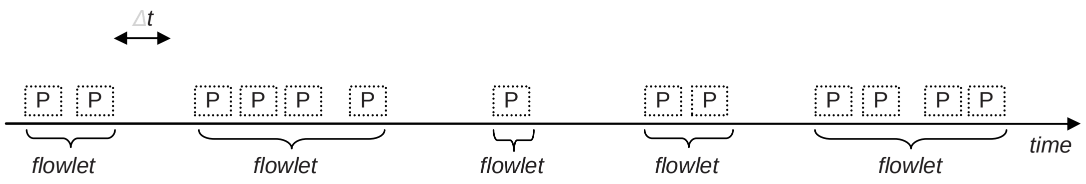


Figure2. Flowlets visualized

If *Δt* is larger than the delay difference of *path a* and *path b*, then the flow can me moved from *path a* to *b* between two flowlets. This certainly means that if the delay is smaller on the original path than the on new one, then the flow can be reassigned between any two packets, regardless of the arrival time difference.

The proposed solution uses active probing to keep track of the delays of different paths. It also stores information for each flow: the timestamp of the last arrived packet and the assigned path is recorded in a hash table. When a packet arrives, this table is looked up using the flow identification information in the packet header, i.e. the source and destination addresses and ports. If the actual time minus the timestamp of the last arrived packet is less than *Δt*, then the stores path is used for this packet as well. If not, a new flowlet is beginning, thus it might be relocated, if desired. Finally, the packet is sent out and the hash record is updated.

This flowlet-based approach seems to be a good tradeoff between the packet-based and the flow-based traffic splitting: it avoids the problem of out-of-order packets, while still providing reasonably equal traffic splitting. On the other hand it requires active probing of the path delays, plus storing state information for each flow.

Application scenarios

Traffic splitting based load balancing can be used in several parts of an IP network. In this subsection a couple of application scenarios are listed, as possible examples.

Let us start with one of the most important fields of application: the already introduced multipath routing. The OSPF routing protocol [RFC2328] for example supports this via a method called Equal Cost Multipath (ECMP).

Another application scenario is the case of local networks [IEEE802]: originally the IEEE 802.3ad task force created a standard for Ethernet, called Link Aggregation. The idea here is two use parallel links between two neighboring Ethernet nodes to achieve higher throughput and/or to employ redundancy for the case of link failure. Later on this part of the standard has been moved to 802.1, when IEEE 802.1AX-2008 came out.

Moreover, the same idea might be used for example within multi-core network processors to distribute load between its cores [FeiHe08]. A network processor is microprocessor specifically designed for networking equipments, such as routers, firewalls, etc. If it is made of multiple cores, the same problem of packet order preserving arises, with the same possible solutions.

Scope of this report

This report compares different traffic splitting algorithms considering several performance metrics, such as the following:

Path utilization fairness

The splitting algorithm should provide fair balancing of the load on the output paths. However, it does not necessarily mean, that all output path should be loaded equally. Generally, if the number of output paths is *n*, it should be possible to provide a set of weight f_1, f_2, ..., f_n, such that $\sum_{i=1}^{n} f_i = 1$. Given these weights, if the total volume of the incoming traffic from a flow in a given time interval is *t* (measured in bytes or in packets), the volume of traffic on the outgoing links should be as close as possible to $t \cdot f_1$, $t \cdot f_2$, ..., $t \cdot f_n$.

Packet reordering

The number of out-of-order packets should be minimal. Zero is the desired value; however, a small percentage of such packets can be an acceptable trade-off for other kinds of benefits.

Router state

Keeping state on a per flow basis allows great flexibility in mapping flows to arbitrary output paths, but on the other hand it might be costly in terms of router memory. Other, for example hash based methods use router memory more sparingly, but perhaps at a cost of the path utilization fairness being degraded.

Related works

The idea of the flowlets was published in [Kandula07]. This paper has an earlier, in many respects more detailed version, [Sinha04], which was published in HotNets04. These papers have surprisingly few follow-up publications, and even those consider flowlets as a complete, finished research. These papers usually just mention that there are basically three kinds of traffic-splitting algorithms: packet based, flow based and flowlet-based.

One of these papers is [Yun08], which deals with the basic problem of traffic engineering: given a traffic matrix, which changes dynamically and given a network, which route should be used for which part of the traffic? In their approach, several routes can be used simultaneously between two endpoints. The proposed method pins a flow to a path, and only mentions the possibility of using flowlets. [Merindol08] and [Kandula05] are similar to the previous paper from our point of view: they describe TE algorithms using flow-based splitting, and solely mentioning the option of flowlets.

