Lehrstuhl für Netzwerkarchitekturen
Fakultät für Informatik
Technische Universität München

Capturing the Variability of Internet Flows in a Workload Generator for Network Simulators

Jörg Wallerich

Vollständiger Abdruck der von der Fakultät für Informatik
der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Die Dissertation wurde am 07. 03. 2007 bei der Technischen Universität München eingereicht
und durch die Fakultät für Informatik am 02. 08. 2007 angenommen.

Abstract

The ever growing amount of transferred data and the constantly increasing expectance on reliability and performance in the Internet pose a fundamental problem for network designers. Traditionally these problems have been tackled by overprovisioning link and router capacity to ensure enough spare room. More recently however careful steering of traffic flows has been deployed as a more sophisticated solution to this problem. This so called Traffic Engineering is mostly done by manually configuration of network components to optimally distribute traffic demands over the available resources. There is also ongoing research on automated traffic management using load adaptive routing algorithms to reduce manual intervention on the network. Both approaches rely on the observation, that the size of traffic flows follows Zipf's Law. This implies that only a small number of flows are responsible for the vast majority of the overall network traffic. Traffic Engineering as well as load adaptive routing try to exploit this property by identifying a small set of traffic flows and, by treating this small set in a special way, to control the flow of data in a network. Unfortunately the high variability of Internet traffic leads to the problem that the set of these few large traffic flows is not constant over time. Some of the traffic flows will cease to exist, new ones appear and existing ones change their size drastically over time. The consequence of this variability is the need for periodic reclassification and intervention into the network configuration. For load adaptive routing this implies the threat of instability through route oscillation.

In this work we investigate the nature as well as the causes of the volatility of Internet flows and their impact on traffic engineering and load adaptive routing. We base our study on packet level and NetFlow traces, covering access networks and backbones of scientific as well as commercial networks and propose a methodology for studying persistence aspects of large Internet flows. We show the feasibility of using coarse grained NetFlow data to perform this kind of analysis. Using this methodology we study persistency aspects of large flows on multiple time resolutions and flow aggregations. Our analysis shows a significant stability of flows with high average flow rates, but that there is a high amount of variability in the set of the largest flows. We then use fine grained packet level traces to understand these persistency aspects. We identify the arrival of new and the departure of active flows and rate fluctuations for aggregated flows as the two main contributors of the observed inpersistency. We then introduce a methodology to automatically derive network and protocol parameters from traces of tightly controlled Web page retrievals in order to configure our simulations and show that the base components of our simulation environment, the ns-2 network simulator and the NSWeb traffic generator, are able to provide us with the needed accuracy and configurability. We then introduce our simulation environment and demonstrate, that we are able to reproduce the persistency properties and the variability of large network flows for the evaluation of traffic management approaches in simulations.

Acknowledgments

I would like to thank the many people who have contributed to this thesis. Without their support, cooperation and guidance this work would not have been possible.

First and foremost, I am indebted to my advisor, Anja Feldmann, for the outstanding motivation, guidance, and support she has provided throughout the course of this work. It was Anja who introduced me to the world of network research when I started working with her during my master studies at the Saarland University, Saarbrücken. Since then, she has become good friend and working with her was always fun and very interesting. Without her advice and support, this work would not exist.

I am also indebted to Walter Willinger, who was my mentor during my internship at AT&T Labs Research in New Jersey, USA. He was always available for discussions and I learned a lot during my work with him. I would also like to thank Balachander Krishnamurthy who worked with Walter and me during my time at AT&T. It was great fun to work with both of them.

I also thank my colleagues and friends in our group at TU München. It has been fun working with them. In particular, many thanks go to Holger Dreger, who helped with the implementation of parts of the necessary software and Manfred Jobmann who provided helpful advice in the area of statistics. I would also like to thank Robin Sommer, who started with me working in Anja's group. Thanks go also to Fabian Schneider for commenting on early drafts of this thesis.

My work has been generously supported by the following institutions by providing access to network traces, something that cannot be taken for granted: AT&T Labs Research, New Jersey who gave me access to NetFlow traces from the AT&T backbone, the Saarland University, Saarbrücken and the Leibnitz Rechenzentum, Munich for providing the means to collect packet level traces at their networks. Without the help of many people at these institutions, this work would not have been possible.

Last, but not the least, I thank my mother and my brother who always believed in me. It has been their lasting love and support that enabled me to reach this point.

Contents

List of Figures

1 Introduction

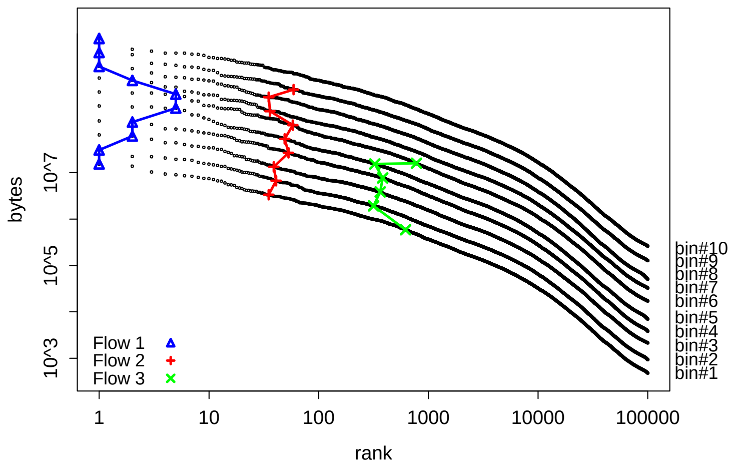


Figure 1.1: Zipf's Law across 10 successive time bins with rank sequences for three flows depicted by connecting the flow ranks on different size-rank curves. The clearly visible linear relation between logarithms of flow rates and ranks, one for each of 10 consecutive time intervals, show that Zipf's Law for flow rates old, even over time.

1.1 Motivation

The Internet is becoming more and more important infrastructure for traditional data services as well as for new applications like Internet telephony and multimedia delivery platforms. The increasing reliance on data networks in conjunction with a growing availability of high bandwidth access at the edge require more and more powerful core networks to satisfy increasing bandwidth demands. In the past higher demands on the network capacity have been addressed by overprovisioning. Based on link and router utilization, network capacities have been increased so that the network was able to cope with the expected raise of the load imposed by ever increasing amounts of traffic. But with the broad availability of the Internet via broadband access technologies and the rise of new applications, this approach will no longer be feasible in the near future. The costs of extending, operating and maintaining an increasingly dense network with the newest technology are growing faster and faster enforcing a much more efficient use of existing and a more careful planning and design of new network capacities. Possible ways to make more efficient use of existing network capacities is Traffic Engineering and the use of load adaptive routing algorithms. Traffic engineering addresses more coarse grained time scales and does therefore not cope well with unexpected or short term changes in traffic. Load adaptive routing mechanisms on the other hand are capable of fast reaction to different traffic patterns, but such routing algorithms tend to introduce instability in the form of oscillation into the routing. Therefore load adaptive routing has been viewed with suspicion in the past.

But with new demands on network capabilities, e.g., by real time applications like Voice or TV over IP, and the insight that overprovisioning is no longer a feasible solution, traffic engineering and load adaptive routing are coming more and more into the focus of network designers and researchers. Developing novel mechanisms that efficiently use available network capabilities in the presence of highly variable traffic is becoming more and more important.

The evaluation of new approaches to control traffic, identify merits and weaknesses and to predict how such new mechanisms behave for existing or planned networks, cannot be done in either real operational networks or in small scale laboratory setups. In order to be able to assess the effects of deploying new mechanisms and algorithms to control the flow of traffic within large networks, the only feasible way is by simulation. But before one can start to develop traffic control mechanisms and use a simulation environment to examine the behavior of such mechanisms, it is necessary to generate realistic traffic for the simulations as a work load to confront new approaches with highly variable traffic. The capability to produce realistic traffic is a vital component of the process of designing and evaluating new traffic control mechanisms in simulations. In order to generate traffic in simulations, a model of realistic traffic is needed. Such a model has to capture the high variability of traffic found in operational networks, especially in the case of traffic control mechanisms as the ability to cope with traffic variability is the main challenge for such algorithms. There are two main contributors to the variability of network traffic in large networks that are relevant for traffic control applications, namely self-similarity and conformance with Zipf's Law of the rate of traffic aggregates.

Starting with the statistical analysis of Ethernet LAN traces in [LTWW94], there has been extensive work towards developing appropriate mathematical and statistical techniques for explaining, describing, and validating the invariant of traffic in packet switched networks to be consistent with self-similar behavior. The self-similar nature or long range dependency of traffic in packet switched data networks implies that there exist a very high variability in the fundamental parameters of network traffic, not only in volume but also in the session arrival, session duration and session size processes. This variability can be observed in the form of large traffic bursts at all time scales. This burstiness is one of the reasons that make it inherently difficult to cope with network traffic, for example in network capacity planning and traffic control mechanisms.

A side effect of the heavy tailed distributions in the On-Off model is the fact, that there are lots of very short sessions or connections that contain together, in spite of their high frequency, only a minor part of the overall packets. The majority of the traffic is generated by only a few very large connections, appearing with low frequency. This phenomenon is also called the '20-80 rule', as about 20% of the connections can be hold responsible for about 80% of the bytes crossing a single point in the Internet (see e.g. [BHGc04]). In a more general context this behavior has been described the in context of Zipf's Law [Li03]. One consequence of this imbalance is that any attempt to cope with network traffic on a per-connection or per-session basis will suffer from the huge number of connections that are responsible for the overall traffic.

This is further complicated by the fact that, due to the self-similar nature of these processes, many connections can arrive in a very short time span (see, e.g., flash crowds [JKR02, LCD04a]). As it is necessary to keep state for every active connection and this state has to be updated at potentially very high rates, this approach requires extensive computing and memory resources. Applications that suffer from this effect are, e.g., quality of service assurance (QoS) and network security [BP01, LCD04b, LCD04a]). But this imbalance also has its advantages. Traffic control mechanism are able to control a large fraction of the traffic by identifying the few very large packet streams to treat them in a special way, e.g., adjust the routing in the network to avoid overloading of links, while ignoring the many small streams. But still a problem remains, as this approach relies on the assumption that once a stream is identified as large, it will stay large. This property of large packet streams is illustrated in Figure 1.1 which shows a log-log plot of the rank-rate relationship for 10 consecutive time intervals for the rates of Internet flows. The curves for the ten time intervals are offset from one another to by a small amount in the vertical direction to facilitate a visual assessment of Zipf's Law across time, indicated by a more or less linear behavior of the ten curves on a log-log scale. The persistence of the ranks of three example flows is visible as by lines that connect the points over the ten bins that belong to the same flow. This lines indicate how the the rates of that particular flows change over time. Note that while the blue line shows about the same amount of movement as the red and the green ones, the range of ranks covered by the medium and the low ranked flows is, due to the logarithmic scaled x-axis, much higher than that of the top ranked flow. Note that, in spite of the high variability in ranks, the medium and low ranked flows show no indication of ever becoming a top ranked flow.

The self-similar nature of network traffic however implies that traffic fluctuations can induce significant changes in the rate of these streams. As a result it is necessary to frequently reclassify traffic streams and identify changes in the set of directly controlled streams. Thus the rate at which traffic streams change, the rate at which reclassification has to be performed and applications have to react to possible changes in the classification, the so called *engineering rate*, are crucial corner stones for these kind of application and will have a significant impact on the feasibility and the kind achievements that can be reached.

Here emerges the questions of how these two fundamental behavior properties of network traffic interact with each other, what are the consequences of such interactions for applications, that rely on the persistence of at least the large traffic streams, in what way should these applications react to possible lack persistence in the used traffic invariants and at what time scale should they react.

Consequently, in this thesis we look into the nature of the variability of traffic volume over time and the persistence properties of large traffic streams as well as the interrelationship between self-similarity and conformance of stream rates with Zipf's Law with the objective to capture these properties in a traffic workload generator for network simulators. We base out workload model on several large packet and NetFlow traces, collected at an access network, the connection of a large scientific network to the Internet, as well as at the backbone of a large commercial network provider. After showing the feasibility of using this kind of trace data, we examine the traffic in respect to self-similarity, Zipf's Law and the nature and stability of these properties, both individually and in conjunction. Finally we incorporate our findings into a network simulation environment based on the widely used network simulator *NS-2* and provide an evaluation of the quality of the implementation of our workload generator.

1.2 Outline

We now briefly summarize the following chapters.

Chapter 2. In the second chapter, we start by introducing necessary background information on various ways to collect network usage information that is needed for assessing current traffic properties used for development of traffic models and also as the basis of traffic engineering approaches. We then focus on one of the introduced traffic monitoring approaches, the network flow concept and its most widely deployed implementation, the CISCO NetFlow module. The we shortly cover basic traffic properties relevant for our modeling approach, namely self-similarity and Zipf's Law for flow rates. We conclude this chapter with a short introduction to traffic engineering and currently used methods to control traffic streams in large operational networks.

Chapter 3. In the third chapter, we introduce our network flow based methodology to identify and characterize properties of network traffic that we need to capture the kind of traffic variability with the highest impact on traffic control applications. We then describe the trace data on which we base our analysis, cover the problems of using network flows for our purposes and show the feasibility of our approach to analyze trace data.

Chapter 4. We start the fourth chapter by looking into basic aspects of flow rankings in terms of stability, followed by a characterization of how the flow rankings behave under different time resolutions and varying degrees of traffic aggregation. We then look into the causes for the observed features of flow rankings using our packet level traces and characterize how they influence the variability of the traffic. Finally, we introduce our metric for quantitively assessing the degree of variability of flow rankings to be able to compare the persistence properties of different data sets, traffic aggregates and time granularity.

Chapter 5. In Chapter 5, we introduce *NS-2* and a basic workload generator, *NSWeb*, that already provides us with the ability to generate self-similar traffic by simulating Web requests. We then show that *NS-2* and *NSWeb* are able to reproduce simple network traffic with high accuracy. We do this by building traces from tightly controlled real Web page requests, automatically configuring simulations based on this traces and compare the simulations with the real Web page requests.

Chapter 6. In this chapter we use the findings of out flow analysis and develop a workload generator that is able to generate simulated network traffic exhibiting the same persistency properties as real traffic. We use Markov modulation as a means to variate flow rates and show that the resulting traffic shows very similar properties as real traffic.

Chapter 7. In the last chapter we provide a summary of this thesis, recur our findings and contributions. We close this work with a discussion of future work.

1.3 Published Work

Parts of this thesis have been published:

Jörg Wallerich and Holger Dreger and Anja Feldmann and
Balachander Krishnamurthy and Walter Willinger
A methodology for studying persistency aspects of Internet flows
ACM Computer Communication Review, Vol. 35(2), 2005

Jörg Wallerich and Anja Feldmann
Capturing the Variability of Internet Flows Across Time
Proc. 9th IEEE Global Internet Symposium, 2006

2 Background

In this chapter we introduce necessary background information on the most important concepts used in this thesis. We start in Section 2.1 with an introduction of various ways to collect network usage information that is needed for assessing current traffic properties. These properties are needed by most approaches to traffic engineering and are therefore the basis of developing traffic models for the design and evaluation of novel traffic control mechanisms. We then focus in Section 2.2.1 on one of the introduced traffic monitoring approaches, the network flow concept and its most widely deployed implementation, the CISCO NetFlow module. Then in we shortly cover basic traffic properties relevant for our modeling approach, namely self-similarity of network traffic in Section 2.3 and Zipf's Law for flow rates in Section 2.4. We conclude this chapter in Section 2.5 with a short introduction to traffic engineering and currently used methods to control traffic streams in large operational networks.

2.1 Monitoring High Volume Network Traffic

With the constantly growing complexity of the network infrastructure in the presence of increasing traffic and complex algorithms on application and network layers, monitoring networks has become both necessary and increasingly difficult. While simple networks consisting of only a few hosts and one connection to the Internet can be easily maintained and controlled, entire provider backbone networks with hundreds of routers and several connections to the rest of the Internet have a complexity that cannot easily be understood and maintained without monitoring what happens on the individual links and routers. Monitoring is a crucial prerequisite for application and user monitoring, capacity planning, security analysis, and traffic engineering. In all of this cases, knowledge about the current behavior of the network in necessary to be able to detect and to adopt appropriate steps to solve problems in the network. Another important field where monitoring is needed is traffic accounting and billing. Here a network carrier needs to reliably determine which of its customers is responsible for what amount of traffic.

Currently, there are five methods readily available to monitor network traffic, packet level capturing, per-interface packet counters on network equipment like switches and routers, sFlow packet sampling and network flow generation.

Packet Capturing

Packet capturing is the most accurate, most fine grained but also the most costly monitoring method. In this case, every packet that passes a network link or a network interface is captured and accounted for, depending on the given application. Capturing packets on links can be achieved by using network taps, that either electrically or optically generate a copy of the link signals and feeds these signals to a monitoring device. Such a device can be as simple as a personal computer using a dedicated network interface card (NIC) to receive the copied packets and forwards them to a monitoring application. There also exist special hardware to perform this task, that can give certain quality guarantees like avoiding packet loss while capturing packets (e.g., DAG cards [End]). While this method is the most accurate one, it poses very high demands on I/O-, CPU and memory capacity, as most networks today have very high bandwidths, in some cases up to 10 Gbps. For such high bandwidth networks, packet rates can be as high as 15 million packets per second and capturing of only the packet header information for one minute needs 16 megabytes of memory for an average packet size of 645 bytes. Because of this lack of scalability, packet level traces are only used in low to medium bandwidth networks or only for small time spans for in-depth analysis of network traffic, e.g., anomaly based network intrusion detection [BP01, LCD04b, LCD04a].

For high bandwidth networks, packet level capturing is only feasible with packet sampling. In this case, not all packets are recorded by the monitor system but only one in *n* packets. The state of the network has then to be inferred by inverting the sampling process using statistical methods. There exists an IETF working group called PSAMP [PSA] that is concerned with defining a standard set of simple to implement capabilities for network components to sample subsets of packets by statistical and other methods. Sampling of packets can be done either randomly by considering every *n*-th packet, or probabilistic by randomly sampling any one out of *n* packets (see, e.g., [CPB93]). Both methods have the disadvantage, that it is not possible to observe the same set of sampled packets at different points in the network, which is needed, e.g., for delay measurements. To overcome this drawback, Duffield and Grossglauser use hash functions to select a subset of the packets passing the network [DG01]. This method, called *Trajectory Sampling*, allows to capture the packets at multiple points along their path (trajectory) through the network and to send information about a packet in the form of a short label to a central accounting machine. Careful adjustment of the hash function allows to capture a particular subset of the traffic or can be used in the same manner as probabilistic sampling.

SNMP and Per-Interface Packet and Byte counters

An alternative to exhaustive packet capturing is provided by modern network equipment like switches and routers in the form of per-interface packet and octet counters that can be accessed remotely using management protocols like the Simple Network Management Protocol (SNMP) [Sta99]. Using these counters it is possible to monitor the amount of packets and bytes that enter and leave a network node, usually on a time scale of 5 minutes [RGK$^+$03]. This allows for a drastic reduction in the amount of monitoring data while enabling an operator to determine link usage even for larger networks. The information that can be won this way is sufficient for local capacity planning and can also be used to detect anomalies in the traffic that show up as distinct increase in link load, e.g., worm attacks [LCD04b].

But even though packets are accounted for on entering the device as well as on leaving the device, it is not easy to discern, where packets originate or where they are forwarded to. No information on end-to-end source and destination or device local forwarding path is available using counters or other statistics provided by the Management Information Bases (MIBs), that come with SNMP enabled devices. While it is possible to derive such information from a set of counters, both per device as well as for entry and exit points of entire networks in the form of traffic matrices, such analysis is computationally expensive and there exist only approximative solutions [ZRDG03, ZRLD03]. SNMP counter values have to be actively requested by a monitoring application at certain time intervals (*pull* principle). This is especially critical if one needs a snapshot of the counter states from a large number of devices, because this results in counter values at many different points in time. This leads to additional problems when correlating counter values from different points in the network.

Network Flows

As packet capturing does not scale in contemporary networks and SNMP based packet and byte counters are too coarse grained, both temporally and spatially, and lacks source and destination information, the concept of a flow was introduced in network monitoring. Flow monitoring delivers highly aggregated counter information while preserving timing information, originator and destination of traffic, routing path information. Moreover, with network flows it is possible to identify protocols and, in a limited fashion, the applications that are responsible for the traffic.

A flow is an abstraction of end-to-end packet streams. It thus constitutes a set of packets, grouped according to common properties like source and destination. Packet and Byte counters are kept on a per-flow basis, allowing for a much more fine grained resolution of the monitoring process as with SNMP. The additional information of packet originator and packet destination allows for reliable accounting and billing based on traffic volume. It enables network operators to determine which links are under what load and who is responsible for network load, allowing for a sensible upgrade of network capacity to accommodate future user demands on performance and reliability. In the case of anomalies, using flow information it is possible to locate the source of the packets causing an anomaly, both for malicious traffic as well as for traffic caused by misconfiguration of devices or applications. In contrast to SNMP based counters, monitoring information is not polled by an application, but exported automatically from the monitored device to a flow collector (*push* principle).

sFlow

sFlow is a simple mechanism for traffic monitoring on high speed network equipment. It uses probabilistic packet sampling in order to cope with large traffic volume and sends information about sampled packets to a central data collector [PPM01].

sFlow thus combines sampled packet capturing with an active export mechanism similar to that of network flows of Trajectory Sampling. It however does not provide complete traffic stream information like network flows nor does its sampling mechanism allow for sampling the same subset of packets at different measurement points to follow the trajectories of the sampled packets. All post precessing of the exported information have to be done offline.

2.2 Network Flows

As the traffic analysis and traffic modeling presented in this thesis is mainly based on network flows we now describe the concepts of network flows in more detail. In addition we take a look at the most widely deployed implementation of network flow probes, namely CISCO's NetFlow.

2.2.1 The Flow Abstraction

The network flow concept was introduced as a means to monitor network traffic in a scalable way while still preserving as much information from the packets as possible [CBP95]. As most communication in IP [RFC81a] based networks does not consist of single packets but of sequences of packets, grouping packets that belong to the same communication stream, e.g., the same TCP [RFC81b] connection, is a sensible solution to the scalability problem. It is a convenient way to reduce monitoring data as one can resort to gather only summarized information instead of storing all packets contained in a packet stream. Information collected for flows usually includes start and end times, counters for packets and bytes and, if available, routing information, e.g., address prefix mask lengths. Network flows are unidirectional by definition, describing packets flowing from a source to a destination. In the case of bidirectional communication, e.g., TCP connections, the packets that make up one of the two directions of such a communication stream have at the very least common source and destination information, and the same protocol. For UDP [RFC80] and TCP, the communication endpoints can be described more precisely by source and destination port numbers. Port numbers are used to distinguish between multiple communication streams terminating at the same network node. This additional information can also be used to identify packets belonging to a single communication stream. The source and destination information used to group packets into flows is called the *flow key*. In order to determine when a flow begins and when it ends, a flow is defined not only by its source and by its destination, there are also timing based restrictions. Consequently a flow is defined by those packets with common source and destination, that are “close in time”. When two packets with the same source and destination are separated by a long time period they are not close in time and thus do not belong to the same flow. Instead, the first packet marks the end of the current flow and the second packet marks the beginning of a new flow.

The IETF working group IPFIX [IPF] is currently concerned with specifying an information model for network flows and a transport mechanism to transfer network flow information from monitoring points to flow collection points. Some of the concepts referred to in IPFIX have been described and defined in a series of RFCs ([Bro99b, BMR99, Bro99a, HSBR99]).

Five-Tuple Flows

The most commonly used and also most fine grained definition of a network flow is the “five tuple flow”. Five tuple flows group packets according to five packet header fields, namely source and destination IP addresses, source and destination port numbers and IP protocol number, e.g., 6 for TCP and 17 for UDP. For protocols without port number information, e.g., ICMP [RFC81a], the port number fields default to zero. As we have seen earlier, flows are uni-directional by definition, so a bidirectional communication, e.g., TCP connections, shows up as two five-tuple flows, one for each direction.

Flow Aggregation

There are cases where it is not necessary to collect such fine grained flow information as five-tuple flows, e.g., for accounting purposes, where information for entire subnetworks are needed.

Therefore it is possible to use other than or only a subset of the five packet attributes of five-tuple flows to group packets into a flow. One commonly used method is grouping packets with common destination address prefix into so called destination prefix flows. The length of the IP address prefix is determined by the corresponding entry in the forwarding table. Destination prefix flows contain all five-tuple flows with destination IP addresses contained in the subnet specified by the destination prefix. This approach is called *flow aggregation*. There are several other schemes to aggregate flows, e.g., AS flows [CBP95, LM97] where the flow key consists of the AS number of the destination IP address, origin-destination [LPC$^+$04] flows, where flows are defined by network ingress and egress points. Aggregation can take place on the probe itself, on the flow collector or by monitoring applications. If the flow probe is able to generate aggregated flows directly from the packet streams, the memory pressure on the flow probe, the network load caused by the export of flow records and storage and processing capacity on the collector and monitoring nodes can be reduced significantly.

The price of flow aggregation is loss of accuracy because of the coarser granularity with which packets are grouped into flows. It depends on the kind of information that is needed for any given application. So is it possible to discard origin and destination related information and only use network ingress and egress points and interfaces to construct very accurate traffic matrices [LPC$^+$04]. Even for accounting and billing applications only IP address prefixes are needed because of the way providers assign IP address blocks to their customers.

It is also possible to perform the aggregation of flows offline. In this case less information is lost as compared to online aggregation on the flow probe because the rates of the contributing five-tuple flows are still available individually and can be used to derive more fine grained flow rate information.

Flow Expiration

As network flows provide only packet and byte counters corresponding to a flow's lifetime, long idle periods can result in misleading flow rate estimations. Therefore it is important to expire flows as soon as they cease to be active and no longer contribute packets to the traffic. Determining the first and the last packets of a flow is not easily possible, as there is usually no mark on a packet that brands it as being the first or the last of a communication stream. For some protocols it is possible to use protocol semantic to recognize start and end packets. An example for such a protocol is TCP, where connections always start with packets carrying a SYN flag and end with packets carrying FIN or RST flags. Nonetheless, TCP flags can be interpreted for finding start and end packets of TCP flows, at least in CISCO's NetFlow implementation and also in other works, e.g. [DLT03]. But even for these protocols, relevant packets that allow one to recognize the end of a transmission, can get lost, appear multiple times or get reordered. Therefore a heuristic based on the notion of a maximum idle time is used. This idle time also allows one to detect the end of flows that do not explicitly signal their termination. But also for protocols with explicit termination signals long idle periods can occur. If there are no packets contributing to a flow for some time, the so called *inactivity timeout*, the flow is considered ended and the statistics collection for this flow is concluded. This heuristic has two advantages. First, it prevents a misleading calculation of flow rates caused by long idle periods. Using the *inactivity timeout*, such a flow is split into two flows, one ending when the idle period begins and another

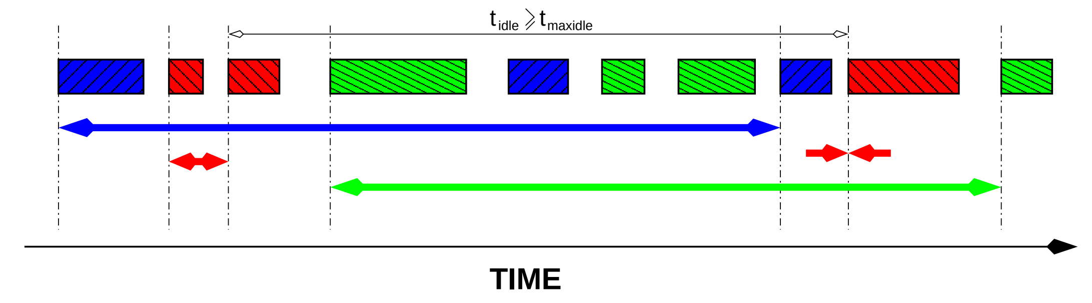


Figure 2.1: Construction of network fbws from packets. Packets with the same fbw key are grouped into fbws if they are 'close in time'. The timing is defi ned by an inactivity timeout. If there are no packets belonging to a fbw for some time, the fbw is considered ended.

one staring when the idle period ends. Omitting long idle periods allows for much more precise flow rate measurements. The second advantage concerns the application of network flows as a cache for routing information. In order to avoid expensive longest prefix match lookups on a per-packet basis, first a lookup is made in the flow cache for a matching five-tuple flow to find information about how to forward the packets of this flow. In order to maintain a high cache hit rate, idle flows had to be replaced in the cache as early as possible by new active ones. This is especially important because of the very limited amount of memory to cache flow entries.

Figure 2.1 shows an example of how flows are generated from packets. The x-axis represents time. Packets are represented by colored rectangles where packets with the same five-tuple are colored in the same way. The length of the rectangles depicts the size of the packets. Timestamps are derived from packet arrival times. The blue packets form together a single flow, starting with the arrival of the first one and ending with the arrival of the third packet. The same holds for the flow consisting of the four green packets. The time between the arrival of two consecutive packets of the same flow is in all cases below the time defined by the flow idle timeout, therefore the flows contain all packets of one color. In the case of the red packets, the time between the arrival of the second and of the arrival of the third red packet exceeds the idle timeout duration. Thus two flows are generated, one consisting of the first two packets and one consisting of only the third red packet. Because the second red flow consists of only a single packet, start and end times are identical and thus the flow is assigned a duration of zero.

Flow Information Export

Contrary to SNMP based counters, where a monitoring application has to regularly request counter values from network devices, network flow statistics are exported actively by the device, that generates flow information from packet streams (*flow probe*). Flow exporting happens as soon as a flow is expired, either by TCP flags or by the idle timeout. In addition, flows are exported, when they have been active for some time, to ensure that the flow information becomes available to monitoring applications in a timely manner. This is important as especially highly aggregated flows may never be idle and would thus never be expired and exported. Flow information, consisting of the flow key, e.g., the five-tuple, timestamps of the first and the last packets of the flow and packet and byte counters, is then sent to a *flow collector*, a network node that receives and stores the flow information for later reference. Common export protocols include UDP and SCTP [SXM$^+$00].

Generating Flows from Sampled Packet Streams

Although monitoring based on (aggregated) network flows is able to reduce the amount of monitoring data significantly, there are cases where this still is too much data. In the presence of, e.g., many short connections or distributed denial of service (DDOS) attacks, only very few packets are really grouped into a single flow. Thus, the number of flows to export is still very high. In order to reduce the load induced by exporting large numbers of small flows, there exists the possibility to generate flows from a sampled packet stream, where only one in n packets, e.g., $n = 100$, is randomly selected and used to generate and update flows. This is usually referred to as *sampled flows*. This is a very different approach from *flow sampling*, where flows are generated using the full packet stream and then only one in n flows is exported. The former is implemented, e.g., in CISCO's NetFlow [Cisb] module while the latter is usually not available on standard network equipment.

Sampled flows have several peculiarities that have to be taken into account. Sampled flows have a bias towards larger flows, as the likelihood of detecting flows consisting of only a few packets is much smaller than the likelihood of detection of large flows with many packets. Very small flows might not even be detected at all. As sampling leads to skipping of packets of flows, the gap between two consecutively seen packets of a particular flow can exceed the configured inactivity timeout and thus lead to packet streams being divided into several network flows. This effect is called *flow splitting* [DLT02, DLT03]. Additionally, the likelihood of sampling the first and last packets of a packet stream is very small. Thus timing information about start and end of a flow is somewhat imprecise.

There exist numerous work on the non-trivial problem recovering statistical traffic information from flows generated from sampled packet streams. In [DLT03] the authors show methods, starting from sampled packet traces, to estimate the size and number of flows in the unsampled packet stream, including those flows that evaded being detected by the sampling process. Other work shows ways to detect elephant flows on sampled network flows using Bayes' theorem [MMK$^+$04] or do a ranking on sampled flows based their rates [BID04]. In [DLT01], the authors propose a method for billing customers on the basis of sampled NetFlow data.

The IETF PSAMP working group responsible for defining standards related to packet sampling is cooperating closely with the IPFIX working group to be able to apply the packet sampling standards to network flows.

2.2.2 The Constant Flow Rate Assumption

As stated above, network flows provide information on packet and byte counts only as total values, accumulated over the lifetime of a flow. However, as we are interested in the dynamics of flow rates over time, we have to derive this information from the total packet and byte counts that a flow contributes to the overall traffic during its lifetime. We need to determine how much traffic a flow contributes during some time interval and compute flow rates using traffic contribution and the length of the time intervals. This traffic contribution however is only available as one value accumulated over the lifetime of a flow. In order to obtain the traffic contribution for some time interval, we need to somehow distribute the total amount of packets and bytes across the flow's lifetime. A natural choice is to assume a constant flow rate, computed as flow volume divided by flow duration. This is referred to as *constant flow rate assumption*.

Practical experience with accounting and visualization systems suggests that this is a reasonable assumption, especially for cases where export for reasonably large aggregation levels [SF02]. This is also consistent with the results of Barakat et al. [BTI$^+$02] who model Internet traffic at the flow level via a Poisson shot noise process where the shape of the "shot" can be rectangular which corresponds to the assumption of constant rate.

In the case of offline flow aggregation based on five-tuple flows, we have can derive more fine grained flow rate information by calculating the rate of aggregated flows as the sum of the average rates of the contributing five-tuple flows. Here the rates of the aggregated flows are no longer constant, but exhibit a significant amount of variability.

2.2.3 CISCO NetFlow Implementation

The most widely deployed implementation of network flow probes is CISCO's NetFlow implementation [Cis98]. As this implementation differs a bit from the concept described above, we now look into this implementation in more detail.

Originally, flow information has been used as a cache of routing information to speed up packet forwarding. As all packets belonging to the same flow can be treated in the same manner by the forwarding process, expensive routing database lookups can be performed once when the first packet of a flow arrives and the forwarding information can be stored in the flow cache for future reference. All successive packets belonging to the same flow would be forwarded according to the information stored in the flow cache entry. This is the reason why CISCO's NetFlows are more complex than simple five tuple flows. They contain additional routing related information in the flow key and the flow statistics. Keeping per-flow packet and byte counters complete the implementation of the network flow concept. We will refer to network flows generated by the CISCO module as *NetFlows* or *NetFlow records*.

Figure 2.2 depicts a NetFlow record in CISCO version 5 format. In contrast to simple five-tuple flows, CISCO uses up to seven fields as flow key. These fields are shown with dark background in Figure 2.2. Besides the fields also used in five-tuple flows, CISCO additionally uses the type of service (TOS) bits of the IP header and the internal index of the incoming interface. These two additional fields can influence the routing decision of a CISCO router and have to be considered because of the flow mechanism being used as a routing information cache. The flow statistics have also been extended to contain routing information in the form of AS numbers and IP prefix mask lengths for both source and destination IP, as well as output interface index and the IP address of the next hop router. In the case of TCP flows, the records also contain information on TCP header flags. The flags are the result of a cumulative OR of all flags seen in the headers of all packets of a TCP flow. CISCO NetFlows are exported as UDP datagrams. As UDP is an unreliable transport protocol, NetFlow records might get lost on their way from the flow probe to the flow collector. In order to be able to detect flow record loss, there exist

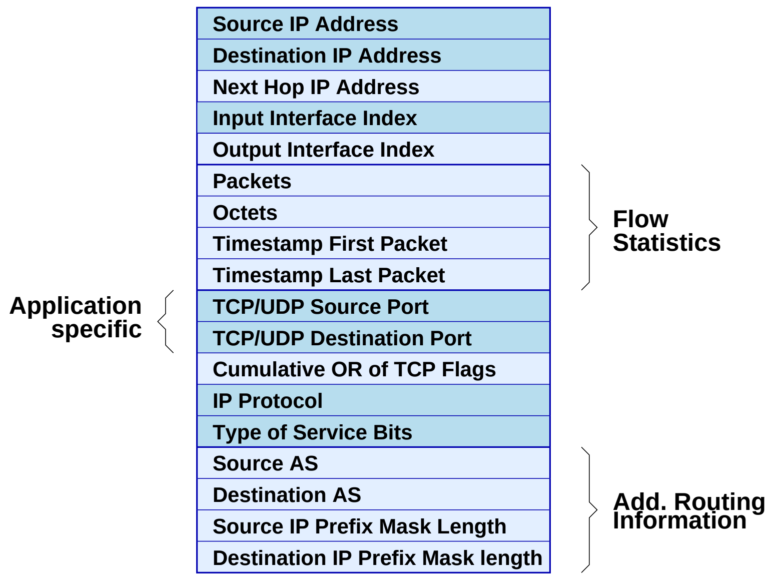


Figure 2.2: CISCO Version 5 NetFlow Records. Fields with dark background are flow key fields.

per-interface flow record sequence numbers that are exported along with the flow records. Flow record loss shows in gaps in the sequence number space.

The timeout values to expire flows are called *inactivity timeout* and *activity timeout*. The inactivity timeout, per default 15 seconds, expires flows that are idle for at least this time, i.e., there has been no packet belonging to the flow. The activity timeout is used to enforce export of long living flows. If a flow is active for a longer time than the value of the activity timer specifies, the flow is exported, regardless of the flow still being active or not. The default of 30 minutes allows for a time granularity that makes it possible to react to undesirable phenomena in reasonable time, e.g., to packet storms due to denial of service attacks. Otherwise, flows might not be exported at all if they are always active. This happens with increasing probability in the presence of aggregation. Both timeouts are configurable, but it is crucial to consider the side effects of these timeouts. The first side effect influences the network load induced by flow record export. If the timeout values are too small, flows might become fragmented, depending on the per-flow packet interarrival time. Each fragment is then treated as its own flow and is exported accordingly, increasing the number of exported flows per time. As CISCO NetFlows are exported in UDP datagrams, a high network load can lead to loss of NetFlow records on the way from the probe to the collector. If the timeout values are too high, the memory consumption on the flow probe is increased. Thus, there is a tradeoff between memory consumption and export induced network load. In addition to the inactivity and the activity timeouts, there exists an additional set of non-disclosed heuristics to export active flows in the presence of memory shortage. The result of poor configuration of the timeout values are coarse time granularity, loss of flow records, and artificially fragmented flows. While a coarse granularity and fragmentation can be tolerated, loss of flow records should be avoided, especially if flows are used for accounting and billing applications. In modern high-end routers, flow generation is no longer used for forwarding purposes. Instead forwarding table lookups are performed using content addressable memory (CAM/TCAM) [MF93, LRV05]. These CAMs are a hardware solution are therefore pose no per-packet load on the CPU. CISCO advertises NetFlow only as a monitoring and billing mechanism.

CISCO's NetFlow implementation starting with version 8 allows flow aggregation on the flow probe using virtually any packet header fields, any routing or forwarding related information like interfaces or prefixes, and even payload information [CISa]. As aggregation is in this case done online on the flow probe, the constant flow rate assumption as to be applied for the resulting NetFlow records.

2.3 Self-Similarity in Network Traffic

One of the most challenging characteristics of traffic in packet switched networks like the Internet is the high variability of the traffic intensity. This variability shows itself in the form of pronounced packet bursts at almost all time scales. This high variability over a wide range of time scales distinguishes traffic in packet switched network from that in, e.g., circuit switched telephony networks. The fact that packet bursts can be observed over a range of time scales implies, that the bursts do not average out over long enough time scales. No matter at what scale one looks at, e.g., on the number of packets crossing a network link, one will always find a similar amount of packet bursts. This phenomenon is referred to as *Self-Similarity* and was first described for network traffic by Leland et al. in [LTWW94]. One of the consequences of the self-similar nature of packet traffic is the failure of traditional network models that use Poisson processes to model packet inter-arrival times [PF95]. While these models have attractive theoretical properties, they lead to a significant underestimation of the burstiness of packet traffic over a wide range of time scales [PF95, GB96].

There exist several methods that can be used to inspect time series for self-similarity, e.g., periodigrams, time-variance plots or Wavelet based methods (see [Gog00] for a more complete overview). In this work, we will use the wavelet based method described in [FHW99]. This scaling analysis technique relies on Wavelets to describe the burstiness of a time series, e.g., packets per time unit, in the form of an energy value as a function of time resolution at different scales. The energy function E_j for a time series $X_{0,k}$ is calculated using the discrete Haar-Wavelet coefficients [Chu92a, Chu92b]

$$d_{j,k} = \frac{1}{\sqrt{2}} \left(X_{j-1,k} - X_{j-1,2k+1} \right)$$

so that E_j can be expressed as

$$E_j = \frac{1}{N_j} \sum_k \left| d_{j,k} \right|^2, \quad j = 1, 2, \ldots, n$$

where N_j is the number of coefficients at scale j. The energy function of a time series is a

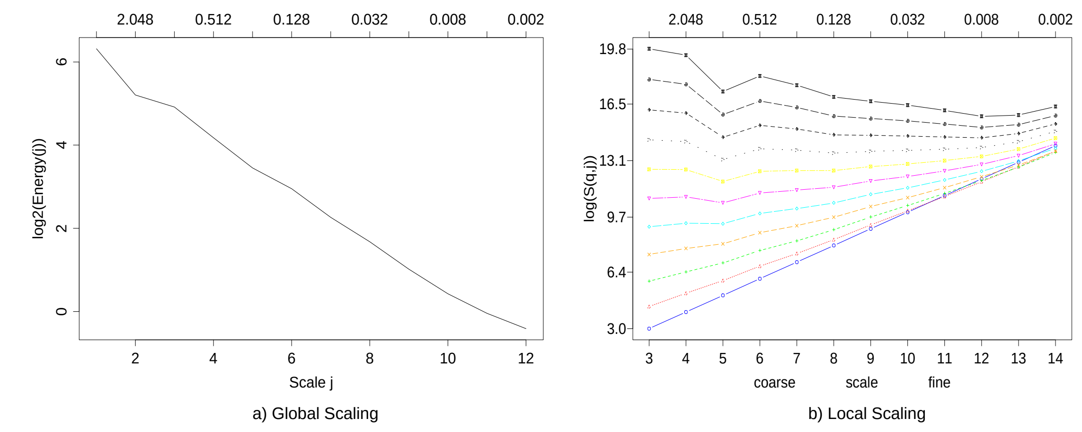


Figure 2.3: Scaling-Plot Examples for a fractionally-differenced ARIMA model, $h = 0.6$: Global scaling plot (left) and Local scaling plot (right). Self-similar scaling is indicated by a linear relationship between the logarithms of scale and energy function (global scaling) or partition function (local scaling) with a non-zero slope. For local scaling plots a linear relationship between the parameter q of the partition function and the scale is indication of multi-fractal scaling.

measure for the amount of variability in the number of events per time unit. Self-similarity is indicated by linear behavior with non-zero slope of the plots over a range of scales. Global scaling plots give information about the average behavior over the complete sample time. Local scaling plots, in contrast, give information about local features like packet bursts. This is accomplished by analyzing the relation between the size of a time interval and the number of events that happen within that interval. The stronger the event bursts are, the less the number of events in an interval depends on the size of the interval. Local scaling plots are plots of the so called partition function $S(q, j)$ as sum over the local maxima of the normalized wavelet coefficients raised to the qth power at each scale j:

$$S(q, j) = \sum_{max} \left| 2^{-j/2} d_{j,k} \right|^q$$

Local scaling plots indicate self-similar scaling behavior, much like global scaling plots, by a more or less linear relationship between scale j and $\log S(q, j)$ over a range of the finest scales. If in addition the plots show a non-linear relation between the slope of the curves and q, the plots indicate multifractal behavior. An example for scaling plots is show in Figure 2.3. Both plots are derived from the same time series. This time series is generated using a simulated long-memory process from a fractionally-differenced ARIMA model with $h = 0.6$ [Bre73]. For both plots, the x-axis shows the time scale in both seconds (top axis) and as a time scale index (bottom axis). The energy function and the partition function are plotted on logarithmic y-axes, respectively. Both plots show approximately linear behavior over all time scales, thus indicating self-similar scaling behavior. Moreover, the slope of the curves in the local scaling plot depends on q in a more or less linear fashion, suggesting monofractal local scaling. In both global and local scaling plots, periodic events, caused for example by packet round-trip times, appear as ’dips’ in the plots, because the variability at the corresponding time scale is lower than at the other time scales. For a more detailed description of the used scaling analyzing method see [FHW99] and [FGWK98].

In [TWS97] the authors show, that self-similar network traffic can be generated with the help of so called ON/OFF sources. An ON/OFF source alternates between sending packets (ON) and being idle (OFF). If the durations of the ON and the OFF phases are consistent with a *heavy tailed* probability distribution, the superposition of a large number of ON/OFF sources results in self-similar packet traffic. Heavy tailed distributions have high or even infinite variance and therefore show extreme variability on all time scales. A statistical distribution is heavy tailed, if the relationship

$$P[X > x] \sim x^{-\alpha}; 0 < \alpha \leq 2$$

holds, at least approximately. More intuitively heavy tailed distributions show a wide range of values, including very large ones, even if almost all values are small. Heavy tailed distributions show a very high variability.

An often used heavy-tailed distributions is the *Pareto* distribution [JK70]. Its probability mass function is defined as

$$p(x) = \alpha k^{\alpha} x^{-\alpha-1}, \quad \alpha, k > 0, \quad x \geq k$$

and its cumulative distribution function is given by

$$F(x) = P[X \leq x] = 1 - (k/x)^{\alpha}$$

The Pareto distribution is an example for a power-law distribution. Note, that the variance of the Pareto distribution is infinite if $\alpha \leq 2$ and for $\alpha \leq 1$ the mean is infinite as well.

A special case of the Pareto distribution that often appear in popularity relations and can be used to describe rate-frequency relations for flow rates are Zipf-like distributions (see Section 2.4). The cumulative distribution function of a Zipf-like distribution is given by

$$P_{\text{Zipf}}[X > x] \sim x^{-(1/\beta)}$$

A Zipf-like distribution with 'slope' β can be generated using a Pareto distribution with $k = 1.0$ and $\alpha = \beta$.

