NetFlow: Information loss or win?

Robin Sommer and Anja Feldmann
Saarland University, Saarbrücken, Germany
{rsommer,anja}@cs.uni-sb.de

Abstract— **Operating a network without accurate traffic statistics is not desirable. Commonly used data sources are SNMP [1], flow-level data, e.g., CISCO's NETFLOW, or packet level data. The first data source provides low volume, coarse-grained, non-application specific data. The latter one provides high volume, fine-grain data and application specific information. NetFlow lies somewhere in between in terms of both: volume and level of detail. In this paper we ask the question how and how accurately can one infer information from NetFlow. More specifically we are interested in TCP connection summaries and accurately aggregated packet and byte counts. The same techniques apply to application specific (per port) summaries.**

I. INTRODUCTION

Even the engineering of small IP networks involves solving a number of difficult challenges. Shifts in user behavior, introduction of new hardware and technology, new application and just plain time of day effects cause significant fluctuations in the volume and the kind of traffic in the network. A network operator needs an understanding of the traffic on the network in order to tune his/her network. Common methods at the disposal of the network operator are either active measurements, using tools such as ping and traceroute, or passive measurements. Passive measurements may take advantage of the routers capability, e.g., SNMP data [1], NetFlow [2], or IP accounting [3] or may require an additional dedicated monitoring devices for sniffing, e.g., [4] or use general purpose hardware together with tools such as `tcpdump` [5].

While some of the tools provide very fine-grained data most operators are more interested in summaries. On the other hand the easily available coarse-grained data does not provide enough information for some tasks that they face, e.g., to determine the traffic mix on their network. One dataset that lies in between is CISCO's NetFlow data[1]. It produces more fine grained data than SNMP but not as detailed and high volume as packet sniffers. But how accurate is the information provided by NetFlow and how can we fit it into the traditional framework of TCP connections and/or aggregated packet and byte count summaries. Therefore in this paper we ask the question how much information do we loose with NetFlow compared to packet level traces and how much information do we win with respect to SNMP like data.

To this end we collect NetFlow and packet level traces for two 24h time periods in March/April 2002 from the external Internet connection of the Saarland University. We extract TCP connection information from both the NetFlow data and the packet level trace and empirically evaluate the strength and weaknesses of using NetFlow data. For large parts of the traffic we show very good agreements, in terms of byte counts, durations, TCP states, and originator classification. Some applications such as the Web perform even better while others such as file sharer, can be a bit troublesome due to their specific use of TCP connections.

Often NetFlow is used to extract SNMP like but application specific packet and byte counts without considering the specific constraints of Cisco's definition of flows and the NetFlow generation process. To understand their impact, especially the burstiness of the NetFlow export process in combination with the varying number of bytes per flow, we evaluate strategies for aggregating NetFlow data to packet/byte counts and compare them to those computed from packet data. While it is possible to derive good counts the simple and most common approach performs rather poorly for small aggregation periods, e.g., < 8 minutes.

The remainder of this paper is organized as follows: In Section II we present a short review of the flow concept and Cisco's variant NetFlow. Section III summarizes our trace environment and data sets. Our methodology for extracting TCP connection summaries from NetFlow data and its evaluation is discussed in Section IV. Our evaluation of different methods for aggregating NetFlow data is presented in Section V. Finally we give a short summary in Section VI.

II. FLOW CONCEPTS

The flow model of Claffy et al. [6] states that "a flow is active as long as observed packets that are meeting the flow specification are observed separated in time by less than a specified timeout value". The model has proven useful in quite a few other studies. For example it is used by Thomp-

[1] We use the term NetFlow for both the concept as well as individual records.

son et al. [7] for traffic measurement and characterization. Lin et al. [8] evaluate the effect of different flow classifiers on switching performance, while Feldmann et al. [9] examine the impact of application-layer aspects on the flow characteristics. Newman et al. [10] propose IP switching based on flows.

Flows have proven to be a very useful tool for measurements and traffic characterization, see for example [11], [12]. This is reflected in the efforts to standardize flow data measurement and collection architectures [13], [14], [15].