[Ye04] analyzes the utility of multipath routing in mobile ad-hoc networks (MANETs). It shows that the benefits of using load balancing in MANETs are not clearly overweight the drawbacks. For example, while for long TCP connections using multipath is usually advantageous, for short TCP connections it can be worse than using a single path.

[He08] is a literature survey about multipath routing. It mentions flowlets in its place, but delivers no further added value.

[FeiHe08] is perhaps the only real follow-up of [Kandula07]: it applies the idea of flowlets for multi core network processors, where the incoming packets are processed in parallel. The problem here is very similar to the multipath routing case, and so is the proposed solution: neither single packets, nor flows, but flowlets should be assigned to a given waiting queue.

Another interesting case of the traffic splitting problem is depicted in [Kandula08]. In this scenario a single WiFi network card with a special device driver was used to connect to two Access Points (APs) at the same time. This is done so that the two uplinks of the two APs can be used simultaneously, thus doubling the throughput, if the uplinks of the APs are independent. The authors here are not using flowlet-based splitting, but are using flow-based division for several reasons. One of them is that the different APs assign different IP addresses to the node, but certainly the sender IP address of an outgoing flow must not change during its lifetime.

[Wu09] deals with packet reordering in line cards before transferring them to the CPU. Although the topic is a bit distant from our current work, this paper well summarizes the problems of TCP stemming from out-of-order packets and the different attempts trying to overcome these problems.

Some IETF RFCs and Internet Drafts are closely related to this work, too. [RFC2991] has already been mentioned: it summarizes the problems with multipath routing. It also suggests different algorithms on how to assign flows to paths. [RFC2992] compares these algorithms, considering how many flows has to be reassigned to another path if the number of paths change (by plus or minus one). [Yong10a] and [Yong10b] are expired Internet Drafts. Their basic idea is promising: treat “large flows” and “small flows” differently (by, for example, storing a state for the large ones and using a hash for the small ones). These drafts proposed to use one bit in the IP header for this distinction. The idea was rejected though in the appropriate IETF working group, as all the functionality can be done in a single box, and thus there is no need to change the IP header.

[RFC6438] describes the usage of the flow label field of the IPv6 header in the case of tunneling. It explains, why it is necessary, and also illustrates, why it can be problematic to use, as [RFC2460] and [RFC3697] strictly regulate the usage of this IPv6 header field. Another work, [Kompella11] deals with a somewhat similar situation: in MPLS networks TCP flows directed to the same destination are bound together into a Label Switched Path (LSP), and it requires a deep packet inspection to distinguish the original flows. This operation, however, is problematic in many ways. [Kompella11] suggests to use a special label, called the “entropy label”, to distinguish the flows within an LSP, and thus to help making a useful load balancing. It has to be noted, that this [Kompella11] is a work in progress, but as the IETF MPLS working group have accepted it as a working group item, it is like to finally become an RFC. Also note, that barely the existence of [RFC6438] and [Kompella11] emphasize the importance of the researched topic.

Pinning a flow (or a flowlet) to an output path can be done in different ways. One of them is to store in a router state for each flow, which path to use. Another way is to use some kind of a hash function on the flow identifier, and decide on the output path based on the output of the function. An interesting hashing algorithm is presented in [Thaler98], called the Highest Random Weight. The basic idea (slightly modified, to be applied to our problem) is to include the identifier of the output paths in the input of the hash function. Using this algorithm the amount of the rerouted flows can be kept minimal, if the number of the output paths changes.

Talking about hash functions, it is important, which part of the IP header should be included in the function input. It is also advantageous to know if a flow-splitting algorithm should be restricted to TCP flows, or UDP should be considered as well. [Lee09] is related to these questions, as it is about observing TCP/UDP ratio and TCP, UDP port distributions using very different kinds of packet traces. The main massage here is: traces from different parts of the network or from different years show very different statistical behavior, thus it is very hard to state anything. Nevertheless, the UDP/TCP ratio in the examined traces was mainly between 5-20%, but without any visible trend.

Another idea mentioned earlier in this section is to treat large and small flows differently. A key question here is how to identify a large flow. [Mori04] proposes an algorithm that can identify large flows (so-called “elephants”) from packet samples, thus largely reducing the required packet processing.