Two other distributions, that are not strictly heavy-tailed according to the above definition, but that both show heavier tails than the Exponential distribution for certain parameters, are the *Lognormal* and the *Weibull* distribution [JK70]. The two-parameter Lognormal distribution is defined as

$$U = \frac{\log(X) - \xi}{\sigma}$$

where U is a unit normal random variable, ξ the expected value and σ the standard deviation of X. A random variable X has a Weibull distribution if there are values for $c > 0$, $\alpha > 0$ and ξ_0, such that

$$Y = \left(\frac{X - \xi_0}{\alpha}\right)^c$$

is exponentially distributed with probability density function

$$p_y(y) = e^{-y}, 0 < y$$

2.4 Zipf's Law and the Consequences for Network Traffic

Zipf's Law [Zip49, Li03] was originally used to describe the relationship between a word's popularity in terms of rank and its frequency in use. It states, that if P is the frequency of use and ρ the popularity of a word, then we have $P \simeq 1/\rho$ or, that the nth most popular word is used twice as often as the $2n$-th most popular word.

This concept has been used to describe the properties of many other systems, so that Zipf's Law can be formulated in a more general context in the following way: if the frequency of occurrences as function of the rank is consistent with a power-law distribution it is referred to as Zipf's-like [Zip49, Mit03]. The rank is determined by the frequency of the occurrence of the studied event, where a low rank index refers to a popular event. Not surprisingly quite a number of different quantities in Internet traffic are consistent with Zipf-like distributions, including the popularity of Web pages [BCF$^+$99, SKW00], intra-domain traffic demands [FGL$^+$01, ZBPS02, CB02], as well as inter-domain Web traffic demands [FKM$^+$04].

If a system behaves in a way that is consistent with a Zipf-like distribution, like the examples above, it is in principle sufficient to optimize the system for the popular events only while ignoring all other events, without introducing a significant performance penalty. In [BCF$^+$98] and [BCF$^+$99] the authors show that Web object access frequencies follow Zipf's-like probability distributions and that this behavior can be used to improve caching of web objects. In [SKW00], the authors show that adjusting the cache replacement algorithm for Web caches according to the Zipf's like access probability distributions, leads to higher cache hit rates, even if the exact distributional parameters are not known a-priori. Breslau et al. however showed in [BCF$^+$99] that the correlation between access frequency and size is very weak, implying that even with a high cache hit rate, the efficiency of caching in terms of bytes affected by caching is limited. Fang and Peterson show in [FP99] that for a set of traces, the size of flows is highly non-uniform with about 9% of AS flows accounting for 86% of all bytes. In order to avoid keeping state for a huge number of concurrent flows, they therefore propose to concentrate on only the small set of large flows when maintaining state for traffic engineering tasks, e.g., in QoS applications. Similar results where found in [UB01]. Other work in the area of network traffic that examines, relies on or has found processes to be consistent with Zipf's Law include load adaptive routing [KKDC05], visualization of network flows [MHCS], flow classification [SST$^+$04] or fair scheduling in routers [NB02].

2.5 Engineering and Controlling of Network Traffic

When operating a network of nontrivial size, the problem arises how to make use of the existing resources as efficiently as possible while at the same time ensuring the best possible performance in transporting data. This problem becomes both more important and complex as the network grows. Efficient use of network resources can mean different things in different situations. For networks that carry transit traffic, the optimization goal is to distribute the traffic over the available links in a way that no link becomes overloaded while others are underutilized, or to minimize the maximum loads on all links in a network [FRT02, PTD04, KKDC05]. A variant of this performance objective is to keep link utilization below 40% [GlFV05]. Similar effects exist for routers, where the capacity of the forwarding engine is exhausted by the joint traffic on all links (hot spot).

For small ISPs or multihomed customer networks, who have to pay for traffic forwarded over upstream links, the goal is to minimize the cost of interdomain traffic while keeping the performance of the interdomain connectivity as high as possible [UB01]. The process of defining an optimization goal and taking measures to achieve this goal is referred to as *traffic engineering*. In traffic engineering network performance is defined by measures like delay, delay variation, packet loss and throughput [ACE$^+$02].

2.5.1 Traffic Demand and Traffic Matrices

Before deciding how to control the traffic crossing a network, it is necessary to first know what kind of traffic one copes with. Thus the process of traffic engineering starts with acquiring information on the traffic demand that is posed on a network. This is usually achieved by deriving a *traffic matrix* from measurements taken from an operational network. A traffic matrix describes how much traffic flows between from all sources to all destinations within an network. More formal, a traffic matrix $T_{i,j}$ provides information on traffic volume entering the network at ingress point i and leaving at egress point j over some time.

Deriving traffic matrices from network measurements is a non-trivial problem. The most obvious approach would be to measure per-destination traffic volume at all routers, but this is not feasible even for moderate sized networks [FGL$^+$01]. An alternative approach is based on NetFlow information from all routers in the network. Since traffic passing through the network generates corresponding flow records on all routers from ingress to egress router, these records can be correlated in a way, that the path through the network can be determined for all incoming packets, together with source, destination and volume information as an *Origin-Destination Flow (OD-flow)* [LPC$^+$04]. The superposition of all OD-flows in conjunction with the path through the network gives then rise to a traffic matrix. Alternatively, one could also collect flow data on ingress routers only and correlate the resulting flow records with routing information to derive the paths of the incoming flows through the network [FGL$^+$01]. The disadvantage of these flow based methods is the huge amount of measurement information needed to derive a traffic matrix.

Therefore, yet another method is based on per-interface SNMP counters (see section 2.1). SNMP counter data is usually collected every 5 minutes [RGK$^+$03], so the amount of measurement information is much lower than for full flow generation. But this comes at a price. SNMP data allows only for estimation of traffic matrices instead of an exact determination [RTZ03].

Existing traffic matrix estimation techniques are, while sufficiently accurate, quite complex [ZRDG03, ZRLD03, MTS$^+$02] because it is a highly underconstrained problem where N ingress and egress points lead to N^2 origin-destination demands [RGK$^+$03].

Usually, demands are stable [TDRR05], which means that they do not change significantly over time. But there are cases, when demands can change dramatically in very short time periods, e.g., Worms, link or router outages or routing changes. If such events are not considered during the process of determining how to route the existing demands, they can lead to link and router overload which in turn results in high delays, packet drops and even router crashes [WXQ$^+$06]. A common approach to avoid such effects is to maintain a history of traffic demand matrices and optimize routing to accommodate for these [ZG05].This way, future matrices are inferred from past ones. In [FT03] the authors show how to optimize routing not only for common case alone but how to ensure that the routing does not become too bad under a preselected set of link or router failures. In [WXQ$^+$06], the authors also propose a methodology to optimize routing for the common case while providing worst case performance guarantees for unexpected scenarios.

A more flexible way to deal with changes in traffic demands is to automate the process of measuring and calculate an optimal routing in real time. Examples for such online algorithms are MATE [EJLW01] and TeXCP [KKDC05]. These approaches are able to detect short-term traffic demands and to adjust the routing accordingly. Another way to deal with unpredictable traffic spikes is oblivious routing [AC03, ABC04, BBCM03]. Oblivious routing does not even try to consider traffic matrices but still manages to minimize the maximum congestion on all links in a network. While oblivious routing can impose an upper bound on congestion, it shows bad performance for the common case [KKDC05].

2.5.2 Traffic Control

After a traffic matrix has been derived, there are currently two major methods for controlling network traffic in use. The first uses intradomain routing protocols like OSPF and IS-IS to control the flow of traffic through the network in way, that the desired performance and utilization goals are met. This is done by carefully computing link metric values that are used by these routing protocols to find the shortest paths from ingress to egress routers [FGL$^+$00, FT00, FRT02]. The solution of this optimization problem ist NP-hard. Consequently, approximation algorithms have to be used, but existing algorithms come very close to the theoretical optimal solution [FRT02].

The second approach makes use of *Multi-Protocol Label Switching (MPLS)*. In MPLS an operator basically defines virtual paths through the network by configuring tunnels on the routers. Then traffic is divided into classes and the packets are labeled according to their classes. Packet forwarding is now done based on this label instead on the destination IP address. For a more complete overview of how traffic engineering with MPLS works, see [Awd99] and [GlFV05].

The advantage of MPLS based traffic engineering over routing-metric based methods is the more fine-grained control of what traffic is treated in what way. This comes from the traffic classification mechanism, that is used to determine which label to use for the forwarding process within an MPLS network. The disadvantage of MPLS stems from the static tunnel configuration, that cannot react automatically to link failure of change in traffic patterns. Backup tunnels have to be configured in advance, to alleviate link failure. Also the setup of network wide tunnels using router-local configuration primitives makes it difficult for operators to maintain an overview of what tunnels are configured and what paths they are using.

There exists also the possibility of using load-adaptive routing protocols, e.g., [QYZS03], [KKDC05] or [FKF06]. These routing protocols can react reasonably fast on changes in traffic demands, something that is not easily possible with static traffic engineering methods like OSPF link weight optimization or MPLS. Disadvantages with load adaptive routing protocols are proneness to oscillations and too frequent route changes. Oscillation effects can appear if shifting traffic from one path to another leads to a situation, where the routing algorithm tries to shift the traffic back from the new path to the old one. There can also be more complex scenarios when many path changes occur that cause traffic to be shifted around in the network without the algorithm converging to a steady state. The second disadvantages, namely changes in routing paths, holds to lesser degree also for static traffic engineering schemes. In the case of adaptive routing algorithms, routing changes appear much more often, leading to undesirable effects, e.g., packet drops and packet delay changes. These two effects have especially negative influences on TCP performance and real time applications like voice-over-IP. The negative impact of changes in the internal routing of an AS is amplified, when they lead to additional changes in the inter-AS routing by choosing different egress points in the presence of hot-potato routing [Tei05]. There exist other adaptive routing schemes based on source routing [ICS96] or overlays, e.g., Detour [SAA$^+$99] or RON [ABKM01].

2.5.3 Concentration on Heavy Hitters

Optimizing routing to accommodate all demands detected in the traffic analysis phase of the engineering process can become a complex task when the amount of different demands growths. In the case of MPLS based traffic engineering, demands are routed over specific paths through a network by configuring tunnels. Covering the entire traffic matrix can result in high number of tunnels. Furthermore, packets entering the network have to be classified to be able to be assigned to their appropriate tunnel, leading to very complex packet classification schemes. The classification problem appears also for adaptive routing algorithms. In order to alleviate this problem, it is a commonly-used approach in traffic engineering to target primarily the large demands and to attempt to optimize system performance mainly for them. This approach exploits the property of traffic demands or traffic flows to be consistent with Zipf's Law. Most of the traffic is contained in only a small portion of the traffic demands. The rest of the demands can be routed without special treatment without significant impact on the overall optimality of the routing. Examples that follow this basic approach to traffic engineering include, among others, [EV02] and [FP99].

2.5.4 Engineering Rate

The traffic engineering process shown above suffers from two major problems. The first one consists in taking a snapshot of the traffic conditions in a network to compute a traffic matrix that in turn is used to decide how to route the traffic to achieve a performance goal. But traffic conditions change over time, making a traffic matrix obsolete and thus lead to detoriation of the initially achieved performance gains. It is possible to alleviate, e.g., time of day effects by taking two traffic matrices, one taken during the day and a second one taken during the night. It is then possible to calculate a routing configuration that is optimal over an entire day by considering both matrices [FRT02]. This approach however does not work for mid- to long-term trends or sudden changes in traffic volume like, e.g., flash crowds.

Therefore, the traffic matrix has to be frequently recomputed and traffic reclassified accordingly. Here, the more often the traffic matrix is updated, the more accurate are current traffic conditions represented by the traffic matrix and the better can the routing be adapted for optimal performance goals. On the other hand, routing changes that result from reconfiguration of paths, disrupt the operation of networks, even to the point of changing the inter-AS routing by influencing BGP routes [Tei05], and are therefore to be done as seldom as possible (see section 2.5.2).

Therefore there exists a tradeoff between accuracy of traffic engineering and stability of routing [FT02], that is controlled by the *traffic engineering rate*. The traffic engineering rate determines how often the routing is adapted to the current traffic conditions. For, e.g., static link-weight based traffic engineering, the engineering rate can range from several minutes to hours [PTD04] or even longer. For adaptive routing mechanisms, the engineering rate has to be at least as high as the rate of traffic changes that are to be considered by the algorithm. Thus the engineering rate for adaptive routing mechanisms takes values from a few seconds to a few minutes.

3 Methodology

As we have seen earlier, both traffic engineering and load adaptive routing mechanisms try to optimize the network load by influencing the routing of large traffic aggregates. This requires some care however, as it is important to distribute packets over the aggregates in a way that has as little influence on end-to-end network characteristics as possible, especially for constructs like, e.g., TCP connections or Voice-over-IP (VoIP) streams. Distributing packets of such an end-to-end communication stream over different aggregates and then using different routes for these aggregates can easily lead to packet reordering or significant variability in packet end-to-end delays. Both delay and jitter are known to have a highly detrimental influence on the performance and quality of such packet streams. One way to avoid these problems is to use network flows as the basic ingredient to build traffic aggregates. Network flows contain by definition those packets that have some common properties, which can be as precise as one direction of a bidirectional TCP connection or as universal as a destination AS. Therefore network flows have become one of the corner stones of traffic engineering. But using network flows can still have its problems in terms of scalability. Depending on the granularity of the used flows, there can be a huge number of them at any given time. In the worst case, one can get one network flow per packet, e.g., for DNS requests. The way that traffic engineering mechanisms cope with this problem exploits the property of network flows, that only a small number of flows is responsible for most of the traffic. This phenomenon, also known as Zipf's Law for flow sizes, allows us to concentrate on a small number of flows while ignoring the rest, and still be able to control the vast majority of the traffic.

In this section we introduce our methodology to analyze network traffic in order to identify and characterize properties of flows that need to be captured by our workload generator. We describe what data we base our analysis on, cover the problems of using network flows for our purposes and show the feasibility of our approach.

3.1 Overview

As we have seen in chapter 2.5.3, traffic engineering applications rely on the heavy tailed nature of flow rates in order to be able to act on large portions of the network traffic by concentrating on only a small number of flows, the heavy hitters, and treating these flows in some special way. Because of the high variability inherent in Internet traffic, dividing network flows into classes like elephants or mice, is a non-trivial task. Because of the self-similar nature of network traffic, flows that are classified as elephants at one point in time can drop their rate drastically and would be classified as mouse some short time later. The result of this volatility in the rate of network flows would then lead to instable classifications.

Therefore relying on this phenomenon as the foundation of operation critical processes makes it necessary to carefully examine the characteristics of heavy hitters and the influence these characteristics exert on the procedures used in traffic engineering. We now turn our attention on how the set of the largest flows behaves at some point in the network and try to identify invariants in the characteristics of this set that either are helpful or harmful for the effectivity of the algorithms used in applications like traffic engineering or load adaptive routing and should therefore be reproducible by the traffic workload generator of our simulation environment.

Accordingly, we concentrate our analysis on heavy hitters and on their dynamics along two dimensions, corresponding to heavy tailed nature of flow rates or flow sizes on the one hand, and the effects related to Zipf's Law for flow sizes on the other hand.

We start this chapter by giving an overview of flow classification methods used to identify a small set of relevant flows. Then we explain our offline analysis approach based on slicing time into bins and derive flow rankings based on per-bin byte contributions. After introducing the trace data we use for our analysis and the necessary preprocessing steps to insure comparability between the different trace types, we show that the feasibility of our methodology relying on less fine grained network flows to perform an in-depth analysis of the behavior of large Internet streams.

3.1.1 Elephants and Mice

When analyzing the properties of large flows, we first need a proper definition for what a large ('elephant') or small ('mouse') flow exactly is before we can isolate such flows in the measurement data. A way that is often used to define elephants is based on traffic fractions. In this case, the *n* largest flows that together account for more than some fraction of the total traffic during a time interval, e.g., 80%, are considered to be elephants. All other flows are then considered to be mice. Instead of using a fraction of the traffic to classify flows as elephants, it is also possible to work instead with a per-flow rate as a threshold. Estan et al. [EV02] define an elephant as a flow whose rate is larger than 1% of the total link utilization. All flows with rates beyond this threshold are then considered elephants. There exist other schemes to classify flows besides using flow rates. In [CB02] Brownlee et al. classify flows by duration into short lived ("Dragonflies") and long lived flows ("Tortoises"). Papagiannaki et al. [KTB$^+$02] use a two feature classification scheme based on both rate and persistence in time. Sarvotham et al. use the burstiness of flows to discern between short high rate flows (alpha traffic) and longer flows without pronounced spikes (beta traffic) [SRB01]. Another classification by burstiness is proposed by Lan and Heidemann based on three different types of burstiness, namely variance burstiness, RTT burstiness and train burstiness [KCL06], referring to very bursty flows as "porcupines". The authors also show in [KCL06] that there exists a strong correlation between the rate and the size of a flow, which explains why the different definitions of elephants that are sometimes based on the flow rate and sometimes on the flow size do not lead significantly different findings.

Adhering to one of the above definitions for elephants might limit our analysis by omitting flows that are, by that definition, not elephants but still might provide insightful observations. We therefore forego an exact definition prior to examining the properties of flows that may suggest a new classification method or reaffirms one of those above. Instead we cover as many of the largest flows as our resources allow us to do. In this way we get the best possible coverage of flow properties of large flows, even if we waste some resources by analyzing much more data than we may actually have to.

3.1.2 Offline Analysis of Network Flows

Independent of the classification scheme used to identify a small set of high rate flows in the overall traffic, the classification of flows itself and the analysis of the properties of the relevant flows is a very resource intensive endeavor.

To avoid having inconsistent flow classifications it is necessary to not only look at the immediate present but to the entire history of flows. If we classify and analyze flows in real time, we do not know the entire history of a flow, but only their past. In such cases, it is possible to predict the future by extrapolating the past. Here one could use statistical methods like, e.g., weighted moving averages [PTB$^+$01, KTB$^+$02] or autoregression based approaches (ARIMA). There are also proposals to use Bloom filters [KXLW03] or sample and hold filters [EV02] to be able to cope with high data rates and large amounts of traffic in real time. These methods however trade accuracy for scalability and thus also provide only approximated results.

These issues might be acceptable when one has to react quickly upon identifying an elephant flow, e.g., by adjusting the routing or for using network flows as the basis of some accounting scheme. Still, using one of these approaches, we would base our analysis on somewhat imprecise flow properties. Moreover, real time analysis needs a large amount of system resources, even when using approximation algorithms, as one has to cover a potentially large number of flows at the same time, consuming system memory and CPU cycles while having to be fast enough to keep up with the amount of monitoring data arriving at the same time.

We therefore chose to follow the path of offline analysis. Analyzing monitoring data offline has several advantages over real time analysis. First, we are not limited in our classification and analysis by system resources and time constraints as we are in the real time case. We are able to spend more time on identifying elephant flows and on analyzing flow behavior. This and the availability of complete per-flow monitoring data allows for a more precise and reliable identification and analysis of elephant flows than what is possible in real-time using methods that can only estimate flow properties and are thus less accurate. Another advantage of offline analysis is the possibility to repeat analysis steps of the same data set using different time scales and different parameters, e.g., different flow aggregation schemes. If the monitoring data is available as packet level trace, it is possible to derive flow properties that are not available in NetFlow records. The most important of these properties are the fluctuations of flow rates over time. NetFlow records provide flow rates only as an average over the flow's lifetime.

3.1.3 Binning: Slicing Network Traffic by Time

Given the above decisions to avoid a limiting pre-definition of elephants and mice and to perform our analysis offline, both providing a maximum of flexibility, we still need a basic common strategy for dealing with measurement data. One of the most fundamental aspects of our analysis of flows is their behavior over time. Note that with packet switched networks like the Internet it makes no sense to look at discrete points in time as at most a single packet belonging to only one flow would be visible at any given point in time. Therefore, in order to be able to observe how flows change their properties during their lifetime and also to compare the behavior of different

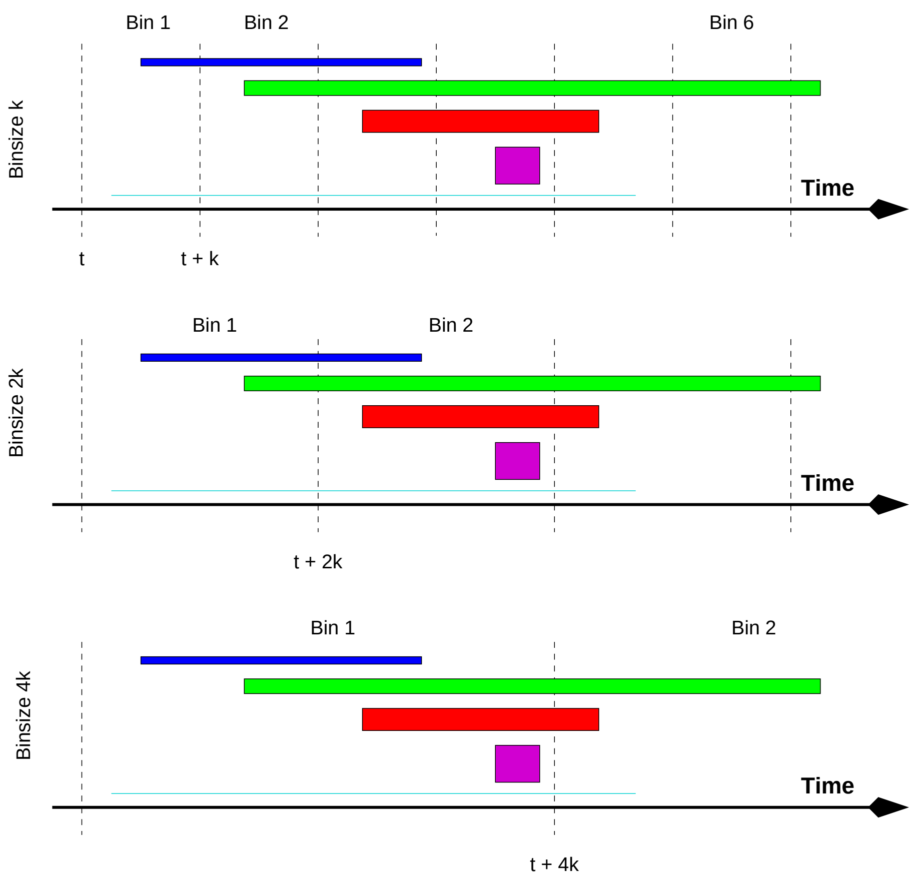


Figure 3.1: Slicing time into bins. The sizes of bins are multiples of powers of 2. Two bins of size k build a larger bin of sizes $2k$.

flows, even under alternative flow definitions or flow aggregation schemes, we divide the time line into constant size intervals called *time bins* or *bins*.

In order to allow for exploration of the influence of time granularity of flow behavior and thus of a sensible engineering rate for controlling flows, bins always start and end on a fixed raster. This raster is defined in a way, that a bin of size $2k$ seconds contains exactly 2 bins of size k seconds, and so on (see Figure 3.1). By looking into the per-bin byte and packet contributions of flows, we get sequences of flow rates, building the cornerstone for the analysis of per-flow behavior over time. On the other hand we can also study the interaction of flows by means of comparing the flow rates during a single bin and also during a sequence of bins (see Section 3.1.4).

Although this approach allows us to analyze flow behavior over time, it also has its drawbacks, partly due to the nature of flows itself, partly as consequence of the fixed raster used to enable us to perform comparable analysis for different time granularity. The first problem is based on the nature of network flow statistics collected on a flow probe (see Section 2.2.3). We only have data on when the first and the last packet of a flow passed the flow probe and how many packets and bytes a flow contained in total. So the only information we have on a flow's rate is an average value, calculated as

$$\text{avg. rate} = \frac{\text{bytes}}{\text{flow end} - \text{flow start}}$$

and the average per-bin contribution as

$$\text{avg. contrib} = \text{avg. rate} \times \text{bin size}$$

We have no information on whether the flow rate was really approximately constant around this average rate or not. There might have been peaks and dips or a drop in the flow rate at some time. The entire dynamics of a flow's rate during its lifetime is lost when the flow with its statistics is being built by a flow probe. In order to assess the influence of this problem on our analysis, we use packet traces with their high level of accuracy to show, that the effect of using only average flow rates is less pronounced than one might think, depending on time granularity (bin size) and aggregation scheme used (see Section 3.4).

The second problem arises from the fact, that bins start and end at a raster, but flows do not. Flows may begin or terminate at any given time within some bin. This leads to flows starting during a bin, thus raising the problem of how to determine the flow rate for the first bin (see Figure 3.1). The same happens when a flow ends. For the solution to this problem, we make use of a flow's average rate. Using the average flow rate and the fraction of the duration of the bin during which the flow was alive, we can calculate how much bytes and packets the flow contributes to the bin. The exact relation for the first bin that a flow contributes to is given by

$$\begin{aligned} \text{contrib}_{\text{start}} &= \frac{\text{bin end} - \text{flow start}}{\text{bin size}} \times \text{avg. contrib} \\ &= \frac{\text{bin end} - \text{flow start}}{\text{bin size}} \times \text{avg. rate} \times \text{bin size} \\ &= \frac{\text{bin end} - \text{flow start}}{\text{flow end} - \text{flow start}} \times \text{bytes} \end{aligned}$$

Finally, the bins should not be too small. The end of a network flow is defined via the inactivity timeout (see Section 2.2.1), therefore it is not possible to chose a bin size that is smaller than this timeout. This lower bound on bin sizes ensures, that each flow contributes to all bins during it's lifetime. Otherwise, an active flow might disappear from a bin, even when it has a high, although bursty, data rate and thereby introducing artifacts into the analysis and distorting the results.

3.1.4 Flow Rankings

Analogue to the way we slice time into bins to be able to look into time dependent properties of flows, we also slice the total traffic into slices using the flow abstraction. This enables us to examine the ways that flows interact with each other. Because of the high variability in flow arrivals, flow departures and the rates of flows, we perform this slicing on a per time bin basis. This allows us to examine in which way the overall traffic is comprised of flows at any time interval. The actual definition of flows can be chosen in advance, making the slicing very flexible. Possible ways to define flows are, e.g., five-tuple flows or destination prefix flows (see Section 2.2.1).

In order to analyze the behavior of heavy hitter or elephant flows, we compute flow rankings for each time bin for all flows active during that bin. This allows us to identify these kind of flows and concentrate our analysis on them. For our analysis we define a ranking as an ordered set R of flows f with

$$r_i < r_j \quad \Leftrightarrow \quad \text{contrib}(f_i) \geq \text{contrib}(f_j)$$

where r_i is the rank of flow f_i and $\text{contrib}(f_i)$ is the contribution of the flow to some bin. Note that $r_i = 1$ represents the flow that contributes the most and therefore has the highest rank. The smaller the rank index i, the higher is the rank and the contribution of the flow at rank r_i to the overall traffic. The ranking is based on the contribution of each flow to the overall traffic during the time bin. Usually the contribution is determined in terms of number of bytes, although it is also possible to use number of packets. As network flows only provide us with total number of bytes and timestamps for the first and the last packets of a flow, and thus with a flow rate averaged over the flow's lifetime, we have to calculate the actual per-bin byte contribution using this average flow rate (see Section 3.1.3). When considering aggregated flows, we calculate the per bin byte contribution as the sum of the per bin byte contributions of all five-tuple flows that are accumulated into the same aggregated flow. As the number of five-tuple flows that together make up an aggregated flow is not constant over time, the rate of an aggregated flow is no longer constant over time but may a significant variability in its rate.

As seen in Section 2.4, we expect flow rates and thus the per-bin contribution to the traffic to be consistent with a Zipf's-like distribution. This is the basis of current traffic engineering and load adaptive routing algorithms. If this is indeed the case with our data we can cut down the size of the per-bin rankings to a small number of the largest flows. How many of the largest flows are necessary to cover a significant portion of the total traffic is highly dependent on the respective measurement points and the nature of the traffic crossing that point. One way to solve this problem is to adjust the size of the rankings so that always a minimum fraction of the traffic, say, e.g., 75% is covered by the ranking. But for the first iteration, we chose to use constant size rankings with the number of ranked flows small enough to be handled with relative ease, e.g., 100, 1000 or 10000 flows.

An example of how the ranking works for up to three flows is shown in Figure 3.2. On the x-axis we plot time and the y-axis represents traffic volume. The figure shows three time bins of size k, starting at times t, $t+k$, $t+2k$ and $t+3k$ respectively. For each bin the three flows with the highest contribution in volume are colored. Each color represents one flow, so the plots contains five different flows. During the first bin, the "green" flow contributes the most, hinted at by having the largest area for the first bin. Thus this flow has rank one and is shown at the top of the flows in this bin. The second most contributing flow for the first bin is the "red" flow and is thus ranked second. The "blue" one with the third highest volume has rank three. The traffic volume in this bin that is not contributed by the three flows with the highest contribution is represented by the shaded area and is comprised of all the remaining flows. These remaining flows only contribute a small part of the total traffic volume and the per-flow contribution for

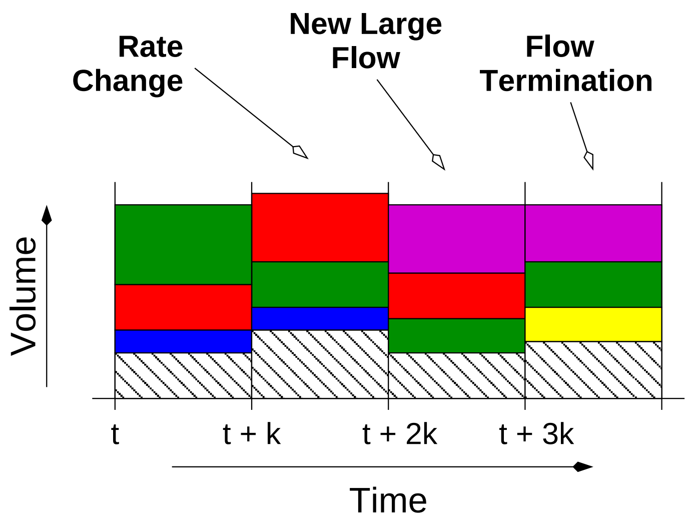


Figure 3.2: Ranking of flows using time bins. Flows are ordered by per-bin byte contribution with the flow with the higher contribution having a higher rank (smaller rank number). Ranks can change by flow rate changes, new flows arriving or active flows terminating.

these flows is so small that the effort to further refine the ranking is not worthwhile. After all we are able to cover three quarters of the total volume by looking at three flows only. In the next bin, starting at time $t + k$, the green flow has reduced its contribution and is now ranked only second, while the red flow has increased its contribution and is now rank at 1 as the flow with the highest traffic contribution. The blue flow has not changed its contribution and is still ranked third. In the third bin a new flow appears in the ranking, colored magenta. This flow has a higher contribution than each of all the other flows and is thus ranked first in this bin. As we only look at the three most contributing flows, the blue flow is now no longer visible and is represented, together with all the rest of the flows, by the shaded area. The blue flow has been pushed out of the ranking by the magenta flow. In the last bin, starting at $t + 3k$, the red flow has disappeared, either because its contribution to this bin has dropped or it has terminated entirely. This leads to the green flow to raise its rank to second and a new flow, colored yellow, to appear in the ranking at rank 3.

The above example shows that there are two major components of variability in the flow rankings over time. The first component is change in flow rates and can cause a flow to raise in the ranking if its rate and thus its per bin contribution grows, or even result in flow with very low rate to newly appear in the ranking. A similar effect can happen when a flow's rate decreases and the flow drops in rank or even disappears from the top ranked flows. As all flows can be expected to change their rates over time at least somewhat, this leads to a very complex interaction between some flows increasing their rates and some decreasing them. Again, due to the heavy tailed distribution for flow rates, we expect that top ranked flows have to show much larger rate fluctuations to change their ranks than lower ranked flows do. This should lead to different dynamics in the top ranks than for the lower ranks. The second component for changes in the rankings stems from high-rate flows terminating or of new ones to start and, due to high flow rates, appear in the top ranks while pushing the lowest ranked flow out of the ranking. The dynamics of this component of ranking volatility depend on the arrival and departure process of flows and whether this process is different for high rate flows than it is for low rate ones. Prior work by Lan and Heidemann [KCL06] suggests, that there is a strong correlation between flow rate and flow size and also between flow rate and flow duration [ZBPS02, KCL06]. Because these effects have a fundamental influence on the dynamics of flow rankings and thus on rank based flow classification algorithms, we explore these dynamics in more detail in section 3.4 before we base any analysis results on flow rankings.

3.2 Available Trace Data

We base our analysis on multiple data sets, taken at different points in the Internet. We also try to cover different types of networks, e.g., backbone links versus access network uplinks. Moreover, as it is not always easy to gain access to high resolution monitoring data, we want to assess the influence of less detailed monitoring data on our analysis. Therefore we apply our analysis methodology to data sets with different levels of detail, from packet traces to NetFlow records built from sampled packet streams and look into what influence a lack of detail has on the results of our analysis.

In this section we now introduce the data sets that build the foundation of our analysis scheme and are also the base for the feasibility evaluation for performing the kind of analysis described in section 3.1 using NetFlow data.

Trace	Type	Sampled	Duration	Records/Packets
NF-I	NetFlow Version 5	no	24 hrs.	211 million records
NF-II	NetFlow Version 5	1:100	24 hrs.	330 million records
WASHng	NetFlow Version 5	1:100	24 hrs.	180 million records
MWN-I	packet level	no	4 hrs.	350 million packets
MWN-II	packet level	no	7 hrs. 30 min.	345 million packets
MWN-III	packet level	no	24 hrs.	2.9 billion packets
USB	packet level	no	3 hrs. 30 min.	100 million packets

Table 3.1: Network traces used in this thesis.

3.2.1 NetFlow Traces

In this section we introduce the NetFlow traces used throughout this thesis. The traces are grouped by measurement points.

Tier-1 Backbone Traces The first CISCO NetFlow trace NF-I was collected from a single backbone router within a Tier-1 ISP. The trace contains a day worth of CISCO Version 5 NetFlow data (see Section 2.2.3), collected on Dec. 11, 2001. The data set NF-I contains over 211 million

Prefix Length	/22	/23	/24	/25	/26	/27	/28	/29	/30	/31	/32
#prefixes	496	459	1791	42	41	30	3	9	30	4	12
affected prefixes	17.0%	15.7%	61.4%	1.4%	1.4%	1.0%	0.1%	0.3%	1.0%	0.1%	0.4%
all prefixes	3.6%	3.3%	13.1%	0.3%	0.3%	0.2%	0.0%	0.1%	0.2%	0.0%	0.1%
Volume [GBytes]	72.5	109.9	488.3	26.1	2.1	1.5	0.1	6.1	1218.1	0.0	0.8
affected prefixes	3.4%	5.1%	33.0%	1.2%	0.1%	0.1%	0.0%	0.3%	56.8%	0.0%	0.0%
all prefixes	0.4%	0.6%	4.0%	0.1%	0.0%	0.0%	0.0%	0.0%	6.9%	0.0%	0.0%

Table 3.2: Effect of Abilene anonymization method on trace quality

flow records or about 4 GB of compressed data. This NetFlow trace is unsampled and has a loss rate of 9% (mostly due to the capacity limits in the monitoring infrastructure, not in the ISPs infrastructure). A second day-long sampled NetFlow trace NF-II was collected on Sept. 5, 2002 and consists of almost 330 million flow records or more than 5.1 GBytes of compressed data. This trace contains NetFlow records built from sampled packet streams considering one in 100 packets.

Abilene Backbone Trace We use a third NetFlow trace, collected at the Abilene network. The Abilene network is an Internet2 research network located in the United States, connecting research laboratories and universities and is intended to enable the development of advanced Internet applications and the deployment of leading-edge network services for the Internet2 project. In contrast to the rest of the traces, this CISCO Version 5 NetFlow trace is accessible at the Abilene Observatory [Abi] for registered research projects. This trace, referenced as WASHng, covers 24 hours worth of data, collected using a packet sampling rate of 1 in 100 packets (see Section 2.2.1) on Nov. 2., 2005 on a core router in Washington, USA.

As this is a publicly available trace, the source and destination IP addresses have been anonymized. The anonymization has been performed by setting the low-order 11 bits to zero, limiting the granularity for network prefixes to /21. This means, that different prefixes that both have the same prefix mask length of more than 21 bits, cannot be distinguished from each other. In order to assess the influence of this anonymization method on the accuracy of our analysis, we look into how many distinct prefixes are affected by zeroing out not only host- but also prefix bits and how much of the traffic is directed to affected prefixes.

The trace contains 13706 different prefixes. Of these prefixes, 2917 (21.8%) have a length of more than 21 bits and are thus affected by the anonymization. In terms of bytes the trace contains flows that account for 15.9 Terabytes of traffic, 12.1% of which are destined for affected prefixes.
In Table 3.2 we show the effect of anonymization on the Abilene trace WASHng. Affected prefixes are those that have a longer prefix mask length than 21 bits. The table shows, that most of the affected prefixes have netmask length of 24 or smaller, with /24 prefixes accounting for more than 60% of the affected prefixes and only 13.1% of all prefixes. When looking at the amount of data sent to the affected prefixes, most of the bytes go to /24 and /30 prefixes.

As a consequence of the effects of anonymization, we avoid using this trace to analyze 5-tuple flows. On the other hand, doing analysis on flows aggregated by destination prefix, both with fixed netmask length, e.g., /24 or /16, and with netmask length as contained in the flow records, is not affected in a major way.

3.2.2 Packet Traces

Because of their usually very large volume, we in this thesis use packet level traces mainly for validation of the feasibility of using network flows for our analysis. We also use packet level traces in cases where NetFlow data was not available. As all available packet level traces were captured at access links with only a limited host population on the inner and the rest of the Internet on the outer side, one needs to be careful when considering destinations for incoming and sources for outgoing targets for analysis. This property of the packet level traces has however no influence on the validity of the evaluation of our methodology.

Münchner Wissenschaftsnetz (Munich Scientific Network) Of the four packet level traces used for this thesis, three were collected at the access link between the Munich Scientific Network (Münchner Wissenschaftsnetz, MWN [MWNa, MWNb]) and the Internet. The MWN, operated by the Leibnitz-Rechenzentrum (Leibnitz-Computing Center, LRZ) provides access for two major universities (Technische Universität München and Ludwig-Maximilian Universität), and several Max-Planck- and Fraunhofer Institutes to the Internet via the Deutsches Forschungsnetz (German Research Network, DFN). It consists of about 50000 individual hosts. The uplink to the Internet has been upgraded during the development of this thesis. Starting with a 622 Mbps PoS link, the uplink has been changed to 1 Gbps Ethernet in 2003. On average the uplink carries about 60 TBytes of data per month as of 2005. The traces were collected using the monitoring facility of the core switch, duplicating the traffic on the uplink to a dedicated 1Gbps fiber to a capturing machine. Although it is in principle problematic to copy both directions of a monitored 1Gbps link onto a single 1Gbps fiber, the peak traffic rate observed using the accounting mechanism of this switch show, that our traces do not suffer from overload of the monitoring port and resulting packet drops. The first trace collected at the MWN, called MWN-I throughout this work, was captured on Friday, Oct. 31, 2003, 11:17–15:22 local time in Munich. This trace contains only packets from and to the CS department of the Technische Universität München. It consists of about 350 million packets or more than 10.3 GBytes of compressed data. The second trace, MWN-II, was gathered at the same location but this time capturing all packets crossing the monitored link without limiting traffic to or from a subset of the internal hosts. It contains all traffic starting from Wednesday, Nov. 13, 2003 , 19:10 up to 02:43 the following Thursday morning, amounting to about 345 million packets or 2.5 GBytes of compressed data. The third trace, MWN-III, contains 24 hours worth of data and was collected on Nov. 15., 2005. This trace is comprised of about 2TBytes worth of packets. As this trace covers an entire day and does thus not limit the maximum flow durations more than any of the NetFlow traces, we use this trace, along with the NetFlow traces, for studying network flow properties.

Universität des Saarlandes (Saarland University) The fourth trace, USB, was collected at the 155Mbps link between the campus network of the Universität des Saarlandes, Saarbrücken, Germany and the DFN backbone. The campus network connects about 10000 hosts using a 1Gbps backbone and generates about 7TB of outgoing and 3TB of incoming traffic per day. The trace was captured at the university uplink on Tuesday, Feb. 02, 2003 between 12:00 and 15:29. It contains approximately 100 million packets or some 2.5 GBytes of compressed data.

3.3 Data Preparation

The traces available for our analysis show a high degree of diversity. Not only do the traces originate from very different network setups, they also come in two entirely different classes of granularity. On the one hand, we have NetFlow traces as exported by CISCO routers. As we have seen in Section 2.2.3, this implementation of the network flow concept comes with certain drawbacks when compared to the theoretic flow model. On the other hand, we use packet level traces. In order for all traces to be comparable, we first have to build network flows from the NetFlow record and packet level traces. In this section we describe how to construct network flows from the available packet level traces and how to cope with the peculiarities of CISCO NetFlow traces.

3.3.1 NetFlow Normalization

All the NetFlow traces we use come from CISCO routers, that construct flows from packet streams in real time. The routers have to do this using limited resources especially in terms of memory. In order to keep the memory pressure low, routers try to export NetFlow records as early as possible. This is achieved by immediately exporting flows that are idle for some time or where the protocol allows for a detection of flow termination, e.g., TCP's FIN and RST packets. A minor problem is the retransmission of such termination packets. Every packet following the last one is exported as a new flow consisting of a single packet. Two much more serious problems are caused by heavy usage of router memory, either by very long flows or by a very large number of flows. In the case of very long flows, routers export portions of a flow, when it has been active for a longer time as the *activity timeout* specifies. The incentive for this behavior is to provide applications with at least near-time information about the network conditions. The second reason for exporting portions of non-terminated flows is memory pressure in the presence of a high number of active flows. In such a situation a router expires active flows following heuristics that try to reduce memory consumption as efficiently as possible.

Unfortunately, exporting a long flow as several flow portions leads to problems in the analysis process, e.g., when looking at the duration of flows. In [SF02] the authors had to cope with the same problem while generating connection summaries from NetFlow records. In order to capture all flows belonging to the same connection, the authors sometimes had to recombine NetFlow records with the same source and destination IP addresses and port numbers to capture all NetFlow records belonging to a connection even if the the time gap between the end of one record and the start of next one exceeded the inactivity timeout value. In contrast to this connection oriented point of view on network flows, we are interested in the behavior of network flows as defined by the more theoretic approach described in Section 2.2.1. Therefore, we recombine NetFlow records that in principle belong to the same network flow but have been exported as several portions because of router memory shortage or high flow duration. We call this procedure *flow restitching*.

We perform flow restitching on five-tuple flows only. A restitched network flow thus consists of all NetFlow records that have the same source and destination IP addresses and port numbers and the same transport layer protocol type and in addition never have gaps between two chronologically consecutive NetFlow records that are longer than allowed by the inactivity timeout value. The procedure of flow restitching is illustrated in Figure 3.3. The figure shows five NetFlow records with the same source and destination IP addresses and ports. The earliest record to the left side, labeled **A**, has a duration that is equal to the activity timeout configured on the exporting router. We assume that NetFlow records of at least this duration are exported because of the configured activity timeout and are thus still active at that time. This means that we expect another NetFlow record, that is the continuation for this flow. In our example, such a NetFlow record is depicted by the record labeled as **B**. This record follows the first record within a short time period, one that is significantly smaller than the configured inactivity timeout. Another reason for a router to export a record for an active flow is memory shortage. In this case, the router prematurely exports a record for an active flow to free memory for new flows. A record that is exported for this reason will, if it is not the first record for a flow, follow another record in less time than the inactivity timeout would suggest. Such a NetFlow record is shown as record **C** in Figure 3.3. The record **D** starts later after the end of record **C** than the inactivity timeout demands. This means, that the network flow represented by the records **A** to **C** ends and record **D** is the beginning of a new network flow. The restitching process therefore merges the records **A** to **C** into a single network flow, that starts with the beginning of record **A**, terminates with the end of record **C** and whose byte and packet counters are the sum of the bytes and packets of the three NetFlow records. A third case, where we merge multiple NetFlow records into a single network flow happens, when parts of a flow carry different type of service (TOS) bits [RFC81a] than the rest of the flow. As CISCO routers do not only use source and destination addresses and ports and the transport layer protocol, to build NetFlow records but also the TOS bits, we merge NetFlow records that fulfill the timing requirements according the two timeouts. Such a case is shown in Figure 3.3 by the records **D** and **E**. Restitching such records is more difficult, as they may overlap in time. In our example, record **E** is completely contained in record **D**, so restitching can be done by simply taking the start and end time of record **D** and sum up the bytes and packets of both records. In general however, we restitch such records into a network flow, that begins with the minimum of the start times of all related records and ends with the maximum of the end times. Summing up packet and byte counters of all the pieces of a network flow leads to loss of flow rate information, if the flow rates for the pieces are different. But as we have to be able to cope with the fact, that we only have flow rates averaged over a flow's lifetime to work with anyway, this does not further complicate the handling of network flows.

Restitching of NetFlow records in this way normalizes the available NetFlow traces and results in network flows that are consistent with the theoretical model and are thus comparable with flow traces collected under different network conditions, independent of flow duration distribution or router architecture or hardware configuration.