CISCO's flow-level aggregation technique NetFlow [2] almost fits into the model by Claffy et al.. According to CISCO's documentation [2] the NetFlow implementation identifies a flow by the tuple $(src_{addr}, src_{port}, src_{if}, dest_{addr}, dest_{port}, ip_{prot}, tos)$. We observed that undefined fields are set to zero. A NetFlow enabled router regularly exports aggregated flows to some predefined collector host using UDP. Using CISCO's terminology we call these *NetFlows*. Table I summarizes some of the relevant fields of NetFlow records. NetFlow is a well-used tool for visualization [16], accounting [17], and traffic analysis [11], [12]. Two common packages for processing NetFlow data are Fullmer's `flow-tools` [18] and CAIDA's `cflowd` [19].

TABLE I
SUBSET OF NETFLOW FIELDS

Name	Description
`srcaddr`	Source address
`dstaddr`	Destination address
`input`	Input interface
`output`	Output interface
`dPkts`	Number of packets
`dOctets`	Number of octets
`First`	Start of NetFlow
`Last`	End of NetFlow
`srcport`	Source port
`dstport`	Destination port
`tcp_flags`	TCP flags
`tos`	IP type-of-service

Since the router stores NetFlows in a rather limited amount of cache memory, it has to use an expiration policy to remove old NetFlows. A NetFlow is expired [2] if

- it has been idle for a specified timeout value called `inactive_timeout`. This is the timeout which is defined by Claffy et al. [6].
- it exceeds a maximum duration[2]: `active_timeout`.
- it contains a `FIN` or a `RST` packet.
- the cache is exhausted; then "a number of heuristics are applied to aggressively age groups of flows simultaneously" [2].

Observe that CISCO's definition of a flow imposes additional constraints on NetFlows. In particular, it is possible that several NetFlow records are exported for one flow as defined by Claffy et al..

III. DATA SETS

Throughout this paper we use two kinds of data sets: NetFlows and packet traces gathered from the connection of the Saarland University, Saarbrücken, Germany, to the external Internet. SNMP like data is derived from the packet trace. Figure 1 sketches an overview of the network infrastructure. The NetFlow data is collected via the router, a Cisco 7200, which is configured to export version 5 NetFlows to a collector host inside the university. The `inactive_timeout` is 15 seconds, the `active_timeout` 30 minutes. The flow switching cache is 6 MBytes. We have collected NetFlows on two different days – on March 21th, 2002, and on April 25th, 2002 – using the `flow-tools` [18] package with some minor custom modifications. We included all NetFlows which start after midnight and end before midnight of the next day. Using the sequence numbers contained in NetFlow export packets we confirm that none of the UDP packets were dropped. The NetFlow traces consist of $17470K/21697K$ individual NetFlows or $1.07/1.29$ GBytes in `flow-tools` format.

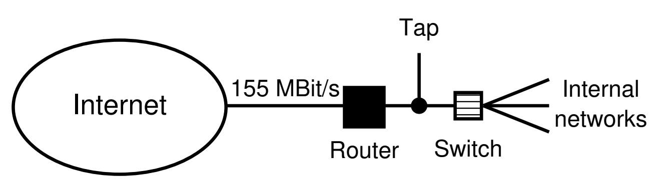


Fig. 1. Network infrastructure

The network tap is a PC running `FreeBSD` which is connected via a GBit network card to the monitoring port of the switch. It captures all traffic passing the link between the router and the switch. Using `tcpdump` [5] we have captured the first 60 bytes of each packet within the same two 24 hour time periods as the NetFlow trace. We have included all packets from midnight to midnight. `tcpdump` has reported negligible ($<$ $0.01\%/0.04\%$) packet losses. The packet traces consist of $2.48 * 10^{11}/4.38e * 10^{11}$ packets or $37.2/56.9$ GBytes if we consider all packets and $2.64 * 10^7/2.82 * 10^7$ packets or $1.87/1.99$ GBytes if we only consider `SYN`/`FIN`/`RST` packets. This implies the average data volume during the day is $3.53/5.40$ Mbit/s. The two clocks involved in the data collection – that of the router and that of the PC - are not tightly synchronized. We observe clock skews [20].

[2]Though in reality, we see NetFlows which last a few seconds longer than configured

For the purpose of correlating the two datasets we apply a clock correction transformation (a simple shift by 19/59 seconds).

Although the network tap and the router basically monitor the same link there are several differences in the traffic they report:

- Some of the internal subnets are routed via the router. Therefore some traffic passes the tap point twice: to the router and from the router. We excluded such university internal traffic from all data sets.
- The router implements an access list based firewall which blocks a subset of the incoming traffic at the router. In addition some traffic is explicitly directed to `/dev/null`. Therefore this traffic cannot be observed at the tap point and we exclude all corresponding NetFlows.
- In addition we excluded all traffic involving one of the router's IP addresses.