Proposed algorithms

Our long-term goal is to further improve the flowlet-based traffic splitting described in [Kandula07]. One way of the possible improvement is to realize their idea without active probing. Another way is to reduce the amount of states stored about flows: either to decrease it considerably, or to propose a totally stateless design.

As a first step to this, we have summed up to requirements as follows. A packet enters the router, and the routing algorithm comes up with several possible output links. The question is, which one to use for the particular packet. The design goals are the following:

- keep each flow on the same path, if possible;
- the actual proportion of the traffic on each output link should be as close as possible to the desired f_1, f_2, ..., f_n set of weights;
- use minimal (or zero) amount of router states;
- use a simple, fast algorithm;
- if the number of output paths, or their weights changes, the amount of rerouted flow should be minimal.

Note that these requirements are partly contradicting each other. Also note that the last requirement is out of our scope in the first phase of the research.

As a first approach, we have examined some simple algorithms:

<u>Method 1: random</u>. In this case a discrete random variable is chosen according to the set of weights, and the packet is sent out on the path corresponding to the random variable. This method is fast, stateless and simple, but does not pin the flows to paths.

<u>Method 2: simple best fit</u>. This algorithm keeps track of the total traffic sent out on each possible outgoing link. Let us denote with g_1, g_2, ..., g_n the amount of traffic (measured either in bytes or in packets) on each outgoing link. Let then

$$h_i = \frac{g_i}{\sum_{j=1}^{n} g_j}$$

denote the observed split rate. Now, after the arrival of a new packet the router checks the desired and the observed split rate for each link, and selects the link that is most underutilized in this sense, i.e. where h_i-f_i is the smallest. This method is also fast and simple, but requires storing of some router state, although fairly little. On the other hand, this method is not suitable for keeping a flow on a single path, either.

<u>Method 3: simple hashing</u>. In this case when a new packet arrives, a hash function is executed with the flow ID as the input. Note that the flow ID can be defined in many ways. Here we have chosen to use the source and destination IP addresses and the source and destination ports. Note also that we run our measurement for TCP packets only. The output space of the hash function is divided into n regions, such that the size of each region is proportional the weights f_1, f_2, ..., $f_{n.}$ This algorithm is a bit more complex than the previous ones, but it is flow-based and require no router state at all.

<u>Method 4: flow-based best fit</u>. This algorithm is another extreme: it stores a state for every single flow. When a new flow arrives, it is assigned to the most underutilized path, as described in Method 2. This comes with using up a large amount of router space, but hopefully results in better utilization fairness. Naturally it keeps each flow on a single path.

Simulation methodology

This section describes the metrics used in the analysis and the scenarios we have examined.

Measured metrics

Path utilization fairness

As a first step, we have calculated the observed split rate, h_i, as defined in the previous section. Secondly, we have calculated the fairness error, defined as

$$r_i = \left| f_i - h_i \right| .$$

However, a single fairness error number for each link is not enough, as if a link is under-utilized for say a second, and overloaded for the next one, it would show in average a nice fair behavior, which is a false result. Therefore we calculated this fairness error periodically, at every *t=0.1..1000* sec, depending on the path utilization. The aim was to keep the number of packets in one period around 10 000-30 000, resulting in a total of 10-20 megabytes per period.

At the end of the file processing, we have calculated the link average of fairness errors above all the periods, separately for each link, resulting in $\overline{r_i}$. Taking the average of these for all the links, we gained a single number, the average fairness error,

$$\overline{r} = \sum_{i=1}^{n} \overline{r_i} .$$

We made these calculations twice: once deriving h_i from the number of the packets, and the other time from the total length of the packets on a link.

We have also calculated the maximum error for each link between the periods, r_{imax}, and taking the maximum of those gave the total maximum, r_{max}.

Note that we represent all errors as a relative percentage of f_i. This means for example that if a link should have a share of 25% of the traffic, but it gets say the 30% of it, then the error is

$$\left| 1 - \frac{30\%}{25\%} \right| = 20\% \cdot$$

Packet reordering

The number of out-of-order packets should be minimal. Zero is the desired value; however, a small percentage of such packets can be an acceptable trade-off for other kinds of benefits.

Router state

Keeping state on a per flow basis allows great flexibility in mapping flows to arbitrary output paths, but on the other hand it might be costly in terms of router memory. Other, for example hash based methods use router memory more sparingly, but perhaps at a cost of the path utilization fairness being degraded.