3.3.2 Generating Network Flows from Packet Level Traces

Although our analysis concentrates on the properties of network flows we do not use only flow traces but also rely on packet level traces. We do this both to broaden our data basis to ensure

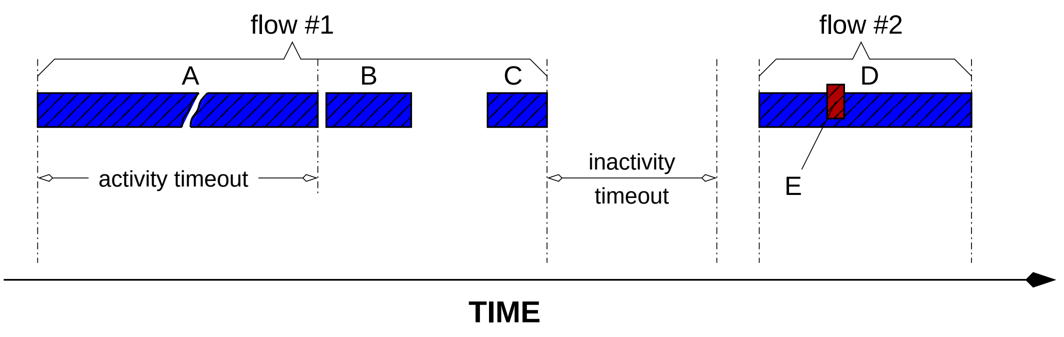


Figure 3.3: Flow restitching. Record B is caused by the activity timeout. Record C is the result of CISCO's export mechanism and starts before an inactivity timeout occurs. Record E has the same five tuple values as record D, but different TOS bits.

better representability of our results as well as to show that it is possible to do traffic analysis using network flows. As we cannot use packet level traces directly in our analysis, we first have to construct network flows from the packet level traces. We construct two types of flows:

The first type are five-tuple flows according to the theoretical model. These flows can be treated in the same way as the normalized flows generated from NetFlow records by restitching. Note, that we can generate network flows from packet traces without having to artificially split flows into several pieces. As we have more memory resources than a real router and because there is no need to report long flows by premature export, we do not have to expire active flows. This way, we can skip the restitching step that is needed for NetFlow records. We can expire flows based on idle time alone. But as we cannot derive additional information like, e.g., prefix mask lengths from the packet level traces, we have to consult BGP [Ste99] tables from either the router used to collect the trace or, in the case the trace has been collected using a passive network tap, from an router adjacent to the tap. Using BGP tables, we can complement the generated flows with AS numbers and prefix mask lengths both for source and for destination addresses.

The second type of flows is intended for the evaluation of the quality of NetFlow based traffic analysis. As already shown in Section 2.2.1, flows contain only averaged information on the flow rate, based on time stamps and packet and byte counters. On the other hand, the fine grained packet level traces can provide us with very accurate flow rate information. In order to use this information to obtain more fine grained flow rates for network flows, we generate, along with the five-tuple flows, short flow fragments. These fragments are oriented along the same time raster that we use for the time bins (see Section 3.1.3). This way, we have precise flow rates that match the finest time granularity we use for our flow analysis. A more fine grained time resolution than the smallest bin size is not needed to obtain exact values for flow rates during a bin and thus the exact traffic contribution of a flow during a time bin. In the remainder of this thesis, we refer to raw, non-aggregated five-tuple flows also as *RawFlows*, while we will refer to the flow fragments as *FragFlows*.

In order to be able to aggregate flows by, e.g., destination IP prefix we need to fill in the prefix mask length fields of the flows with correct values. Therefore, we extracted BGP tables from the

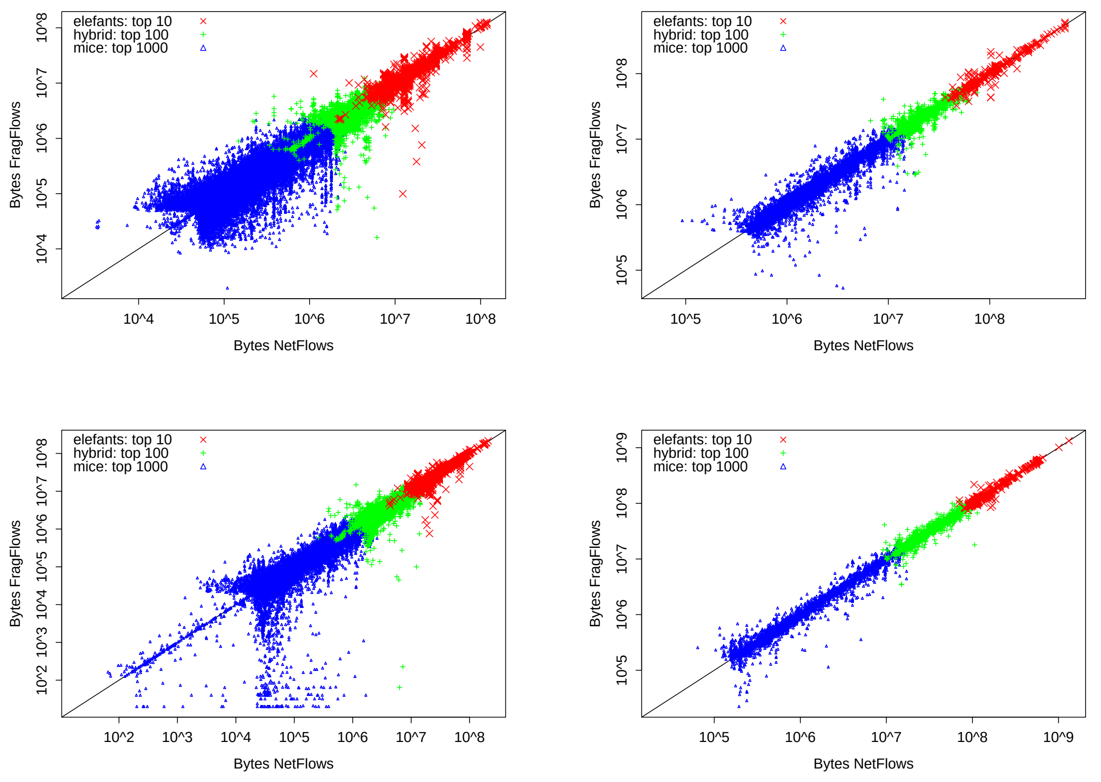


Figure 3.4: Scatterplots comparing the per-bin byte counts for NetFlows and FragFlows (top: IP fbws, bottom: aggregated destination prefi x fbws using a fi xed 16 bit mask; left: bin size = 60 sec; right: bin size = 480 sec). Already strong concentration around diagonals, especially for top ranked fbws, increases with larger bin sizes and higher aggregation levels.

corresponding access routers, reconstructed the routing tables from the BGP information and then used longest prefix lookups into these routing tables to get per IP prefix mask lengths for inclusion into the generated NetFlows. As both collection points, MWN and USB announce the interior networks in only a few large blocks, this procedure is somewhat inaccurate for hosts within both academic networks. However the number of affected IPs is sufficiently small for this kind of error to be tolerable. In cases were this inaccuracy might have a noticeable impact on the aggregation results, we omit aggregation for internal hosts, e.g., by only considering destination prefix flows that are outgoing from the viewpoint of these networks.

3.4 Feasibility of NetFlow Based Analysis

Before embarking on a study of the persistence aspects of Internet flows, we first examine in this section the implications of the constant flow rate assumption and the impact of possible deviations of actual flow rates from a flow's average rate. As discussed in Section 2.2.2, this assumption is unavoidable when using non-aggregated NetFlow data in this context.

To this end, we rely on our packet-level traces for which we can derive both FragFlows (see Section 3.3.2) and RawFlows. We generate the rankings for both FragFlows and RawFlows and compare the resulting rankings in terms of per bin byte contribution and, as we want to concentrate on the dynamics of the rankings, also in terms of per bin ranks. After obtaining the per bin rankings for the top 5000 flows for both RawFlows and FragFlows at various abstraction levels, we match the appropriate RawFlows and FragFlows and compare them to assess the impact of the constant flow rate assumption on the validity and quality of our findings. More precisely, we select the top 1000 entries from each bin and locate the matching counterpart if it existed among the top 5000 entries. In order to assess the influence of different time granularities on our results, we perform this comparison for different small to medium bin sizes in the range of 1 to 8 minutes. The bin sizes are chosen in a way that one size is an integral multiple of the next smaller bin size. Starting with 1 minute as the smallest bin size, we additionally use the rather small bin sizes of 120, 240 and 480 seconds for the purpose of evaluation.

3.4.1 Per-Bin Byte Differences

We start by comparing how the per-bin byte counts of the FragFlows differ from those of the RawFlows for different time aggregations and flow abstractions. The working hypothesis is that we should expect differences, but that they will diminish as we consider larger bin sizes and/or higher flow aggregates. In this context, one of the objectives is to try and identify the causes of and quantify to some extent the expected differences for small bin sizes and unaggregated flows.

We expect the constant flow rate assumption for network flows to have its highest impact when comparing how many bytes a FragFlow is contributing to a particular bin and how many bytes the corresponding network flow contributes. To illustrate this comparison, Figure 3.4 shows scatterplots of the per-bin contributions of RawFlows (x-axis, log-scale) against the per-bin contributions of the corresponding FragFlows (y-axis, log-scale) for the trace MWN-I. The top row is for unaggregated RawFlows and bin sizes of 60 seconds (left) and 480 seconds (right), while the bottom row is for aggregated destination prefix flows using fixed 16 bit prefix masks and the same two bin sizes. Note that the plot only shows distinct points; duplicates are removed before plotting. We use different symbols to indicate the ranking that a particular point is associated with. A small "$\triangle$" corresponds to a byte count that has at least one ranking (FragFlow or network flow) in the top 1000 but none in the top 100. A "+" marks those byte counts that have one ranking in the top 100 but not top 10, and a "$\times$" identifies the byte counts that have a top 10 ranking.

The most pronounced feature in all of these plots is a strong concentration of the points around the diagonal, with varying degrees of deviation as we consider different bin sizes and/or flow abstractions. In relative terms, these deviations from the diagonal seem to be smallest for byte counts with a top 10 ranking and tend to get larger as we consider more of the more frequently occurring lower-ranked byte counts. Also, as we consider either larger bin sizes (left to right in Figure 3.4) or larger aggregation levels (top to bottom), or a combination of larger bins and aggregates (top left to bottom right), the concentration along the diagonal is accentuated. As far as increasing the bin size is concerned, one explanation for this observation is that more flows will completely fall within a single bin, and that this feature impacts not only the many lowly-

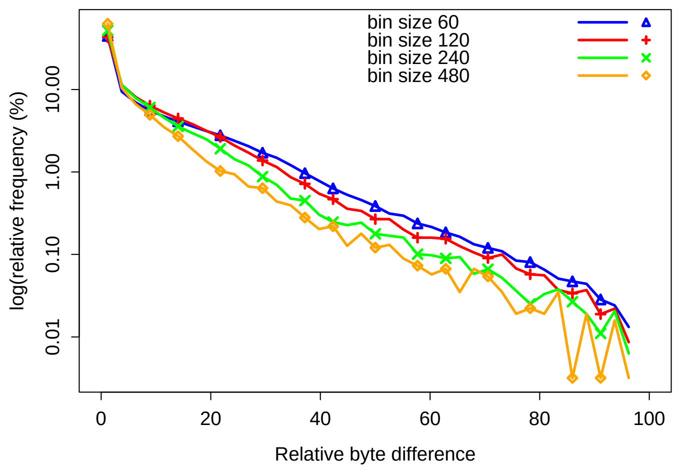


Figure 3.5: Histogram of the relative byte differences between FragFlows and network flows with linear x- and logarithmic y-axis (non-aggregated flows, bin size = 60, 120, 240, 480 sec). Relative error is mostly small and further shrinks as larger bin sizes are considered.

ranked flows whose durations tend to be shorter than those of the few top-ranked flows. In terms of increasing the level of flow aggregation, the variability of the per-bin byte counts of large aggregates is bound to decrease as predicted by the Central Limit Theorem (see, e.g., [JS87]).

To quantify the degree of (in)accuracy of the approximation resulting from the constant flow rate assumption, we compute for each per-bin byte count the relative byte difference between the FragFlow and the network flow entries. This is done for each bin and flow by taking the absolute value of the difference between the two byte counts, dividing it by the maximum of those two values, and multiplying by 100 to get percentages. Figure 3.5 shows histogram plots of the relative byte differences for different bin sizes and illustrates that the error caused by the constant flow rate assumption decreases with bin size. Overall we observe that—as we expected—the accuracy of using the more widely available network flows instead of the hard-to-come-by FragFlows increases with bin size and with aggregation level, implying that the constant throughput assumption may be appropriate for certain flow abstractions.

While concentration around the diagonal in the plots in Figure 3.4 is highly desirable, points that clearly deviate may also be informative, especially if they concern top-ranked byte counts, and deserve closer inspection. An obvious artifact in Figure 3.4 are the vertical bands that are associated with one and the same network flow byte count and result from having in general many different FragFlow byte counts for the same flow. For example, in the top left plot in Figure 3.4, we can identify 18 bins associated with 16 flows where the difference in FragFlow-vs. flow-derived byte counts was large enough to cause the byte counts to be classified as top 10 for FragFlow and as top 100 for Flow, or vice versa. Of these 18 “outliers”, 14 occurred at the start (6) or the end (8) of the flows. Possible explanations include TCP slowstart and initial protocol overhead for those at the beginning of their lifetime, and timeout effects for those at the end.

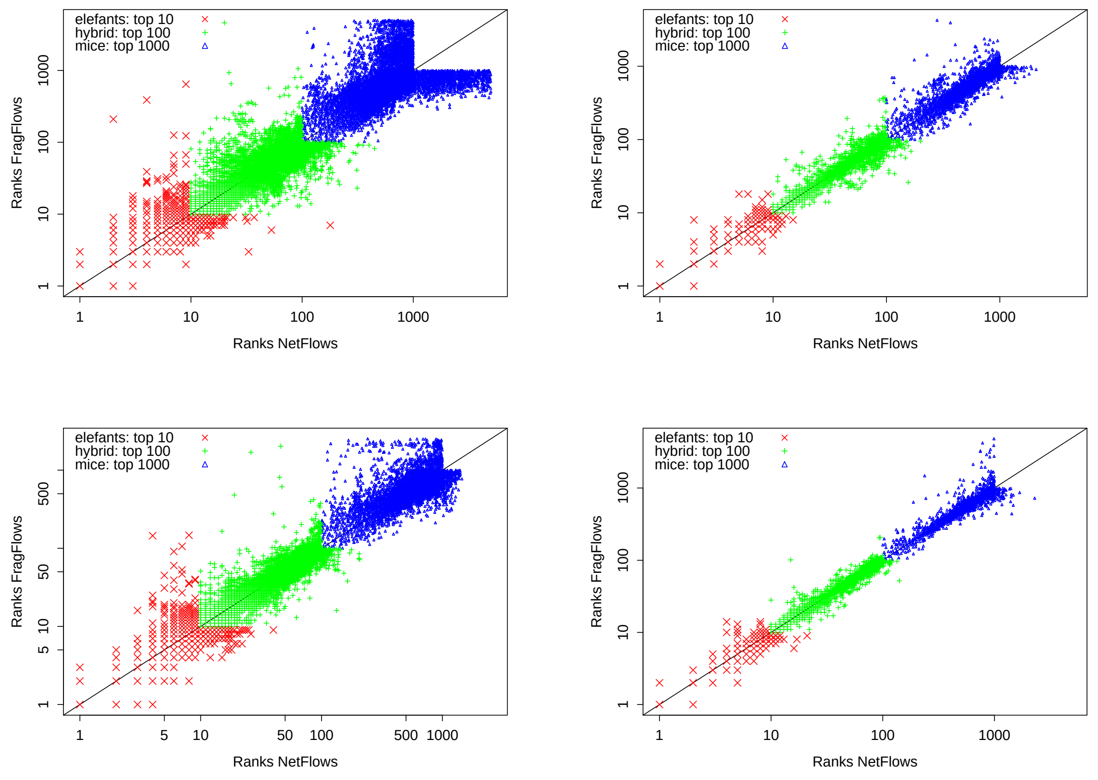


Figure 3.6: Scatterplots comparing the per-bin ranks for network fbws and FragFlows (top: non-aggregated fbws, bottom: destination prefi x fbws using a fi xed 16 bit mask; left: bin size = 60 sec; right: bin size = 480 sec). The deviation in ranks decreases with larger bin sizes, higher aggregation levels, and higher ranks (smaller rank numbers) where the influence of the rank on the error is somewhat hidden by the logarithmic scales.

3.4.2 Per-Bin Rank Differences

Next we examine what impact the observed differences in per-bin byte counts that are the result of the constant throughput assumption have on the per-bin ranking of the flows. While the previous subsection focused on the impact of the constant flow rate assumption on byte counts, we now examine the differences that are imposed by this assumption on the ranking. The raw rank data derived from the MWN-I trace are given in Figure 3.6 which shows scatterplots of the per-bin network flow derived ranks (x-axis, logarithmic scale) against the corresponding FragFlow derived ranks (y-axis, logarithmic scale). As in Figure 3.4, the top row is for unaggregated flows and bin sizes of 60 seconds (left) and 480 seconds (right), while the bottom row is for aggregated flows using a fixed 16 bit mask and the same two bin sizes. The symbols have the same meaning as in Figure 3.4, but note that the top-ranked per-bin flow rates are now concentrated in the lower left rather than in the upper right corners of the four plots as high byte counts correspond to small rank numbers. After accounting for the artifacts caused by selecting only the top 1000

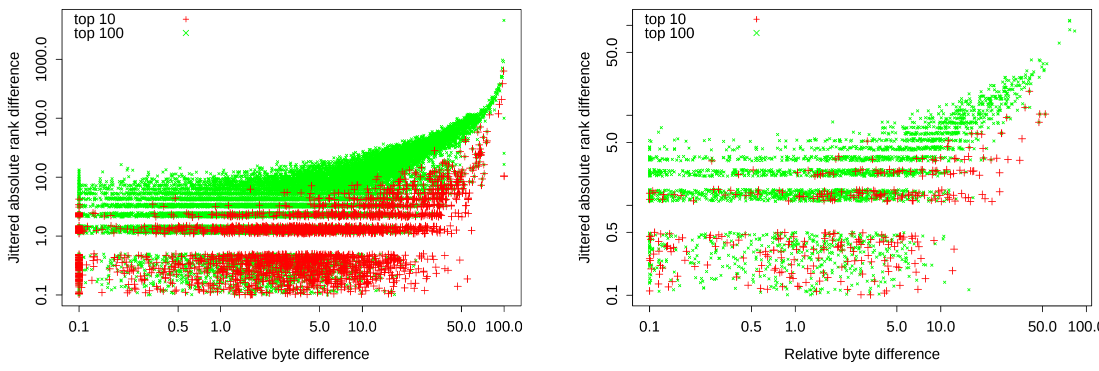


Figure 3.7: Scatterplots of the relative per-bin byte count differences between network fbws and FragFlows vs. jittered absolute per-bin rank differences between FragFlows and network fbws (left: RawFlows, bin size = 60 sec; right: aggregated destination prefi x fbws using a fi xed 16 bit mask, bin size = 480 sec). There can be relative byte count changes of up to 50% (20%) without a difference in ranks. The same relative byte difference leads to smaller absolute rank changes for elephants than for other fbws.

entries (cut off in the upper right corners) and by using a logarithmic scale in conjunction with ranks that can only have discrete values, the common dominant feature in these plots is again a pronounced concentration of the points around the diagonal, with some obvious deviations. However, these deviations from the diagonal diminish significantly as either larger bin sizes or higher flow aggregates are considered. The deviations are also smaller for higher ranks (smaller rank numbers). This fact is not readily discernible due to the logarithmic scales of the plots.

Before addressing the issue of how the constant flow rate assumption impacts the ranking of individual flows, we first examine how much of a per-bin rank difference can be expected as a result of a given per-bin byte count difference. In effect, in answering this question, we combine the information from Figures 3.4 and 3.6 to generate Figure 3.7. More precisely, to generate the relevant information, we consider the relative per-bin byte count differences instead of their absolute values because generally, for elephants, larger absolute byte changes are needed to switch ranks than for mice. At the same time, in terms of rank differences, it seems more sensible to consider absolute rather than relative rank changes. To be able to examine the relative byte differences in conjunction with the smaller absolute rank differences in more detail, we manipulate the data by adding a constant offset (jitter) of 0.1 to both the absolute rank differences as well as to the relative byte differences; we also introduce some jitter to the absolute rank differences by adding a uniform random amount between 0 and 0.4 to each absolute rank difference so as to avoid the situation that all points with the same (integer-valued) rank difference appear as a single point in the plot.

The resulting two plots, one for unaggregated flows and a bin size of 60 sec (left), and one for aggregated flows using a fixed 16 bit mask and a bin size of 480 sec (right), are shown in Figure 3.7. The two plots correspond to the top left and bottom right plots shown in Figures 3.4 and 3.6. The clearly visible band with (jittered) rank differences between 0.1 and 0.5 corre-

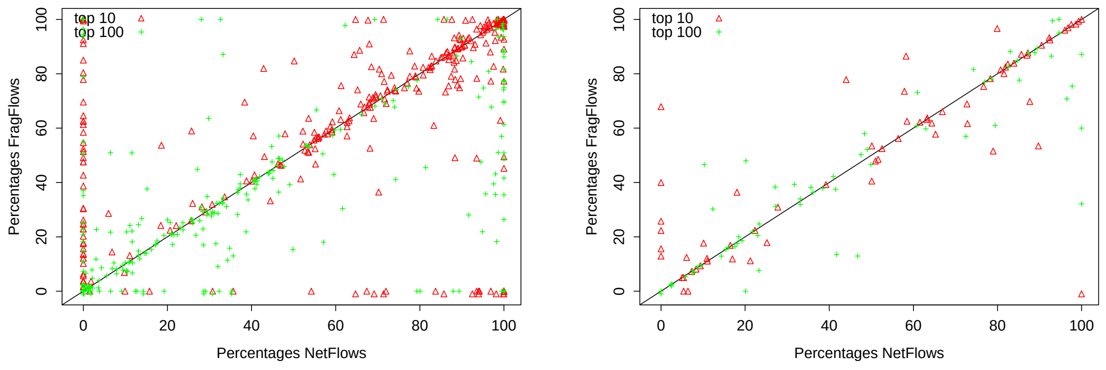


Figure 3.8: Scatterplots of percentages of bytes from elephant/hybrid bins for network flow-derived elephants/hybrids vs. percentages of bytes from elephant/hybrid bins for FragFlows-derived elephants/hybrids (left: RawFlows, bin size = 60 sec; right: aggregated destination prefix flows using a fixed 16 bit mask, bin size = 480 sec). The concentration around the diagonals mean the same ranks for FragFlows and network flows. The points at $x = 0$ represent bins where the flow rate is misjudged by network flows because of their inability to capture rate fluctuations. The points at $y = 0$ are caused by edge effects where the rank of network flows is just barely overestimated.

sponds to those bins where FragFlow and the corresponding network flow entries have the same rank (absolute rank difference of 0 plus jitter). It is interesting to note that in the non-aggregated case (left plot), rather large relative byte difference (up to 50%) can occur without influencing the rank too much. Once we include the next few discernible bands corresponding to rank differences of ± 1, ± 2, to ± 5 and so on, the number of top-ranked bins is drastically reduced, more so for the aggregated case (right plot) than for the non-aggregated one. The remaining elephant bins are the ones where a large relative byte difference leads to a relatively large rank change. Figure 3.7 begs the question how a flow rate can more or less keep its rank in going from network flows to FragFlows even if the byte count difference is relatively large. There are at least two arguments that can be put forward. For one, the byte difference may not be big enough to either reach the byte count of the next higher ranked entry or let it drop below the next lower ranked entry. Alternatively, another flow that was lower or higher ranked than the current one has a large relative byte difference and is therefore now ranked higher or lower than the current one. The latter argument also explains why a flow may change its rank from one bin to the next even if the byte difference extremely small or even zero. Similar comments apply when considering different bin sizes and/or flow aggregation levels. Figure 3.7 also illustrates that because the byte differences between the individual ranks are much smaller for the lower-ranked entries, the observed rank differences for the latter will be larger than for the top-ranked items.

3.4.3 Per-Flow Rank Differences

Until now, we have been mainly concerned with the per-bin byte count differences between FragFlows and network flows and with their impact on the resulting per-bin rank differences.

Here we will use the insight gained so far at the per-bin level and apply it to determine the impact that the constant throughput assumption has on the flows as a whole. To this end, we focus on the heavy hitters, where we define a heavy hitter to be a flow that is at least once ranked an "elephant" (i.e., in the top-10) in any bin during its lifetime. Other choices of defining an elephant (e.g., in the top-5, or top-20) yield similar results. For such flows we are interested in determining what percentage of the total bytes contributed by an "elephant" flow can be attributed to bins that are ranked within the top-10 or the top-100. Accordingly, a flow is considered a "hybrid" flow if its top ranked bin is a "hybrid" (i.e., in the top-100, but not in the top-10).

Figure 3.8 shows two scatterplots of the percentage of bytes contributed by an elephant or hybrid flow during bins that were ranked within the top-10 (for elephants) or top-100 but not top-10 (for hybrids) using network flows (x-axis) against the same FragFlow-derived quantity (y-axis). The left plot deals with the non-aggregated case (i.e., IP flows and a bin size of 60), and the right plot is for the aggregated case (i.e., aggregated flows using a fixed 16 bit mask and a bin size of 480 sec).

Figure 3.8 shows a number of informative properties related to relying on network flows as compared to FragFlows. For one, most of the points in both plots scatter around the diagonal, some are right on the diagonal (indicating a perfect match between network flows and FragFlows), some occupy the line $x = 0$ (vertical line through 0) and others the line $y = 0$ (horizontal line through 0). Looking first into the 24 (total points in the top plot is 524) flows satisfying $y = 0$, we find that the median distance of the FragFlow and the network flow ranking for the bins that cause each of these flows to be considered an elephant is 1 (mean is 1.8). This suggests that network flows just barely overestimated the ranking in comparison with FragFlows. Unfortunately, these edge effects cannot be avoided whenever one chooses a simple static elephant classification such as top-10 ranking, but time aggregation helps in alleviating this problem. Next the 51 flows satisfying $x = 0$ have little to do with edge effects, but represent in some sense the price one has to pay when using network flows instead of FragFlows in classifying Internet flows. Indeed, the reason for this obvious mismatch in between network flows and FragFlows in this case is that while FragFlows are capable of capturing the dynamics of the within-flow data exchange, these details are invisible to network flows. To illustrate, a sample sequence of per-bin ranks for a FragFlow is: 135, 9, 11, 7, 15, 18, 18, 19, 17 classifying the flow as an elephant; the network flow-derived per-bin rank sequence for the same flow is: 92, 13, 12, 15, 15, 18, 18, 19, 17 which qualifies for a hybrid. One consequence of network flows "mis-classifying" some elephant bins as hybrid bins is that the affected flows tend to get a large percentage of their bytes from hybrid bins when the actual (FragFlow-derived) percentage is in fact smaller. This explains the set of hybrid flows clustering around the line $x = 100\%$. In general, this problem can be alleviated with flow aggregation, which illustrates yet again that aggregation is the proper tool for achieving a desirable degree of accuracy when using network flows instead of FragFlows.

4 Network Flow Variability Analysis

Many traffic engineering methods rely on identifying a small set of flows, the *heavy hitters*, and control how these flows are routed through the network. Thereby they rely implicitly on the large flows to stay large, at least for some time. But there also is the the fact, that the rate of traffic aggregates is consistent with self-similarity, showing high variability on a wide range of time scales. Here the question arises of how much traffic individual Internet flows (especially the heavy hitters) contribute during their lifetime to the overall traffic. A better insight into the persistency properties of heavy hitters and the causes for that properties, especially in the presence of self-similar behavior of network flow rates, is vital for the understanding of existing and the development of new mechanisms for network flow based traffic engineering and load adaptive routing algorithms.

A simulation environment that is designed to enable the evaluation of novel traffic engineering or adaptive routing mechanisms therefore should be able to generate workload traffic that shows the same properties and characteristics as real network traffic. This is important not only on the packet level, where traffic shows self-similar scaling properties. Such workload traffic must also show realistic characteristics on the flow level where it has to exhibit flow rates that are conformant with Zipf's Law. Finally, it is also important that the workload traffic is not only self-similar and conforms with Zipf's Law, but that it also shows the same interactions between these two characteristics.

Consequently, we look into the persistency properties of heavy hitter flows and characterize as well as look into possible causes for these properties. Instead of analyzing the exact behavior of every network flow we classify flows according to their significance in terms of traffic contribution. We do this by deriving rankings for flows based on their per bin byte contribution and analyze the dynamics of such rankings over time.

The remainder of this chapter is structured as follows: We start by looking into basic aspects of flow rankings in terms of stability, followed by a characterization of how the flow rankings behave under different time resolutions and varying degrees of traffic aggregation. We then look into the causes for the observed features of flow rankings using our packet level traces and characterize how they influence the variability of the traffic. Finally, we introduce our metric for quantitatively assessing the degree of variability of flow rankings to be able to compare the persistence properties of different data sets, traffic aggregates and time granularity.

4.1 On the Dynamics of Flow Rankings

For traffic control applications it is not without problems to rely on flow rates to be consistent with Zipf's Law. On the one hand, flow rates are in fact consistent with Zipf's Law, at least for all our traces. This observation even holds over time. On the other hand, the dynamics inherent

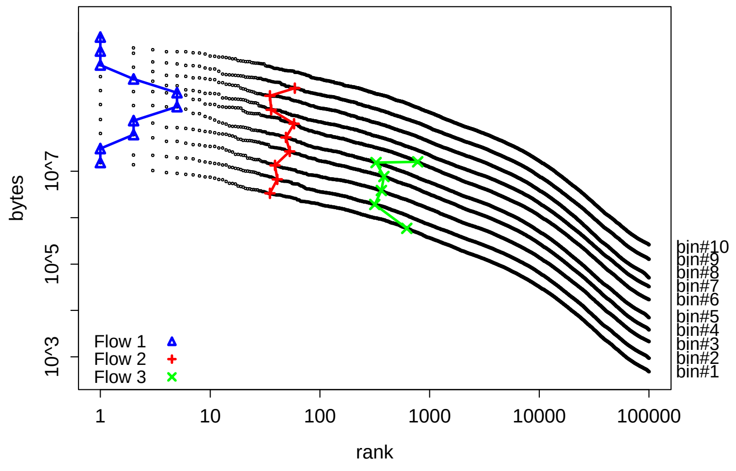


Figure 4.1: Zipf's Law across 10 successive time bins with rank sequences for three flows depicted by connecting the flow ranks on different size-rank curves. The colored lines connect corresponding ranks of the same flow. Although the rank fluctuations for the example flows seem to be equal in size because of the logarithmic x-axis, their real extent is much higher for low ranked flows than for top flows.

in network traffic lead to problems when trying to exploit Zipf's Law to identify a set of heavy hitter flows and then concentrate traffic control on this flow set.

To illustrate that Zipf's law applies bin-by-bin over time, we reconsider Figure 4.1 which shows a plot of 10 rank-rate relationships for flow rates of IP flows corresponding to 10 consecutive 1-minute bins. On the x-axis, the rank is shown on a logarithmic scale, while the y-axis, also in logarithmic scale, shows the rate of the flow. The curves are offset from one another by a small amount in the vertical direction to facilitate a visual assessment of Zipf's law across time, i.e., an approximate straight line behavior for each of the 10 curves. This kind of behavior also holds for other time periods, flow abstractions, and bin sizes. Given that Zipf's law for flow rates applies on a per-bin basis across time begs the question whether a flow that lasts for a number of bins and has been classified as “heavy hitter” has earned this distinction because of being top-ranked only sporadically because the flow rates associated with just one or two bins made it into the top ranks, or if the the flow is persistently top ranked, because its rates are top ranked in most of the bins throughout its lifetime. In Figure 4.1, we emphasize the ranks of three flows across the 10 bins by connecting the appropriate points of the rank-rate curves. All three flows show a certain amount of variability in their respective ranks. But they also stay either top ranked (blue curve), medium ranked (red curve) or low ranked (green curve). Note however that, although Figure 4.1 at first glance suggests otherwise, the movement in ranks of the low ranked flow is much higher than that of the top ranked flow. Note that flow #3, though seemingly spanning the same range of ranks as flow #1, the logarithmic scale on the x-axis shows, that while flow #1 spans over 5 ranks, flow #3 does so over the range of several hundred ranks! Still the low ranked flow shows no indication of ever becoming a medium or top ranked flow. This shows that although there exists a certain amount of variability in the ranks of a flow, there is also a

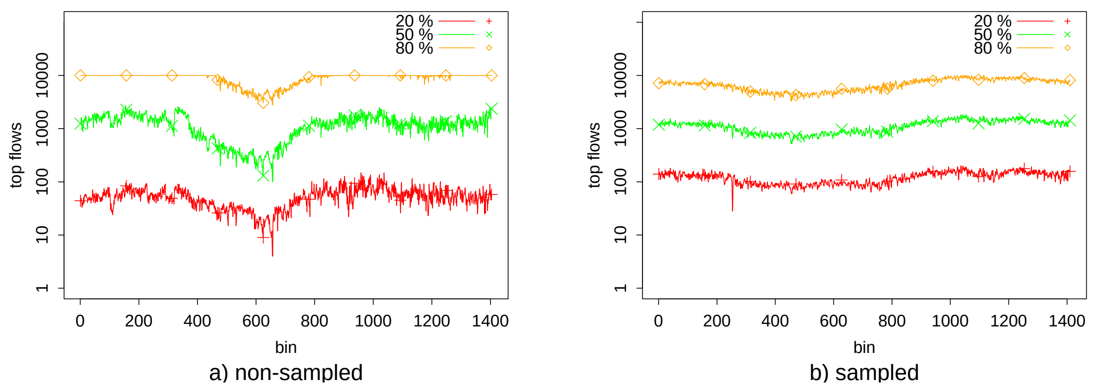


Figure 4.2: Number of top-ranked fbws needed to account for a given portion of the total traffi c per bin (IP fbws, bin size = 60 sec). With sampled fbws, more top fbws are needed to account for 20% of the traffi c, whereas less sampled fbws are needed to cover 50% or more of the traffi c. This is the result of a bias towards higher rates when considering fbws from sampled packet streams.

tendency for a flow to maintain at least its classification as top, medium or low ranked. It is this persistence of classification of flows that makes them interesting for traffic control applications.

Informally, Zipf's law and its variations are often interpreted as 80 – 20 or 90 – 10 rules, which state that about 80% (or 90%) of consequences stem from some 20% (or 10%) of causes. In the present context, this translates into "a significant portion of the total number of bytes in a bin is due to a relatively small percentage of the top-ranked flows (i.e., flows with the highest flow rates). Relying on the unsampled NetFlow trace, NF-I, and the sampled NetFlow trace NF-II, Figure 4.2 considers network flows and 1-minute bins and shows how many of the top-ranked flows are needed for each bin to account for 20%, 50%, and 80% of the bin's total traffic volume. For example, we note that most of the time the top 100 flows account for 20% of the total traffic per bin, the top 1000 flows are responsible for about 50%, and to account for 80% of the total traffic, we need to consider way more than the top 1000 flows (e.g., there are times that require more than the top 10000 flows). We note that the relative bytes per top ranked flows for the unsampled NetFlow trace, NF-I, is larger than those for the sampled NetFlow trace, NF-II. Yet for the lower ranked flows they are less. Also note that the overall per bin volume of the unsampled trace is about a factor 2 – 3 smaller than for the sampled trace. This is a direct consequence of constructing flows from sampled packets streams. The high sample rate of 1 in 100 packets on average leads to a bias towards large flows while missing lots of very small ones entirely.

According to Figure 4.2 it is necessary for traffic engineering applications to consider in the order of about 10000 of the largest flows to be able to control the majority of the traffic passing a router. Therefore, we construct rankings for our analysis that include the top 10000 ranks to cover enough of the traffic to obtain meaningful results. However in most cases, we want to concentrate on the largest flows only, so we show only flows with ranks 1 – 10 (elephants), ranks 11 – 100 (hybrids) and ranks 101 – 1000 (mice). Using this flow classification we focus on more manageable amount of the largest flows, while still covering about 50% of the overall traffic.

4.1.1 Stability of Flow Ranks

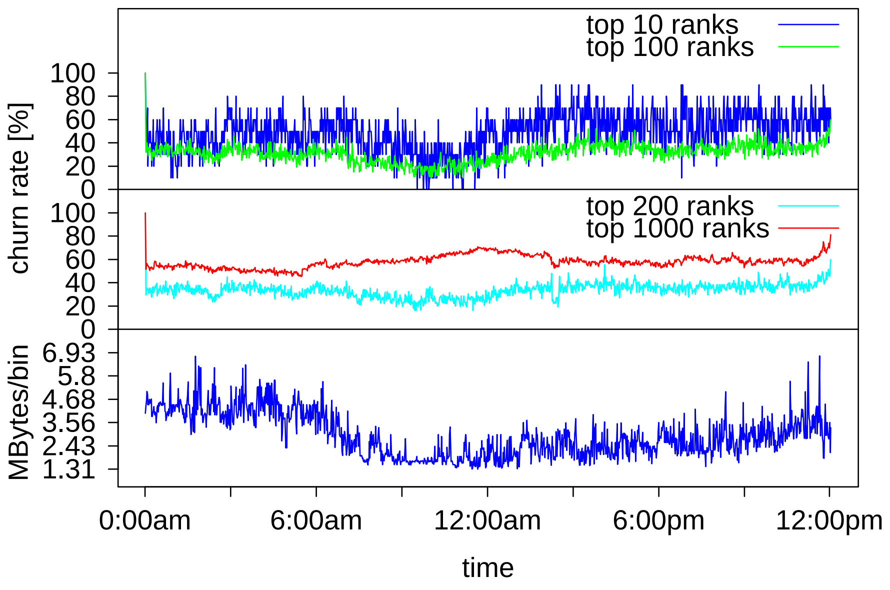
Figure 4.2 leaves open the possibility that the cast of top-ranked flows can vary considerably from one bin to the next due to the arrivals of new and the departure of existing flows. There always exist opportunities for newly arriving flows to make it into the top ranks if their rates are high enough and for existing flows to fall out of the top ranks, either due to them ending or being pushed out of the top ranks by new large flows.

To illustrate the degree of variability among the per-bin sets of top-ranked flows over time, Figure 4.3 considers network flows for 1-minute bins again and shows the “churn rate” among the top 10, top 100, and top 1000 flows, respectively. Here, for each bin, the churn rate is defined to be the percentage of top 10 (top 100, top 1000) flows in that bin that were not among the top 10 (top 100, top 1000) flows in *any* of the previous bins. Thus, a high churn rate is an indication of significant non-persistency among the top-ranked flows, while a low churn rate reflects a considerable degree of stability among the cast of top-ranked flows in time. In the upper and middle sections of Figures 4.3 a) and 4.3 b) we can see that, while the different churn rates are roughly comparable, they show subtle but nevertheless important differences. Overall, the variability of the churn rate drops as one considers more ranks. For the non-sampled trace NF-I (Figure 4.3 a), the churn rate among the top 100 and top 200 ranks is lower than the churn rate for the top 10 ranks. During the early morning hours this difference is reduced. The churn rate for the top 1000 ranks shows the opposite behavior. It is larger than the churn rate for the other ranks and it increases as the traffic volume decreases. This indicates that most flows that are in the top 1000 but not the top 200 are short lived and interchangeable with small byte rate differences in the order of tens of bytes.

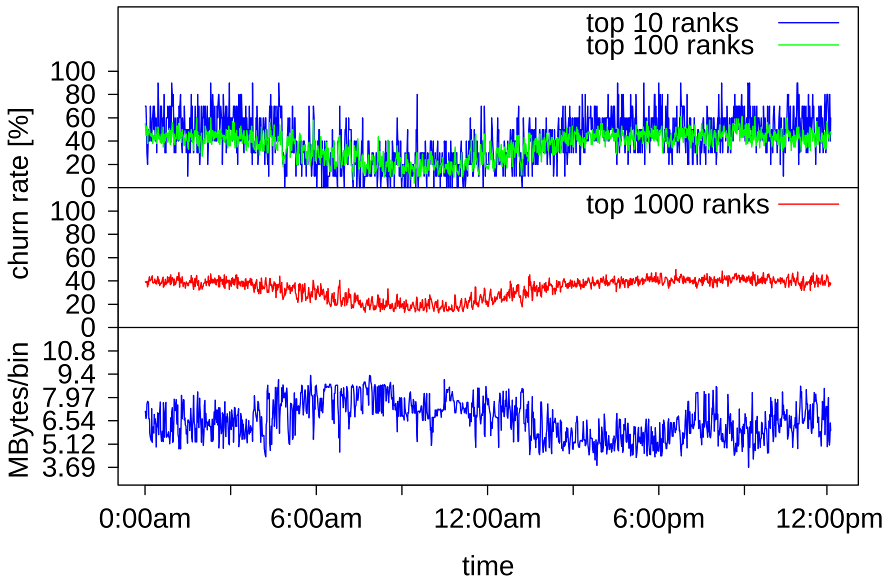
For the sampled trace NF-II (Figure 4.3 b), the churn rate among the top 100 and top 1000 ranks is slightly lower than the churn rate for the top 10 ranks during the early AM hours. During the later AM/early PM hours it is slightly higher, and appears to revert to the early AM behavior during the late PM hours. For this trace the churn rate for top 1000 ranks does not show the opposite behavior as we saw for NF-I since the byte volume differences between flows at rank 1000 are in the order of thousands of bytes. This is caused by the bias towards larger flows when using sampling to construct flows from packet streams.

For both, NF-I and NF-II, the churn rate appears to be correlated with the total number of flows. However during the less busy periods in NF-I, a flow needs to contribute a smaller number of bytes to a bin in order to be top ranked than during the corresponding periods in NF-II. For a visual assessment of this observed behavior of the churn rates, we show in the lower sections of Figure 4.3 the time series representing for each bin the rate (i.e., number of bytes per bin) at which a newly arriving flow would have to send data to move into the top 10 ranks (i.e., be classified as elephant). The differences observed in the plots in Figure 4.3 again resulting from the use network flows derived from unsampled (trace NF-I) vs. sampled (trace NF-II) packet streams clearly warrants further investigations.

To assess the influence of this variability in the set of top-ranked flows on traffic engineering and load adaptive routing, Figure 4.4 shows 10%, 50% and 90% percent quantiles of the fraction



a) non-sampled



b) sampled

Figure 4.3: Upper and middle parts: Churn rate processes associated with the top 10, top 100, top 200, and top 1000 fbws, respectively. Lower part: Time series of fbw rates needed for a newly arriving fbw to move into the top 10. (IP fbws, bin size = 60 sec.)
The churn rate for top 1000 fbws for the non-sampled trace (a) hows the opposite behavior than for the top 200, top 100 and top 10 fbws due to very small byte volume differences. This is not the case for the sampled trace (b) where the sampling leads to a bias towards higher fbw rates. For the non-sampled trace, the volume needed to enter the top 10 ranks during the less busy hours is lower than for the sampled trace.

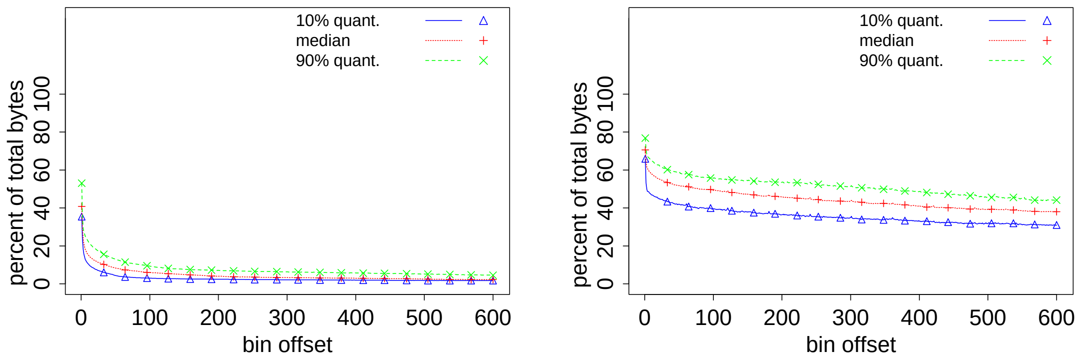
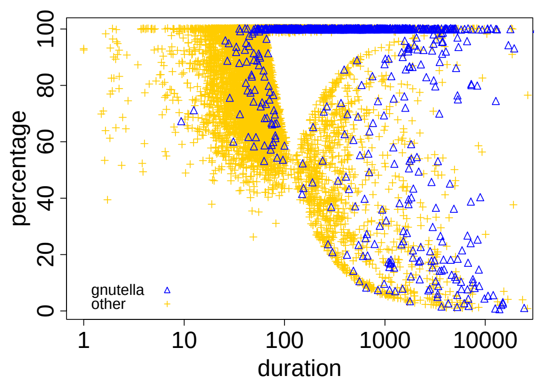


Figure 4.4: Byte contribution of a fi xed set of top 1000 fbws for WASHng and 1 minute bins. The traffi c fraction contributed by any fi xed set of fbws diminishes quickly over time as top fbws will terminate with time. Using aggregated fbws reduces this effect due to higher fbw durations.

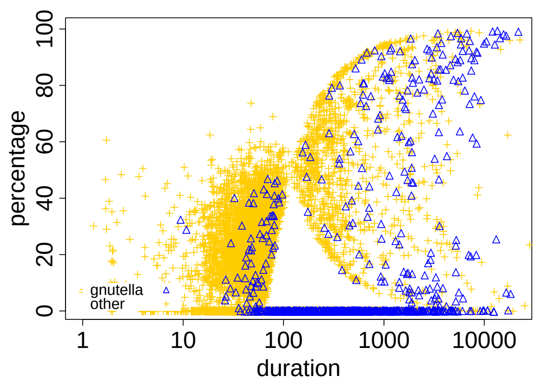
of bytes covered by considering only the top 1000 flows of some bin (y-axis) and how this fraction develops with time in terms of distance in bins. For both non-aggregated and aggregated flows, the fraction of bytes contributed by a fixed set of flows diminishes quickly over time. In the case of non-aggregated flows, the byte contribution halves within about 20 minutes on average. Although the reduction in byte contribution for aggregated flows is not as dramatic as for non-aggregated flows, the plots show, that it is necessary to reclassify the set of top flows on a regular basis if one wants to control a significant part of the overall traffic. The curves in this plots are also consistent with similar assessments of Papagiannaki et. al. in [PTD04].