Applying these filters yields two kinds of data sets, packet traces and NetFlows, that should both more or less summarize the same traffic (see Section V). Application specific traffic is identified based on the source or destination port number. In addition to `ftp` (ports 20/21), we consider all applications in the per-application analysis which contribute at least 1% of the TCP connections.

One significant difference in the network configuration is that on March 21th an intercepting proxy was imposed on a large part of the internal network which was disabled on April 25th. If the proxy is enabled the router intercepts all relevant Web connections and tunnels them to the proxy, a CISCO CACHEENGINE which may or may not issue a corresponding request. This implies that NetFlows will record the original and the proxy request but not the response from the proxy to the original client nor the tunneled request. The latter two are considered to be internal traffic and therefore eliminated from both traces.

If not stated otherwise we will concentrate on the March data set. The results for the April data set confirm those of the March data set.

IV. ANALYSIS OF TCP CONNECTIONS

In this section we examine how accurately one can infer TCP connection summary information from NetFlow. We differentiate three kinds of connections:

1. A *TCP connection* is the transport layer session between two endpoints.
2. A *packet connection* is the reconstruction of a TCP connection based on packets captured at some tap point.
3. A *flow connection* is the reconstruction of a TCP connection based on NetFlows.

Our goal is to generate and compare TCP connection summaries based on both packet and flow connections. Figure 2 shows example TCP connection summaries. The first two fields contain start time and duration. The next two stem from the port numbers. Next are transfered TCP byte counts for both directions and the IP addresses of originator and responder. The last field summarizes the state of the TCP connection (see Table III).

```
5354.386 0.110 6346 1283 ? ? A.B.C.D E.F.G.H REJ
5354.957 0.116 6346 1283 ? ? A.B.C.D E.F.G.H REJ
5355.558 0.346 6346 1283 152 222 A.B.C.D E.F.G.H SF
```

Fig. 2. Example: TCP connection summaries

Therefore we first discuss how to construct packet connection summaries and compare their characteristics to NetFlows. Motivated by the unsatisfactory outcome we discuss our post-processing methodology which combines NetFlows into flow connections. We end this section with an evaluation of our methodology.

A. Generating packet connections from traces

TCP connection summaries are typically computed from packet traces using tools such as TCP-REDUCE [21] or BRO [22]. In an ideal world it should be possible to extract all information contained in the TCP connection summaries by using the packet trace to simulate the TCP state machine at the two endpoints. For this it suffices to consider TCP control packets (`SYN`/`FIN`/`RST`). In reality there are two main reasons why tools are sometimes fooled into producing erroneous TCP summaries. First the packet traces themself may miss packets due to the high network traffic volume. Second interpreting the packet stream to infer the behavior of the end-systems is a difficult problem in itself due to ambiguities and "crud" in network traffic [20], [23]. Nevertheless, packet connections yield the best known approximations for TCP connections.

Due to memory problems we found the commonly used simple tool TCP-REDUCE insufficient for our purposes. TCP-REDUCE accumulates the necessary connection state information in main memory while processing the trace. This implies that TCP-REDUCE cannot be used for real time generation of connection summaries and on large traces from high-volume network links as is necessary in our case. Therefore, we use an alternative tool for generating TCP-REDUCE-compatible connection summaries: BRO [22]. BRO is an intrusion detection system, build on top of a robust implementation of the TCP/IP stack, designed to operate in real time relying on timeouts rather than memory use. Since we only need a small part of BRO we use default values and a stripped down configuration script. We substitute some of BRO's standard timeouts with a bro_inact_tout to purge connections that are idle for more than 5 minutes. This implies that BRO needs to consider all packets and not just control packets. We modified BRO to defer emitting summaries until the connection state is removed from BRO's memory[3]. We furthermore modified BRO's connection summaries slightly to include all fields summarized in Table II.