Examined packet traces

We have analyzed real-life, anonymized traffic traces, taken at different parts of the Internet. The algorithms described in the previous section were implemented in Perl, using the Pcap library for captured packet file processing. Only the TCP packets were dealt with, but by far the majority of all the packets were actually TCP packets. The assumption was that all these packets are traveling to the same direction (at least in the beginning of their paths), and a load sharing algorithm distributes them amongst *n* outgoing links, with the weights $f_1, f_2, ..., f_n$. In our simulation *n* was 4, and $f_1 = f_2 = f_3 = f_4 = 0.25$.

Real packet traces

Dataset1 has been downloaded from [CAIDA], and contains anonymized passive traffic traces from CAIDA's "equinix-chicago" monitor on an OC192 Internet backbone link. The snapshot has been taken at January 2009.

Dataset2 is from [SWEB], Trace 4: "the 1 Gbit/s aggregated uplink of an ADSL access network has been monitored. A couple of hundred ADSL customers, mostly student dorms, are connected to this access network. Access link speeds vary from 256 kbit/s (down and up) to 8 Mbit/s (down) and 1 Mbit/s (up). The average load on the aggregated uplink is around 150 Mbit/s. These measurements are from February - July 2004." The actual file used is loc4-20040208-2001.bz2.

Dataset3 is from the same trace collection as *Dataset2*, the examined file is loc4-20040214-0410.bz2.

Dataset4 is also from [SWEB], Trace 6: "a 100 Mbit/s Ethernet link connecting an educational organization to the internet has been measured. This is a relatively small organization with around 35 employees and a little over 100 students working and studying at this site (the headquarter location of this organization). All workstations at this location (100 in total) have a 100 Mbit/s LAN connection. The core network consists of a 1 Gbit/s connection. The recordings took place between the external optical fiber modem and the first firewall. The measured link was only mildly loaded during this period. These measurements are from May - June 2007." The selected file is loc6-20070501-2055.gz.

Dataset5 is from the same trace collection as *Dataset3*, but the file is loc6-20070523-0005.gz.

Note that these are fairly large (several gigabytes long) files, and we have only used the first few seconds of them.

Derived packet traces

Dataset6: "tunnels". When tunneling is heavily used, few, but very large flows represent the majority of the traffic. In this case we expect the conventional algorithms to act rather weakly. Unfortunately, it is not easy to get such a trace. Therefore we have created *Dataset6* as follows: each flow in *Dataset1* has been assigned a random number *x* between 1 and 6. If *x* was between 1 and 5, then the packet header has been modified to some predefined IP source address, destination address, TCP source port, destination port values, such that they seem to form 5 streams, emulating 5 tunnels. If *x* was 6, then the packets are left unmodified, representing some background traffic along with the tunnels.

Dataset7: "mice, pigs, elephants". The ratio of small/ medium/large sized flows is expected to be crucial from the path utilization fairness point of view. Small flows are usually called "mice", large ones are referred to as "elephants". We have enhanced this a bit, introducing a middle size category, named as "pigs". The flows in the first 9 million packets of *Dataset1* have been categorized into these three sets, having the limits at 10 000 and 1 000 000 bytes. With these settings we found 249 860 mice, 24 964 pigs and 455 elephants. From these flows we have selected 100 mice, 100 pigs and 10 elephants, thus getting *Dataset7*.

Results

This chapter summarizes the results of the analysis, grouped by datasets. The shown fairness errors are relative errors, as described in the previous section.

TABLE I. summarizes the average and maximal errors for the packet numbers and packet sizes for the different methods for *Dataset1*.

TABLE I. FAIRNESS ERROS IN DATASET1

	Avg,	**Avg, size**	**Max,**	**Max,**
Method	*0.77%*	*1.24%*	*2.36%*	*3.83%*
Method	1.06%	0.01%	3.90%	0.02%
Method	*4.08%*	*1.64%*	*11.42%*	*8.40%*
Method	3.44%	5.15%	10.38%	17.67%

These numbers show that the non-trivial methods 3 and 4 have a considerable maximal error, even for this fairly large and evenly loaded link. For methods 1 and 2 the error is much smaller, but keep in mind that they are not preserving the packet order.

As expected, the stateful, and thus more “expensive”, flow-based best fit (Method 3) produces better results than the stateless simple hashing (Method 4) regarding the packet sizes. On the other hand, Method 4 performs slightly better for packet numbers: this is because the “best-fit” optimization in Method 3 was tuned to packet sizes rather than packet numbers.