Figure 4.2 also does not answer the question, whether the observed dynamics are due to churn only. It might be the case that there is no persistency of flows across bins and our observations automatically follow from the well-known heavy-tailed distribution of flow rates or flow durations. However, a simple comparison of the flow duration distributions shows that this is not the case for the top ranked flows. For example, for the trace NF-I the median of the flow length distribution for 60 second bins increases from 4.2 seconds for the top 10000 flows to 44.7 for the hybrid flows and to 57.8 seconds for the elephant flows. In the case of 480 second bins, the larger bin length increases the possibility of flows with moderate rate but high duration to be ranked higher. In fact, for the same trace, the median of the flow length distribution changes from 44.7 seconds to 136.0 seconds for hybrids and from 57.8 to 296.5 seconds for elephants. The maximum flow length coincides with the trace duration.

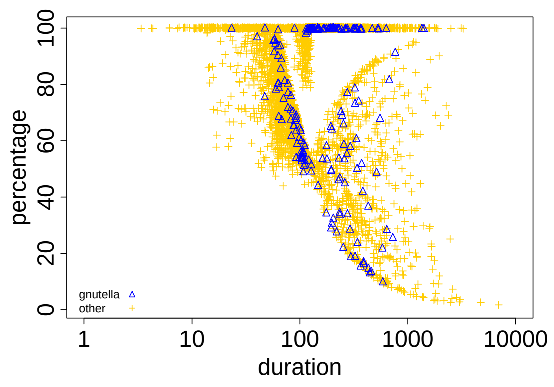
In general instabilities can have many causes including flow rate fluctuations. But even if one assumes that all flows operate at a fixed flow rate, a top ranked flow can become a medium ranked one if a number of larger flows pop up. Alternatively a medium ranked flow can become a highly ranked one if enough top ranked flows end.



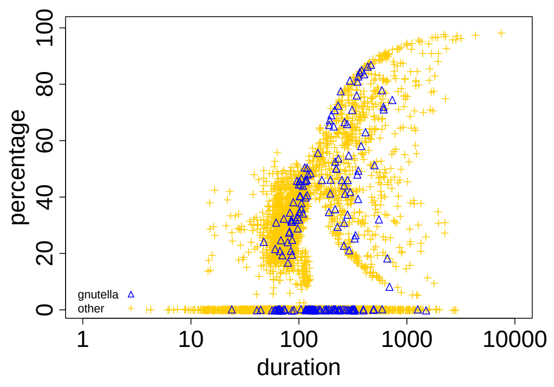
a) Heavy hitters (non-sampled) as Elephants



b) Heavy hitters (non-sampled) as Hybrids



c) Heavy hitters (sampled) as Elephants



d) Heavy hitters (sampled) as Hybrids

Figure 4.5: Scatterplots of lifetimes of heavy hitters (log-scale on x-axis) against relative amount of time spent as elephants (left plots) or hybrids (right plots) for NetFlow trace NF-I (top) and NF-II (bottom). More than 95% of the heavy hitters are elephants for more than half of their lifetime. This is largely independent of what application is responsible for the flows. The half-moon shapes are caused by the constant flow rate assumption and the handling of partially covered bins in conjunction with the logarithmic x-axis.

4.2 Persistency Properties of Heavy Hitters

Given the persistency behavior of Internet flows suggested by Figure 4.3, we next focus on the heavy hitters, where we define a heavy hitter as in Section 3.4, and ask whether or not heavy hitters have a distinct persistency property. Put differently, we are interested in whether “once an elephant” implies “always an elephant”, at least with high probability. Evidence of such persistency properties for the largest Internet flows is crucial for approaches to traffic engineering that rely on the persistence in time of flows to remain elephants. For the trace NF-I with a bin size of 1 minute and considering non-aggregated network flows, we extracted a total of 5666 heavy hitters and show in Figure 4.5 scatterplots of the heavy hitters' lifetimes against the percentage of the time they were ranked "elephants" (ranks 1-10; top left plot) or "hybrids" (ranks 11-100; top right plot); we use a logarithmic scale for their lifetimes on the x-axis and linear scale for percentages on the y-axis. When computing the percentage of time that flows were ranked as elephants/hybrids, there are flows that start during some bin and subsequently cover one or more full bins before terminating during another partially covered bin. The contributions to the partially covered bins are calculated as a portion of the average per-bin contribution, depending on how much of the bin is covered by a flow (see Section 3.1.3). This partial contribution results in the rank of a flow to be too low for the first bin. To avoid artifacts caused by this effect, the time a flow spends in the first (partial) bin is counted towards the rank of the flow in its first full bin; ending partial bins are handled in a similar way. Figure 4.5 reveals a number of interesting features as far as the heavy hitters are concerned. Ignoring for the time being the different coding of the points, we first note that about half of the heavy hitters are elephants during their entire lifetime (i.e., out of a total of 5666 (5830) points, some 2875 (3075) fall on the $y = 100\%$ line). Second, heavy hitters who are alive for 2 or more bins and are not elephants during their whole lifetime, but only part of it, have about a 40% chance to be elephants for more than half their lifetime and a 60% chance to be elephants for less than half their lifetime (i.e., half-moon shaped cluster starting at $x = 120$ sec and $y = 50\%$). The half-moon shape is the result of the way we handle partially covered bins and the assumption of a constant flow rate for network flows in conjunction with the logarithmic x-axis. Expressing the lifetime of flows as n times the bin size, each flow can be an elephant (or hybrid) for a fraction on the range of $1/n$ to $1 - 1/n$ of its lifetime. The half-moon shape vanishes if we do not handle handle partially covered bins as described above. Finally, when comparing the left and right columns of Figure 4.5, the anti-symmetry between heavy hitters as elephants and heavy hitters as hybrids is not an coincident. In fact, the right plots show some 60% of the heavy hitters are never hybrids, and those heavy hitters that are alive for 2 or more bins and are hybrids for some time have about a 60% chance to be hybrids for less than half of their lifetime. These observations are largely independent of whether we use non-sampled data (top row, trace NF-I) or sampled data (bottom row, trace NF-II). Note that the remaining structure in the left sections of the plots of Figure 4.5 are relatively uninteresting since those points correspond to heavy hitters that are alive for less than 120 seconds (2 bins). Flows that last less than 120 seconds and are elephant for only part of their lifetime are split across two bins. Since they are more likely to be elephants in the bin in which they spend most of their time it is not surprising that most of the fractions are larger than 50%. In summary, Figure 4.5 shows that more than 95% of the heavy hitters are elephants for more than half of their lifetime. A breakdown of all the heavy hitters by application is easily possible, but simply shows the usual suspects (e.g., web, nntp, peer-to-peer, ftp, and others) and individually, they produce plots similar to the ones shown in Figure 4.5. To illustrate, we use in Figure 4.5 the symbol "$\triangle$" to denote heavy hitters associated with the well known peer-to-peer application Gnutella.

4.2.1 Variability across Bin Sizes

Part of the multi-scale aspect of our flow analysis involves considering different time scales and performing the same type of persistency study across a range of time scales. The effects of using

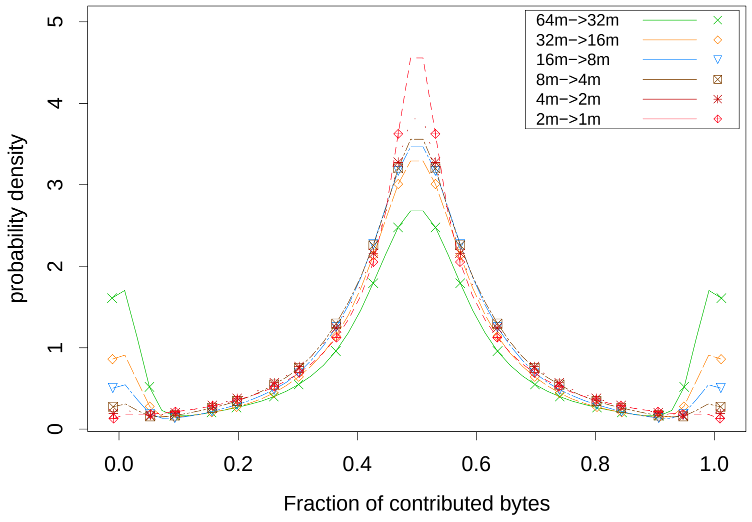


Figure 4.6: Distribution of the contribution to large bins over the two corresponding smaller bins. Heavy hitters tend to be invariant under time (dis)aggregation for a limited range of time scales. This allows certain persistency properties to be gleaned from analysis at coarse time scales, increasing the effi ciency of the analysis.

different bin sizes depend on the granularity of our trace data, as especially for non-aggregated network flows we have only averaged information on the per-bin byte contribution and so have to distribute the byte contribution of a flow evenly over all fully covered bins. In this case, there is no variability across the size of bins. For FragFlows, where we have access to byte contribution in a sufficiently fine grained resolution, or for aggregated network flows, e.g., destination prefix flows, the situation is more complicated, because the contributions of a flow to a coarse scale bin may generally no longer be distributed evenly among the smaller scale bins. Therefore, we use the FragFlows generated from the MWN-III packet level trace and inspect how the contributions to bins are distributed over the corresponding two half-sized bins.

Figure 4.6 shows a probability density plot of how the byte contribution of a large bin is distributed over the two corresponding bins on the next finer time scale. The x-axis shows the fraction of the contribution to the first of two fine scale bins and the y-axis shows the probability density for a given fraction. The curves show a pronounced concentration about 50% with a small probability that the entire contribution goes (almost) completely to either one of the fine scale bins. This holds over a wide range of time scales with little variation except for the fact that with larger bin sizes the probability for the entire contribution going to a single bin is increasing. This effect is not surprising and increases further as the bin size approaches or exceeds the duration of a significant part of the flows. Nevertheless, Figure 4.6 shows that for aggregate heavy hitters across a limited range of time scales (from a few seconds to hundreds of seconds) the heavy hitters tend to be invariant under time (dis)aggregation. This explains why certain persistency properties associated with heavy hitters can already be gleaned from

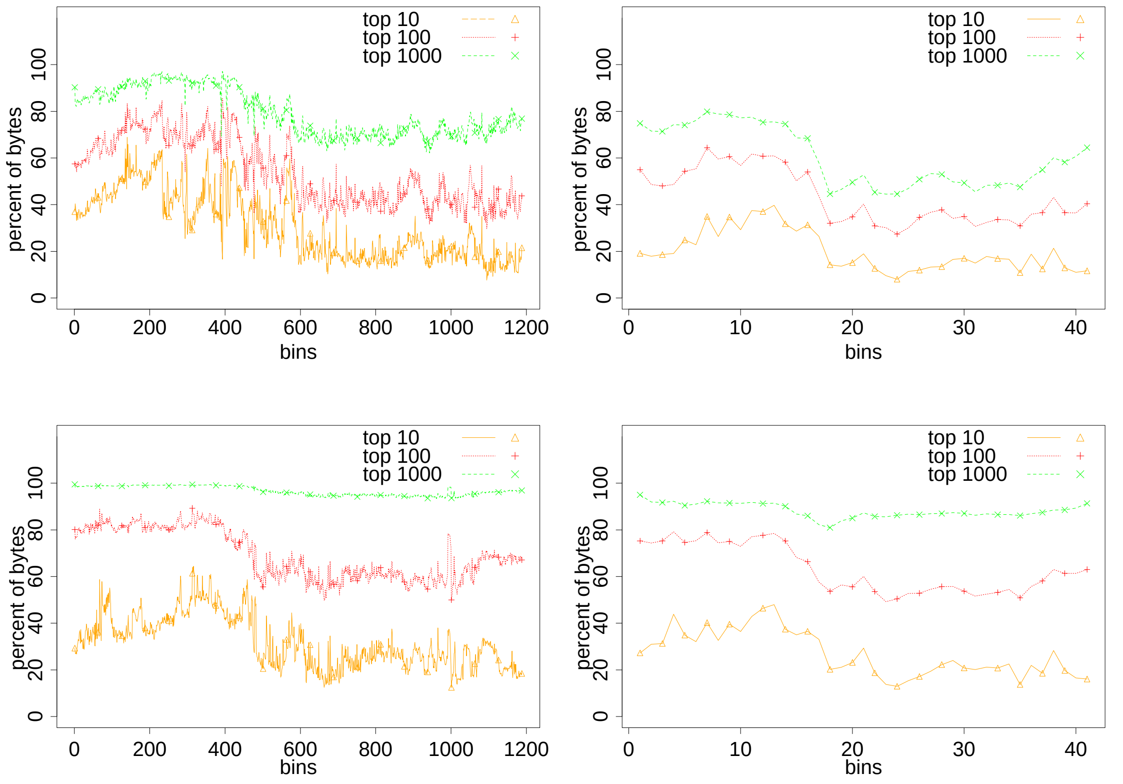


Figure 4.7: Percentage of traffi c covered by top ranked fbws depending on bin size for the MWN-III trace (top row: unaggregated, bottom row: aggregated by destination prefi x, left column: one minute bins, right column: 30 minute bins). Fraction of traffi c covered by top *n* fbws is largely independent of the bin sizes, but is signifi cantly influenced by fbw aggregation.

an analysis at coarse time scales which in turn generally involves a substantially reduced data set and is therefore faster and more efficient. This is also confirmed by Figure 4.7 where we show what fraction of the overall traffic is contributed by the top 10, top 100 and top 1000 flows, respectively. The plot shows curves for non-aggregated (top row) and for destination prefix flows with a fixed 16 bit prefix length (bottom row), both for 1 minute bins (left column) and for 30 minute bins (right column). In both cases, the bin size has no distinct influence on the fraction of bytes covered by the top *n* flows. With aggregation however, the fraction of bytes covered by the top *n* flows is clearly larger than for non-aggregated flows. Here, the top 1000 flows cover more than 90% of the overall traffic, while for non-aggregated flows, this fraction is about 50% on average.

Our analysis suggests that the observations reported in the previous sections are largely invariant under different choices of bin sizes and hold in a genuinely multi-scale fashion. This property is the consequence of the fact that heavy hitters at the level of IP flows and at large time scale tend to remain heavy hitters at finer time scales. Considering a fine scale to mean a *k*-second bin size and coarse scale to mean a 2*k*-second bin size, an IP flow that is a heavy hitter at coarse scale will simply re-distribute the bytes in each large bin nearly evenly among the two corresponding k-minute bins at the finer time scale and is thus likely to cause the resulting flow to be a heavy hitter at the finer time scale.

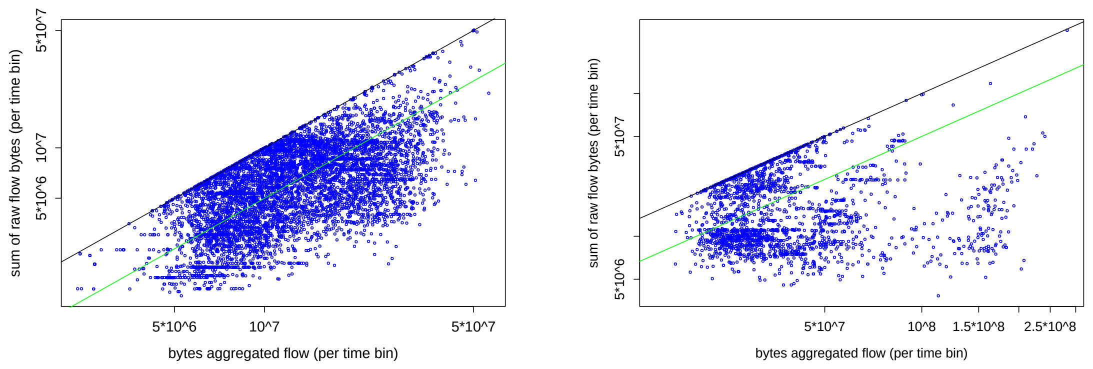
4.2.2 Variability across Traffic Aggregation

Another aspect of our multi-scale analysis of Internet flows concerns aggregation in IP or flow abstraction. NetFlow data lends itself naturally to different levels of aggregation, from five-tuple flows (defined by source and destination IP addresses and port numbers and protocol) to prefix flows (defined by source and destination prefix) to AS flows (defined by source and destination AS). As we have seen in the previous section, flow aggregation has a direct influence on the fraction of the traffic that is covered by the top n flows, with a significantly larger fraction for higher aggregations. To illustrate that flow aggregation is in many other ways more intricate than time aggregation, we explore for traces NF-I and NF-II in Figure 4.8 the question whether or not flows that are heavy hitters at some coarse scale of flow aggregation (e.g., prefix flows) are in general made up of constituents that are heavy hitters at a finer scale of flow abstraction (e.g., unaggregated five-tuple flows). That is, what is the observed behavior of heavy hitters under flow (dis)aggregation? To this end, for a 1-minute bin size, Figure 4.8 shows scatterplots of the per-bin contributions of the heavy hitters at the aggregate level (destination-prefix, logarithmic scale on x-axis) against the sum of the per-bin contributions of those flows that were elephants (top row), hybrids (middle row), and mice (bottom row), respectively, at the level of five-tuple network flows (logarithmic scale on y-axis). Thus we get one point per aggregated flow for each of the three flow classes. We observe a certain number of points between the lines $y = x$ and $y = x/2$ in the top an middle rows (about 50%), representing aggregated flows that have a per-bin byte contribution that comes at least by half from elephant and hybrid five-tuple flows, respectively. However, in the bottom row, the vast majority of points concentrate in that area, showing that there are a large number of aggregated elephant flows whose per-bin byte contributions come almost exclusively from five-tuple flows that are classified as mice. This observations hold for both unsampled data (left column, trace NF-I) and sampled data (right column, trace NF-II).

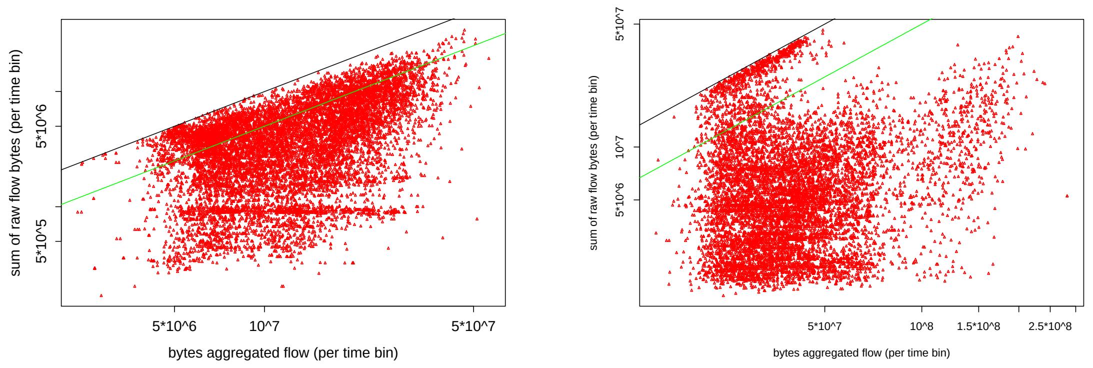
4.3 Causes of Instability

In the previous sections we have seen that, although our datasets show that flow rates are consistent with Zipf's Law even over different time resolutions and across time, the cast of flows that contribute most to the overall traffic can vary considerably over time. So far, we have concentrated on characterizing this variability without looking into its causes. We therefore explore now possible causes for the instabilities in the rankings of large flows.

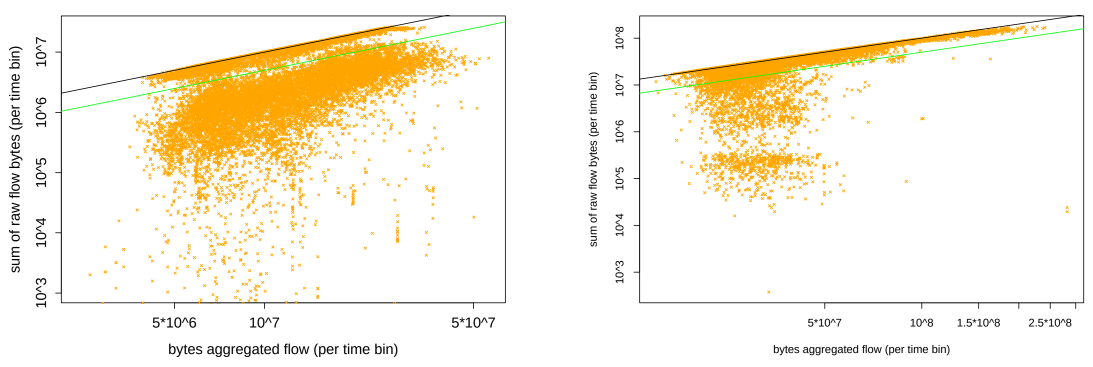
We have already seen, that one cause for instability is the arrival and departure of flows. The set of top n flows changes whenever a new flow arrives with a rate higher than any one of the top n flows. Also if a top n flow terminates, it disappears and makes room for a new flow to enter the set of top n flows. We refer to the characteristics of flows entering and leaving the cast of top n flows as *entry*- and *exit process*, respectively. The arrival of new and the departure of



a) Elephant Contributers



b) Hybrid Contributers



c) Mice Contributers

Figure 4.8: Scatterplots of the per-bin byte contribution of aggregate heavy hitters (log-scale on x-axis) against the sum of the per-bin byte contributions of the constituent fi ve-tuple elephant flows (top), hybrid flows (middle), and mouse flows (bottom) for non-sampled NetFlow trace NF-I (left column) and the sampled trace NF-II (right column) and a bin size of 1 minute. Many elephant flows (about 50%) consist mostly of elephant and hybrid fi ve-tuple flows, but there are a large number of elephant flows that are made up almost exclusively by fi ve-tuple flows that are mice. This holds for both for sampled and the unsampled NetFlow traces.

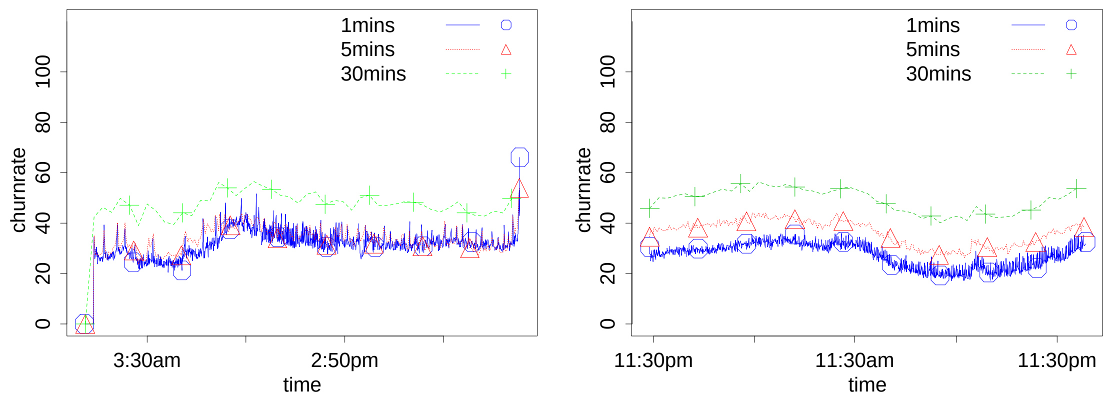


Figure 4.9: Churn rates for the top 1000 destination prefi x fbws for traces MWN-III (left) and WASHng (right). For aggregated fbws, the churn rate increases with the bin size, as the fbw rates are no longer constant over time. This makes it harder for a fbw to retain a high rank over time.

old flows is one of the two underlying causes for instability among the top *n* flows. How much the entry and exit processes can influence the set of top *n* flows has already been shown by the churn rate for non-aggregated flows (see Figure 4.3). The same findings hold for aggregated ones as well. This can be seen, e.g., when considering Figure 4.9 which shows the churn rate among the top 1000 flows. The churn rate is now defined as the percentage of top *n* flows in a bin that were not among the top *n* flows in the previous bin. Figure 4.9 shows that indeed a significant portion of the top 1000 flows, about 30%, are no longer in the top 1000 ranks in the next bin. An interesting aspect here is that the churn rate increases with larger time bins, as it gets more and more difficult for a flow to retain a top 1000 rank over longer time periods. This change in the top ranks results in the undesired fact, that a set of flows once classified as top *n* is at first responsible for a significant portion of the overall traffic, but that this portion quickly diminishes over time. This confirms previous observations by Papagiannaki et al. [PTD04] for different data sets.

In the case of aggregated flows, where we cannot assume constant flow rates because of the way we perform the aggregation (see 3.1.4), the entry and exit processes are not the only cause of the observed instability in flow rankings. Here we also have variability in the rankings that is caused by flow rate fluctuations. A flow changes its rank, as soon as its flow rate increases beyond the rate of the next higher ranked flow or decreases below the rate of the next lower ranked flow.

In the following sections we look into both causes for the variability in the set of top flows over time, the flow entry and exit processes as well as fluctuations of the flow rates. We rely on the MWN-III and the WASHng traces for our analysis. We compute the entry/exit processes and the flow rate fluctuations for 1, 5 and 30 minute bins, representing small, medium and large bin sizes. Because the MWN-III trace was collected at an access network and thus has a limited IP address variability on the access side, we separate this trace into incoming and outgoing flows and consider only aggregation by destination address prefix for outgoing flows. The offline aggregation process results in 9.1 million destination prefix flows for the MWN-III trace and roughly 1.9 million for the WASHng trace.

4.3.1 Flow Entry and Exit Process

In this section we go beyond previous results [PTD04] and decompose the churn rate into its components, flow entry process, flow exit process, and duration, and study their behavior. The entry process determines the rank at which a flow enters the ranking and the flow exit process determines the rank at which it leaves the ranking. The exit process is the result of combining the entry process with the flow durations.

Figure 4.10 (top and middle rows) shows histograms for the first ranks of those flows that have an average rank of at least 100. Each bar summarizes the counts for 5 consecutive ranks. For the MWN-III trace (left column), the counts increase sublinearly with the entry rank. This also holds in general for the WASHng trace (right column), although there is a pronounced spike at the top ranks. We assume this spike to be the effect of the special kind of traffic carried by the Abilene network, generated to a large degree by experimental protocols and transfers of measurement data (see also Section 3.2). For both traces, the entry process is influenced to only a small degree by different bin sizes. For the MWN-III trace, the entry ranks level out to be almost uniformly distributed. For the WASHng trace, the pronounced spike at the top 5 entry ranks is somewhat distributed over the top 25 ranks. Still the general shape of the histograms persists over a large range of bin sizes and shows that new flows arrive with all kinds of different ranks, including top ranks, to a non-negligible degree.

The exit processes show almost identical behavior to the entry processes. This is not surprising, as entry and exit processes are tightly coupled to the duration of flows. For every flow that enters the top 100 ranks, another of the top 100 flows has to terminate and leave the ranking. As to the rank of the flow leaving the top 100 flows, the histograms for the exit processes in Figure 4.10 (bottom row) show, that the probability for top ranked flows to leave the rankings is nearly as high or even higher than for all other flows. Again this observation holds for all considered bin sizes.

The above properties of the flow entry and exit processes show that newly arriving flows as well as departing flows are with a non-negligible probability top ranked. This causes a lack of predictability for the entry and exit ranks of flows and therefore it is not clear how to take advantage of the “heavy-tailed” nature of flow rates for the purpose of better engineering the network. Still, Papagiannaki et al. [PTD04] found that they were able to identify a subset of flows that contribute more traffic than the average flows, i.e., they are highly ranked, and persist longer than the large majority of the flows. For this to be possible some of the highly ranked flows have to have longer durations. We find that the top rank of a flow (its highest rank) is not a good indicator to decide if a flow is large and has a longer than average duration. But flows with top average ranks over their duration usually persist longer than those with lower average ranks. This is shown in Figure 4.11 where we show on the y-axis the probability density for flows of three different classes of top ranks to have a certain duration (y-axis). The plot shows that flows with average ranks from 1 to 10 have a high probability to last for more than 5000 seconds, which is longer than for flows with an average rank from 11 to 100, which in turn is longer than for flows with lower average rank. The peaks at the right hand side of the plot are

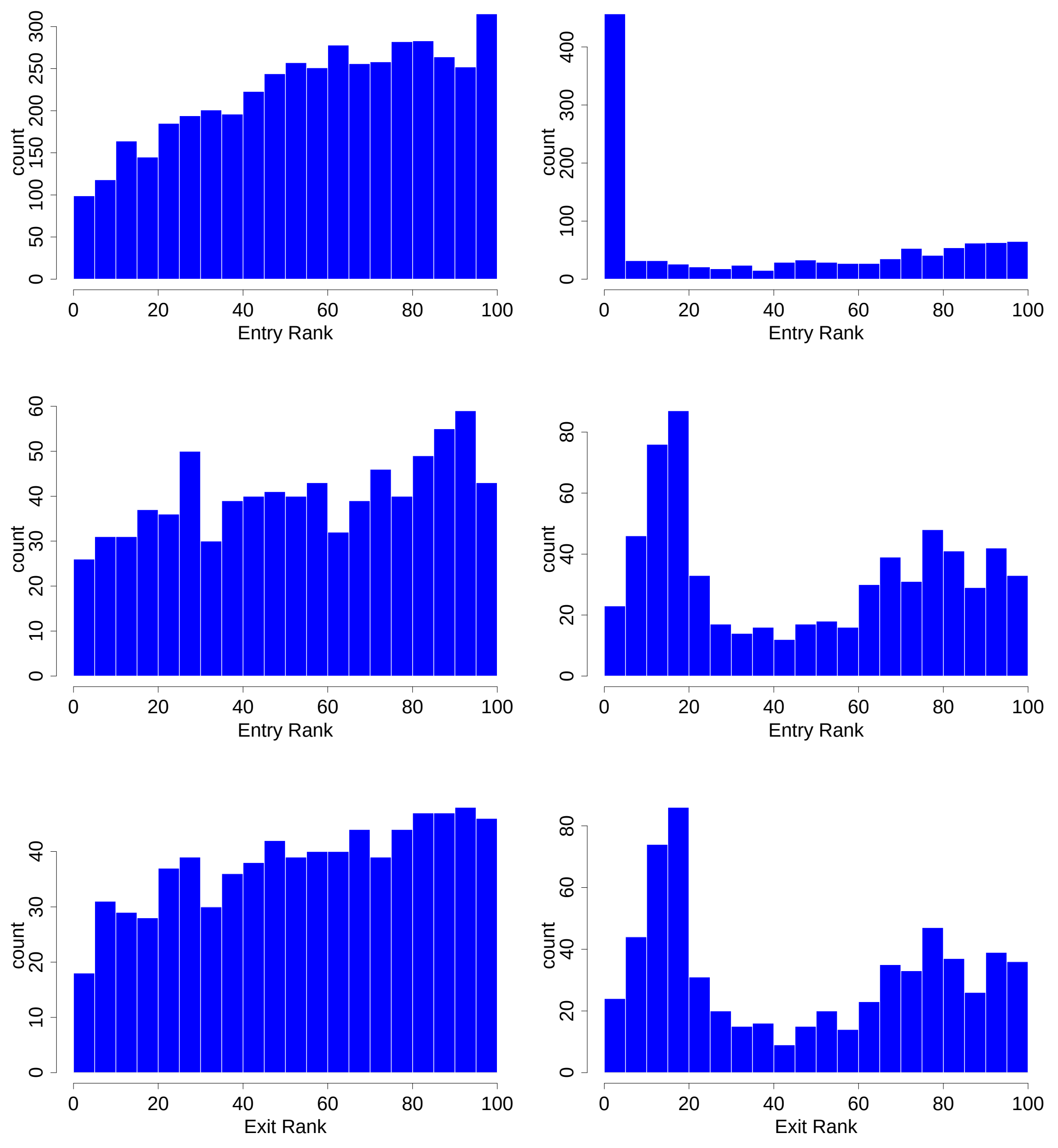


Figure 4.10: Histograms of entry/exit ranks with a granularity of 5 ranks: left column MWN-III, right column WASHng; entry 1 minute bins (top row), entry 30 minute bins (middle row), exit 30 minute bins (bottom row). The counts increase sub-linearly with the ranks, indicating that flows can begin with all kinds of ranks, including top ranks. The spike for the WASHng trace is the result of the special traffi c carried by the Abilene network. The exit process behaves almost identical to the entry process. Both processes are only slightly influenced by the bin size.

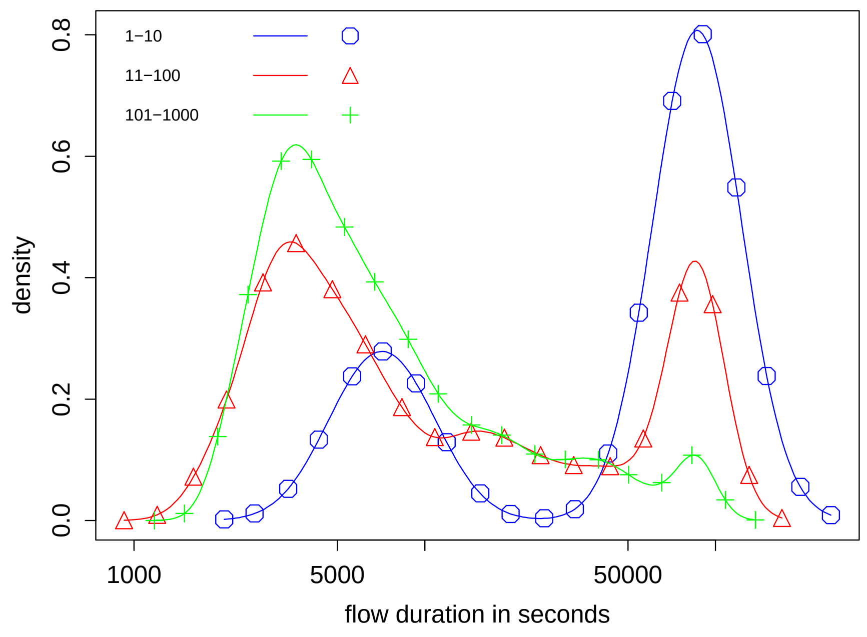


Figure 4.11: Probability density of the logarithm of the flow duration for WASHng. The higher the average rank of an aggregated flow, the higher the probability that it has a large duration. The right peaks are caused by the limited duration of the traces of 24 hours (86400 seconds).

artifacts of the traces covering only 24 hours (86400 seconds). But the probability mass covered by these spikes is higher if the average ranks are higher which means that there are more long lived flows in the higher ranks than in the middle and lower ranks. These findings explain the observations by Papagiannaki et al. [PTD04], as there is a strong correlation between average rank of a flow and its duration.

4.3.2 Flow Rate Variability

So far our analysis has ignored the important aspect of flow rate fluctuation and its impact on rank changes. As flow rates are consistent with heavy-tailed distributions, the difference of the rates of top ranked flows are very high, so only large deviations are able to influence the ranking of top flows. Correspondingly, we do not care much about small additive deviations. We choose instead to consider a multiplicative metric, called *relative deviation* (or *reldev*). We compute this metric for all bins of each flow in the following manner:

$$\text{reldev} = \log_2 \left(\frac{\text{actual byte contribution}}{\text{mean byte contribution}} \right)$$

A *reldev* value 1 (2) stands for a deviation from the mean by a factor of 2 (4) and a value -1 (-2) represents a deviation by a factor of $1/2$ ($1/4$).

Figure 4.12 (left) shows, for a typical destination prefix flow, how the relative deviation be-

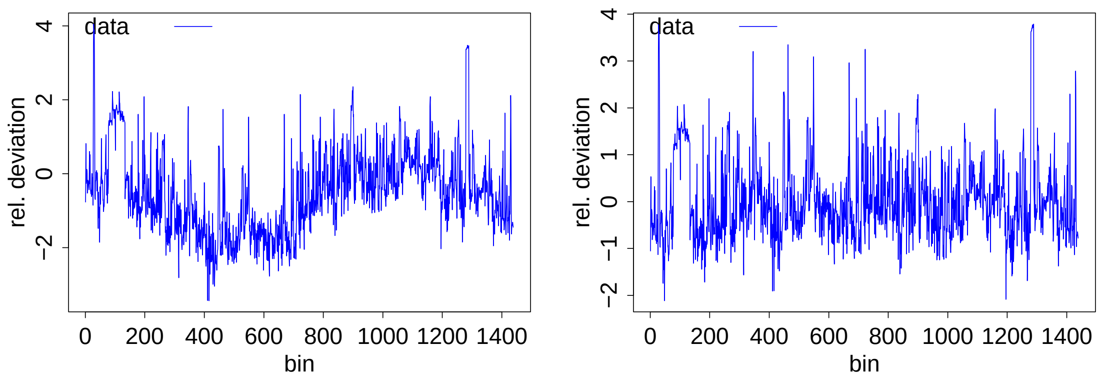


Figure 4.12: Relative deviation over time for a typical example fbw, before (left) and after (right) removing time of day effects. The detrending using Wavelets removes the time of day effects while it conserves the spikes.

haves over time for 1 minute bins. The plot shows a clearly discernible time-of-day effect. This means that we see the effects of the combination of different processes, a periodic trend caused by time of day effects and the rate fluctuations we are interested in. To be able to perform a meaningful analysis of the flow rate fluctuations stemming from the nature of the flows themselves, we detrended our data to remove time-of-day effects. We use a wavelet based filtering methodology for detrending our trace data. Wavelets have the advantage over, e.g., Fourier based filter mechanisms, that they are able to work on non-periodic data. They are also able to preserve pronounced spikes in the data with a much smaller risk of introducing artifacts. The filter threshold depends the frequency of the time of day effect and is internally expressed in terms of bins. We use the bi-orthogonal spline wavelet *bs1.3* [BG96] because it is known to be resilient against the introduction of artifacts [Chu92a, Chu92b].

The result of the detrending of the relative deviation sequence of our example flow is shown in the right plot of Figure 4.12. The time-of-day effect has been removed while the characteristics of the curve, even the distinct spikes, have been preserved.

The overall behavior of the relative deviation for this example flow is again shown in the left plot of Figure 4.13. The plot shows on the right hand side the probability density over the per-bin relative deviation values for this flow which lasts for more than 1400 one-minute bins. The values range from -3 to $+4$, which means that there are times when this flow shows only an eighth of its average rate but also increases its rate at time to 16 times its average rate! The distribution of the values of the relative deviation shown in Figure 4.13 (right) is close to a normal distribution with the same mean and variance as the relative deviation values, with a slight bias to higher than average rates.

However, this is not always the case. Especially for aggregated flows, where contributors on the level of five-tuple flows come and go, there is not always a cleanly centered distribution of the relative deviation. Figure 4.14 shows the a probability density plot for the relative deviation of a different destination prefix flow. This plot shows bimodal behavior with a spike at -0.5 and a second one at $+0.4$. Bimodal distributions are not uncommon and there are even tri- and

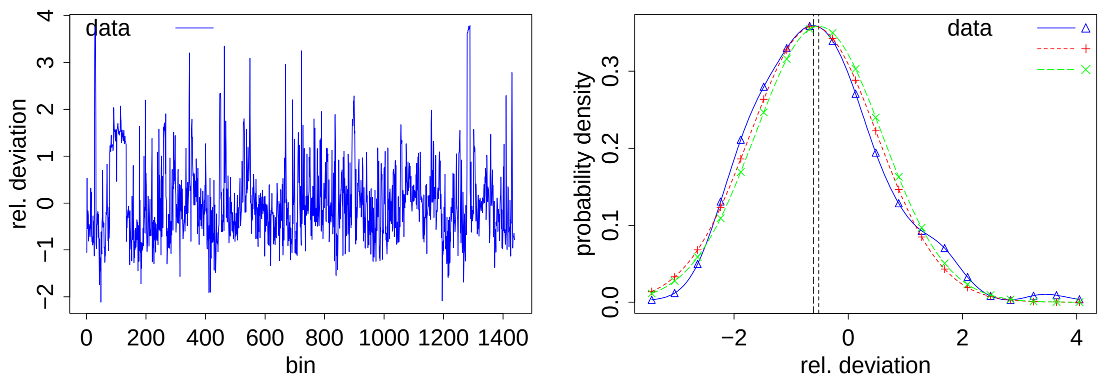


Figure 4.13: Probability density of relative Deviation for an example destination prefi x fbw. The rate fluctuations are with one eight up to 16 times the average rate high enough to cause rank changes.

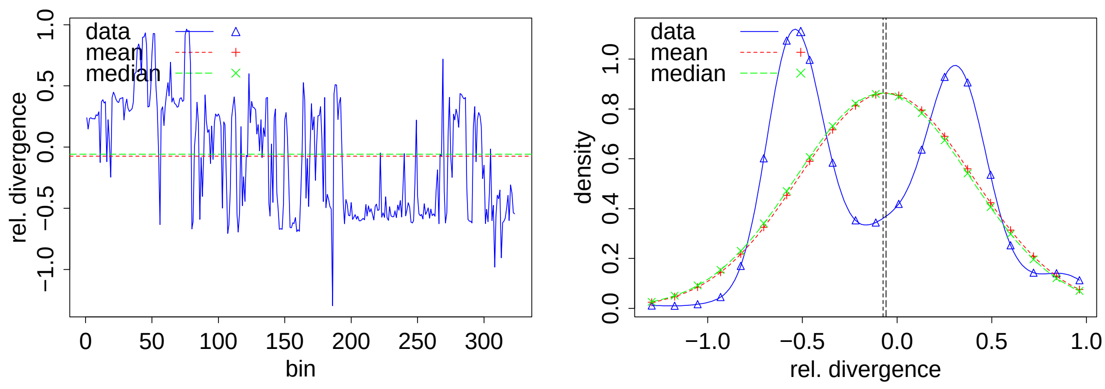


Figure 4.14: Probability density of rel. Deviation for an example multimodal destination prefi x fbw. The modes are caused by transient changes in set of constituent fbws, often caused by a single fi ve-tuple fbw.

higher multimodal distributions to be found. A manual inspection showed, that the modes are mostly caused by only a few bins deviating from the average rate, especially in the bimodal cases. We found that for the MWN-III trace with aggregation by destination prefix 60% of the 116 flows spanning at least 20 bins and with an average rank in the top 1000 flows, are consistent with a normal distribution and 37% show bimodal distributions. For 30-minute bins, we found 522 destination prefix flows spanning at least 10 bins and with an average rank also in the top 1000. Of these, 65% are consistent with a normal distribution with same mean and variance, while 30% follow a bimodal distribution. Both Figures 4.13 and 4.14 show however that there is a significant variance in the relative deviation values throughout the lifetime of the flows and that the changes are not necessarily predictable or restricted to only some part of the lifetime of a flow. These observation also hold for larger bin sizes.

In order to get a more general view on the properties of the relative deviation of aggregated

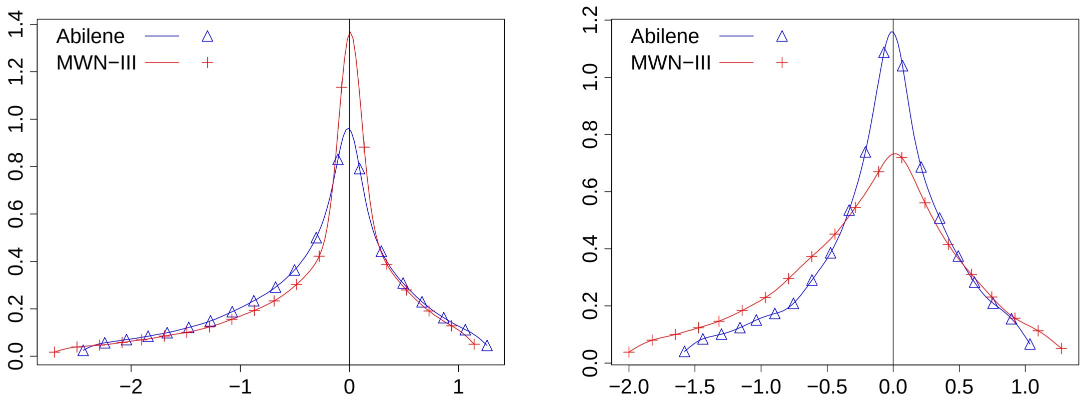


Figure 4.15: Probability density for relative deviation for top 1000 flows for 1 minute bins (left) and 30 minute bins (right). Overall per-bin flow rates are usually close to the average rate of the flow with fluctuations mostly ranging from $1/8$ times up to 2 times the average rate. The range decreases with larger bin sizes.

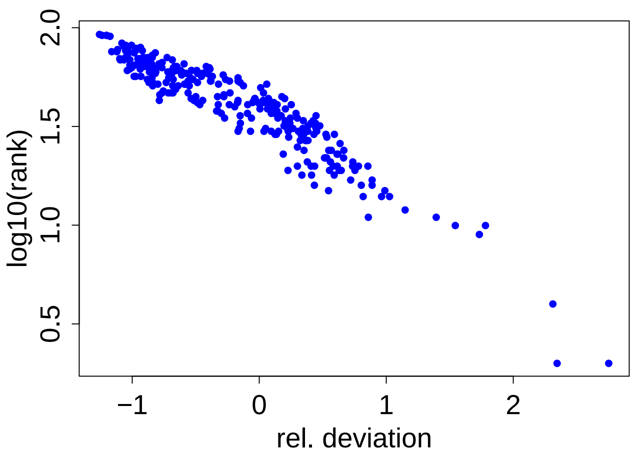


Figure 4.16: Correlation of relative deviation and rank change for an example flow. There exists a strong linear correlation between decadic logarithm of ranks and the relative deviation. This means that flow rate fluctuations are a major ingredient of rank changes.

flows, Figure 4.15 shows, for both data sets, the probability density of the relative deviation for all flows with average rank 1–1000 that span at least 10 bins for 1 minute bins (left) and 30 minute bins (right) respectively. The plot omits all values outside the 5% and 95% quantiles. For 1 minute bins, both curves have a similar form and show a pronounced spike at 0 while covering a range from -3 up to $+1$. which corresponds to flow rates between $1/8$-th of the average rate use up to two times the average rate. With 30 minute bin size the curve for MWN widens slightly while it almost stays the same for the WASHng data set. The spike around 0 indicates that the flow rate of most highly ranked flows does not diverge significantly from their average rate. Still, the broad base in both curves highlights that there are in some cases deviations from the average flow rates, especially towards lower rates. Some of the larger absolute *reldev* values are due to short flow rate spikes or bandwidth drop-offs.