TABLE II
BRO'S (MODIFIED) CONNECTIONS SUMMARIES

Field	Description
0	Timestamp of first packet
1	Duration
2	Port of originator (client)
3	Port of responder (server)
4	Bytes transfered by originator
5	Bytes transfered by responder
6	IP address of originator
7	IP address of responder
8	IP protocol
9	Connection's final state (TCP-REDUCE-compatible, see Table III)

TABLE III
SUBSET OF OF BRO'S STATE SUMMARIES

Name	Meaning
`S0`	Connection attempt, no reply seen
`S1`	Connection established, not terminated
`SF`	Normal establishment and termination
`REJ`	Connection attempt rejected
`RSTO`	Connection established, originator aborted
`RSTR`	Connection established, responder aborted

B. Raw NetFlows vs. packet connections

While one needs a dedicated box for capturing network packets, NetFlows are generated by the router itself and offer several advantages. In particular, we do not need to address the administrative, security and privacy issues of installing a packet sniffer. Therefore collecting NetFlows is much simpler and needs little additional setup. Also the data sizes are comparably small. While still sizable our NetFlow traces are about a factor of 1.75 smaller than packet traces of `SYN`/`FIN`/`RST` packets and a factor of 33.3 smaller than the full packet traces. In addition NetFlows include additional data not available in packet traces, e.g., interfaces, and autonomous systems.

Yet, NetFlows in themself have three severe limitations. First they are uni-directional whereas packet connections are bi-directional and distinguish originator and responder. Second their definition is tricky and intricate. The exact output depends on the specific configuration of each NetFlow router. Indeed we will show that the characteristics of NetFlows differ significantly from those of packet connections. Third NetFlows do not contain TCP state information (see Table III). They only contain the cumulative `OR` of all TCP flags observed in any of the packets associated with a NetFlow.

C. Generating flow connections from NetFlows

We now present a post-processor, FLOW-REDUCE, for NetFlows that allows us to overcome these three limitations. The main idea is to combine all NetFlows, which contribute to a TCP connection, to a *flow connection*. The result is a well defined bi-directional flow connection whose summaries are remarkably close to those generated by BRO from packet connections.

We first introduce some terminology. A TCP connection is uniquely identified by the 3-tuple $(conn_{start}, conn_{end}, conn_{id})$ where $conn_{id}$ is a 4-tuple $(orig_{ip}, orig_{port}, resp_{ip}, resp_{port})$. We say that a NetFlow *matches a* $conn_{id}$ if its IP protocol is TCP and if it contains the tuple $(orig_{ip}, orig_{port})$ as source and $(resp_{ip}, resp_{port})$ as destination, or vice versa. We say a NetFlow *matches a TCP connection* if it matches the connection's $conn_{id}$ and if the interval defined by its start time and its end time is a subset of $[conn_{start}, conn_{end}]$. Finally, we say that two NetFlows *match* if they match the same TCP connection. FLOW-REDUCE's task is to identify matching NetFlows and combine them into flow connections.

Identifying matching NetFlow: Since the same $conn_{id}$ may be used more than once during a trace the main difficulty is that it is impossible to decide when two NetFlows match. Therefore we match using a heuristic based on the lifetime of TCP connections. Our heuristic relies on two assumptions:

1. TCP endpoints will usually not reuse a socket within a short time period. TCP's `TIME_WAIT` state even enforces this after a regular connection tear-down. Typically TCP uses a different originator port for the next connection.
2. An active TCP connection will typically generate NetFlows on a regular basis. This assumption is based on the observation that long-living but inactive TCP connections are usually associated with human activity, e.g., telnet or ssh connections rather than bulk data transfers, and do not contribute a large fraction of the TCP connections anymore.

The idea of our heuristic is simple but effective (see Figure 3). For each $conn_{id}$ we identify all NetFlows that match this $conn_{id}$ and then combine them to a flow connection. To this end we sort all NetFlows matching a $conn_{id}$ according to their start time. The first connection

[3] BRO uses TCP sequence number to calculates byte counts of connections; this can in rare cases, mainly due to RST packet handling, result in unrealistically high transfer volumes (even the unmodified BRO reports some of them). We removed $1711/1036$ packet connections whose throughput exceeds the available bandwidth.