The results for the other datasets are shown in TABLE II.

TABLE II. FAIRNESS ERROS IN DATASETS2-7

	Avg,	**Avg, size**	**Max,**	**Max, size**
Ds2, M3	*6.77%*	*3.33%*	*21.34%*	*20.48%*
Ds2, M4	6.49%	13.91%	23.77%	41.86%
Ds3, M3	*5.23%*	*2.53%*	*16.65%*	*17.32%*
Ds3, M4	5.29%	8.57%	14.84%	22.84%
Ds4, M3	*31.94%*	*34.85%*	*83.78%*	*196.42%*
Ds4, M4	44.45%	66.36%	127.54%	257.68%
Ds5, M3	*97.41%*	*139.02%*	*291.65%*	*297.45%*
Ds5, M4	115.89%	143.89%	298.42%	299.54%
Ds6, M3	*32.17%*	*32.34%*	*42.33%*	*43.92%*
Ds6, M4	83.17%	82.99%	121.63%	122.57%
Ds7, M3	*22.23%*	*12.24%*	*51.80%*	*49.78%*
Ds7, M4	21.77%	57.42%	61.44%	133.72%

From TABLE II. the results for the first two methods have been omitted to save space, as their practical usage is very limited, and they have been already shown in Table I for comparison.

TABLE III. below shows the number of flows for each trace. This shows the approximate volume of state that should be stored using Method 3.

TABLE III. FLOW NUMBERS IN THE DATASETS

Ds1	**Ds2**	**Ds3**	**Ds4**	**Ds5**	**Ds6**	**Ds7**

67 580	43 168	35 854	5 200	1 238	11 251	210

The main conclusion form the above results is that the widely used hashing algorithm has a huge error not only for the derived, but for the real datasets, too. In some situations Method3 performs better, but in one of the access network traces, it also produces errors up to 300%.

Summary

In this report we have overviewed the different traffic splitting algorithms that can be used for load balancing. After an elaborated introduction to the topic and to the related works, a simulation study has been described.

We have set up an experimental framework for comparing different splitting algorithms using different metrics. For this examination several traffic traces have been used: partly from real networks (both core and access), and partly artificially derived ones, emulating a tunneling scenario, and a hypothetical, future network load.

Our results show, that the today widely used flow-based methods seem to perform acceptably on a backbone router. In the access network, however, the link utilization fairness measure might show several hundred percents of error, showing that more sophisticated methods are required here. Regarding the derived traces, like the tunneling scenario, the classical algorithms again perform unacceptably.

This shows that further studies and probably more sophisticated algorithms are required in this field, as the widely deployed flow-based algorithm seem to be just not good enough in many practical cases.

References

[CAIDA] Colby Walsworth, Emile Aben, kc claffy, Dan Andersen: The CAIDA Anonymized 2009 Internet Traces, 20090115: equinix-chicago.dirA.20090115-130000.UTC.anon.pcap, http://www.caida.org/data/passive/passive_2009_dataset.xml

[FeiHe08] Fei He, Yaxuan Qi, Yibo Xue, Jun Li: *Load Scheduling for Flow-based Packet Processing on Multi-Core Network Processors*, Parallel and Distributed Computing and Systems (PDCS 2008), November 16 – 18, 2008, Orlando, Florida, USA

[He08] Jiayue He, J. Rexford: *Toward internet-wide multipath routing*, IEEE Network Magazine, March-April 2008, Volume: 22 Issue:2, pp. 16 – 21, ISSN: 0890-8044

[IEEE802] IEEE 802 series standards are available at http://standards.ieee.org/getieee802

[Kandula05] Srikanth Kandula, Dina Katabi, Bruce Davie, Anna Charny: *Walking the Tightrope: Responsive Yet Stable Traffic Engineering*, ACM SIGCOMM, August 2005, Philadelphia, PA, USA

[Kandula07] Srikanth Kandula, Dina Katabi, Shantanu Sinha, Arthur Berger: *Dynamic load balancing without packet reordering*, ACM SIGCOMM Computer Communication Review, Volume 37 Issue 2, April 2007, Pp. 51-62, ACM New York, NY, USA

[Kandula08] Srikanth Kandula, Kate Ching-Ju Lin, Tural Badirkhanli, Dina Katabi: *FatVAP: Aggregating AP Backhaul Capacity to Maximize Throughput*, 5th USENIX Symposium on Networked Systems Design and Implementation, San Francisco, CA, USA, April 16-18, 2008