Given the size of the flow rate fluctuations one can expect them to have an impact on the flow ranking. To verify this we plot the relative deviation against the rank of the flows. Yet as the relative deviations are logarithmic we also consider the logarithm of the ranks of the flow. Figure 4.16 shows for a typical flow, the corresponding scatterplot (*reldev* (x-axis) vs. decadic logarithm of flow rank (y-axis)) for 1 minute bins. There is one data point for each time bin that shows the relation between the flow rate during a bin and the rank that the same flow had in that bin. The plot shows a clear correlation between rank and *relative deviation*. This is a clear indication that for aggregated flows bandwidth fluctuations can have a significant influence on the flow ranking. The linear relation visible in the plot with both axes having logarithmic scales (*reldev* is in itself a logarithmic metric) relates well to the observation that the bandwidth use is consistent with Zipf's Law. For a change in the top 10 ranks one needs a much larger rate shift than for a change in the ranks 900 to 1000. Such a correlation is not just apparent for this flow but seems to be present for all flows for all bin sizes.

4.4 Rank Change Metric

The churn inherent among the top ranked flows as shown above in Section 4.1.1 is a major ingredient of instability of flow rankings. The churn rate shows us how much of the flows enter or leave the rankings from bin to bin. However, according to Zipf's Law for flow rates, an elephant entering or leaving a ranking indicates a much more pronounced change in the traffic constitution than when a mouse flow enters or leaves a ranking. In this section we introduce a novel way to quantitatively characterize the volatility of flows in terms of their ranks. More precisely we show how to measure the difference between two rankings R_1 and R_2 in a way that takes into account that the quantities we base our rankings on are consistent with heavy-tailed distributions. Accordingly, changes in the ranking that involve popular elements, i.e., those with high ranks, should have more weight than those involving only non popular elements, i.e., those with low ranks. We propose to use a metric that is based on the inversions needed to transform R_1 into R_2, thereby weighting inversions according to the importance of the ranks.

In the following sections we introduce such a metric and explore its properties. We examine how the metric captures various changes in the ranking and finally apply it to our data sets.

4.4.1 Rank change metric definition

Given two rankings $R_j = r_{j,1},\ldots,r_{j,n}$ with $r_{j,i} < r_{j,i+1}$ and $j \in \{1,2\}$ over the same set of flows, R_2 is a permutation π of R_1 with $r_{2,i} = \pi(r_{1,i})$. There are many ways in which one can define a metric M for $R_1 \rightarrow R_2$. For example one could count the number of rank changes, e.g., for $R_1 = (1,2,3,4)$ and $R_2 = (2,3,4,1)$ the metric would yield a value of 4, or one could compute the sum of the absolute distances between the two vectors, e.g., $1+1+1+3$, or the number of inversions, e.g., 3. Note, that the sum of the distances is twice the number of inversions. The problem with these ideas is that they do not capture the fact that low ranked elements should have a different weight than high ranked ones. The problem with adding a weight to the first alternative of counting the rank changes is that it is unclear how to weight a change. The same applies to computing distances. Only the inversion based approach can be easily extended to use weights. Each inversion is weighted according to the rank where it is applied. The crucial insight is that the order in which the inversions that lead from R_1 to $R_2 = \pi(R_1)$ take place does not matter. Therefore the metric can be computed based on any sequence of inversions that result in π and always yield the same result. Accordingly we define our *rank change metric M* in the following manner:

$$M = \sum_{r=1}^{n} (\mathrm{W}(r) \cdot \mathrm{inv}(r))$$

where $\mathrm{inv}(r)$ counts the number of inversions at rank r and $\mathrm{W}(r)$ is a function that specifies the weight of rank r.

Unfortunately it is not always the case that R_2 is a permutation of R_1. In our case some flows are leaving the ranking while others are entering. Indeed, recall from Section 4.1.1, that the churn rate which captures this aspect is rather sizable. In order to account for this problem the metric is computed on the closure of the two rankings. The closure is computed by first determining the set of *leavers* $\subset R_1$, the elements that appear in R_1 but not R_2, and the set of *newcomers* $\subset R_2$, the elements that appear in R_2 but not R_1. $\overline{R_1}$ is now the extended ranking (R_1, newcomers) while $\overline{R_2}$ is defined as (R_2, leavers). Hereby it is important that the newcomers and the leavers retain the order they have in R_1 (respectively R_2) to ensure that no additional inversions are introduced.

Weight function The weight function W is used to account for the disparity in byte contribution between the flows. As we assume that flow sizes follow a Zipf's-like distribution, we choose the weight function as

$$\mathrm{W}(r) = \begin{cases} 1/r^{\alpha} & : \quad r \leq n \\ 0 & : \quad r > n \end{cases}, \quad \alpha > 1$$

where r is the higher of the two ranks of the inversion, n is the size of the ranking and α is the parameter of the Zipf-like distribution. Note that $r = 1$ represents the highest rank and thus the largest flow. Therefore any inversion involving the top ranked flow is weighted by 1. Moreover, defining the weight to be 0 for ranks greater than the number of flows in R_1 and R_2 ensures that our metric is insensitive to differences in the ranks for the leavers and the newcomers. The weight function has been chosen to match the characteristics of the relative frequency of the occurrence relation $f(x)$ for Zipf-like distributions. It has the nice property that the relation of the logarithms of rank and weight is linear, as is the case for any Zipf-like distribution. If, e.g., the rank of a flow changes by a factor of two the weight decreases by a factor of $1/(2^2)$.

An alternative way of defining the weight of an inversion is based on the actual byte contributions of the two flows at the ranks r_1 and r_2.

$$\mathrm{W}(r) = \begin{cases} \frac{\mathrm{bytes}(r_1) - \mathrm{bytes}(r_2)}{\mathrm{total\ bytes}} & : \quad r \leq n \\ 0 & : \quad r > n \end{cases}$$

where total bytes is the sum of all bytes in the bin and r is the higher of rank of r_1 and r_2. The advantage of using this weight is that distributional properties of the flows are captured implicitly even if they change over time. The disadvantage is that it is more difficult to determine the expected values analytically. Therefore we in this paper focus on the rank based weight function.

Metric Computation Algorithms for computing the inversions to transform one permutation into another are readily available. Using an approach similar to merge sort, it is possible to compute the metric in time $O(n\,log\,n)$ with $n = |\overline{R_1}|$.

The implementation of the metric computation first determines the leavers and the newcomers, and then constructs the rankings $\overline{R_1}$ and $\overline{R_2}$ ensuring that no additional inversion is introduced. Then the inversions are applied one by one to transform the first ranking into the second one. During this process the weight of the inversions is accrued according to the metric definition.

4.4.2 Experimental evaluation

If we assume that the entry process and exit processes are consistent across the ranks with the uniform distribution, it is possible to compute the expected contribution of the arriving and departing flows to the rank change metric. More precisely, if the i-th flow leaves a ranking with probability p_i then the expected cost $E_{i,n}$ of it departing and a newcomer being added at the lowest rank is:

$$E_{i,n} = p_i \sum_{j=i}^{n} W(j)$$

If the weight function is defined as $W(j) = 1/j^2$ the the expected cost of q flows departing and q newcomers being added to the at the lowest ranks the expected cost is:

$$\begin{aligned} E_{q,n} &= q \cdot \sum_{i=1}^{n} \cdot E_{i,n} \\ &= q \cdot \sum_{i=1}^{n} \left(1/n \cdot \sum_{j=i}^{n} W(j) \right) \\ &= q/n \cdot \sum_{i=1}^{n} i \cdot 1/i^2 \\ &= q/n \cdot \sum_{i=1}^{n} 1/i \end{aligned}$$

Even though the entry and the exit process are symmetric the expected cost of replacing one entry with a new entry is not twice the cost of one entry leaving the ranking. This difference, while minor, is caused by a different base value n. The number of ranks decreases by one which decreases n by one. The same applies if one replaces 300 random elements from a ranking with 1000 entries: the expected metric value is $300 * E_{q,1000} + 300 * E_{q,700} = 4.38$. The base value changes from 1000 to 700.

For a ranking with 100 entries and one newly arriving flow the above calculation would predict

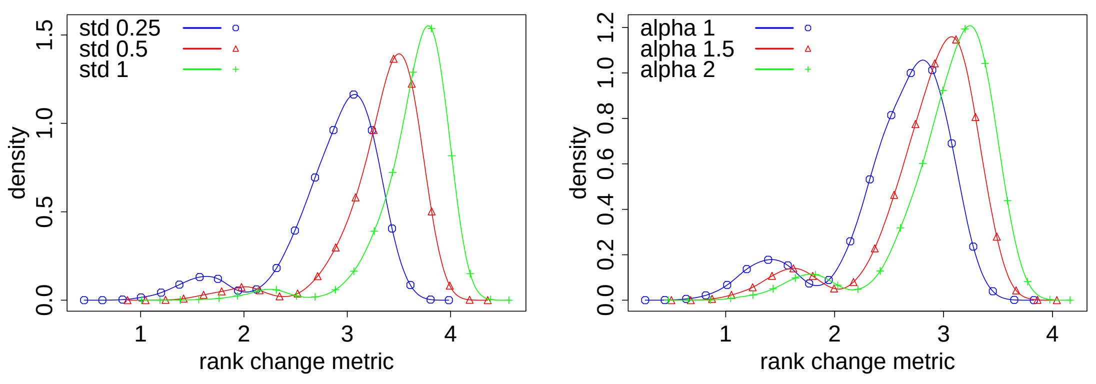


Figure 4.17: Impact of fbw rate fluctuations on the rank change metric assessed by numerical simulations. Bandwidth fluctuations are simulated using normal distributions with different standard deviations (left) and for Pareto distributions with different alpha values (right). The plots show that the metric reacts with growing values to the increased variance of the simulated fbw rates. The amount of metric increase is smaller for Pareto because of the fluctuations are not always suffi cient to result in rank changes.

a metric value of 0.052. An experimental evaluation using an ordered vector of samples from a Pareto distribution with $k = 100$ and $\alpha = 1.0$, where we determined the rank of one additional sample, gives a value of 0.049, only a small difference.

In order to assess the impact of flow rate fluctuations on our rank change metric, we use another simple numerical simulation as in the previous section. We again use 100 samples from a Pareto distributed random variable as flow rates. For this experiment, we chose $\alpha = 1.5$ fairly arbitrary and derive k from the mean rate of /24 destination prefix flows with five minute bins from the MWN-III trace as $29k$. We now modulate the synthetic flow rates using relative deviation values sampled from a normal distributed random variable with mean zero and a standard deviation of 0.54. The standard deviation is derived from the same data as k. Then we calculate the ranking for the modulated rates and compute the rank change metric between the the two bins. We repeat this step 100 times. To explore the impact of different amounts of bandwidth fluctuations, we repeat the entire simulation two more times, but with twice and half the standard deviations for the relative deviations. This gives us data for three sets of experiments with the following standard deviations: 0.25, 0.54, and 1. Figure 4.17 (left) shows probability density plots of the resulting metric values. An increase in bandwidth fluctuation clearly causes the mean metric value to increase (from 2.96 over 3.33 to 3.62). If there would be no bandwidth deviations then $25-35$ of the 100 flows would have to be replaced to get similar metric values. Large fluctuations do not always have to imply a large value for the metric. Indeed, if two flows with consecutive ranks experience bandwidth increases/decreases such that no inversion of the two occurs then the bandwidth fluctuations may not have any impact on the metric. This is the reason why some of the experiments have a significantly lower metric value than others.

The likelihood of whether an inversion occurs depends on the distances between the bandwidth values of the two flows. The larger the distances the less likely is the inversion. This explains why flows ranked towards the bottom are more likely to change ranks than those ranked

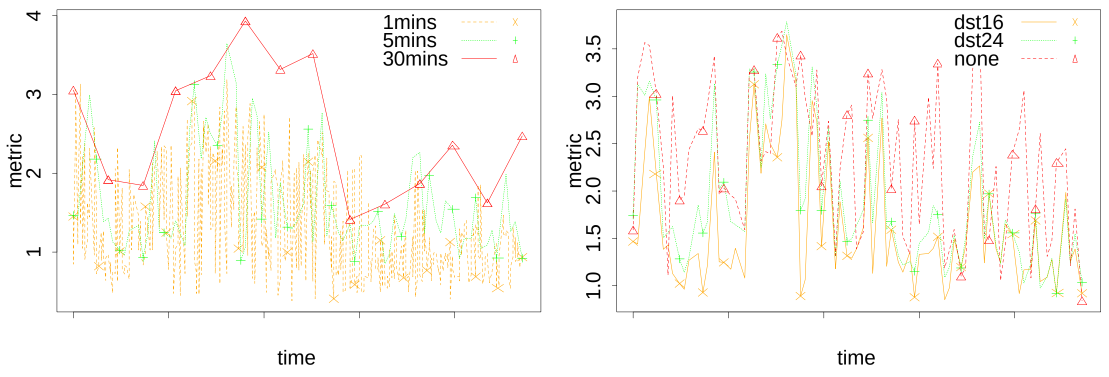


Figure 4.18: Change metric values over time: for different bin sizes (left) and for different aggregations (right). The metric is sensitive to larger bin sizes as the churn is generally higher because of accumulation effects. It is less sensitive to different aggregation schemes as the rate fluctuations of aggregated flows are mostly the result of the behavior of a few contributing five-tuple flows.

towards the top. It also indicates that there may also be a relationship with the degree of heavy-tailedness of the bandwidth values. To explore this aspect we focus on one value for the relative deviation (0.25) but vary the α value from 1 over 1.5 to 2 and repeat the above experiment. This choice of α values ensures a good coverage of the degree of heavy-tailedness. A heavier tail in the Pareto distribution ensures that the byte differences between top ranked flows are larger. Therefore it is less likely that bandwidth fluctuations lead to rank changes in the top ranked flows. Accordingly, the rank change metric values should be less for $\alpha = 1$ than for $\alpha = 1.5$ and for $\alpha = 2$. Figure 4.17 (right), which plots the probability density for the change metric values for the simulation with the different α values, confirms this.

We now return to the actual data and compute the change metric for consecutive time bins. Figure 4.18 shows the metric value for different time granularities and different aggregations vs. time for the time from 9:00am to 3:30pm. As the time granularity changes from 1 minute to 30 minutes the metric increases from a mean value of 1.15 over a value of 1.58 to 2.57. This is not too surprising as the churn rate, Figure 4.3, increases from about 20% to 40%. Yet, when comparing the churn rate with the metric, it becomes apparent that the churn rate is only part of the story. The variability in the metric is significantly higher than in the churn rate. Also note that the metric value for the larger time bins seems to accumulate the value for the lower time granularities. This again confirms that available degree of persistency decreases as the time granularity is increased.

Figure 4.18 (right) explores the relationship between aggregation and rank change metric values. Interestingly, while the metric value for higher aggregations is usually smaller than without aggregation, all curves are rather spiky. This is fairly surprising if one considers that the churn rate at higher aggregations is smoother. This indicates that the spikiness in the metric values is due to bandwidth fluctuations. Bandwidth fluctuations at higher aggregations are likely to be caused by flows that by themselves contribute significant amounts of bandwidth. Therefore it is to be expected that they induce bandwidth fluctuations across all aggregation levels and hence spikes in the metric values. A churn rate of 20% would impose an expected contribution to the rank metric of 1.03. The remaining parts of the rank change metric are due to the bandwidth fluctuations and their resulting rank changes. Overall we note that the metric very nicely captures the variability due to the churn rate as well as due to bandwidth fluctuations.

5 Simulation Framework

The goal of this thesis is the development of a simulation environment that is capable of generating workload traffic, that has the same properties as we have found in our NetFlow and packet level traces. But instead of implementing another network simulation engine, we rely on an existing network simulator to provide the basic components to simulate network traffic, the *NS-2* network simulator. In this chapter, we introduce *NS-2* and a basic workload generator, *NSWeb*, that already provides us with the ability to generate self-similar traffic using a Web based traffic model. After this short introduction, we show that *NS-2* and *NSWeb* are able to reproduce simple network traffic with high accuracy and are thus suitable core components for our simulations environment. We do this by building traces from tightly controlled real Web page requests, automatically configuring simulations based on this traces and compare the simulations with the real Web page requests.

5.1 The NS-2 Network Simulator and the NSWeb Workload Generator

In this thesis, we make use of an existing network simulator called *NS-2* [Pro]. *NS-2* is a discrete event simulator, providing support for the Internet core protocols like HTTP[FGM$^+$99] and several flavours of TCP[RFC81b]. It provides mathematical support in the form of random number generators which are able to generate random number sequences that underlie certain probability distributions, like the exponential or the Pareto distribution. There is also extensive support for logging of events and tracing of protocol interactions. *NS-2* also provides us with the topology and routing issues of our simulation environment. Moreover, we rely on the implementation of the different versions of TCP that comes with *NS-2*.

Only a short time ago, the World Wide Web (WWW) was the predominant contributor to Internet traffic Therefore traffic generators for self-similar traffic has so far been realized by imitating WWW traffic using empirical parameter sets. As shown in Section 2.3, self-similar network traffic is the result of the cumulative traffic of many sources that each generate traffic alternating between sending of data and being idle and the length of both periods is consistent with heavy-tailed distributions, e.g., the Pareto distribution [TWS97]. In the case of the WWW, distributions and parameters for this model have been first described in [Mah97] and [CB97]. Based on these insights, the *SURGE* workload generator [BC98] was proposed. This workload generator models ON/OFF sources indirectly with the help of *User Equivalents* (UE) with user behavior described in terms of page access probabilities and probability distributions for accesses in time (see Figure 5.1). The indirection is necessary because WWW follows the client-server paradigm where clients request data from servers. The actual traffic is thus *initiated* by clients but is *generated* by the servers.

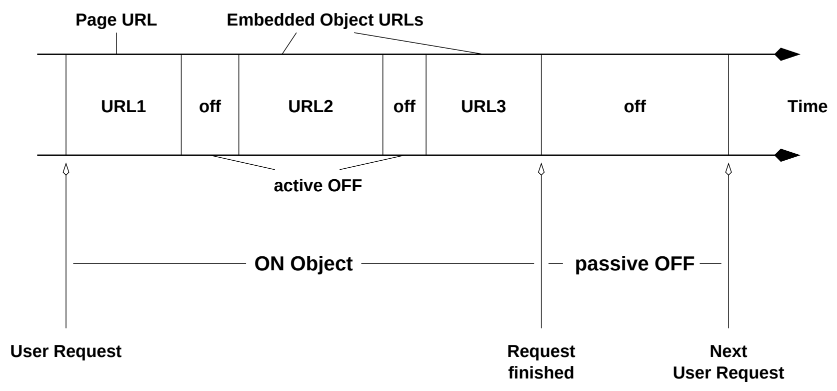


Figure 5.1: ON/OFF Model in *SURGE*. The model is based on user equivalents, and uses two types of inactivity phases. It is defi ned for one transfer per connection and one connection per time.

For this work we use the *SURGE*-based workload generator *NSWeb* [Wal01]. *NSWeb* is an extension for the *NS-2* simulator. It uses the same user and WWW content model as *SURGE*, modeling clients as user equivalents. Such a UE alternates between being idle and issuing requests to a server and downloading data. Figure 5.1 illustrates this process. In *SURGE*, a UE knows two different kinds of OFF phases, the passive OFF or user think time, and the active OFF periods that are responsible for modeling network overhead and client processing time. In addition to the *SURGE* client behavior model (Figure 5.1), *NSWeb* adds the notion of a *user session*. A user session refines the user behavior model by grouping client requests to be near in time and separating these groups with longer idle times. User sessions are modeled much like a *SURGE*-UE with similar distributions and parameters for the ON and OFF times. In addition to the inter-request times that correspond to the client OFF times, *NSWeb* introduces inter-session times. Inter-session times are basically the same as inter-request times, but their duration is usually higher. The lengths of sessions are described in terms of number of requests per session and also follow a heavy-tailed distribution. Figure 5.2 shows how client requests are grouped into sessions. This model has important effects on the burstiness of WWW traffic. It leads to long idle periods followed by phases of high activity when actual WWW requests occur. In [BC98] the authors show, that omitting OFF phases from the user model destroys self-similarity in the resulting traffic under high loads.

NSWeb also provides support for the different variants of HTTP. The first published version of HTTP processed exactly one client request per transport layer connection, using the termination of that connection as an end-of-response indicator (Figure 5.2). Modern Web browsers are capable of using multiple parallel connections and support persistent or even pipelined connection to process user requests. These features have a great influence on network packet traffic, as the overhead of setting up and towing down TCP connections for every Web file can grow very large when requesting many small files. Persistent and pipelined connections are able to reduce this overhead significantly [Mog95].

Web content in *NSWeb* is modeled as pages and embedded objects. For each page there is a

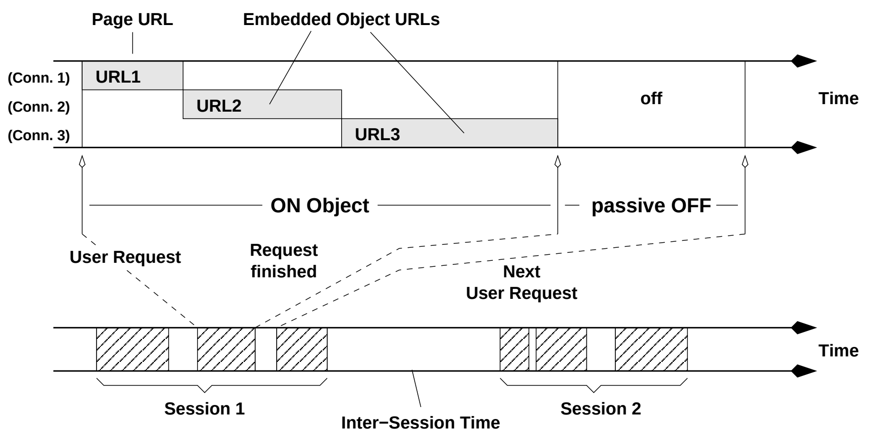


Figure 5.2: The Session Model of *NSWeb* (Simple Connections). The active OFF phases found in the *SURGE* model are gone. Instead requests are grouped into sessions and a intersession timeout is introduced as additional type of OFF phase.

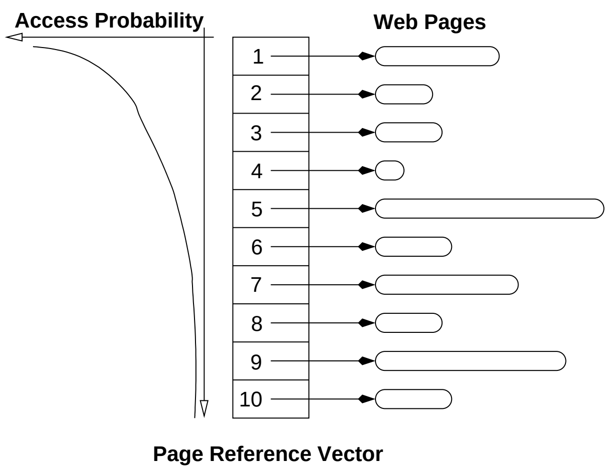


Figure 5.3: Page access probability control in *NSWeb*. Pages are referenced indirectly via a Page Reference Vector using an access probability distribution. This indirection allows for changing the access probability of pages while maintaining the overall distribution.

number of embedded objects, e.g., images, that are loaded by a client after or during the download of the page itself. A page together with its embedded objects are called *web objects* in *NSWeb*. The sizes of pages and embedded objects are consistent with heavy-tailed distributions gleaned from actual network traffic. This is similar to the *SURGE* workload model. In *NSWeb* however, it is possible to place web content explicitly on multiple server instances and have embedded objects referred to by multiple pages simultaneously. This allows for both more realistic scenarios and the simulation of *Content Distribution Networks* (CDN). Content distribution networks are intended to improve access performance by replicating content near to the clients.

In order to generate self-similar traffic by simulating Web requests in such a scenario requires a flexible and highly configurable mechanism to determine, what client requests what content from what server. Such a mechanism is provided by *NSWeb*. In addition to be able to control such complex web content scenarios, it also allows to control page access probabilities to follow a desired probability distribution. This is a key property of *NSWeb* as it allows page access probabilities or page popularity to be configured to follow Zipf's Law and provides us with the second major component to reproduce traffic with a variability that is conformant with our observations in the previous chapters. The core of the page selection mechanism is an ordered vector containing references to every configured page as illustrated in Figure 5.3. The page selection process first determines at which index in the page reference vector to look for the page to request next. This index is selected using a configured probability distribution, which in turn indirectly defines the page access probability distribution. The advantage of this approach is the possibility to change page access probabilities for specific pages without changing the overall access probability distribution. In the case that, e.g., a page is configured on more than one server the same mechanism is used to select a server from which to download the page. We make use of this mechanism to implement a simulation environment that provides for controlled variability in page access probability to reproduce the variability in the set of heavy hitters as described in Chapter 4. For a more in-depth description and evaluation of *NSWeb* see [Wal01].

5.2 On the Realism of Network Simulations

Earlier work by, e.g., [KR01], has shown that the behavior of TCP and its interaction with application layer protocols have a significant impact on the nature of packet traffic caused by these protocols [KW99]. Effects caused by TCP itself are mostly due the congestion control algorithm. While the slowstart algorithm in TCP has a significant impact on Web performance, it is the closed-loop sender control that is responsible for the intricate properties of TCP traffic. This feedback loop leads for example to a phenomenon called *ACK compression* [ZSC91] which has been identified as one cause of the intricate scaling properties of TCP at more fine grained time scales in the order of typical round-trip times observed in operational networks [FHW99]. The interaction of TCP, and especially of the congestion control mechanisms in TCP, with application layer protocols has a major impact on the performance of applications. This has been shown for WWW traffic in, e.g, [Mog95, NGBS+97], and has lead to major changes in HTTP to improve Web performance [FGM+99].

In order to improve the accuracy of our simulation framework, we look in this section into the protocol parameters for TCP and HTTP, how they are configured and what their impact is on the accuracy of our simulations. To this end we first show how to automatically extract values for these parameters from packet level traces of controlled Web accesses. We then evaluate how exact *NS-2*and *NSWeb*are able to reproduce those web accesses using the extracted parameter values. Finally we look into the impact the respective parameters have on the accuracy of our simulations.

5.2.1 Methodology

In order to “reproduce” reality in a simulation framework we need real network traffic to compare our simulations to. To acquire a basis of comparison, we make use of measurements of real world Web transfers. On the one hand, we rely on active probing by performing Web page retrievals in order to determine factors like what real Web content looks like in terms of number and size of files, which HTTP protocol features are supported by the servers and how much time it took the client to retrieve the page and all embedded objects. On the other hand, we passively monitor what happens on the transport layer during the page retrievals. This is done by generating packet level traces of the actively performed Web page retrievals directly at the client end of the network. These packet level traces allow us to determine network related properties like packet round trip times and the bandwidth used by the downloads. In addition, we can glean a set of TCP parameters from the traces, for example maximum segment size and congestion window size. The extraction of TCP parameters and network path properties from passive measurements like packet level traces is a non-trivial task. Therefore we use different heuristics to extract information from the traces and evaluate which heuristics are best suited for our purposes.

To avoid location based or temporal biases in our measurements, we perform measurements from two different locations, one in Europe (EU) at the University of the Saarland (see also Section 3.2), and a second one in New Jersey, USA (US). In order to be able to cope with variations in network conditions, we repeat every measurement multiple times both on weekdays and on weekends. Our measurements also cover different connectivity to the Internet, starting from very low bandwidth connectivity using a modem up to 100 MBit LANs connected to the Internet via a 155 MBit backbone.

Starting from our measurements we extract values for a set of parameters for configuration of topology, Web content and of the protocols in *NS-2* and *NSWeb*. All parameters we consider have a direct or indirect influence on Web performance. Because of the high number of measurements to perform, we automate the entire process of measurement, parameter extraction and simulation as far as possible. We use the time a client needs to fetch a the complete page with all embedded objects as a metric to compare real Web page retrievals against simulated ones.

5.2.2 Configuration Parameter Sets

Before we start analyzing what influence the different protocol parameters have on the quality of the simulations, we first need to determine what parameters have to be considered to achieve this goal.

We start by looking at the application layer protocol HTTP. The main HTTP parameters have to to with how a client connects to servers. There are a number of strategies to do so, starting from a *Single HTTP/1.0 connection*, were a client requests and receives one file per TCP connection and requests a Web page and all embedded objects one after another. In order to speed up page retrieval, several improvements to this scheme have been made. First, clients started to retrieve the files not one after another but used *HTTP/1.0 with parallel connections*. Here, a client can request and retrieve multiple objects simultaneously. A further improvement are *persistent connections* HTTP/1.1. To avoid the setup of a new connection for every file existing

TCP connections are reused. This avoids performance problems due to the TCP slow start algorithm. Moreover it is possible to us *HTTP/1.1 with pipelined connections*, were requests can be streamed over persistent connections. All these features are supported by *NSWeb* and we are thus able to adjust the way that clients retrieve Web pages and so improve the accuracy of our simulations.

Considering that clients retrieves Web pages that consist of several files, it is important to reproduce the number and sizes of the files that make up a Web page and how these files are distributed over a set of servers. For persistent connections to be useful, several files have to be located on the same server. Distributing the files over multiple servers defeats the intended performance improvement. In the case a persistent connection is used to retrieve a single file it is de-facto the same as a simple connection without persistence. We therefore explore the properties of Web retrieved content. This enables us to accurately reproduce traffic of real page retrievals. *NSWeb* provides a powerful interface to configure Web content in exactly this way.

On the other hand, there are HTTP features seen in real Web traffic, that cannot be modeled using *NSWeb*. Earlier studies have shown that caching [Dav01, FCDR99] can have a significant influence on the user perceived latency of Web page retrievals. While caching can improve performance it can also hinder reproducibility and predictability. With caching enabled it is much harder to asses which Web component actually delivered the data or part of the data to the client. Other aspects of HTTP such as bandwidth optimizations (byte ranges requests [KR01] Chapter 7, compression [MDFK97]) or message transmission are not considered.

The behavior of TCP is influenced by a number of parameters. Parameters that have a significant influence on the throughput, especially for small files typical for Web traffic, are the *maximum segment size* and the size of the *initial congestion window* These parameters determine how fast TCP is able to increase its sending rate and adjust it to the network conditions. Another parameter is the size of the advertised *receiver window*. This parameter is able to limit the sender rate according to client capabilities, independently from network conditions. A more complex influence on the behavior of TCP is exerted by using *delayed acknowledgments* and *Nagle's algorithm*. Both algorithms are intended to avoid sending of overly many or unnecessarily small packets.

Finally, we need to consider network conditions and topology as the network itself has a major influence on how fast the protocols can do their work. Because we have no way to extract either exact network conditions nor the topology from actively retrieving Web pages or passively monitoring page retrievals, we have to resort to model the network in an indirect way. We do this by abstracting from the real conditions and reduce the influence of the network to end-to-end metrics in the form of *Roundtrip Times* and *available bandwidth* between all encountered client-server pairs. These metrics already include the influence of the network topology, so we are able to use very simple topologies for our simulations.

Even determining what kind of congestion a Web page download was subjected to is hard. We only observe the results of congestion and packet losses as a reduction of the throughput. The estimation the available bandwidth is also a difficult question in itself (e.g., [All01, Pax97, BA96, LB01]). An alternative to using the available bandwidth is to identify the bottleneck link on the network paths between the client and each server and use its capacity. This implies identifying the link capacities of the network path, which again is a hard question [Dow99, Mah01, CC96].

The TCP throughput is determined by the interactions of the network path characteristics round trip time (RTT) and available bandwidth, the TCP implementation details of the end-systems, and the amount of data to transfer (see e.g., [PFTK98]). We derive the link properties from the path characteristics: delay from round-trip time, capacity from the available bandwidth. This works well under the assumption that these values do not change rapidly. An alternative to such estimation is random choice or default values. Random choice is not desirable while using default values may be appropriate as fallback or in scenarios where the necessary information cannot be obtained, e.g., for downloads via modem, where the modem link is quite likely to be the network bottleneck. However, most of the time measurements can be expected to generate better estimates than default values. Thus we use measurement results to drive the simulations where available and use default values otherwise.

5.2.3 Performing Controlled Web Access Measurements

The candidate set of Web sites used for our evaluation relies on the notion of "popular sites" as determined by some rating site. More specifically we obtain a list of top 135 Web sites from the November 2001 Hot100 [hot] list. Only 122 Web sites existed in late February 2002 when we started our experiments, six of which used redirections and were resolved manually. For the measurements to complete in a reasonable time we choose $40/10$ Web sites for the wide area and modem measurements respectively. We selected the Web sites to represent the distribution of number of embedded objects per page and number of servers involved in each download. To this end we downloaded each page and determined the number and sizes of embedded objects and the number of servers. Based on this information we tried to pick one Web page for each combination: number and size of embedded objects and servers, without checking if the servers supported HTTP/1.1. We perform active measurements in order to estimate the user perceived end-to-end latency of Web page downloads. The basic engine for our active measurements is `httperf` [MJ] which supports all the HTTP protocol features needed for our study. The two key drawbacks [KW00, KA01] of `httperf` are that it does not parse HTML code to retrieve embedded images and that a single run of `httperf` can communicate with only one server. To overcome these limitations we used the same approach and software as used in [KW00]. The drawback of this approach is that it serializes the downloads of the main page and its embedded objects and the downloads from different servers (if more than one server stores objects embedded in the Web page). The advantage of using the method from [KW00] is that the structure/list of the Web page and its embedded objects is determined once. Afterwards this list of objects can be used for repeatedly downloading the Web page, as long as these downloads occur within a short measurement period. To evaluate the performance bottlenecks of the active measurements and estimate the network path properties, we traced each Web page download via `tcpdump` [JLM]. This produces a separate packet level trace file for each Web page retrieval.

To eliminate location biases the measurements are performed from two different locations on two different continents, Europe (EU) and USA (US). Our sample sites, Saarland University, Saarbrücken, Germany and New Jersey, USA, differ in connectivity and network bottlenecks. To increase the range of considered access bandwidth we used a client connected to the LAN and one connected via a modem. The experiments were run from using the same software. At Saarland, we used a 600MHz Athlon PC with 256 MByte memory running Linux 2.2.17. At the US site, we used a 200MHz R4400 with 64 MByte memory running Irix 6.5. Saarland University is connected to the general Internet via a 155 Mbit/s link to the German research network (DFNnet) which is richly interconnected with other networks. The US site is dual-homed to the commercial Internet via a fractional T3 line and a backup T1 line. To increase the diversity in terms of access network bandwidth we run the tests from a similar machine at Saarland University but this time connected via a 28.8 Kbit/s modem line. We denote the Saarland University experiments as WAN(EU) and the US site experiments as WAN(US). Downloads via the modem are denoted as Modem(EU).

We choose a time interval of 6 hours in-between experiments to enable us to capture four experiments within a 24 hour interval. Each experiment consisted of 10 downloads of each Web page. In the modem case the latencies are significantly larger. Therefore we had to limit the number of servers to 10 instead of 40 and the number of downloads to 5 instead of 10. Measurements were performed at 1am, 7am, 1pm, 7pm UTC time during the $3, 4, 5, 15$th of April 2002 for weekdays and $6, 7, 13, 14$th of April 2002 for weekends.

Based on the trace data from the active measurements we performed some sanity checks on the datasets. We found the servers that did not support persistent/pipelining connections on their base server ($10/11$ of 40). Therefore we excluded these servers from the evaluation of these features. Other servers send a `Connection: close` header for the main web page but otherwise support persistent connections and pipelining. This leads to overhead that is not accounted for in the simulations.

5.2.4 Parameter Estimation

In this section we discuss how to extract values for our parameter set from our measurements. Because there are no exact ways to determine values for some of our parameters from passive measurements at a single point near the clients, we resort to heuristics to estimate parameter values.

Page Retrieval Time: We determine our main metric, the page retrieval time, as the time from observing the first SYN packet (from downloading the main page) until the last data packet has been received for each download. Note that we use the last data packet since upon receipt of the last data packet the Web page can be presented completely to the user. Waiting for the close of the connection can be a mistake especially when using persistent connections. In order to eliminate measurement outliers we then compute the median value of all measurements to estimate the user perceived latency for a certain client and Web page at a specific time using a certain HTTP feature.

Web Content: Determining the information about the Web page is fairly straightforward. Using `tcpreduce` [Pax] on a `tcpdump` trace file from retrieving a Web page using a single TCP connection and HTTP/1.0 produces a per connection summary with one connection per object. The size of each object and its meta-information is the number of bytes transfered via the connection. For dynamically generated data there is a chance that another Web page download generates an object of a different size—this is not accounted for. The retrieval order is determined by the start time of each connection. We approximate the number of servers by the number of different IP addresses observed in one of the traces. This ignores IP aliasing. Each object of a given size is mapped to the server corresponding to the destination IP address of the connection. We do not account for Web sites that use DNS or other methods for distributing Web page requests among different servers.

Round Trip Times: In the context of TCP, the round-trip time is understood to be the time difference between the sending of a packet to the receipt of the acknowledgment for this packet. Therefore we just need to find pairs of packets and acknowledgments for estimating the round-trip time. Unfortunately delayed acknowledgments, packet loss, and retransmissions complicate the estimation. There are several ways to cope with them: only consider unaffected packet pairs or explicitly deal with the complication. The latter can be done using the same approach used in TCP. We consider both alternatives.

The packets pairs involved in the initial TCP handshake are not subject to delayed acknowledgments and losses are easy to detect, due to long, deterministic timeouts. Thus, for each TCP connection, we approximate the round-trip time by the time difference between the last packet sent by the client with the SYN flag set and the first packet from the receiver with the SYNACK flag set. The advantages of this estimation technique, denoted SYNACK, is its simplicity, that it is unbiased by application layer overhead, and that it works well for short connections. The disadvantages are that it ignores TCP- and server overhead and only uses a small fraction of the available packet pairs.

We contrast the simple SYNACK method to one that uses a TCP-like mechanism. For this, we use the round-trip estimates from `tcptrace` [Ost]. `tcptrace` computes an estimate of the average round-trip time and the minimum round-trip time for each TCP connection. The advantages of this estimation technique, denoted TCPTRACE, are that it is based on a larger sample set of packet pairs and uses a well tested tool that has proven to be useful in other contexts. The disadvantage is that, since it is based on a TCP like mechanism, it needs more packets in order to derive good round-trip time estimates and that many packet losses can significantly alter these estimates.

Each measurement downloads each Web page repeatedly using different HTTP protocol features. Therefore our set of tcpdump traces should contain several samples of TCP connection between the same endpoints. Accordingly, we derive several round-trip time samples using each round-trip time estimator. Based on these samples we derive an estimate of the round-trip time by computing either the *minimum value* or the *average value* of the round-trip time estimates. This value is then divided by a factor of two to account for the difference between one-way delay and round-trip time. By choosing the minimum (and in the case of `tcptrace` the minimum of the minimum round-trip time estimates) we focus only on the static portion of the round-trip time delay. By using the average round trip time delay (in the case of `tcptrace` the average of the average round-trip time estimates) we have the option of considering some of the variable portions of the round-trip time and thus the influence of congestion.

Throughput: We benefit from the fact that the Web page downloads are sampling the available bandwidth between the client and each server. We use two different techniques for the bandwidth estimation. The first one uses throughput estimations from `tcptrace` for each TCP connection, which is computed by dividing the goodput of the connection by the elapsed time.

To account for slow-start effects we only consider TCP connections which transfer at least 6 data packets from the server to the client. The advantages of this technique, denoted TCPTRACE, are that it is based on a well tested tool, that it only considers goodput, and that it requires a minimal number of transmitted data packets. The disadvantages are that the TCP connection could have received a much higher throughput throughout parts of its lifetime, that elapsed time may include idle periods due to the use of persistent connections, and that the slow-start phase is included in the throughput computation.

To circumvent some of the disadvantages of `tcptrace`'s estimation technique we wrote our own throughput estimation tool which uses a tighter window for computing throughput. The end of the time window is determined by the last packet that carries data from the server to the client. This eliminates the time for closing a connection. The start of the time window depends on the number of packets in the TCP connection. If there are sufficiently many packets (> 12) then we skip part of the slow-start phase by using the timestamp of the 6th data packet as the start of the window. Our justification is that ack-clocking should be established once the window size is larger than 4. Throughput is estimated based on the number of data bytes (goodput) transfered within the time window. The advantages of this method, denoted TAIL, are that it tries to skip the initial slow-start phase and the last idle time before a connection is closed. The disadvantages are that it cannot skip slow-start phases due to packet losses and other longer idle periods (e.g., due to server problems or idle persistent connections). As before, we only applied this tool to TCP connections with at least 6 data packets from the server to the client.

We compute several samples for each bandwidth estimator from the various TCP connections. Based on theses samples we derive an estimate for the throughput by taking either the *maximum value* or the *average value* of the throughput estimates. Our motivation for choosing the maximum value is that the throughput is limited by TCP effects and congestion. By choosing the maximum we use an optimistic estimate. By using the average throughput we are more pessimistic and place a larger emphasis on the influence of congestion.

TCP parameters: Using the `tcpdump` trace information it is possible to analyze what TCP options are supported by the various Web servers and what TCP parameters are used. We examined the traces for the following factors: The *maximum segment size* can be determined using the TCP option fields of SYN packets. The factor that mainly influences end-to-end latencies is the *receiver window size* of the client which is a `httperf` parameter. We use the default value: 16 KBytes. The window value actually used is available via the window information in the packets from the client to the server. To understand if *Nagle* is disabled by the server, one can examine if the server sends several small packets in a row. By default `httperf` disables Nagle on the client. To check if the client is using *delayed Acknowledgments* one can see if each data packet is acknowledged immediately. Certain systems, such as Linux, are known to disable delayed ACKs during slow-start. Another factor that impacts the speed of page retrievals is the size of the *initial congestion window* at the server. Most systems by default use an initial window of one or two. But larger values are also conform with the TCP RFCs [MA02, KP98] and can improve user-perceived latency. To check the server initial window size we inspect the packet level trace to see how many packets are sent by the server at the beginning of the slow-start phase before waiting for an acknowledgment from the client.

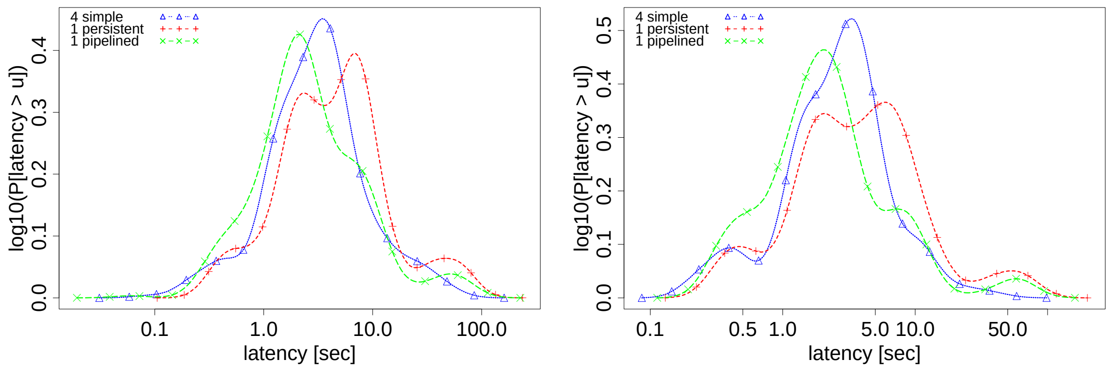


Figure 5.4: Probability density of the end-to-end latency of the EU Web page retrievals: (left) weekday; (right) weekend. The plots show a representative range of latencies. It also shows the improvement in latency of persistent and pipelined connections over simple ones. The latencies for weekends (right) is smaller than for weekdays (left) due to congestion effects.

5.2.5 Measurement Results

Using our estimation techniques we generate and simulate 24 simulation scenarios per Web page, HTTP flavour, and each combination of the techniques for estimating round-trip times and throughput for each experiment. To facilitate a detailed comparison we also collect packet level traces from the simulation. Overall a typical experiment results in roughly 400 MBytes of measurement and simulation traces and logs.

To verify that the Web sites that we picked indeed cover a range of different end-to-end latencies Figure 5.4 shows the density of the logarithm of the end-to-end latency of the Web page downloads from the EU location. The left plot summarizes all weekday experiments while the right plot summarizes all weekend experiments. As expected overall latencies are best for HTTP/1.1 with pipelining, next are the downloads with four simple TCP connections, while using just a single persistent connection is the worst. Somewhat surprisingly pipelined and persistent connections show a peak at 60 seconds. This is `httperf`'s standard timeout value which applies if the server does not send an answer. Since this peak is due to an anomaly it should be ignored which improves the results even more. The latencies for the downloads during weekends are a bit better than those during the weekday, indicating that congestion effects are more prevalent during the weekday than during the weekend, just as one would expect. Similar observations hold for daytime vs. nighttime.

For some of the Web sites the end-to-end performance is subject to significant fluctuations: the median end-to-end latency differs rather significantly from the minimum/maximum values and the standard deviation is of the same order of magnitude as the original latency. Upon further inspection of the packet level traces we found that one factor contributing to such fluctuations is the number of packet drops (approximated via the number of packet retransmissions). Therefore we decided to split the evaluation into two cases. Case one includes all measurements while case two only contains measurements without packet retransmissions. Of the approx. 19000 (3800) measurements from WAN(EU) (modem(EU)) 13.4% (30.4) had retransmissions, respectively. Of the 22300 measurements from WAN(US) 63.9% had retransmissions. We observed more retransmissions in the modem(EU) datasets and the US datasets than in the WAN(EU) datasets. While surprising it can be explained by considering that the transmission of packets over the modem line takes time and that those packets are buffered on the remote end. If the retransmission timeout value is too small the client will retransmit packets before the data can be acknowledged. A similar effect seems to be happening for the US based client.

Except for one server which uses a segment size of 512 bytes all servers use a segment size bigger than 1300 and most servers use the expected segment size of 1460. Other anomalies include Web sites that use embedded objects located on servers that require the use of cookies. Requests without these cookies will just hang until the connection is closed. We observed this on two of the Web sites and eliminated them from further consideration.

5.2.6 Comparison of Parameter Estimators

In Section 5.2.4 we discussed several different ways of estimating round-trip times and throughput. The scatter plots shown in Figure 5.5 plot for each experiment and each involved server the round-trip time estimate using one method vs. the round-trip time estimate derived using another method using logarithmic scales for both axes. In addition we added a line for $x = y$ indicating identical results by both estimators. This plot includes all experiments independent of the location of the client, the time of day or the time of week. This means that we consider more than 3200 bandwidth and over 6200 round-trip time samples.

The left plot of Figure 5.5 compares the minimum of all round-trip time samples to one server during one experiment to the average round-trip time for SYNACK. Note that the round-trip times range from such small values as 2.96 milliseconds all the way up to 1.91 seconds for SYNACK(min). In general we observe that the average round-trip time does not differ too much from the minimum round-trip times. Yet, there are cases where they differ significantly. In the case of SYNACK some of these are due to retransmissions and some are due to the overhead of TCP connection establishment and the necessary server overhead and some are due to congestion effects. It appears that the minimum value is a better estimator than the average value since the effects of larger outliers are minimized. The penalty of using the minimum could be that we put too much emphasis on small values (in effect using the small outliers). In addition we can observe that there is a clustering of the round-trip values around some expected values such as, e.g., 20 ms (approx. time to EU–US), 100 ms (approx. modem delay).

The right plot of Figure 5.5 compares the round trip time estimators SYNACK(min) and TCPTRACE(min). Overall they agree rather well. There are only a few cases in which the SYNACK estimator is significantly smaller than the TCPTRACE estimator. These are likely due to short TCP connections which limits the effectiveness of the TCPTRACE estimator. In a small number of cases the TCPTRACE estimator is smaller than the SYNACK estimator, most likely highlighting the drawback of using a much smaller sample set and the influence of the TCP connection establishment. Overall the results show that even simple round-trip time estimators can provide good estimates for round-trip times.

The scatter plots of Figure 5.6 each compare two throughput estimators. Note that the difference between the estimators for throughput is significantly larger than the difference between the estimators for round-trip time. The plot contains more “noise”. Interestingly one can distin-

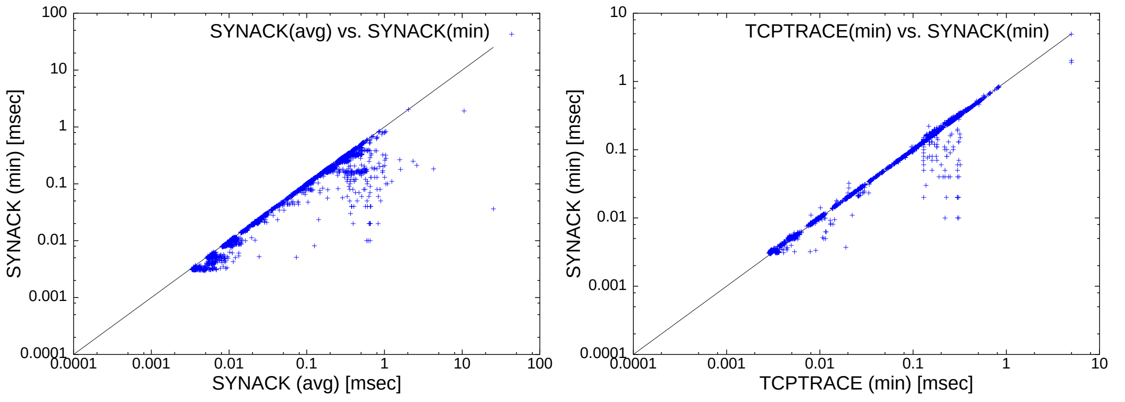


Figure 5.5: Scatter plots of RTT estimates. Each plot compares two different estimation techniques: round-trip time estimates in seconds SYNACK(avg) vs. SYNACK(min) (left) TCPTRACE(min) vs. SYNACK(min) (right). Minimum and average SYN-ACK RTTs do not differ signifi cantly in most cases. Differences are caused by retransmissions and server processing overhead. SYNACK(min) and TCP-TRACE(min) generally agree very well, except for very short connections where TCPTRACE gets too few samples. Overall the very simple SYNACK(min) estimator can yield very good RTT estimates.

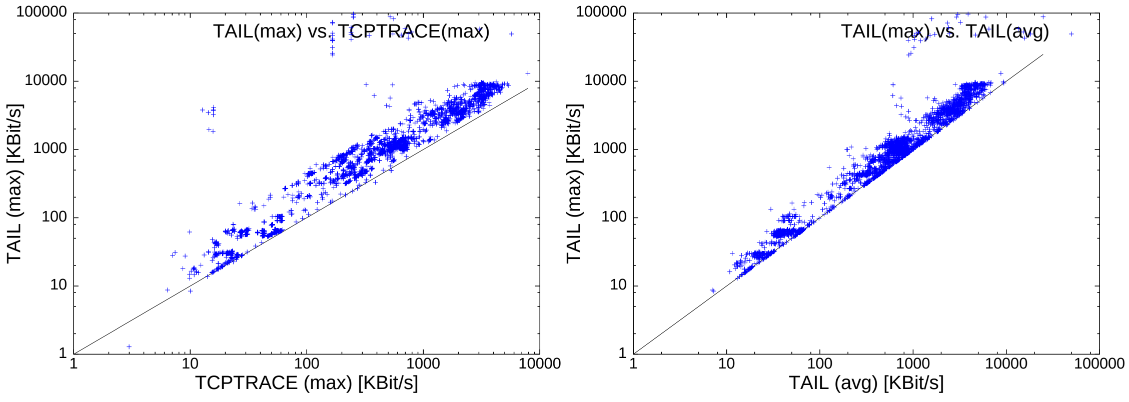


Figure 5.6: Scatter plots of throughput estimates. Each plot compares two different estimation techniques: throughput estimates in Kbit/s for TAIL(max) vs. SYNACK(max) (left) and TAIL(max) vs. TAIL(min) (right). TAIL sometimes overestimates throughput for short connections but is otherwise relatively accurate due to its ability to ignore connection setup and teardown. The clusters are due to common link capacities, e.g., T1.

guish three clusters: one with smaller bandwidth estimates and less data points for the modem case, one for T1 and similar speeds, and one for high bandwidth connections. Another interesting aspect is that the results spread across a large range of bandwidth values: from 130 bit/s to 8.9 Mbit/s for TAIL(max). As is to be expected TCPTRACE(max) is significantly smaller than the TAIL(max). This is the result of ignoring slow-start and connection teardown in the TAIL estimator. It is interesting to observe that for some cases TAIL(max) seems to overestimate the throughput. This is most likely due to cases where the number of packets is just large enough so that TAIL can ignore the slow-start phase but where the next set of packets arrives more or less back to back. These are the clusters far to the left of the $x = y$ line. But otherwise TAIL(max) looks promising. Overall based on the above results we choose the most optimistic network characteristic estimator combination, SYNACK(min) for round trip times and TAIL(max) for available bandwidth, as the base case for our simulations.

5.2.7 Simulation Setup

In order to configure our simulations according to the findings in the previous sections, we chose our parameter values as follows. TCP parameters were chosen so that *NS-2* mimics a TCP RENO implementation without selective acknowledgments (SACK) [MMFR96] and the following options: Nagle is disabled, delayed acknowledgments are enabled with a maximum timer value of 200 ms, the MSS is 1460 bytes, the receiver window sizes are 11 segments (corresponds to roughly 16 KBytes for full segments) and the initial window is one segment. This configuration differs from vanilla TCP only in the sense that Nagle is disabled. But this is a common performance enhancement of Web servers and we have found no evidence of Nagle in the traces.

The topology parameters for each server are derived from the estimated round-trip times and throughput values. In the case where we cannot derive throughput estimates from the packet level measurements, we use the following default values: 56 Kbit/s for the modem (speed of serial interface), 2 Mbit/s for US and 1.54 Mbit/s for EU. We used a slightly higher default speed for the US since most accesses didn't involve transatlantic links that are likely bottle-neck links. Overall we use several different estimation techniques: min/avg from SYNACK and TCPTRACE for round-trip time estimation and max/avg from TCPTRACE and TAIL for throughput estimation. The round-trip time is translated into the link delay by dividing it by two and subtracting the transmission delay for a 40 byte packet. The throughput estimation translates directly into link capacities.

The choice of topology is hard since we do not know the location of the servers in relationship to the client nor do we know the exact path and its characteristics between the client and the servers. We thus choose a rather simplistic topology consisting of a single client and a set of servers which correspond to the servers contacted by the active measurement client. Figure 5.7 shows an example of such a topology. The client node has a direct link to each server node with delay and bandwidth corresponding to the measured roundtrip time and throughput values. The direct links ensure that the path characteristics estimated from the actual downloads interfere witch each other in the simulations.

In order to be as accurate as possible, we have to ensure that the order that the embedded objects of a Web page are loaded is the same between measurements and simulation. Unfortunately due to our approach for actively downloading the Web pages using `httperf` [MJ] the simulation scenario sketched above consisting of one Web page which contains all embedded objects, does not achieve this. In the simulation the client can simultaneously contact all servers and download all embedded objects in parallel. To ensure that in the simulation the client cannot start downloading the embedded objects until it has received at least the main page, the client is

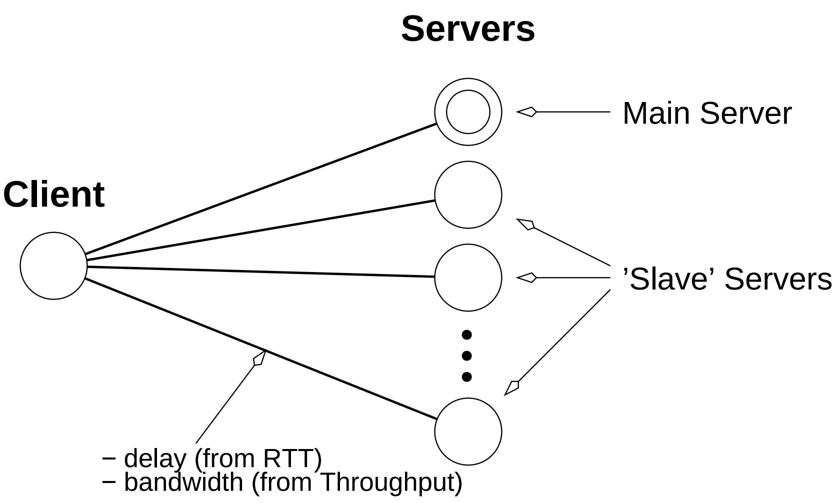


Figure 5.7: Example topology for simulation accuracy evaluation. Every server used by a Web Object is connected to the client according to the estimated capacities and delays. The direct connection prevents that traffi c from a server does not influence the characteristics of the traffi c from the other servers.

configured to delay requests for embedded objects from the main server until the entire page has been received. To ensure that the client can only download objects from a single server at a time, a real Web page request is simulated not as a page with embedded objects but rather as a series of page requests, one page request per server. This implies that for each server, we choose one embedded object and make *NSWeb* treat it as if it was a page object. This page embeds all other objects residing on this server and the client is configured to *not* delay requests for embedded objects for these servers and instead load page and embedded objects simultaneously. Thus, from the view of the transport layer, the simulated client behaves in the same way as the real one.

5.2.8 Evaluation of Simulation Accuracy

To compare the results for each measured Web page we consider the relative error between the median end-to-end latency of the active measurements to the end-to-end latency of the simulation. The relative error is the percentage difference between the larger value and the smaller value. If the latency from the simulation is faster we force the value to be negative otherwise the value is positive. For example a relative error of 100% means that the latency derived from the simulation is twice the latency of the active measurement. A relative error of -50% means that the latency from the simulation is 1.5 faster than the latency from the active measurement. A relative error of 0 means that the latencies are equal. The advantage of using the relative error is that is captures both faster and slower deviations in the same fashion and that it is centered at zero.

Figure 5.8 shows two density plots of the relative error for the 1pm weekday experiments for different client locations, one with four simple connection and one with one persistent connec-

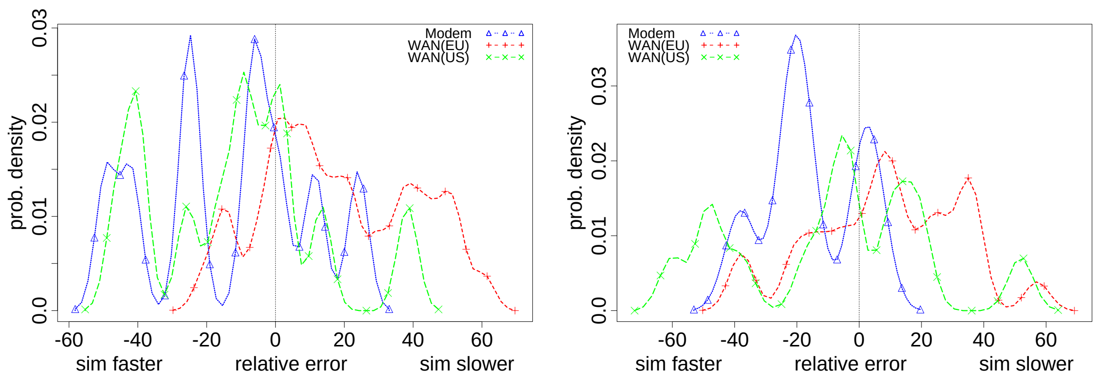


Figure 5.8: Density plots of the relative error for different client location and connectivity for the 1pm weekday experiments using 4 simple connections (left) and 1 persistent connection (right). The relative error is mostly smaller than ±30% and overall smaller than a factor of 2. Simulations are mostly faster for the modem case (capacity overestimation) and slower for WAN(EU) (throughput underestimation because of large RTT values). Using different HTTP connection methods has only a small influence on the relative error.

tion. We only consider measurements without retransmissions for the estimation of the parameters and the latencies. The good news is that most (68% for four simple, 70% for persistent, 78% for pipelined connections) of the relative error values are fairly small: in the range of -30% to 30%. The bad news is that some of the results are of by more than 30%. For the modem case the simulation results are almost always faster than the measurements but not by a huge factor. This indicates that our bandwidth default might be a bit too generous with 56 Kbit/s. This is in contrast to the WAN(EU) case were the simulation tends to be slower than the measurement. This is indicative of a systematic error which causes us to use too large link delays, too small link capacities, or incorrect TCP parameters. The fact that this is only happening for the European measurements is interesting and can be explained in the following way: downloads via modems are mainly constrained by the speed of the modem; download from the US experience shorter round trip times and most likely a more homogeneous network environment; this enables them to open their congestion windows faster and achieve a higher throughput. On the other hand the larger round trip times for the EU environment implies that the throughput estimator might significantly underestimate the actual throughput. In addition this means that underestimating the initial window size has a stronger impact on the EU measurements than the US measurements. This may explain why the simulations are in general slower than the measurements for the WAN(EU) scenarios. Overall the relative errors from the US client cover a much wider range of relative errors. In part this is due to the larger variances already present in the measured latencies.

Taking a closer look at the Web pages where the relative error is large leads us to the following observations. One Web server uses an initial window size bigger than one while the simulation uses an initial window size of one. While in the case of WAN this leads to slower simulated latencies this somewhat surprising yields faster (factor of 1.5) simulated latencies for the modem case. There is a second effect that dominates the first. The bandwidth estimate for the modem

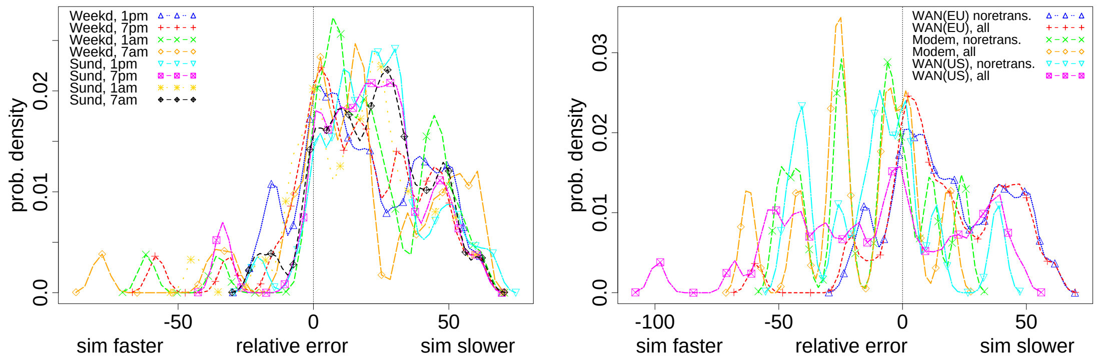


Figure 5.9: Density plots of the relative error for 4 simple connections for the WAN(EU) client for different times of day and different weekdays and no retransmissions (left) and for different clients for 1pm weekday measurements including and excluding measurements with retransmissions. The quality of the measurements (left) is not signifi cantly influenced by the measurement time except for a few cases of bad RTT estimations and high server load. There seems to be however a small systematic error of about 25%) towards slower simulations. If measurements with packet drops are included (right), this error seems to vanish, although there are more outliers caused bey, e.g., retransmission timeouts.

appears to be too small and since the modem client is rather sensible to accurate bandwidth estimations this may explain this surprising difference. The simulated latencies for another server, which also uses an initial window size bigger than one, are also off by about a factor of 1.5 from the measured latencies for all HTTP versions and WAN clients.

In the case of WAN(EU) the outliers with the five biggest positive relative errors ($> 45\%$) are all due to Web pages that consist of lots of small files and where the servers have a bigger initial window size, causing a mismatch to the simulation setup. This is especially pronounced for the case of four simple connections. With persistent/pipelining only $3/2$ still show relative errors $> 30\%$. As reasoned above the impact of the mismatches in initial window size although noticeable is not as strong for the WAN(US) case as for the WAN(EU) case.

We noticed that the typical standard deviation of the measured latencies are significantly larger for the US than for the EU experiments. These variations leave their signature not just on the latency estimates but also on the parameter estimators. For example for one Web page the maximum estimated throughput differs from the average estimated throughput by more than a factor of two. In this case the average would have been a better estimator. The use of the maximum leads to deviations of the simulated and the measured latencies in the order of 50%. Another contributing factor to the differences between the simulations and the measurements is that we used a fairly old measurement machine in the US. It needs roughly 150 milliseconds to start each new `httperf` process. (Note, we need a separate `httperf` processes per server.) This overhead impacts especially Web pages that are spread across many different servers, for example one page spread across four servers.

One of the interesting questions is what influence the time when we conduct the experiments has on the quality of the simulations derived from the measurements. Therefore on the left side of Figure 5.9 we plot the relative errors from each of the eight experiments for four simple connections for WAN(EU). We note that the shapes of the curves do not change dramatically from experiment to experiment. But there are some differences visible. For example the weekday measurements include two pages that suddenly show a much larger negative relative error at -78.7, -61.6 and -57.9. The outlier at -78.7 is caused by a minimum round-trip time that is representing the network state at that particular time rather badly. The average round-trip time is about twice that of the minimum and would have been a better estimator in this case. The second and third outliers (plus the weekend 1*am* outlier) are due to high load on the server, which causes it to send packets at irregular intervals.

On the other hand the Web page responsible for the largest outliers during the weekday 7pm experiment shows a much smaller relative error for almost all other time periods. This underlines that at certain times network effects and server load limit the accuracy of our approach. In addition we note that the accuracy of the simulations for the weekend experiments is smaller than for the weekday experiments. The number of outliers is significantly smaller. The fact that in most cases the simulation is slower than the real measurements and that a lot of the weekend experiments have an error in the order of 25% may indicate that we are suffering from some systematic error, e.g., over estimating the round-trip time, underestimating the throughput, or using wrong TCP parameters. (See Section 5.2.9 for the sensitivity analysis.)

So far we have ignored measurements with retransmissions. The penalty of this is that we are excluding some measurements. On the other hand the estimations of the measured latencies and the network path properties should be more accurate. Figure 5.9 (right) plots the relative errors including/excluding measurements with retransmissions for four simple connections. In this case it is especially noticeable that we get faster simulated latencies indicating that our network path estimators can be fooled by retransmissions. In the case of retransmissions using the maximum of the estimated throughput values is most likely responsible for the larger relative error values. We verified that the extreme outliers (relative error of -98.8% and -96.3%) are indeed related to packet drops. In these cases the client side experienced retransmission timeouts.

Another question one may ask is what is the influence of the network path characteristics estimation technique. We already pointed out a few cases where using a different network parameter estimator might have produced better results. Figure 5.10 (top) shows how varying the estimation technique for round-trip time has an influence on the relative error for the WAN(EU) client keeping the throughput estimator constant. As the scatter plots shown in Figure 5.5 suggest, most of the relative errors stay the same. But some change drastically. For example the number of outliers where the simulation is too fast is reduced by using averages instead of minimum values. On the other hand using averages leads to a larger number of errors in the $+40\%$ range. The results look similar for the other HTTP version and clients.

Varying the bandwidth estimators causes bigger shifts in the relative errors as can be seen in Figure 5.10 (middle) vs. (bottom). Given how much more noise the corresponding scatter plots shown in Figure 5.6 exhibit, this really should not be surprising. Since TAIL estimators yields higher network capacities than the TCPTRACE estimators one expects that the simulated latencies are reduced. Accordingly the plots show the expected shift towards negative relative errors. Especially the relative error for the US client with pipelining indicate that using the least conservative bandwidth estimator maximum of TAIL can be a good idea. But it comes at a price: sometimes it is too aggressive. Probably the best would be something in between the maximum and the average: e.g., some small quantile.

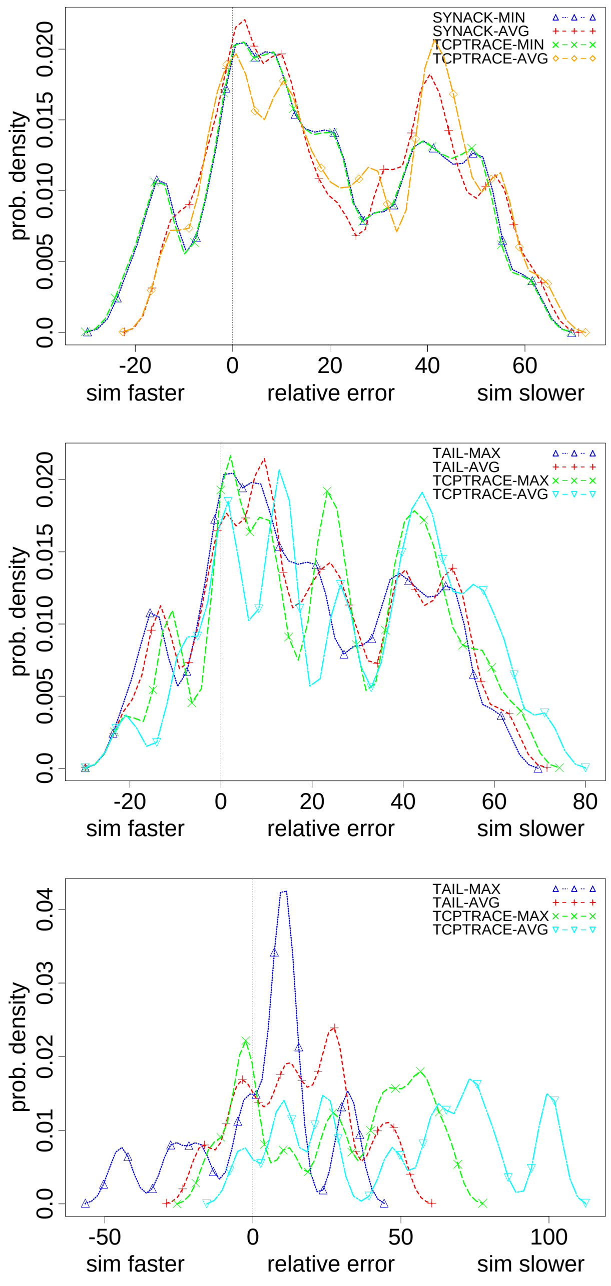


Figure 5.10: Estimation effects: Density plots of the relative error for different link delay estimators for WAN(EU) 1pm weekday measurement and 4 simple connections (top) different bandwidth estimators for 1pm weekday measurement for WAN(EU) and 4 simple connections (middle) and different bandwidth estimators WAN(US) and pipelining without retransmissions (bottom). Using average based RTT estimators instead of minimum based ones leads to more simulations with an error at about +40, but otherwise yield simulations of similar accuracy (top). Using average based bandwidth estimators instead of maximum based ones has little influence for simple connections (middle) but leads to significant errors when using pipelined connections (bottom).

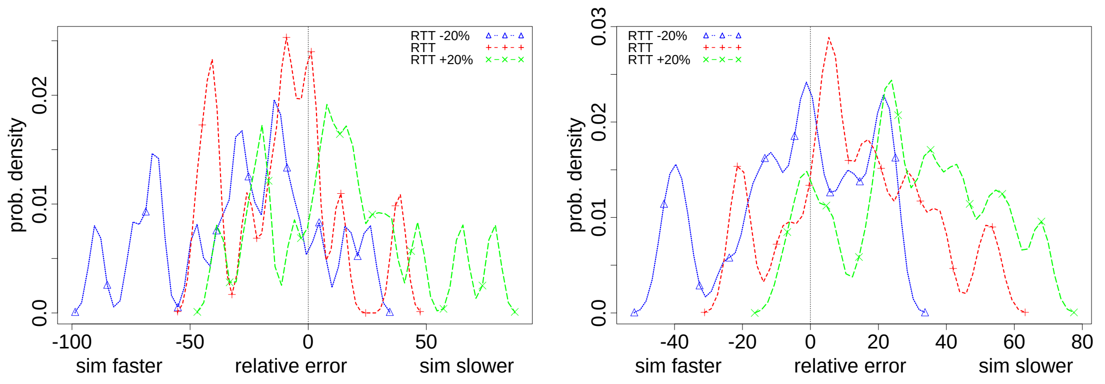


Figure 5.11: Sensitivity analysis: density plots of the relative error for different shifts of round-trip time estimations for 1pm weekday measurement: US client for with 4 simple connections (left) and EU client with pipelining (right) (no retransmissions). In both cases, a relatively small deviation in the RTT estimates leads to an increase in the relative error of up to 50%. This effect is strongest for simple connections due to them mostly being in delay dominated slow start phase.

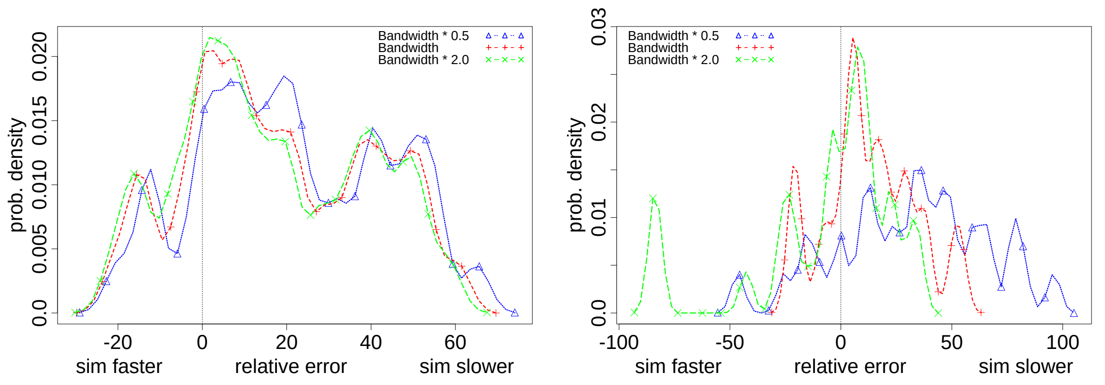


Figure 5.12: Sensitivity analysis: density plots of the relative error for shifts of throughput estimations by 0.5 and 2 for 1pm weekday measurement EU client for with 4 simple connections (left) and with 1 pipelined connection (right) (no retransmissions). Changes in estimated bandwidth have little effect for simple connections but can influence the accuracy of simulations signifi cantly for persistent and pipelined connections that are more susceptible to available path capacities.

5.2.9 Sensitivity Analysis for Protocol and Network Parameters

From the above discussion it should be obvious that there is no “right” set of parameters for the simulation scenario. Matching the real scenario is not always possible. Therefore we, in this Section, present the results of a sensitivity analysis by changing some of our parameters in a systematic fashion. Our base case is a page download without retransmissions using the base estimators: minimum SYNACK and maximum TAIL.

Figure 5.11 shows the effect of changing the value of the estimated round-trip time by $\pm$20%. This relatively small shift can increase or decrease the relative error by as much as 50% for

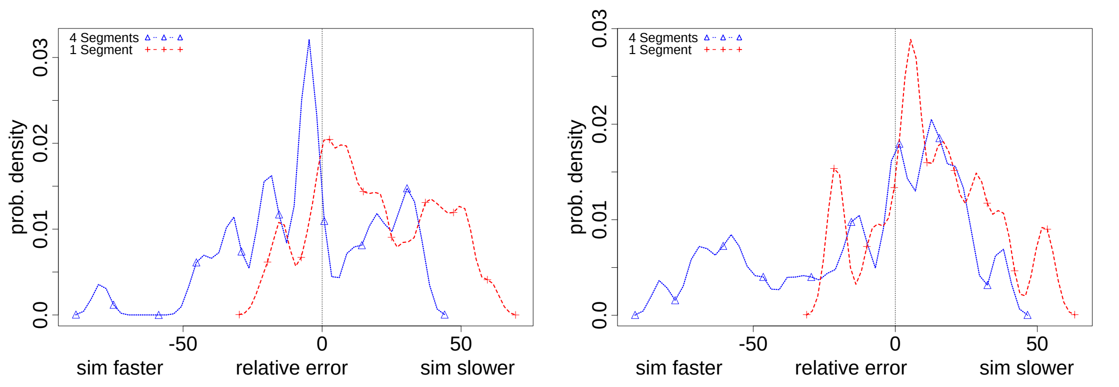


Figure 5.13: Density plots of the relative error for initial window size for EU client for 1pm weekday measurement and no retransmissions with 4 simple connections (left) and 1 pipelined connection (right). In both cases, a larger TCP initial congestion window has a clearly visible influence on the simulated latencies with a larger effect for simple connections. The correct initial window size is crucial for highly accurate simulations.

the US based client using simple HTTP/1.0 connections. The same effect, although not as pronounced, can be seen for different HTTP connection types as well as for different client locations. This effect is due to the small file sizes which leave many TCP connections stuck in the round-trip time dominated slow-start phase. This effect is especially strong for simple HTTP/1.0 connections and for relatively small estimated round-trip time, e.g., for the WAN(US) client. For the modem case, the effects are not as pronounced as for the WAN cases due to the small capacity of the network path. In this case the round-trip time is dominated by packet transmission delay instead of the link propagation or queuing delay. Overall it appears that both, under as well as overestimating the, round-trip time can have quite an impact on the accuracy of the simulations.

Correspondingly one would assume that changing the bandwidth by a factor of ± 2 should not change the simulated latencies for the WAN(EU) client with simple connection too much. Figure 5.12 (left) confirms this assumption. On the other hand clients using pipelining can take advantage of the increased bandwidth (Figure 5.12 (right)). But for some servers the relative error becomes rather large with -95%. Decreasing the bandwidth again has a dramatic effect on some servers. The simulation is now an additional 50% slower. Overestimating the available bandwidth seems, in most cases, to have a lesser impact than underestimating it.

Earlier we noted that a sizable number of the outliers with positive relative error are due to a mismatch of the initial congestion window size. In order to examine this effect more closely, we compared the relative errors for initial congestion windows of 1 and 4 segments. The results are shown in Figure 5.13. As expected the servers with slower simulated latencies improve relative to the error measure. On the other hand servers whose relative error fell in the range of ± 30 may now show much larger errors. It seems that including the initial window estimation in the simulation setup is necessary and could significantly improve our quality measure.

5.2.10 Summary

Given the above results, we find that we are able to very closely reproduce real Web page retrievals using properly configured and parameterized simulations. We have also seen, that the impact that some parameters have on the accuracy of our simulations are far more significant than it is the case for other. Especially the correct configuration of the network link delays and the size of the initial congestion window are far more crucial, than for example link capacities, at least when using such simple scenarios as we did in our evaluation. The accuracy with which our simulations are able to reproduce network traffic at the level of detail corresponding to transport and application layer protocols seems to be sufficient to rely on *NS-2* and *NSWeb* to provide a robust platform to base our simulation framework on.

6 Capturing Traffic Variability in Simulations

In this chapter, we set out to take advantage of the findings on the nature of variability of large network flows and develop a workload generator that reproduces the observed variability characteristics and thus provides the means to develop and evaluate new as well as refine existing traffic engineering and load adaptive routing strategies. We provide a simulation framework that considers the variability characteristics described earlier in this thesis and generates network traffic consistent with self-similarity for per link packet and byte rates, follows Zipf's Law for flow rates, sizes and durations and, in addition, considers the variability and interaction properties of network flows as found by our trace analysis.

6.1 Introduction

As the main goal of our efforts is to provide a simulation framework that captures the variability of network flow rates as observed in Chapter 4 and is thus suitable for evaluating traffic engineering and load-adaptive routing algorithms, we need to extend the *NSWeb* workload generator by two major components. The first component is support for the generation of network flows with properties much like that of flows with higher aggregation levels while maintaining heavy-tailed flow size and flow rate distributions over multiple levels of flow aggregation and time granularity. The second missing component covers support for introducing the same variability in flow rates as characterized in Section 4.1.1 to also capture the dynamics of large Internet flows in simulations.

Using the *NSWeb* workload generator for *NS-2*, we are able to generate self-similar traffic, the first major component of traffic variability observed in operating networks. Because this is done using a *SURGE*-like workload model with heavy-tailed page access probabilities, this implies that we are also able to reproduce the Zipf-like transfer size distribution and thus also for five-tuple flows resulting from TCP connections used to download of files with a heavy-tailed size distribution [Wal01]. In this chapter we extend this framework to allow us to generate a Zipf-like distribution for flow rates on a bin-by-bin basis. As the traffic granularity most commonly used in traffic engineering applications and load adaptive routing algorithms is on the level of destination prefix flows (see Section 2.5), we concentrate our efforts on generating traffic, that not only exhibits a Zipf-like transfer size distribution but also maintains Zipf's Law for flow rates on the level of destination prefix flows. As a first step in developing such a simulation framework, we extend *NSWeb* by the ability to reproduce destination prefix flows using normal Web requests, relying on the page selection mechanism provided by *NSWeb*. Moreover, we enable our framework to generate destination prefix flows that follow a Zipf-like distribution for flow rates, much like we found in our analysis efforts in Chapter 4. In a final step we further

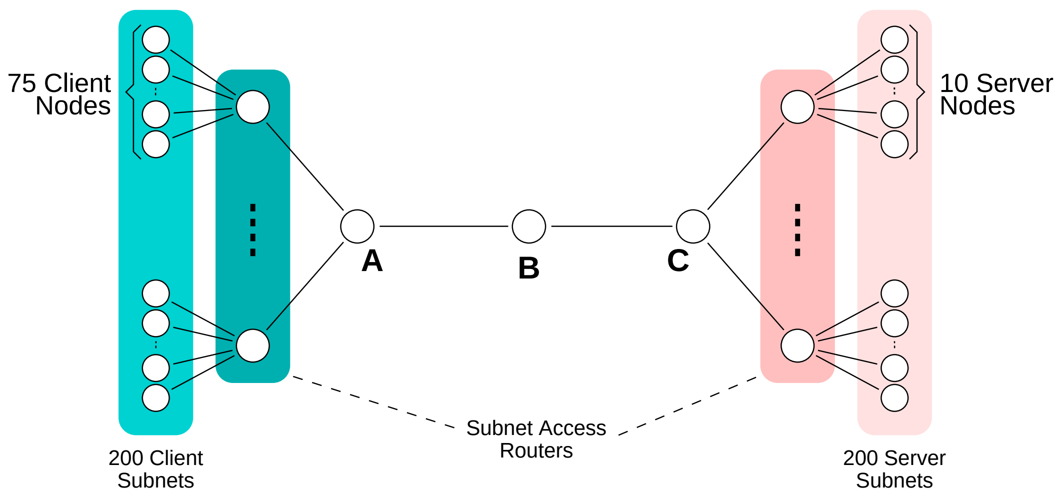


Figure 6.1: Flexbell topology with 200 client subnets for destination prefi x fbws. The subnet access routers are used to assess the behavior of prefi x fbws where the server and client subnets represent the network prefi xes. To control the rate of fbws over a wide range of possible values, we confi gure a relatively high number of clients per subnet. The nodes A, B and C mimic backbone routers and are used to assess the characteristics of the overall traffi c.

refine our framework to introduce the same variability into flow rates as we found in the flow rankings derived from our trace data.

6.2 Static Configuration

We start out by showing how we generate traffic using *NS-2* and *NSWeb* that exhibits self-similarity and a transfer size distribution that is consistent with Zipf's Law. Because the transfer size distribution translates over TCP connections to five-tuple flows, this setup is already able to generate the intended flow size distribution as observed in real world traffic.

Another feature we are able to support is the consistency of flow size and flow rate distributions for destination prefix flows with Zipf's Law. However, the controlled generation of destination prefix flows using *NSWeb* is not straightforward. Remember, that the traffic model of *NSWeb* relies to a large degree on client behavior. In the case of WWW, traffic is caused by client requests, but generated by the server responses. Controlling how much data and with what distributions for rates and flows sizes is generated is thus only possible in an indirect way. Also the temporal and spatial properties of the traffic we need are only achievable by carefully scheduling client requests considering both timing and client selection constraints.

6.2.1 Simulation Setup

In order to generate destination prefix flows, we group the traffic destinations, in this case the clients, into subnets. These subnets are defined by the topology of the simulation (see Figure 6.1). All traffic that is caused by requests by clients of the same subnet travels in the form of the according responses from the servers to the clients of this subnet. Together the clients build the destination of a number of file transfers that individually correspond to five-tuple flows yet are aggregated into a single destination prefix flows in the core of the topology. This is shown in Figure 6.1. The topology consists of a backbone built by the links between routers **A** and **B** and between **B** and **C**. Router **A** connects all client subnets to the backbone via the client subnet access routers. In Section 4.2.1 we have seen that for destination prefix flows it usually is sufficient to consider roughly the 100 largest flows to account for half of the total traffic. In order to increase the range of possible aggregated flow rates, we further increase this number and configure 200 client subnets. Each of these subnets is connected to the backbone using a subnet access router and contains 75 client nodes. There are also server subnets that are connected to the backbone in the same way using router **C** with 10 servers per subnet. We put 10 servers into each server subnet to avoid link overload at the server nodes. The Web page retrievals issued by the clients cause the servers to send data in the direction of the requesting clients. These downloads appear as five-tuple flows at the core router **B** with one of the server nodes as source and one of the client nodes as destination. All five-tuple flows with destination node in the same client subnet can then be aggregated to build destination prefix flows at the core routers. At the same time, these five tuple flows can also be aggregated to source prefix flows when using the source of the five-tuple flows as flow key. As we use *NSWeb* to configure *SURGE*-like Web content and have the clients issue page requests using the *NSWeb* session model, we can expect that the five-tuple flows are consistent with the desired properties regarding the distribution of the size of the five-tuple flows.

So far we have been able to configure *NSWeb* to generate traffic that can be aggregated to destination prefix flows. These aggregated flows however still need to have a flow-rate distribution that is consistent with Zipf's Law. As we have already seen, we can control the rate of flows only indirectly by carefully controlling the access rate of the clients. We have to reduce client access frequency for pages on our servers without destroying the self-similarity of the traffic generated using the session model of *NSWeb*. The only way to do so is to "deflect" requests from the servers that the clients reach over the backbone of our topology to *off-site* servers, that are reachable by network paths that do not include the topology backbone links and routers. This ensures that the *NSWeb* session model is preserved while reducing requests to the servers on the right hand side of the topology. For performance reasons however, we do not really issue requests to off-site servers, but make the clients only pretend to do so. This way, there are much fewer requests to process which significantly reduces the number of packets that have to be processed by the simulation engine. Pretending to perform Web page retrievals in our simulation setup is done by omitting to issue server requests on a per-session basis while still considering when sessions end and new sessions start. The duration of sessions to off-site servers can be inferred using the inter-request times and the Web content configuration. This way the traffic demand of the clients can be reduced while preserving the *NSWeb* session model.

Now that we are able to control the request rate of clients, we still need to adjust this rate in a way, that the combined requests of all clients of a subnet cause the destination prefix flows to have a Zipf-like byte rate distribution. Therefore we have to adjust the request rates not only on a per-client but on a per subnet basis. We achieve this by limiting the number of concurrent sessions for each client subnet. Each time a client starts a new session, it has to check whether adding this new session to the already active sessions would exceed the session limit for its subnet. If this is the case, the client omits this session and schedules a new one using inferred session duration and the configured inter-session time. In order to achieve a Zipf-like byte rate distribution for destination prefix flows, we assign ranks to subnets and determine the per-subnet session limits of a subnet with rank r and number of clients c to be

$$\text{max. sessions}(r) = c \cdot \frac{1}{2^{(r-1)}}$$

This allows at most as many sessions as there are clients in a subnet while enforcing an ordered Zipf-like distribution of the session numbers with the top ranked subnet being allowed to have all its clients issue page requests simultaneously, the subnet ranked second to have half as many active sessions, the subnet ranked third to have a quarter of the sessions as the top ranked subnet, and so on.

Unfortunately, this method only works at coarse time scales while leading to uniformly distributed flow rates at finer time scales in the order of 1 to 5 minutes. This is a result of the granularity of our client control mechanism that operates on the level of entire sessions. The individual page requests and file transfers are still limited only by the available end-to-end bandwidth between clients and servers. In order remedy this problem we limit the available bandwidth between client subnet and servers depending on the targeted rank for the destination prefix flow to the corresponding client subnet. We achieve this by configuring the capacities of the links between the backbone and the *server access routers* as bottleneck links with appropriate capacities and direct all requests of the clients of a subnet to the server subnet that is connected by an appropriate bottleneck link. The link capacities are chosen according to the intended rank r of the destination prefix flows as

$$\text{capacity}(r) = \frac{\text{max. capacity}}{r-1}$$

This approach requires the use of a variant of the basic page selection mechanism in *NSWeb* as described in Section 5.1. Our variant allows a client to first select a server and then select a page that is available on that particular server. As the content of all servers is the same in terms of distributions of file sizes and number of embedded objects referenced by pages, this has no influence on the workload model and thus on the other properties of the generated traffic. In our case we determine the server by first selecting the server subnet according to the targeted end-to-end path capacity and then randomly select a server from that subnet with uniform probability.

In order for clients to issue requests to one of the servers in the correct server subnet we add an additional level of indirection by using Access Maps, the same mechanism that is used for page selection in *NSWeb* (see Section 5.1 and Figure 5.3). There is one Access Map per server subnet that contains references to all servers within a subnet. As we want to select every server in a subnet with the same probability, we use a uniform distribution to determine the server access probabilities.

Whenever a client is scheduled to issue a page request, the page selection mechanism works as follows:

1. Determine the subnet the client resides in. The subnet implicitly defines a single destina-

tion prefix flow with a given rank *r*.

2. Using this rank *r*, we look up the server access map of the server subnet related to rank *r*. Select a server *s* by using the appropriate Access Map.

3. Lookup the page Access Map of server *s*. Select a page *p* by using this Access Map.

In order to complete the topology setup, we still need to configure the capacities of the rest of the links and assign link delays yielding sensible end-to-end round trip times. Nowadays, a large number of clients have broadband access to the Internet like ADSL or cable modems. Moreover our evaluation in Section 5.2.9 has shown that the quality of our simulations is not likely to suffer from using higher capacities than found in real networks. Therefore the only limitation we have on link capacities is that the server subnet access links have to be the bottleneck on the path from servers to clients. Thus the link capacities are configured as follows: The links between clients and their respective access routers have a capacity of 1 Gbps. The corresponding links for servers are configured as 100 Mbps links. The links between the core router **A** and the client access routers as well as the backbone links between routers **A** and **B** and between **B** and **C** all have a capacity of 1 Gbps. The links between core router **C** and the server subnet access routers are configured to support the rank-dependent rate limiting for destination prefix flows described above. Because these links need to be bottleneck links, we configure the capacities with a maximum of 1 Gbps.

In order to have a variety of different realistic round trip times on the paths between servers and clients the links between the clients and their respective access routers have delays of 2–20ms, the delays for the links between the servers and their access routers are between 2ms and 30ms. For the links between the client access routers and the core router **A** we use delays of 3–10ms. To avoid interactions between delays and capacities for the links between the server access routers and core router **C** we use 3ms delay for all these links. The backbone links both have a delay of 2ms. These delay values result in round trip times between 28ms and 134ms.

Besides the topology, client sessions and Web content, we need to configure the protocol parameters for both HTTP and TCP. For HTTP, the only parameter we need to configure is which connection scheme clients use to access the servers. Available connection schemes are four parallel HTTP/0.9 connections supporting one request per connection, one persistent connection per server without request pipelining and one persistent connection per server with request pipelining (see Sections 5.1 and 5.2.2). According to our measurements in Section 5.2.3, 25% of the servers are configured to support pipelined connections, another 25% to support persistent connections without pipelining and the rest to support up to four parallel simple connections. The clients are configured to adjust themselves to the connectivity of the servers so that we get a similar connection mix as found in our Web request measurements.

For TCP we also use parameters as suggested by our measurements. Specifically, we use a maximum segment size of 1460 bytes instead of the *NS-2* default of 512 Bytes, a receiver window of 22 segments or about 32 KBytes and an initial congestion window of one segment. We enable delayed acknowledgments with a delay of 200ms and disable the Nagle algorithm as both features are not enabled by default on *NS-2*2.