[Kompella11] K. Kompella, J. Drake, S. Amante, W. Henderickx, L. Yong: *The Use of Entropy Labels in MPLS Forwarding*, IETF draft (i.e., work in progress), draft-ietf-mpls-entropy-label-01, October 31, 2011, expires: May 3, 2012

[Lee09] DongJin Lee, Brian E. Carpenter, Nevil Brownlee: *Observations of UDP to TCP Ratio and Port Numbers*, Fifth International Conference on Internet Monitoring and Protection (ICIMP), 9-15 May 2010, Barcelona, Spain. Print ISBN: 978-1-4244-6726-6, Pp. 99-10

[Merindol08] Merindol, P., Pansiot, J.-J.. Cateloin, S.: *Improving Load Balancing with Multipath Routing*, Proceedings of 17th International Conference on Computer Communications and Networks, 2008 (ICCCN '08), 3-7 Aug. 2008, St. Thomas, US Virgin Islands, pp 1-8

[Mori04] Tatsuya Mori, Masato Uchida, Ryoichi Kawahara, Jianping Pan, Shigeki Goto: *Identifying Elephant Flows Through Periodically Sampled Packets*, Proceedings of the 4th ACM SIGCOMM conference on Internet measurement (IMC '04), Taormina, Italy, October 25-27, 2004, ISBN: 1-58113-821-0, Pp. 115-120

[RFC2328] J. Moy: *OSPF Version 2*, IETF RFC 2328, April 1998

[RFC2460] S. Deering, R. Hinden: *Internet Protocol, Version 6 (IPv6) Specification*, IETF RFC 2460, December 1998

[RFC2991] D. Thaler, C. Hopps: *Multipath Issues in Unicast and Multicast Next-Hop Selection*, IETF RFC 2991, November 2000

[RFC2992]	C. Hopps: *Analysis of an Equal-Cost Multi-Path Algorithm*, IETF RFC 2992, November 2000
[RFC3697]	J. Rajahalme, A. Conta, B. Carpenter, S. Deering, *IPv6 Flow Label Specification*, IETF RFC 3697, March 2004
[RFC6438]	B. Carpenter, S. Amante: Using the IPv6 Flow Label for Equal Cost Multipath Routing and Link Aggregation in Tunnels, IETF RFC 6438, November 2011
[SWEB]	SimpleWeb trace files, http://traces.simpleweb.org/
[Sinha04]	Sinha, Kandula, Katabi: *Harnessing TCP's Burstiness with Flowlet Switching*, 3rd ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets), November 2004, San Diego, CA, USA
[Thaler98]	David G. Thaler, Chinya V. Ravishankar: *Using Name-Based Mappings to Increase Hit Rates*, IEEE/ACM Transactions on Networking (TON), Vol. 6, No. 1, February 1998, Pp. 1-14
[Wu09]	Wenji Wu, Phil Demar, Matt Crawford: *Sorting Reordered Packets with Interrupt Coalescing*, Computer Networks: The International Journal of Computer and Telecommunications Networking, Volume 53 Issue 15, October, 2009, Pp. 2646-2662, Elsevier North-Holland, Inc. New York, NY, USA, ISSN: 1389-1286
[Ye04]	Zhenqiang Ye, Krishnamurthy, S.V., Tripathi, S.K.: *Effects of multipath routing on TCP performance in ad hoc networks*, Global Telecommunications Conference, 2004. GLOBECOM '04. IEEE, 29 Nov.-3 Dec. 2004, pp 4125 - 4131 (Vol.6)
[Yong10a]	L. Yong, P. L. Yang: *Large Flow Classification in IPv6 Protocol*, expired IETF draft (i.e., abandoned work), draft-yong-6man-large-flow-classification-00.txt, June 18, 2010
[Yong10b]	L. Yong, P. L. Yang: *Large Flow Classification in Flow Aware Transport over PSN*, expired IETF draft (i.e., abandoned work), draft-yong-pwe3-lfc-fat-pw-01.txt, July 11, 2010
[Yun08]	Jung-Hoon Yun, Anseok Lee, Song Chong: *Multi-path Aggregate Flow Control for Real-time Traffic Engineering*, Global Telecommunications Conference, 2008. (IEEE GLOBECOM), Nov. 30 2008-Dec. 4 2008, New Orleans, LO, USA, pp. 1-5