In order to be able to generate flows of a length comparable to those found in our NetFlow and packet level traces, we configure the simulation to run for 24 hours of virtual time. As a measure

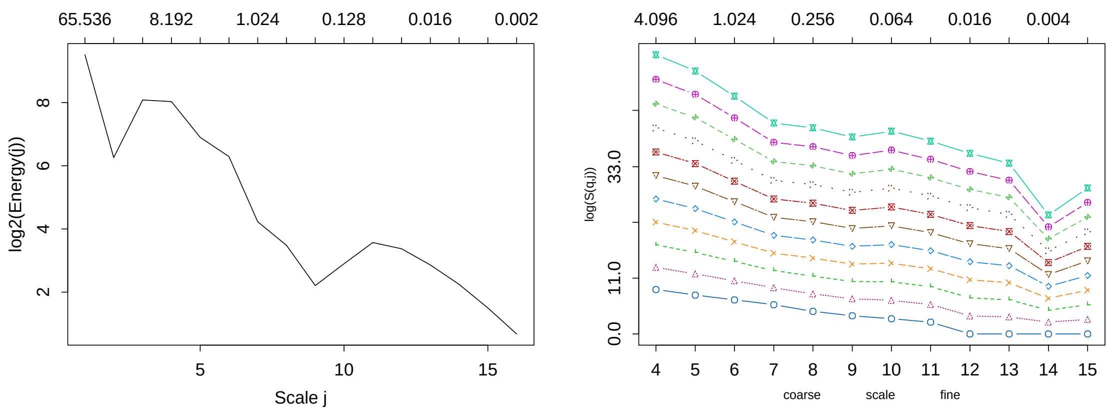


Figure 6.2: Scaling plots for simulated traffi c at backbone link C-to-B, static confi guration. The global scaling (left) shows an approximately linear behavior with a dip at 256*ms* indicating self-similar scaling. The local scaling plot (right) shows similar behavior with no distinct influence of *q* suggesting monofractal scaling. Both plots show the traffi c to be self-similar.

to reduce synchronization effects, the clients start issuing requests at a randomly chosen time within the first 200 seconds of a simulation.

6.2.2 Evaluation and Results

In this section we show, that the traffic generated by our simulation setup shows the desired features, namely self-similarity and Zipf's Law for flow rates.

We therefore first generate packet level traces from the simulation log files at the backbone link between routers **C** and **B** and at all subnet uplinks between router **A** and the client subnets and between the server subnets and router **C**. Starting from the packet level trace at the backbone link, we first generate network flows and then process these flows in the same way we did with our real network traces. We then generated flow rankings for both non-aggregated and destination prefix flows for bin sizes of one and five minutes. Because of the limited number of possible destination prefix flows of 200, we perform the rankings for the top 10, top 20 and top 100 flows only. The packet level traces collected at the various client subnet uplinks are used to analyze the scaling behavior of individual destination prefix flows.

In order to show that the generated traffic is self-similar, we make use of the technique introduced in section 2.3. This method captures variability at different time scales with the help of a Wavelet based energy function. Self-similar behavior is then indicated by a more or less linear dependency of the logarithm of the energy function with non-zero slope over a range of different time scales. For local scaling, a partition function is used instead of the energy function to capture very short bursts. Self-similar local scaling is then indicated, similar to global scaling, by a linear relationship between the logarithm of the partition function and of the time scale. In order to detect multi-fractal scaling behavior, local scaling plots consider a family of partition functions using powers $q = 0, 2, 4, \ldots, 20$ of the Wavelet coefficients. A linear relationship

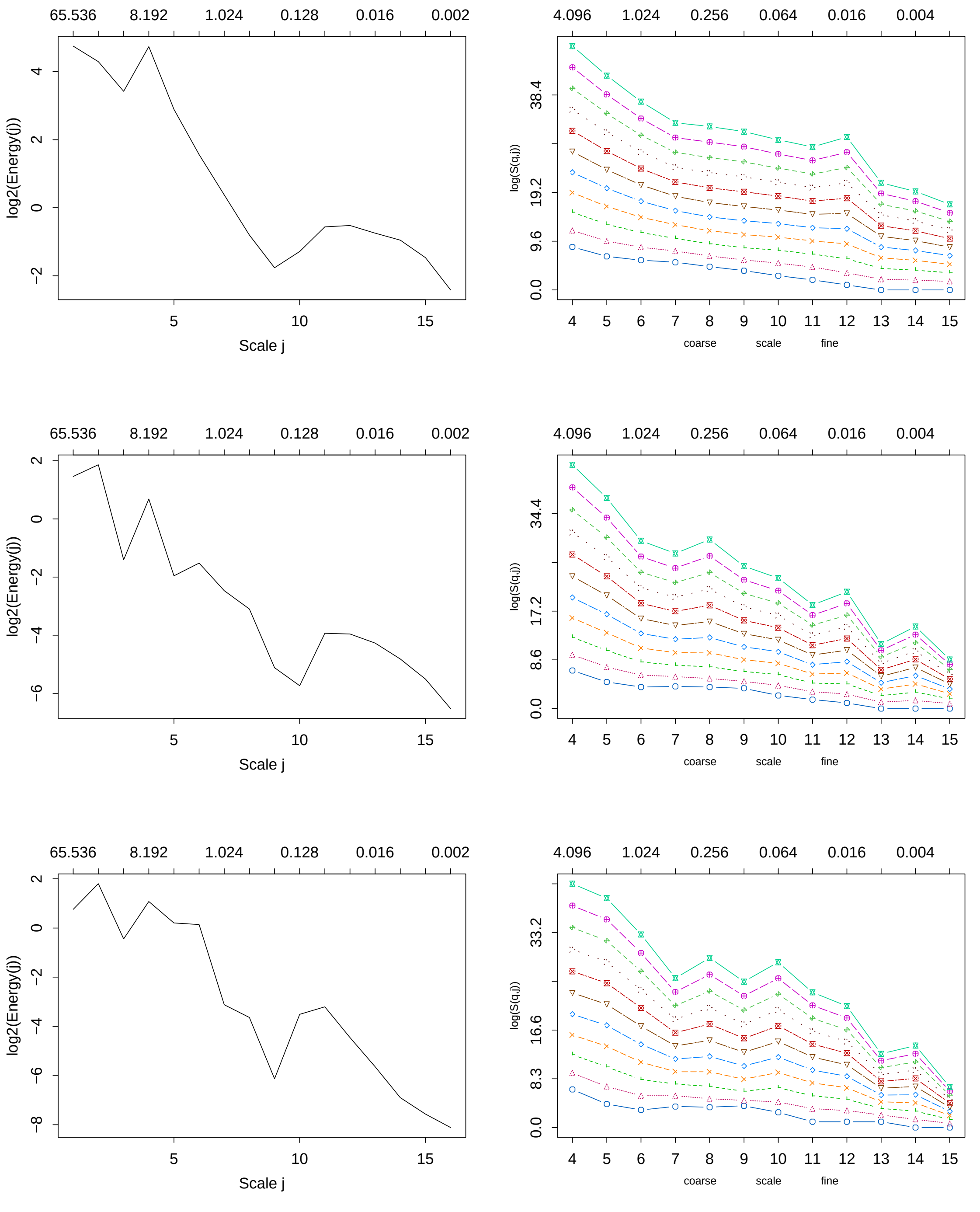


Figure 6.3: Scaling plots for simulated traffi c at client subnet access links for prefi xes 1 (top), 30 (middle) and 100 (bottom), static confi guration. In all three cases, the approximately linear relationship between scale and the logarithm of the energy and partition functions indicates self-similar scaling for the aggregated fbws at that links. The local scaling with its almost linear relationship between scale and the slope of the partition function family hints at multifractal scaling. This shows that the traffi c caused by the individual destination prefi x fbws is self-similar.

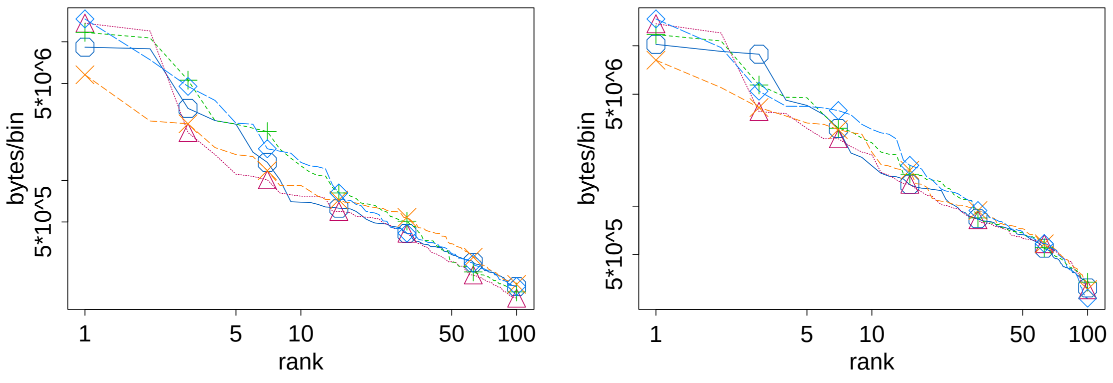


Figure 6.4: Per-bin byte contribution vs. rank for five-tuple flows (left) and destination prefix flows (right) for five consecutive one minute bins (simulation with static configuration). The linear relationship between the logarithms of ranks and flow rates for both non-aggregated and aggregated flows shows, that Zipf's Law holds in both cases, even over time.

between the partition function and q shows multi-fractal scaling behavior. See Section 2.3 for more details on this scaling analysis technique.

Figure 6.2 shows global (left) and local (right) scaling plots for the traffic rate at the link from router **C** to router **B**. The global scaling shows approximate linear behavior over the time scales between 3 and 9 and again from scale 11 to scale 16 indicating self-similarity over a wide range of time scales. The change in scaling behavior between scales 9 and 11 , corresponding to times from about 64ms to 256ms is the result of periodic behavior caused by RTT effects. The time periods are of a magnitude that correspond to the range of RTTs (28ms to 134ms) and multiples thereof. The local scaling plot shown in Figure 6.2 also shows linear behavior over almost all time scales. The linear relationship between scale j and the power q indicates monofractal scaling behavior. The dip apparent in the global scaling plot is only marginally visible for high values of q. The absence of a pronounced dip at middle to coarse time scales is indication that there is no major congestion on the path from the servers to the clients. This is to be expected considering the high link capacities in our simulation setup.

Figure 6.2 shows that the traffic as an aggregation of all destination prefix flows shows self-similar scaling over a wide range of time scales. There remains however the question whether this also holds for the individual flows themselves. As the ranks of the flows are constant during the simulation, we can analyze the scaling behavior flows of a particular rank at the client subnet access links of the respective client subnets. Accordingly, Figure 6.3 shows scaling plots for the destination prefix flows with the static ranks 1 (top row), 10 (middle row) and 100 (bottom row), as measured on the respective client subnet uplinks. The global scaling plots (left column) show approximately the same features as the corresponding plot for the entire backbone traffic. The curves are only slightly less linear for the lower ranked flows at coarser time scales. This observations also hold for the local scaling plots (right column). They show similar behavior as the plot for the entire backbone traffic, again with a slight jitter over the coarser time scales that increases with rank. We assume that the scaling plots for the individual destination prefix flows

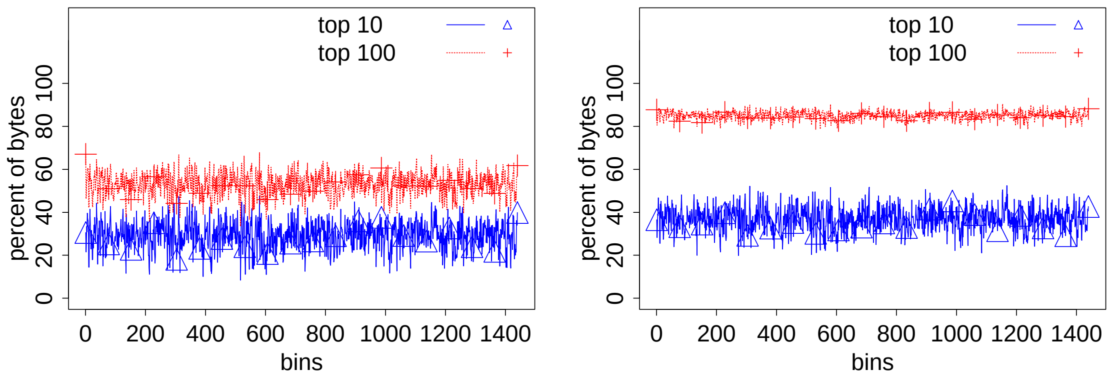


Figure 6.5: Fraction of total traffic contributed by the top 10 and top 100 flows, without aggregation (left) and with aggregation by destination prefix (right) (simulation with static configuration). The byte contribution of the top 10 and the top 100 five-tuple flows is, apart from missing time-of-day effects, very similar to real traffic. For aggregated flows, the simulated flows account for more traffic than real flows. This is caused by the limited number of 200 destination prefixes in our simulations.

are not as clear as for the entire backbone traffic because of the much smaller traffic volume contributed by the lower ranked flows. Still the scaling plots demonstrate that the entire traffic at the backbone as well as the individual destination prefix flows show self-similar behavior over a wide range of time scales.

The second major traffic property we want to reproduce in our simulation framework besides self-similarity is conformity with Zipf's Law for the rates of the destination prefix flows. In order to show that Zipf's Law holds over time for our generated traffic, we plot in Figure 6.4 for five consecutive one minute bins the per-bin byte contribution versus the rank of the top 100 flows on a log-log scale for non-aggregated (left) and destination prefix flows (right). Both plots show approximately linear relationship between logarithms of rank and byte rate over all five plotted bins. This is clear indication, that Zipf's Law for flow rates holds with and without flow aggregation, and even over consecutive time bins. We also find the same behavior for the larger bin size of five minutes. These results show, that our simulation environment is able to reproduce Zipf's Law for flow rates.

The implication of flow rates being conformant to Zipf's Law is that a small number of flows contributes a large fraction of the overall traffic. Figure 6.5 shows the fraction of bytes contributed by the top 10 and top 100 flows for non-aggregated (left) and for destination prefix flows (right). These plots correspond to Figure 4.7, where we show the same relationship for the MWN-III trace. Although our simulations do not show a time-of-day effect as the plots for the trace data do, the top 10 flows contribute about 30% of the total traffic on average and the top 100 flows are responsible for about half of the total traffic. Apart from the time-of-day effect, our simulations come very close to the real world traffic in terms of traffic contribution of top flows, at least for the non-aggregated flows. In the case of destination prefix flows (right) the simulated traffic shows with almost 40% a slightly higher fraction of the traffic contributed by the top 10 flows, than we have for our trace data with about 30% on average. If we look at the top 100

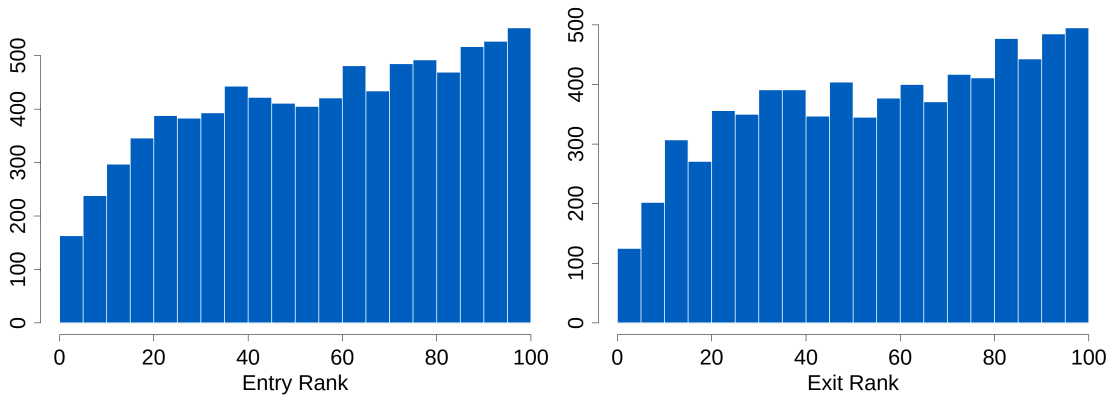


Figure 6.6: Entry (left) and Exit (right) ranks for aggregated fbws and 60 second bins (simulation with static confi guration). The entry exit process look almost identical. They are also very similar to the entry/exit processes of real data, except for higher absolute frequencies as a result of a limited number of prefi xes in our simulations.

destination prefix flows, we see a contribution of almost 90%, which is much higher than for the trace data, where we have a contribution of only 70% on average. Again, we assume this to be caused by the limited number of different destination prefixes (200) in our simulations. Still, there is a remarkable agreement in the byte contribution of top ranked flows between simulated and measured traffic.

Another aspect of the traffic that should already be visible even for statically ranked flows is the entry and exit process as described in Section 4.3.1. For our trace data we have seen that both entry and exit ranks show basically the same distribution with a significant fraction of the flows entering and leaving the rankings at top ranks. For our simulations, Figure 6.6 shows histograms for the entry (left) and exit (right) ranks for aggregated flows and one minute bins. Both histograms are almost identical to each other and also show a rank-frequency distribution that is very similar to that of the MWN-III trace as shown in Figure 4.10. The only visible difference are the slightly increased absolute numbers of flow appearing or disappearing with a rank in one of the 5 ranks wide intervals used in the histograms. This can be explained by the limited number of destination prefix flows in our simulations which leads to a concentration of all aggregated flows in the top 200 ranks, while this effect has no influence on the entry and exit process for our trace data with a much higher number of destination prefix flows. Still, Figure 6.6 shows, that we are able to reproduce this aspect of the entry and exit process with a relative high accuracy.

One of the most important assumptions made in traffic engineering is that large Internet flows usually have a longer duration. This correlation of rate and duration has been described in [KCL06] and we could also show this in our own analysis in Section 4.3.1. Therefore we show in Figure 6.7 probability density curves for destination prefix flows to have particular duration. We discern between flows having an average rank between 1 and 10, between 11 and 100 and lower than rank 100. The curves show quite a different picture of the correlation between flow rate and flow duration than our trace data does. For flows with an average rank lower than 100

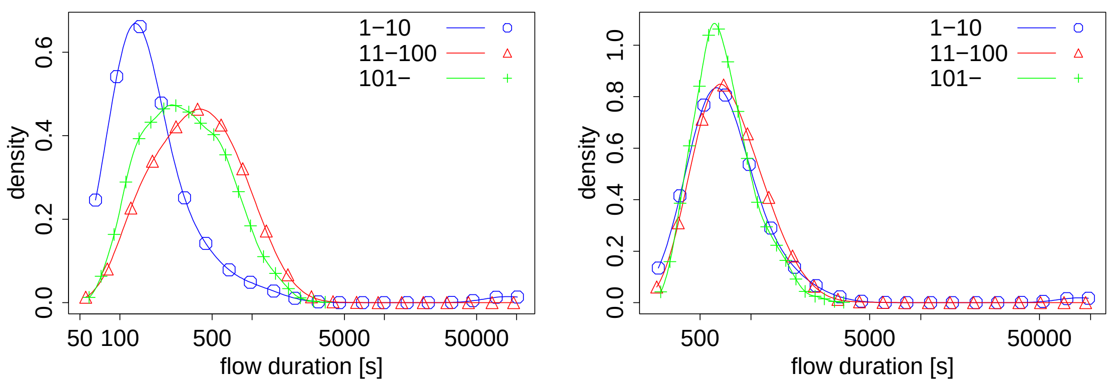


Figure 6.7: Duration of destination prefi x fbws, classifi cation by avg. rank for 1 minute (left) and 5 minute bins (right) for simulation with static confi guration. For our simulations, the duration of top ranked fbws is in most cases shorter than for other fbws, although there are some top 10 fbws, that last as long as the entire simulation. This observation holds for both small and medium sized bins.

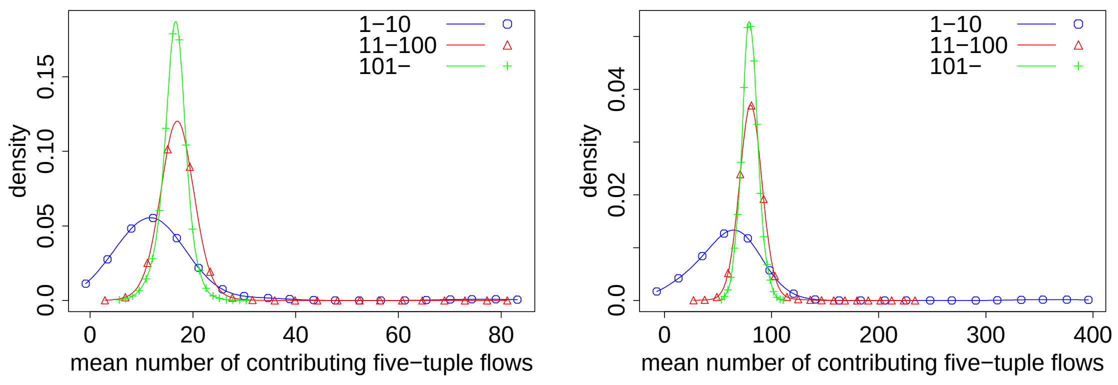


Figure 6.8: Mean number of contributing fi ve-tuple fbws for destination prefi x fbws, classifi cation by avg. Rank for 1 minute (left) and 5 minutes (right) for simulation with static confi guration. Top 10 destination prefi x fbws consist of less fi ve-tuple fbws than lower ranked ones. This increases the probability for top 10 fbws to terminate due to lack of contributing fi ve-tuple fbws. This explains the limited duration of aggregated top fbws.

there is an upper limit for the duration at slightly less than 5000 seconds or approximately one hour and 20 minutes. Moreover flows with average rank between 11 and 100 have a higher probability to last for a long time than flows with low average rank. However aggregated flows with very high mean ranks are in most cases shorter than all others with high probability at about two minutes and a slight probability mass concentration at 24 hours, the duration of our simulations. This observation holds for 1 minute bins as well as for 5 minute bins, where this effect is less pronounced for longer bins. The duration of aggregated flows however directly depends on the constant presence of at least one contributing five-tuple flow. As soon as there is no contributing five-tuple flow for a longer time period than specified by the flow inactivity timeout (15 seconds), an aggregated flows is terminated. Accordingly, we show in Figure 6.8 density plots for 1 minute (left) and 5 minute bins (right) of the average number of five-tuple flows contributing to the same aggregated flow. For aggregated flows with a mean rank of 1 to 10, the number of contributing five-tuple flows is generally smaller than for lower ranked aggregated flows. Moreover, the number of contributing five-tuple flows scales approximately linear with the bin size, independent of the mean rank of the aggregated flows. This is clear indication, that the top ranked aggregated flows consist mostly of a small number of short high-rate five-tuple flows. Therefore the short durations of the top ranked aggregated flows in our simulation are most likely the effect of a too small number of contributing five-tuple flows. To rectify this problem, we have to generate more five-tuple flows which, in our simulation environment, we can only accomplish by configuring more clients. Unfortunately, this is not possible due to the high memory consumption of the *NS-2* network simulator.

We do not yet look into relative deviation as we have not yet introduced our mechanism to change the ranks of aggregated flows in a controlled manner and thus the variability is not comparable to that of real traffic. Yet for the characteristics that do not depend on changing flow ranks we can show a reasonable accuracy of the consistency of our simulated traffic with Zipf's Law, for individual bins as well as over time, and also with self-similarity as can also be found in real traffic.

6.3 Dynamic Configuration

In the last section, we have shown that we are able to simulate self-similar traffic that is consistent with Zipf's Law for the rates of flows aggregated by destination prefix and does so over time. The reproduction of the dynamics of flow rankings highlighted in Section 4.1 however is still missing. Starting from the setup described in the previous section, we now extend our simulation framework to be able to reproduce variability in the ranks of the destination prefix flows. To this end, we add a mechanism, that is able to variate the average ranks of the aggregated flows in a way that the same or at least very similar variability as we found in the trace data is introduced.

We assume that the rank change behavior can be described by a memoryless system, that is, the rank in bin $i+1$ is not dependent on the ranks in bins 1 to i, except that the average of all ranks is preserved. In other words, we assume that there are no temporal dependencies in the sequences of ranks. Based on this assumption, we rely on Markov Chains to model the rank changes for aggregated flows. As flow rankings describe the rates of flows only in relation to the rates of all other flows, this does not necessarily contradict the property of aggregate traffic rates showing self-similar scaling behavior.

6.3.1 Markov Modulation for Ranks

In this section, we now describe the mechanism we use to vary flow ranks over time so that the rank change dynamics found in Section 4.1 can be reproduced in our simulation framework. This mechanism uses a Markov Chain to determine how to change the target ranks for flows over time.

A Markov Chain is defined as a finite state machine with probabilities for each transition. The transition probabilities describe the likelihood that the next state of the machine is s_{i+1}, given that the current state is s_i [Fre71]. This definition implies, that the sequence of transitions leading to s_i has no influence on the transition from s_i to s_{i+1}. The system is memoryless. In our case we need to model the way that flow ranks change over time. We want to know with what probability the rank r_i of some flow during time bin i changes to rank r_{i+1} in the next bin.

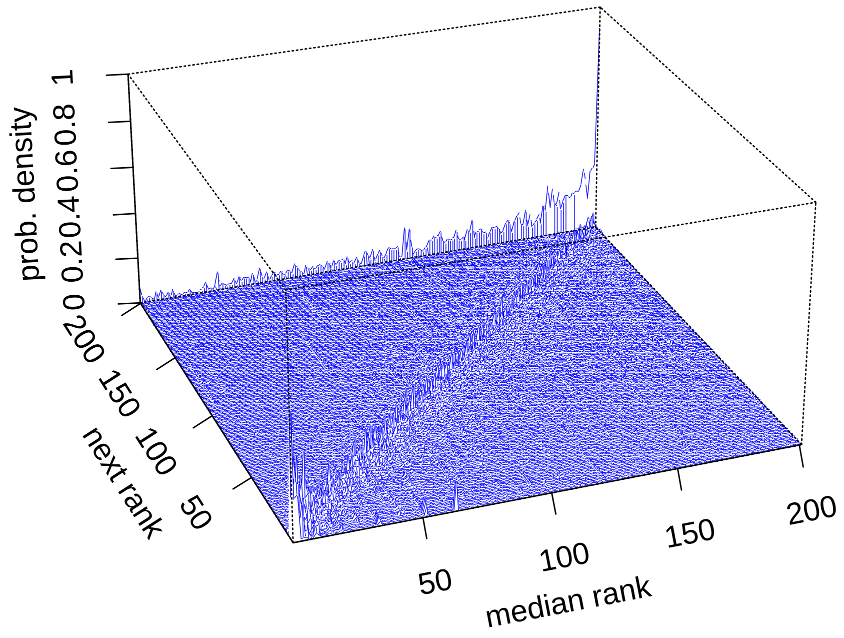
However we have seen in 4.1 that there exists a significant amount of stability in the ranks of flows, depending on the average ranks of flows. This stability is highest for elephants and decreases for hybrids and mice. This means, that a process with states consisting only of the current rank of a flow cannot be memoryless. Accordingly, we need to model the states of the Markov Chain not simply as ranks but as as tuples $(r_{avg};r_i)$, where r_{avg} is the average rank and r_i is the current rank of some flow. This process is memoryless if the transition probabilities for $(r_{avg};r_i)$ are equal for all i, that is, the transitions do not depend on the current rank. Then the transitions depend only on r_{avg} which is constant for any given flow. In order to model flows leaving a ranking, i.e., flow termination, we use a special state $(r_{avg};r_{max+1})$ to account for the corresponding probability.

Changing the ranks of the flows using this model is implemented in the following way. Using a sample u from a uniformly distributed random variable in $]0;1[$, we look for the largest rank with a *cumulative* transition probability that is smaller than u. With p_j being the transition probability to rank j, we thus determine the new rank r_{new} of a flow as

$$r_{new} = max(i) \text{ with } \left(\sum_{j=1}^{i} p_j\right) \leq u$$

When changing the ranks of all destination prefix flows, we get collisions, where we attempt to assign the same new rank to two different flows. We resolve such collisions by prioritizing high ranks. This is achieved by determining new ranks for flows in the order of descending current rank. Whenever we want to change the rank of a flow to a rank already configured for another flow, we chose instead the next lower rank (higher index) that is not yet assigned to any flow. The resulting ranking is used to replace the mapping between destination prefix flows and client subnets used in the first step of the page selection algorithm described in the previous section.

The transition probabilities for a flow with average rank r_{avg} to change its rank from any rank r_i to r_{i+1} are empirically derived from the rankings we derived from our trace data. We do this by first determine the integer average rank for all flows. We then count, for all flows of a given average rank, the ranks these flows had during their lifetime and normalize the counters to probability values. This way, we get one Markov chain for every average rank. As we want to use the Markov chains to configure our simulations and we are not able to simulate more than 200 prefixes, we construct the Markov chains for 200 ranks and one meta rank to account for flows outside the top 200 flows. Because Markov chains derived in this manner show a high dispersion of the probability values, we use median ranks for flow classification and Markov chain construction. Median ranks lead to a much clearer rank distribution than mean ranks. Figure 6.9 shows surface plots (density plots of 2 variables) of the Markov Chains for the WASHng trace for destination prefix flows and 60-second bins covering 200 ranks. On the x-axis, we plot flow median rank and on the z-axis the new rank that a flow is changed to.



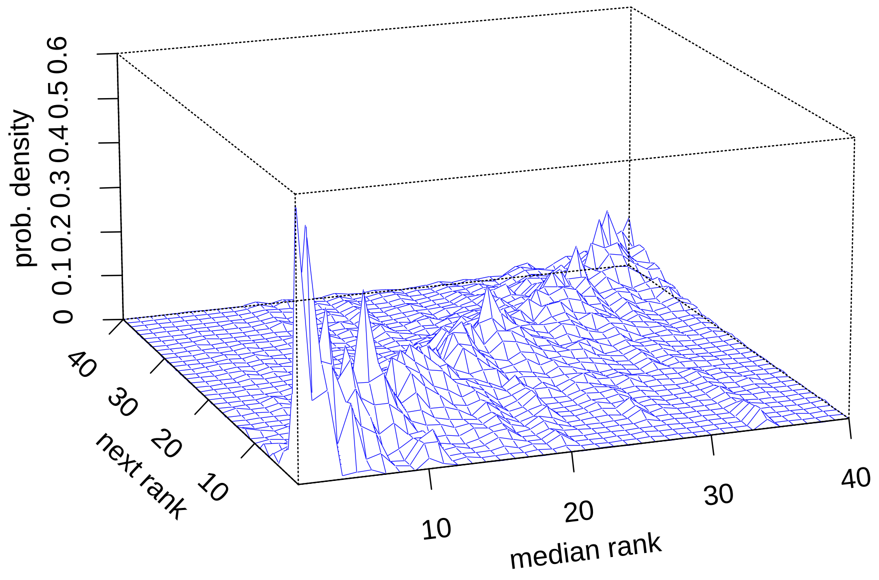


Figure 6.9: Markov chain for WASHng data set, destination prefi x fbws, 1 minute granularity. with probability density for ranks depending on median rank. On the x-axis we plot the fbw median rank and on the z-axis we plot the rank that a fbw is changed to. On the y-axis we show the transition probability for a fbw to change its rank to a given new one. The probability mass is concentrated along the diagonal with a high probability mass at the top ranks. The mass for next rank beyond 200 is caused by fbws disappearing from the rankings. This mass concentrates at fbws with low median rank. The top plot covers all ranks while the bottom plot shows a detailed cut-out of the top 40 ranks.

The y-axis represents the transition probabilities for a flow with a given median rank to change to the respective new rank. Not surprisingly, the plot shows a high probability concentration along the diagonal where the new rank is near the median rank. Moreover, the concentration is clearly more pronounced for the top average ranks (elephants) as for lower ranks (hybrids and mice). This is consistent with the results of our analysis in Section 4.1.1, were we found a higher stability in ranks for top ranked flows than we did for lower ranked ones. The plot also shows that the probability to change to a rank beyond 200, which is, as we have described above, equivalent to disappearing entirely from the ranking in some bin, increases approximately exponentially with the median rank of a flow. Again, this is consistent with the findings of our analysis.

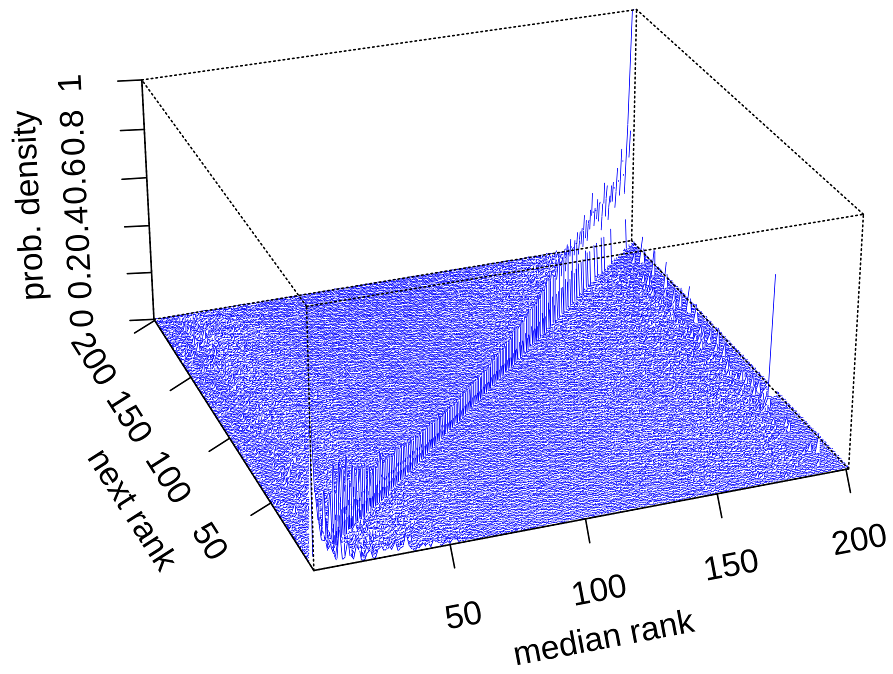
6.3.2 Simulation Setup

To validate our simulation environment with support for varying the ranks of the destination prefix flows, we build upon the setup of the static configuration described above. We reuse topology, link delays and capacities, Web content, the *NSWeb* session configuration and all protocol parameters from our static simulation setup.

In order to incorporate the Markov modulation mechanism, we schedule calls to the stepping function of the Markov modulator that calculates a new mapping vector between destination prefix flows and their targeted ranks. According to the most fine grained time scale used in our simulations, we schedule these calls every 60 seconds in terms of simulated time. In order to examine, if the frequency of modulating the ranks of flows has an influence on the variability of the traffic, we also performed simulations where we change the prefix flow to rank mapping every 5 minutes.

6.3.3 Evaluation and Results

As the dynamic configuration of our simulation environment differs from the static configuration only by the Markov modulation system, we begin our evaluation by ensuring that the modulation mechanism works as intended. We do this by deriving the Markov chains from rankings, generated from simulation trace files in the same manner as described above for the static configuration. The resulting Markov chain for 60 second bins and destination prefix flows is shown in Figure 6.10 as a surface plot. There are two features in the top plot that are clearly different from the Markov chain derived from the MWN-III trace. The first difference is the zero probability for flows to have a rank beyond 200, which is not surprising, as there only are 200 destination prefix flows in our simulations. The second variation is the increasing probability concentration along the diagonal for flows with a median rank larger than 100, which shows that low ranked flows stay low ranked but cannot leave the ranking. This also is an effect that is caused by the limited number of aggregated flows. On the other hand, we can see on the bottom plot, which shows the Markov chains for flows with a median rank of 40 or higher in more detail. The high probability for flows with a median rank in the top 10 ranks to stay in the top 10 ranks shows, that we are able to generate a certain persistence in ranks for the top flows. It is also visible that the probability concentration along the diagonal is far less distinctive for flows with median rank in the top 10 than for the rest of the flows. This shows, that the modulation mechanism intro-



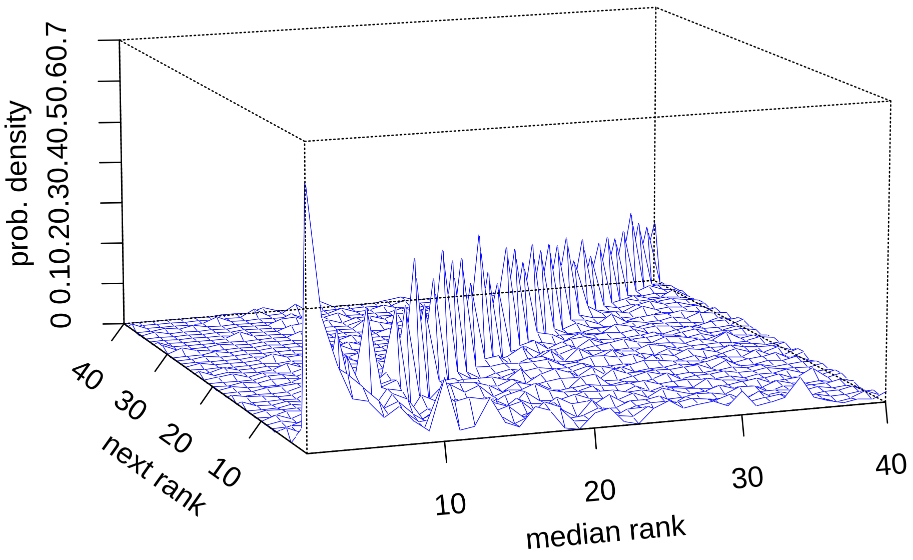


Figure 6.10: Markov Chains derived from simulated traffi c. The Markov Chains derived from the simulated traffi c shows a much higher concentration of the probability mass along the diagonal than was the case for real traffi c. The probability mass broadens out at for top 10 median rank flows which is the result of the desired high flow rate fluctuations causing top flows to change their ranks.

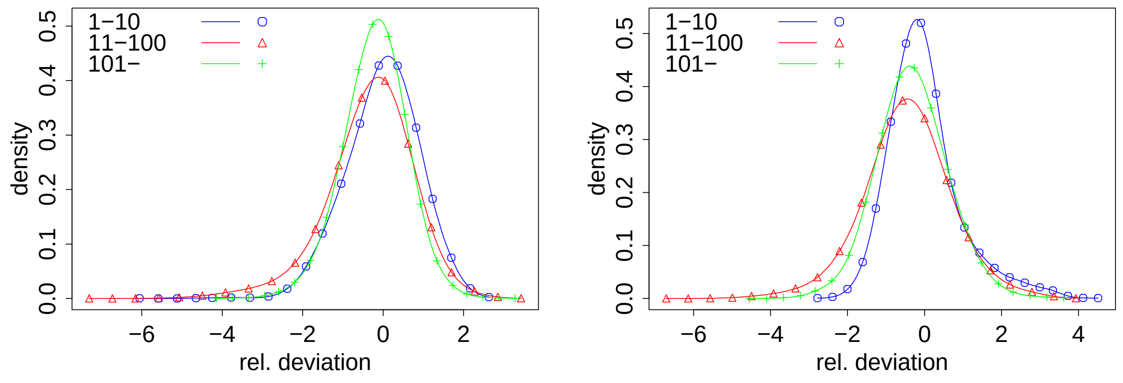


Figure 6.11: Probability density plots for the relative deviation of destination prefi x fbws for static (left) and dynamic (right) simulation confi guration. There is a signifi cant probability mass for relative deviation between +2 and +4 for the dynamic simulation confi guration than for the static one. Especially for top 10 fbws, the probability mass is moved towards positive relative deviation values. This indicates that the Markov modulation mechanism is able to increase the variability of fbw rates signifi cantly.

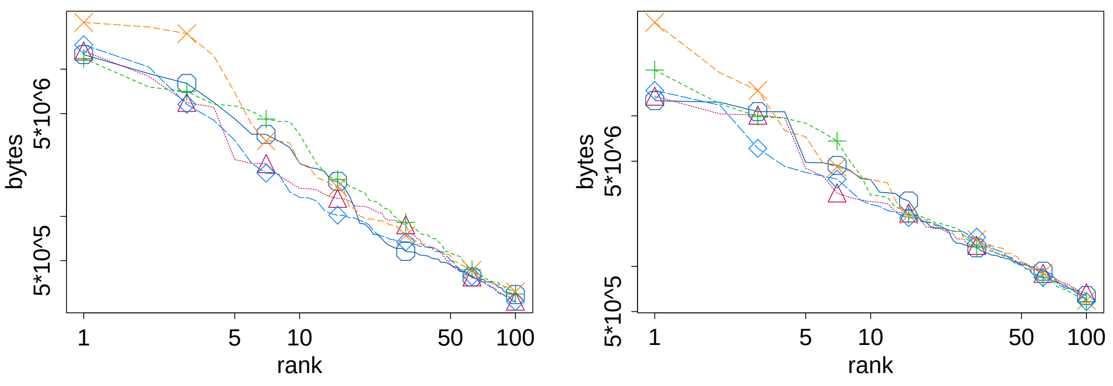


Figure 6.12: Per-bin byte contribution vs. rank for non-aggregated (left) and aggregated fbws (right) for Markov modulated fbw ranks for fi ve consecutive bins. The plots show for both cases a clearly linear relation between logarithms of rank and per-bin byte contribution. This shows that the Markov modulation preserves Zipf's Law for fbw rates even for different aggregation levels and over time.

duces a notable variability in ranks for the largest flows, which in turn requires a high amount of variability in the flow rates.

In order to verify, that the rate variability for top ranks has indeed been increased by the Markov modulation mechanism, we show in Figure 6.11 probability density plots for the relative deviation of destination prefix flows for the static (left) and the dynamic (right) configuration. The plots show, that the relative deviation of aggregated flows with a mean rank in the top 10 shift a significant amount of probability mass to larger values for the dynamic configuration. The spike at zero is more pronounced and there is a higher probability density at +1 to +4, whereas the few very small values in the static configuration between -6 and -2 vanish in the dynamic case. The higher probability density between 1 and 4 in the dynamic case correspond to rates that are 2 to 16 times higher than the average rate of the flows! A similar observation, although less distinct, can be made for flows with an average rank from 11 to 100. We therefore conclude, that the Markov modulation mechanism is able to significantly increase the variability of the rate of large aggregated flows.

After verifying the effectiveness of the Markov modulation mechanism, we have to ensure that the other traffic invariants discovered during our analysis, e.g., Zipf's Law, are still present in the aggregated flows. We start with self-similarity. The only difference between the static and the dynamic configuration in this regard is where the packets passing the monitored backbone link are destined. We still have the same high number of clients representing ON/OFF sources with heavy tailed durations for the ON and OFF phases. Therefore we should still see the same scaling behavior for the dynamic configuration as we did for the static one. This is confirmed by the respective scaling plots that are almost indistinguishable from the plots for the static configuration. In order to verify, that the rate of the destination prefix flows still follow Zipf's Law, we show in Figure 6.12 for five consecutive bins the per-bin byte contribution versus the rank of the top 100 flows on a log-log scale for non-aggregated (left) and destination prefix flows (right). Both plots show a linear relationship between byte contribution and rank over all five bins. Moreover, the plots look very similar to the corresponding plots for the static configuration shown in Figure 6.4. This a is clear indication that the Markov modulation mechanism is able to increase the flow rate variability without destroying self-similarity or conformance to Zipf's Law for flow rates.

After covering the two main traffic features self-similarity and conformance to Zipf's Law, we now take a more detailed look at the traffic properties we already examined for the static configuration and verify that these properties, with the exception of the flow rate variability itself, are unaffected by the Markov modulation mechanism used in the dynamic simulation configuration. We start by ensuring that the fraction of bytes contributed by the top 10 and top 100 ranked flows is unaltered when compared to the static simulation configuration. Figure 6.13 shows the percentage of bytes contributed by the top 10 and top 100 ranked five-tuple (left) and destination prefix flows (right) on a per-bin basis. This plots are almost identical to the corresponding plots for the static configuration shown in Figure 6.5, except for a slightly higher variability in the curves. For the non-aggregated flows for example, there are several bins where the top 10 flows contribute one half of the total bytes although the average contribution is still about 30%. For the static configuration, the top 10 flows contribute only little more than 40% during a small number of bins. The same observation can be made when we aggregate flows by destination prefix. This shows that this traffic property is nearly invariant under Markov modulation of the rates of aggregated flows, except for a slight increase in variability of the contribution of top ranked flows. However, it also shows that the Markov modulation mechanism does not help to overcome the the effects of having too few flows in our simulations which leads to a significant deviation of the contribution of top flows when compared against real traffic as shown in Figure 4.7.

Next, we focus on the entry and exit process for aggregated flows in the dynamic setup. Accordingly, we show in Figure 6.14 histograms of the entry (left) and the exit (right) ranks of destination prefix flows. The histograms show a slightly higher probability for a flow to first appear in the rankings with a rank in the top 40 than for the static configuration. For the

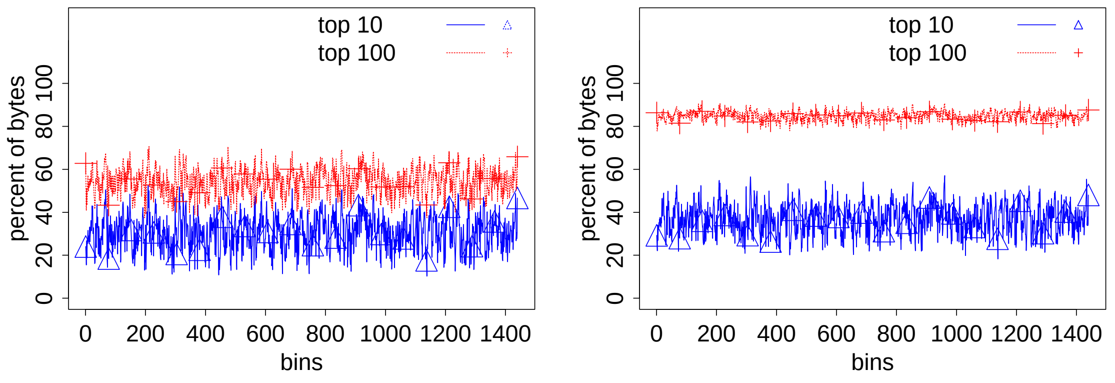


Figure 6.13: Fraction of total traffi c contributed by the top 10 and top 100 fbws, no aggregation (left) and aggregated by destination prefi x (right) for the dynamic simulation confi guration. The curves are almost identical to the corresponding ones for the static confi guration, except for a slightly higher variability. This shows that the Zipf's-like per-bin fbw rate characteristic is largely invariant under Markov modulation, even under different aggregation levels.

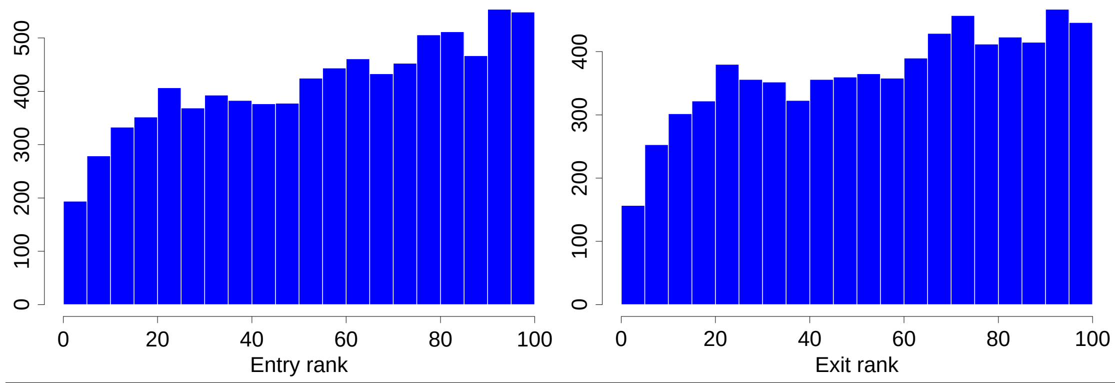


Figure 6.14: Entry (left) and Exit (right) ranks for aggregated fbws for the dynamic simulation setup and 60 second bins. The histograms look almost identical to the ones for the static confi guration with more fbws entering and leaving with a rank in the top 40 caused by the higher fbw rate variability. Still the plots show that the entry end exit processes are only slightly influenced by the Markov modulation mechanism.

remaining ranks, there is almost no discernible difference between dynamic and static simulation setup. A similar observation can be made for the exit ranks of aggregated flows. This is not surprising, as in all cases we have considered so far, the entry and exit processes are almost identical. Although the absolute numbers are again much higher for the simulation as for the trace data, the plots show that the entry and exit processes are only slightly influenced by the higher flow rate variability and the tendency towards higher flow rates for top ranked flows shown in Figure 6.11.

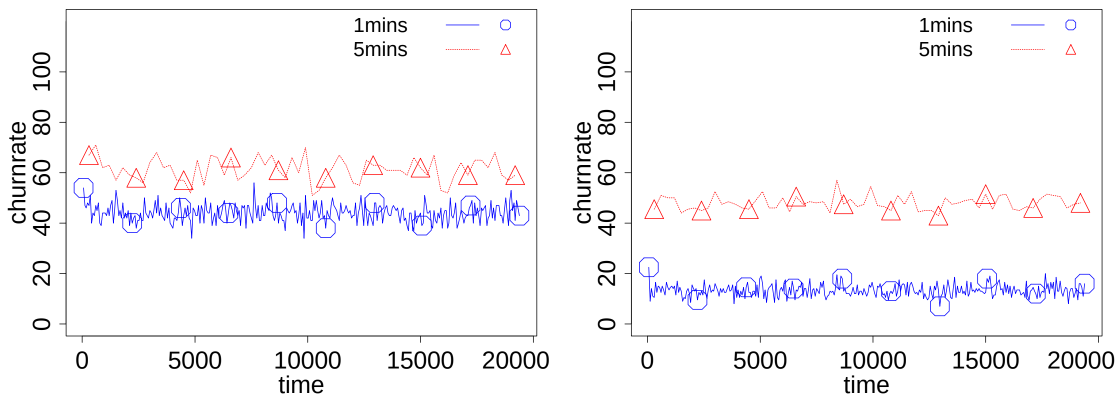


Figure 6.15: Churnrates for the top 100 (left) and the top 200 ranks for the dynamic simulation setup using destination prefi x aggregation. Both plots show that we have much higher churn rate for our simulations than we found in the trace data. Especially when considering the top 200 fbws, the high churn rate suggests that the top ranked aggregated fbws in our simulations are in general much shorter than those found in the trace data.

Another way to characterize the variability of network traffic on the level of flows is the churn rate. The churn rate is a measure for how many flows appear and disappear from the ranking from one bin to the next. A high churn rate means that a high number of flows are either short or have a rate that is so volatile that, even if high ranked in one bin, the rate is not sufficiently high to make it in the top ranks again in the next bin or that there are short bursts that push the flow into the rankings. Figure 6.15 shows the churn rate for the dynamic simulation setup for the top 100 (left) and top 200 (right) destination prefix flows for bin sizes of one and of five minutes. The plots show, that for the top 100 ranks, at least half of the flows in every bin were not among the top 100 flows in the previous bin. This fraction is with about 60% even higher for the longer bin sizes of five minutes. The values are much higher for the simulations than those we found in our trace data in Section 4.2. There we have similar churn rates not for the top 100 but for the top 1000 ranks. When we consider the top 200 ranks (right), the churn rate for one minute bins is significantly smaller (by about 20%) for five minute bins, and the churn rate for one minute bins is, with a factor of about one third, much smaller than for the top 100 flows. The churn rate for top 200 ranks and five minute bins drops only slightly compared to that for the top 100 flows. Note that even though our simulations have only 200 distinct destination prefixes, there can be more than 200 destination prefix flows per bin because packet streams to the same destination prefix with gaps that are longer than the inactivity timeout value generate multiple flows. The churn rate plots in Figure 6.15 show that we have much more flows appearing and disappearing from one bin to another than we can see in our trace data. Especially when considering the top 200 flows, the high churn rate affirms that the top ranked aggregated flows in our simulations are in general much shorter than those found in the trace data.

So far, we have shown, that a large part of the variability of the simulated traffic is caused by flows entering or leaving the rankings from one bin to the next. There are however a larger number of top ranked flows that do not disappear, but stay in the rankings over two or more bins.

In order to assess how much influence the behavior of these flows has on the overall variability of the simulated traffic, we rely on our rank change metric introduced in Section 4.4. Accordingly, we compare in Figure 6.16 the rank change metric over time of the WASHng trace (left column) with the metric of our simulations with Markov modulation (right column) for the rates of non-aggregated and for destination prefix flows. As the rank change metric is sensitive to flows appearing and disappearing, we look at the metric values computed for only the top 10 (top row) and the top 100 flows (bottom row). The maximum values our metric can have depending on the number of top ranks considered (see Section 4.4) when all top flows disappear and are replaced by new ones, are 2.9 for the top 10 and 5.19 for the top 100 flows. If we consider the top 10 flows (upper row), we can see that for the trace data (left), the metric values for non-aggregated flows clearly stay below their upper bound, and are mostly concentrated from 2.0 to 2.8 with some bins where they drop to almost 0.0. In the case of our simulations (right), the upper bound is almost reached during a number of bins, which means that in these cases, almost all top 10 flows are replaced by new ones from one bin to the next. During most of the bins however, the metric values are significantly smaller than those derived from the trace, and without a visible concentration around some value. This metric behavior is caused by the different size of the flow population of the trace data and the simulations. The top *n* flows for our simulations represent a much higher fraction of the overall number of flows than for the trace data. In our simulations we saw at most a few thousand concurrent flows, while this number is larger by several degrees of magnitude for the trace data. According to the results from our flow analysis, the more flows are active, the harder it gets for a flow to reach the top ranks.

When we look at the metric values for destination prefix flows, we can see that for the WASHng trace these values are about half as large on average as for the non-aggregated flows. In the case of our simulations the metric for non-aggregated and for destination prefix flows shows the same or at least very similar values over large time spans. This is another indication that the aggregated flows generated using our simulation environment consist only of a small number of contributing five-tuple flows and therefore both kinds of flows show the same behavior in the metric values.

When we compare the metric values between the top 10 and the top 100 flows, we can see an increase in the metric values for the trace data by factor of about 1.5, while the values for our simulations are almost twice as high as before. There is also a slight increase in the variability of the metric values for both trace data and the simulations and the nature of the variability described for the top 10 flows above is still basically the same. The larger increase in the metric values for our simulations can be explained by the generally shorter durations of the flows when compared to the trace data, which leads to a higher portion of the flows to leave the top ranks. But in contrast to the top 10 flows, we can see that the metric values for non-aggregated flows are generally higher by a factor of about 1.3. This difference in metric values between five-tuple and aggregated flows indicates a higher persistency of aggregated flows than of the five-tuple flows, which is a direct result of the aggregation process.

In summary, the plots show that we are able to capture most of the important variability properties of network flows in our simulation environment. We can reproduce self-similar traffic from which network flows can be derived that have rates consistent with Zipf’s Law on a per-bin basis and over time. We can also reproduce the rate variability of large network flows using the Markov modulation mechanism of our simulation environment. On the other hand, we were

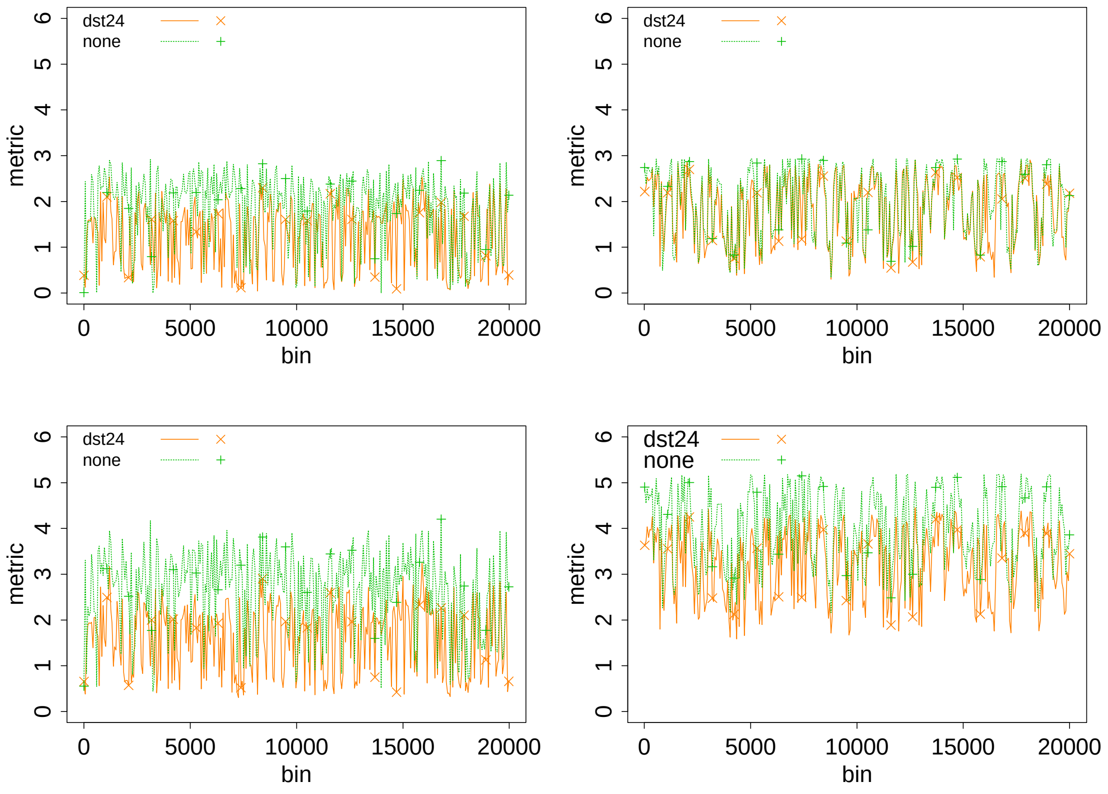


Figure 6.16: Rank change metric for trace WASHng (left column) and for simulation (right column), top 10 flows (top row) and top 100 flows (bottom row). During most of the bins the metric values are significantly smaller than those derived from the trace and without a visible concentration around some value. This is caused by the different size of the flow population of the trace data and the simulations. The difference in metric values between five-tuple and aggregated flows indicates a higher persistency of aggregated flows as a result of the aggregation process.

not able to accurately generate aggregated flows with similar durations as we find in our trace data. The cause of this shortcoming is that, using the *NS-2* network simulator as the core of our simulation environment, we are not able to generate enough five-tuple flows that contribute to the aggregated flows. *NS-2* needs too much memory in order to configure more clients and more destination prefixes. In spite of this limitation, the techniques and mechanisms used by our simulation environment are able to reproduce almost all the aspects of real world traffic needed to generate simulated network traffic that exhibits a high degree of variability.

7 Conclusion

In this chapter we close this thesis by summarizing our findings and contributions and presenting a discussion of possible future work.

7.1 Summary

With the current increase in bandwidth demands, network operators rely more and more on traffic engineering or load adaptive routing algorithms to use existing capacities as efficiently as possible. Both approaches rely on a particular property of network traffic, namely that a very small number of traffic flows are responsible for the vast majority of the overall network traffic, a consequence of flow rates being consistent with Zipf's Law. This approach however is somewhat problematic, as there is a high amount of variability in the rate of Internet flows. This variability is a major problem for the efficiency and stability of traffic management mechanisms

The goal of this work is the development of a workload generator for network simulators that is able to capture the variability aspects of real traffic and that is therefore suitable to evaluate existing and upcoming traffic management algorithms. Such a workload generator has to capture traffic variability properties not only for the entire traffic but also on the level of network flows. This includes the two most important properties, self-similarity nature of packet traffic and the consistency of flow rates with Zipf's Law.

In order to understand the influence of Zipf's Law and the self-similar nature on the variability of network traffic, we proposed a new methodology to analyse the persistency and variability properties of Internet traffic. Because traffic management applications operate of network flows and because it is easier to work with flow traces than with high volume packet level traces, we base our method on flows. For our analysis, we slice time into constant size bins and compute rankings of active flows based on their per-bin byte contribution. The flows with the highest ranks are those that contribute most of the traffic during a bin. Our methodology allows for the analysis of flow properties using different time scales and different flow abstractions. The main difficulty with this approach is the coarse detail level of network flows. The only information a flow provides on its rate is an average value computed over its lifetime. By comparing rankings derived from network flows with those from packet level traces, we found that the assumption of a constant flow rate always leads to errors and mismatches. But these errors can be significantly reduced by flow aggregation or time aggregation and by focusing on the heavy hitter flows.

After showing the feasibility of flow based traffic analysis, we applied our methodology on a set of NetFlow and packet level traces. We found the rates of flows to be consistent with Zipf's Law for all our data sets, even across time and under different time and aggregation granularities. For all our traces, a reasonably small number of top ranked flows covered about half of the total traffic. This number was smaller by an order magnitude for sampled network flows than for unsampled ones, because of sampling leading to a bias towards large flows. We also found, that this set of top ranked flows changed significantly over time. On the other hand, flows that were top ranked at least once during their lifetime had a high probability to stay in the top ranks. Such flows also showed a tendency to have longer durations than lower ranked ones. This behavior again was largely invariant under different bin sizes. For aggregated flows, we were not able to find a distinct pattern in the properties of the contributing five-tuple flows. Aggregated flows can become heavy hitters both because of a high number of small contributors and because of a few very large ones.

Next we explored the causes for the significant degree of variability in the set of top flows and found two major contributing factors. The first factor is the arrival of new and the departure of active flows. New flows entering at high ranks can lead to lower ranks for a significant number of active flows, while their departure leads to higher ranks. We found that there is a relatively high probability for new flows to arrive, and for departing flows to leave at top ranks, leading to a lack of predictability of the behavior of flows that have a high rank at some particular bin. The second factor are flow rate fluctuations of aggregated flows. We used a multiplicative metric to assess the variability of flow rates around their mean values. After removing time of day effects using a wavelet based approach, we found that the rate fluctuations are often high enough to lead to changes in the flow rankings, even though the flow rates were consistent with Zipf's Law. We could also show, that for top ranked flows, the rate deviations were relatively small in most cases. In addition, we found a strong correlation between the relative deviation and rank on a per-bin basis, a clear indication of the strong influence of flow rate fluctuations on the rankings. These observations imply, that the average rank of a flow is better suited for controlling large parts of the traffic than, e.g., the highest rank.

After identifying the causes for the inpersistency of network flows we introduced a metric that is able to capture the the variability in the constitution of network traffic based on flow rankings. This metric describes the difference of the rankings of two bins in terms of inversions where inversions are weighted according to the involved ranks. This way, we take into account, that changes at the top ranks have more impact on the traffic than changes in lower ranks. We could show, that this metric shows more details in the behavior of rankings over time than the churn rate we used during our analysis. The higher precision comes from the fact that the metric does not only accounts for flows entering or leaving the rankings but also considers the rank change dynamics of flows that are active in both compared bins.

We then set out to incorporate our findings of the behavior of Internet flows into a simulation environment, based on the *NS-2* network simulator and the *NSWeb* traffic generator. In order to first assess the accuracy of *NSWeb* we compared simulations with real Web page retrievals. To this end, we generated packet level traces from tightly controlled Web accesses for a set of popular pages. We then automatically extracted parameter values for TCP and HTTP and derived a topology and Web content from the traces to configure our simulations. We found that using *NS-2* and *NSWeb* we are able achieve a high accuracy for our simulations. We were moreover able to assess the degree to what certain parameters are able to influence the accuracy. So has, e.g., an increased TCP initial congestion window a much stronger influence on the behavior of the simulations than an overestimated bandwidth between clients and servers.

For our simulation environment, we concentrated again on destination prefix flows. We were able to generate such flows by clustering client nodes into subnetworks and coordinating client behavior on a per subnet basis. Traffic flowing from the servers to a client subnet represents a destination prefix flow. In order for the rates of the aggregated flows to be consistent with Zipf's Law, we had to control the behavior of the clients, as the actual traffic is generated by the servers upon client requests. We achieved this by limiting the number of concurrent client sessions on a per-subnet basis. The session limits were chosen to also follow Zipf's Law. In addition to controlling the number of concurrent client sessions, we also had to impose bandwidth limits between servers and client subnets by routing the traffic over links with logarithmically scaled capacities. We showed, that our approach to control flow rates did not affect the ability of *NSWeb* to produce self-similar network traffic and that our setup indeed shows Zipf's Law for the rates of aggregated flows. We were moreover able to reproduce the entry- and exit processes of real traffic in our simulations. We had however problems to recreate the duration distribution for the aggregated flows. The cause of that problem are the too small numbers of clients per subnet, that we were not able to increase because of the high memory consumption of *NS-2*.

In a final step, we completed our simulation environment by integrating a mechanism to change the rate of destination prefix flows over time. This mechanism uses an approach based on Markov chains to modulate the rates of the destination prefix flows by adjusting the client session limits. We derived the Markov chains from our trace data. An analysis of the simulation results showed, that using Markov modulation, we were able to increase the variability of the flow rates as intended while preserving the variability aspects from the simulations without modulated flow rates. The modulation mechanisms had also no additional harmful influence on the durations of the aggregated flows.

The simulation environment we developed in this work therefore allows for a more accurate evaluation of traffic engineering and load adaptive routing mechanisms by confronting them with more variable and more realistic workload traffic.

7.2 Outlook

There are number of questions and problems that open up ways for future work.

First and foremost there are the scalability problems of *NS-2* that prevented us from configuring our simulation scenarios to generate enough five-tuple flows to sustain traffic aggregates like, e.g., destination prefix flows long enough to be able to reproduce realistic flow durations. This requires the use of a different network simulator than *NS-2*. There are a number of possible candidates for such an endeavor. Recently, work has begun on the successor of *NS-2*. One of the design goals of this successor is a better scalability to enable larger simulation scenarios. Another possible choice is SSFNet, a Java based network simulator that is able to cope with very large simulation setups.

In this work, we have largely concentrated on destination prefix flows with prefix lengths either chosen as fixed values, e.g., /16 or /24, or derived from routing tables. A promising direction to extend this work is to go beyond destination prefix flows as most commonly used flow aggregation and consider other aggregation schemes like, e.g., origin-destination flows which are used as basis of some traffic management algorithms. This flow type uses network entry and exit points as flow key and is independent of source and destination addresses. We believe that applying our analysis methodology to other aggregation mechanisms will lead to a better understanding of the behavior of traffic aggregates in the Internet and thus to the development of new ways to control the flow of traffic through large networks.

Other interesting questions concern the flow rate modulation mechanism. In this work we used a Markov modulation based approach that requires the modulation to be a memoryless process. But that might not be the case. A more in-depth analysis of the properties of flow rates to explore the behavior of flow rates of aggregated flows over time opens the possibility of a more realistic flow rate modulation.

Independent of how the modulation mechanism is implemented, we believe that it is an interesting question to go beyond trace derived parameterization of the modulation and develop a formal model to be able to configure the modulation in a more generic and representative way. Additional ways to improve the modulation mechanism includes the addition of time-of-day effects.

The goal of this work was to develop a workload generator for simulators that is able to provide traffic with a variability properties that relevant for traffic engineering and load adaptive routing mechanisms. This of course opens up the wide area of developing and evaluating such mechanism with the help of simulations.

As overprovisioning is proving to be more and more infeasible to ensure that there is enough network capacity to cope with current and future traffic demands, the development of novel approaches to traffic management becomes more important. The evaluation of new traffic management systems according to correctness, stability and convergence behavior under realistic workloads and resilience against failure of links or routers is an important prerequisite for the application of these systems in operational networks. Using our workload generator it is now possible to actually perform simulation based evaluations of traffic management systems under highly variable traffic demands and this way improve the quality of such evaluations.

The interaction between overlay and underlay routing in the presence of traffic management mechanisms is another interesting question, independent of what management approach is actually used. Overlay networks that base their routing on measured path quality are able to dynamically circumvent the traffic management decisions of the underlay and are therefore a major challenge for network operators. A simulation based evaluation of traffic management algorithms can be used not only to explore correctness, stability and convergence behavior of such algorithms, but are also suitable to look into the interaction between traffic management on the underlay on the one hand and the overlay routing on the other.

Bibliography

[ABC04] David Applegate, Lee Breslau, and Edith Cohen. Coping with network failures: routing strategies for optimal demand oblivious restoration. In *SIGMETRICS '04/Performance '04: Proceedings of the joint international conference on Measurement and modeling of computer systems*, pages 270–281. ACM Press, New York, NY, USA, 2004. ISBN 1-58113-873-3.

[Abi] Abilene observatory. `http://abilene.internet2.edu/observatory/`.

[ABKM01] D. G. Anderson, H. Balakrishnan, M. F. Kaashoek, and R. Morris. Resilient overlay networks. In *Proceedings of the 18th Anual ACM Symposium on Operating System Principles*, October 2001.

[AC03] David Applegate and Edith Cohen. Making intra-domain routing robust to changing and uncertain traffic demands: understanding fundamental tradeoffs. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 313–324. ACM Press, New York, NY, USA, 2003. ISBN 1-58113-735-4.

[ACE$^+$02] D. Awduche, A. Chiu, A. Elwalid, I. Widjaja, and X. Xiao. Overview and Principles of Internet Traffic Engineering. RFC 3272, 2002.

[All01] Mark Allman. Measuring end-to-end bulk transfer capacity. In *Proceedings of the ACM SIGCOMM Internet Measurement Workshop 2001*, 2001.

[Awd99] D. O. Awduche. Mpls and traffic engineering in ip networks. *IEEE Communications Magazine*, December 1999.

[BA96] Suman Banerjee and Ashok K. Agrawala. Estimating available capacity of a network connection. *Performance Evaluation*, 27, 1996.

[BBCM03] Nikhil Bansal, Avrim Blum, Shuchi Chawla, and Adam Meyerson. Online oblivious routing. In *Proceedings of the Fifteenth ACM Symposium on Parallelism in Algorithms and Architectures SPAA'03*, 2003.

[BC98] Paul Barford and Mark Crovella. Generating Representative Web Workloads for Network and Server Performance Evaluation. In *Joint International Conference on Measurement and Modeling of Computer Systems – Performance Evaluation Review (SIGMETRICS '98/PERFORMANCE '98)*, pages 151–160, 1998.

[BCF$^+$98] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, and Scott Shenker. On the implications of zipf's law for web caching. 3rd International WWW Caching Workshop, 1998.

[BCF$^+$99] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, and Scott Shenker. Web Caching and Zipf-like Distributions: Evidence and Implications. In *Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOMM '99)*, pages 126–134, 1999.

[BG96] Andrew Bruce and Hong-Ye Gao. *Applied Wavelet Analysis with S-PLUS*. Springer, 1996. ISBN 0-387-94714-0.

[BHGc04] Andre Broido, Young Hyun, Ruomei Gao, and kc claffy. Their share: diversity and disparity in ip traffic. In *The 5th anuual Passive & Active Measurement Workshop PAM2004 (LNCS3015)*, 2004.

[BID04] Chadi Barakat, Gianluca Iannaccone, and Christophe Diot. Ranking flows from sampled traffic. Technical Report IRC-TR-04-015, Intel Research Cambridge, 2004.

[BMR99] N. Brownlee, C. Mills, and G. Ruth. Traffic Flow Measurement: Architecture. RFC 2722, 1999.

[BP01] Paul Barford and Dave Plonka. Characteristics of Network Traffic Flow Anomalies. In *Proc. ACM Internet Measurement Workshop*, 2001.

[Bre73] R. Brent. *Algorithms for Minimization without Derivatives*. Prentice-Hall, Englewood Cliffs, NJ, 1973.

[Bro99a] N. Brownlee. SRL: A Language for Describing Traffic Flows and Specifying Actions for Flow Groups. RFC 2723, 1999.

[Bro99b] N. Brownlee. Traffic Flow Measurement: Meter MIB. RFC 2720, 1999.

[BTI$^+$02] Chadi Barakat, Patrick Thiran, Gianluca Iannaccone, Christophe Diot, and Philippe Owezarski. A flow-based model for internet backbone traffic. In *Proc. ACM Internet Measurement Workshop*, 2002.

[CB97] Mark E. Crovella and Azer Bestavros. Self-similarity in World Wide Web traffic: Evidence and possible causes. *IEEE/ACM Trans. Networking*, 5(6):835–846, 1997.

[CB02] Kimberly C. Claffy and Nevil Brownlee. Understanding Internet traffic streams: Dragonflies and Tortoises. *IEEE Communications*, 2002.

[CBP95] Kimberly C. Claffy, Hans-Werner Braun, and George C. Polyzos. A parameterizable methodology for Internet traffic flow profiling. *IEEE Journal on Selected Areas in Communications*, 13(8):1481–1494, 1995.

[CC96] Robert L. Carter and Mark E. Crovella. Measuring bottleneck link speed in packet-switched networks. Technical report, Boston University, 1996.

[Chu92a] Charles K. Chui. *An Introduction to Wavelets*, volume 1 of *Wavelet Analysis and its Applications*. Academic Press, 1992.

[Chu92b] Charles K. Chui. *Wavelets: A Tutorial in Theory and Applications*, volume 2 of *Wavelet Analysis and its Applications*. Academic Press, 1992.

[CISa] Cisco IOS flexible netflow technology white paper. `http://www.cisco.com/en/US/products/ps6601/products_white_paper0900aecd804be1cc.shtml`.

[Cisb] Cisco Netflow. Cisco netflow documentation. `http://www.cisco.com/en/US/products/ps6601/prod_white_papers_list.html`.

[Cis98] Cisco. *Cisco IOS Configuration Fundamentals*. Cisco Press, New Riders Publishing, 1998. Documentation from the Cisco IOS reference Library.

[CPB93] Kimberly C. Claffy, George C. Polyzos, and Hans-Werner Braun. Application of sampling methodologies to network traffic characterization. In *SIGCOMM '93: Conference proceedings on Communications architectures, protocols and applications*, pages 194–203. ACM Press, New York, NY, USA, 1993. ISBN 0-89791-619-0.

[Dav01] Brian Davison. Http simulator validation using real measurements: A case study. In *In Proceedings of the International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'0 1)*, August 2001.

[DG01] N. G. Duffield and M. Grossglauser. Trajectory sampling for direct traffic observation. *IEEE/ACM Transactions on Networking*, 9(3), June 2001.

[DLT01] Nick Duffield, Carsten Lund, and Mikkel Thorup. Charging from sampled network usage. In *IMW '01: Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, pages 245–256. ACM Press, New York, NY, USA, 2001. ISBN 1-58113-435-5.

[DLT02] Nick Duffield, Carsten Lund, and Mikkel Thorup. Properties and prediction of flow statistics from sampled packet streams. In *IMW '02: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pages 159–171. ACM Press, New York, NY, USA, 2002. ISBN 1-58113-603-X.

[DLT03] N.G. Duffield, C. Lund, and M. Thorup. Estimating flow distributions from sampled flow statistics. In *Proc. ACM SIGCOMM*, 2003.

[Dow99] Allen B. Downey. Using pathchar to estimate Internet link characteristics. In *Proc. ACM SIGCOMM*. ACM, Cambridge, Massachusetts USA, September 1999.

[EJLW01] Anwar Elwalid, Cheng Jin, Steven H. Low, and Indra Widjaja. MATE: MPLS adaptive traffic engineering. In *Proceedings of the Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2001)*, pages 1300–1309, 2001.

[End] ENDACE Measurement Systems. `http://www.endace.com`.

[EV02] Cristian Estan and George Varghese. New directions in traffic measurement and accounting. In *Proc. ACM SIGCOMM*, 2002.

[FCDR99] A. Feldmann, R. Caceres, F. Douglis, and M. Rabinovich. Performance of web proxy caching in heterogeneous bandwidth environments. In *Proc. IEEE INFOCOM*, 1999.

[FGL$^+$00] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, and J. Rexford. Netscope: Trafic engineering for ip networks. *IEEE Network Magazine*, 2000.

[FGL$^+$01] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, J. Rexford, and F. True. Deriving traffic demands for operational IP networks: Methodology and experience. *IEEE/ACM Transactions on Networking*, pages 265–279, 2001.

[FGM$^+$99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, 1999.

[FGWK98] A. Feldmann, A. C. Gilbert, W. Willinger, and T. G. Kurtz. The changing nature of network traffic: Scaling phenomena. *ACM Computer Communication Review*, 28(2):5–29, 1998.

[FHW99] Anja Feldmann, Polly Huang, and Walter Willinger. Dynamics of IP Traffic: A Study of the Role of Variablility and the Impact of Control. In *Computer Communication Review, Proceedings of ACM SIGCOMM ’99 conference*, pages 301–313, 1999.

[FKF06] Simon Fischer, Nils Kammenhuber, and Anja Feldmann. REPLEX — Dynamic traffic engineering based on Wardrop routing policies. In *Proceedings of the 2n Conference on Future Networking Technologies (CONEXT 2006)*. Lisboa, Portugal, December 2006.

[FKM$^+$04] Anja Feldmann, Nils Kammenhuber, Olaf Maennel, Bruce Maggs, Roberto De Prisco, and Ravi Sundaram. A methodology for estimating interdomain Web traffic demands. In *Proc. ACM Internet Measurement Conference*, 2004.

[FP99] Wenjia Fang and Larry Peterson. Inter-as traffic patterns and their implications. In *Proceedings of the 4th Global Internet Symposium*, December 1999.

[Fre71] David Freedman. *Markov Chains*. Holden Day, 1971. ISBN 0-8162-3004-8.

[FRT02] Bernhard Fortz, Jennifer Rexford, and Mikkel Thorup. Traffic engineering with traditional ip routing protocols. *IEEE Trans. Communications*, 40(10):118–124, October 2002.

[FT00] Bernard Fortz and Mikkel Thorup. Internet traffic engineering by optimizing OSPF weights. In *INFOCOM (2)*, pages 519–528, 2000.

[FT02] B. Fortz and M. Thorup. Optimizing OSPF/IS-IS weights in a changing world. *IEEE Journal on Selected Areas in Communication*, 20(4):756–767, 2002.

[FT03] B. Fortz and M. Thorup. Robust optimization of OSPF/IS-IS weights. In *Proc. INOC 2003.*, 2003.

[GB96] M. Grossglauser and J.-C. Bolot. On the relevance of long-range dependence in network traffic. In *SIGCOMM '96: Conference proceedings on Applications, technologies, architectures, and protocols for computer communications*, pages 15–24. ACM Press, New York, NY, USA, 1996. ISBN 0-89791-790-1.

[GlFV05] J. Guichard, F. le Faucheur, and J. P. Vasseur. *Definitive MPLS Network Designs*. Cisco Press, 2005.

[Gog00] Helmut Gogl. *Measurement and Characterization of Traffic Streams in High-Speed Wide Area Networks*. PhD thesis, Technische Universität München, 2000.

[hot] Hot100. `http://100hot.com/`.

[HSBR99] S. Handelman, S. Stibler, N. Brownlee, and G. Ruth. RTFM: New Attributes for Traffic Flow Measurement. RFC 2724, 1999.

[ICS96] N. Chiappa I. Castineyra and M. Steenstrup. The Nimrod routing architecture. RFC 1992, 1996.

[IPF] IETF Working Group on IP Flow Information Export (IPFIX). `http://www.ietf.org/html.charters/ipfix-charter.html`.

[JK70] Norman L. Johnson and Samuel Kotz. *Continous Univariate Distributions - 1*. Distributions In Statistics. Houghton Mifflin, Boston, MA., 1970.

[JKR02] Jayeon Jung, Balachander Krishnamurthy, and Michael Rabinovich. Flash Crowds and Denial of Service Attacks: Characterization and Implications for CDNs and Web Sites. In *Proceedings of the World Wide Web Conference*. Honolulu, Hawaii USA, May 2002.

[JLM] Van Jacobson, C. Leres, and S. McCanne. Tcpdump. `ftp://ftp.ee.lbl.gov/tcpdump.tar.Z`.

[JS87] J. Jacod and A. N. Shiryaev. *Limit theorems forstochastic processes*. Springer, 1987.

[KA01] Balachander Krishnamurthy and Martin Arlitt. PRO-COW: Protocol Compliance on the Web—A Longitudinal Study. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems (USITS0́1)*, March 2001.

[KCL06] John Heidemann Kun-Chan Lan. A measurement study of correlations of internet flow characteristics. *Comput. Networks*, 50(1):46–62, 2006. ISSN 1389-1286.

[KKDC05] Srikanth Kandula, Dina Katabi, Bruce Davie, and Anna Charny. Walking the Tightrope: Responsive Yet Stable Traffic Engineering. In *ACM SIGCOMM*. Philadelphia, PA, August 2005.

[KP98] K. Nichols K. Poduri. Simulation studies of increased initial TCP window size. RFC 2415, 1998.

[KR01] Balachander Krishnamurthy and Jennifer Rexford. *Web Protocols and Practice*. Addison-Wesley, May 2001.
`http://cseng.aw.com/book/toc/0,3830,0201710889,00.html`.

[KTB$^+$02] K.Papagiannaki, N. Taft, S. Bhattacharyya, P. Thiran, K. Salamatian, and C. Diot. A pragmatic definition of elephants in internet backbone traffic. In *Proc. ACM Internet Measurement Workshop*, 2002.

[KW99] Balachander Krishnamurthy and Craig E. Willis. Analyzing factors that influence end-to-end web performance. Technical Report WPI-CS-TR-99-35, Worcester Polytechnic Institute, 1999.

[KW00] Balachander Krishnamurthy and Craig E. Wills. Analyzing Factors that influence end-to-end Web performance. In *Proceedings of the 9th International World Wide Web Conference (WWW9)*, pages 17–32, May 2000.

[KXLW03] Abhishek Kumar, Jun (Jim) Xu, Li Li, and Jia Wang. Space-code bloom filter for efficient traffic flow measurement. In *IMC '03: Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, pages 167–172. ACM Press, New York, NY, USA, 2003. ISBN 1-58113-773-7.

[LB01] Kevin Lai and Mary Baker. Nettimer: A tool for measuring bottleneck link bandwidth. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, 2001.

[LCD04a] Anukool Lakhina, Mark Crovella, and Christiphe Diot. Characterization of network-wide anomalies in traffic flows. In *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 201–206. ACM Press, New York, NY, USA, 2004. ISBN 1-58113-821-0.

[LCD04b] Anukool Lakhina, Mark Crovella, and Christophe Diot. Diagnosing network-wide traffic anomalies. *ACM Computer Communication Review*, 34(4):219–230, 2004. ISSN 0146-4833.

[Li03] Wentian Li. References on zipf's law. `http://linkage.rockefeller.edu/wli/zipf/`, 2003.

[LM97] Steven Lin and Nick McKeown. A simulation study of IP switching. In *Proc. ACM SIGCOMM*, pages 15–24, September 1997.

[LPC$^+$04] A. Lakhina, K. Papagiannaki, M. Crovella, C. Diot, E. Kolaczyk, and N. Taft. Structural analysis of network traffic flows. In *Proc. ACM SIGMETRICS*, 2004.

[LRV05] Karthik Lakshminarayanan, Anand Rangarajan, and Srinivasan Venkatachary. Algorithms for advanced packet classification with ternary CAMs. In *Proc. ACM SIGCOMM*, 2005.

[LTWW94] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson. On the self-similar nature of ethernet traffic (extended version). *IEEE/ACM Trans. Networking*, 2: 1–15, 1994.

[MA02] C. Partridge M. Allman, S. Floyd. Increasing TCP's Initial Window. RFC 3390, 2002.

[Mah97] Bruce Mah. An Empirical Model of HTTP Network Traffic. In *Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '97)*, pages 592–600, 1997.

[Mah01] Bruce A. Mah. A tool for measuring Internet path characteristics, 2001. `http://www.employees.org/~bmah/Software/pchar/`.

[MDFK97] Jeffrey C. Mogul, Fred Douglis, Anja Feldmann, and Balachander Krishnamurthy. Potential benefits of delta encoding and data compression for HTTP. In *Proc. ACM SIGCOMM*, pages 181–194, August 1997.

[MF93] Anthony J. McAuley and Paul Francis. Fast routing table lookup using CAMs. In *INFOCOM (3)*, pages 1382–1391, 1993.

[MHCS] J. S. Marron, Felix Hernandez-Campos, and F. D. Smith. Mice and elephants visualization of Internet traffic.

[Mit03] M. Mitzenmacher. A brief history of generative models for power law and lognormal distributions. *Internet Mathematics*, 2003.

[MJ] David Mosberger and Tai Jin. httperf — a tool for measuring web server performance. `http://www.hpl.hp.com/personal/David\_Mosberger/httperf.html`.

[MMFR96] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP selective acknowledgment options. RFC 2018, 1996.

[MMK$^+$04] T. Mori, Uchida M., R. Kawahara, J. Pan, and S. Goto. Identifying elephant flows through periodically sampled packets. In *Proc. ACM Internet Measurement Conference*, 2004.

[Mog95] Jeffrey C. Mogul. The Case for Persistent-Connection HTTP. In *Proceedings of the ACM SIGCOMM '95 Conference on Applications, Technologies, Architectures and Protocols for Computer Communication*, pages 299–313, 1995.

[MTS$^+$02] A. Medina, N. Taft, K. Salamatian, S. Bhattacharyya, and C. Diot. Traffic matrix estimation: existing techniques and new directions. In *SIGCOMM '02: Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–174. ACM Press, New York, NY, USA, 2002. ISBN 1-58113-570-X.

[MWNa] Das Münchner Wissenschaftsnetz (MWN). `http://www.lrz-muenchen.de/services/netz/mwn-netzkonzept/mwn-netzkonzept.pdf`.

[MWNb] Überlick über das Münchner Wissenschaftsnetz. `http://www.lrz-muenchen.de/services/netz/mhn-ueberblick`.

[NB02] Nan Ni and Laxmi N. Bhuyan. Fair scheduling and buffer management in Internet routers. In *INFOCOM*, 2002.

[NGBS$^+$97] Henrik Frystyk Nielsen, James Gettys, Anselm Baird-Smith, Eric Prud'hommeaux, Hakon Wium Lie, and Chris Lilley. Network performance effects of HTTP/1.1, CCS1 and PNG. In *Proc. ACM SIGCOMM*, August 1997.

[Ost] Shawn Ostermann. tcptrace – TCP dump file analysis tool. `http://jarok.cs.ohiou.edu/software/tcptrace/tcptrace.html`.

[Pax] Vern Paxson. tcpreduce. `http://ita.ee.lbl.gov/html/contrib/tcp-reduce.html`.

[Pax97] Vern Paxson. *Measurements an Analysis of End-to-End Internet Dynamics*. PhD thesis, University of California at Berkeley, April 1997.

[PF95] V. Paxson and S. Floyd. Wide area traffic: The failure of Poisson modeling. *IEEE/ACM Trans. Networking*, 3:226–244, 1995.

[PFTK98] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: a simple model and its empirical validation. In *Proc. ACM SIGCOMM*, 1998.

[PPM01] P. Phaal, S. Panchen, and N. McKee. Inmon corporation's sflow: A method for monitoring traffic in switched and routed networks. RFC 3176, 2001.

[Pro] The VINT Project. The ns-2 network simulator. `http://www.isi.edu/nsnam/ns/`.

[PSA] IETF Working Group on Packet Sampling (PSAMP). `http://www.ietf.org/html.charters/psamp-charter.html`.

[PTB+01] Konstantina Papagiannaki, Nina Taft, Supratik Bhattacharrya, Patrick Thiran, Kave Salamatian, and Christophe Diot. On the feasibility of identifying elephants in internet backbone traffic. Sprint ATL Research Report RR01-ATL-110918, Sprint ATL, November 2001.

[PTD04] K. Papagiannaki, N. Taft, and C. Diot. Impact of flow dynamics on traffic engineering design principles. In *Proc. IEEE INFOCOM*, 2004.

[QYZS03] L. Qiu, Y.R. Yang, Y. Zhang, and S. Shenker. On selfish routing in internet-like environments. In *Proc. of ACM SIGCOMM '03*, 2003.

[RFC80] User Datagram Protocol. RFC768, 1980. John Postel.

[RFC81a] Internet Protocol. RFC791, 1981. John Postel.

[RFC81b] Transmission Control Protocol. RFC793, 1981. John Postel.

[RGK+03] M. Roughan, A. Greenberg, C. Kalmanek, M. Rumsewicz, J. Yates, and Y. Zhang. Experience in measuring internet backbone traffic variability: Models, metrics, measurements and meaning. In *Proceedings of the International Teletraffic Congress (ITC-18)*, pages 379–388, 2003.

[RTZ03] Matthew Roughan, Mikkel Thorup, and Yin Zhang. Traffic engineering with estimated traffic matrices. In *IMC '03: Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, pages 248–258. ACM Press, New York, NY, USA, 2003. ISBN 1-58113-773-7.

[SAA+99] S. Savage, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collins, E. Hoffman, J. Snell, A. Vahdat, G. Voelker, and J. Zahorjan. Detour: a case of informed Internet routing and transport. *IEEE Micro*, 19, January 1999.

[SF02] R. Sommer and A. Feldmann. NetFlow: Information loss or win? In *Proc. ACM Internet Measurement Workshop*, 2002.

[SKW00] D. N. Serpanos, G. Karakostas, and W. H. Wolf. Effective caching of web objects using zipf's law. In *ICME*, 2000.

[SRB01] Shriram Sarvotham, Rudolf Riedi, and Richard Baraniuk. Connection-level analysis and modeling of network traffic. In *Proc. ACM Internet Measurement Workshop*, 2001.

[SST+04] A. Soule, K. Salamatian, N. Taft, R. Emilion, and K. Papagiannaki. Flow classification by histograms or how to go on safari in the internet. In *Proc. ACM SIGMETRICS*, 2004.

[Sta99] William Stallings. *SNMP, SNMPv2, SNMPv3 and RMON 1 and 2*. Addison-Wesley, 1999.

[Ste99] John W. Stewart. *BGP4: Inter-Domain Routing in the Internet*. Addison-Wesley, 1999.

[SXM+00] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson. Stream control transmission protocol. RFC 2960, 2000.

[TDRR05] R. Teixeira, N. Duffield, J. Rexford, and M. Roughan. Traffic matrix reloaded: impact of routing changes. In *Proc. of Passive and Active Measurement Workshop (PAM'05)*, 2005.

[Tei05] R. Teixeira. *Network Sensitivity to Intradomain Routing Changes*. PhD thesis, University of California San Diego, August 2005.

[TWS97] Murad S. Taqqu, Walter Willinger, and Robert Sherman. Proof of a fundamental result in self-similar traffic modeling. *ACMCCR: Computer Communication Review*, 27, 1997.

[UB01] Steve Uhlig and Olivier Bonaventure. Implications of interdomain traffic characteristics on traffic engineering. Technical Report Infonet-TR-2001-08, University of Namur, Belgium, 2001.

[Wal01] Jörg Wallerich. Design and Implementation of a WWW Workload Generator for the NS-2 Network Simulator. Master's thesis, Universität des Saarlandes, Germany, August 2001. `http://www.net.in.tum.de/~jw/nsweb/`.

[WF06] Jörg Wallerich and Anja Feldmann. Capturing the variability of internet flows across time. In *Proc. 9th IEEE Global Internet Symposium*, 2006.

[WXQ+06] Hao Wang, Haiyong Xie, Lili Qiu, Yang Richard Yang, Yin Zhang, and Albert Greenberg. Cope: traffic engineering in dynamic networks. In *SIGCOMM '06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 99–110. ACM Press, New York, NY, USA, 2006. ISBN 1-59593-308-5.

[ZBPS02] Yin Zhang, Lee Breslau, Vern Paxson, and Scott Shenker. On the characteristics and origins of Internet flow rates. In *Proc. ACM SIGCOMM*, 2002.

[ZG05] Y. Zhang and Z. Ge. Finding critical traffic matrices. In *Proceedings of DSN'05*, June 2005.

[Zip49] G.K. Zipf. *Human Behavior and the Principle of Least Effort*. Addison-Wesley, 1949.

[ZRDG03] Yin Zhang, Matthew Roughan, Nick Duffield, and Ablert Greenberg. Fast accurate computation of large-scale ip traffic matrices from link loads. In *Proc. ACM SIGMETRICS*, 2003.

[ZRLD03] Yin Zhang, Matthew Roughan, Carsten Lund, and David Donoho. An information-theoretic approach to traffic matrix estimation. In *Proc. ACM SIGCOMM*, 2003.

[ZSC91] L. Zhang, S. Shenker, and D. Clark. Observations on the dynamics of a congestion control algorithm: The effects of two-way traffic. In *Proc. ACM SIGCOMM*, pages 133–147, September 1991.