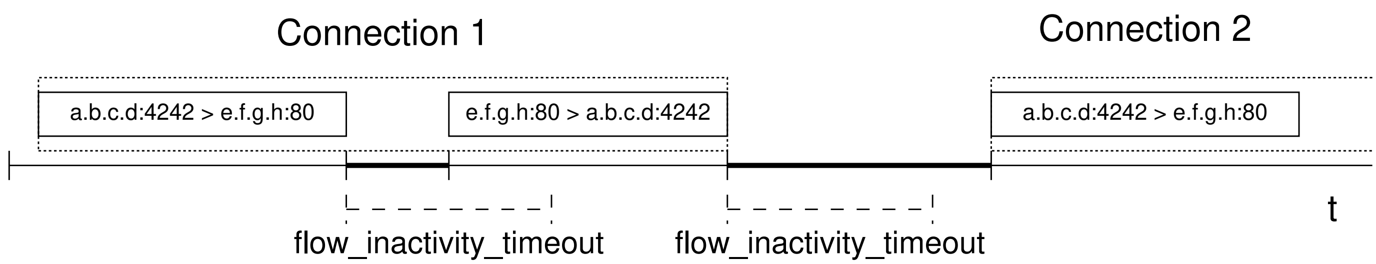


Fig. 3. Example for combining NetFlows to flow connection.

starts with the first NetFlow. The next NetFlow is associate with this connection if the gap between the end of the previous NetFlow and the start of this NetFlow is less than `flow_inact_tout`. Otherwise this NetFlow starts a new connection. The first assumption should ensure that there is a reasonable value for `flow_inact_tout` while the second assumption ensures that all flows from a TCP connection are associate with the same flow connection. Choosing `flow_inact_tout` is not straightforward since the first assumption recommends using a small value while the second assumption suggests a larger value. We solve this by modifying the heuristic to take advantage of TCP FIN flags. After observing connection close packets from both endpoints we can decrease `flow_inact_tout` to `flow_inact_tout_f`, approximately the duration of the `TIME_WAIT` state.

After combining NetFlows to flow connections we can generate most fields of the flow connection summaries, see Table II, quite easily: the start time is the start of the earliest NetFlow; the duration is the difference between end of the last NetFlow and the start time; the transmitted bytes are the sum of the NetFlows' bytes. Two fields however, the originator and the state of the connection, are more complicated.

Flow connection originator: Often the originator of a connection is the one that starts the communication. Therefore the originator of the flow connection is the source of the first NetFlow associate with the connection. Unfortunately the timestamps of NetFlow's have a limited resolution (milliseconds) which often leaves us with two possible first NetFlows. We determine the originator based on the heuristics outlined in Figure 4. The first two checks help with misclassified NetFlows. The next two rely on an augmented list of well-known ports to identify the servers. The next check uses our original idea and the last one is the fallback.

Flow connection state: Due to the aggregated nature of NetFlows we cannot follow the endpoints' TCP state-machine transitions to infer the state of the connection. Instead we derive the state from the roles of the end-points (originator/responder) and the total TCP flag information

```
If SYNs from both hosts and
   SYNs in earliest NFs of both hosts and
   start of hosts' earliest packets differ:
      Host with earliest NF is originator
If SYN only from one host and
   SYN is in connection's earliest NF:
      Host is originator
If one host uses port 20 (ftp-data):
      Host is originator
If only one host uses a well-known port:
      Host is responder
If start of hosts' first packets differ:
      Host with earliest NF is originator
Arbitrarily choose originator
```

Fig. 4. Pseudo-code implementation of originator identification

of each direction. The total flag information is accumulated from the individual NetFlows. Based on this information we determine the state via the decision procedure outlined in Figure IV. A `0` corresponds to an absent TCP flag, a `1` to a present one, and a `*` to a wildcard. The list is traversed from top to bottom and we choose the state of the first match.

TABLE IV

DECISION PROCEDURE FOR DETERMINING STATE (TOP TO BOTTOM; FIRST MATCH IS TAKEN)

Originator			Responder			State
SYN	FIN	RST	SYN	FIN	RST	
1	⋆	⋆	0	⋆	1	REJ
0	⋆	⋆	⋆	⋆	1	RSTRH
⋆	⋆	⋆	⋆	⋆	1	RSTR
1	⋆	1	0	⋆	⋆	RSTOS0
⋆	⋆	1	⋆	⋆	⋆	RSTO
1	1	0	1	1	0	SF
1	1	0	1	0	0	S2
1	1	0	0	⋆	0	SH
1	0	0	1	1	0	S3
0	⋆	0	1	1	0	SHR
1	⋆	0	0	⋆	0	S0
1	0	0	1	0	0	S1
⋆	⋆	⋆	⋆	⋆	⋆	OTH

Implementation: Our prototype implementation of FLOW-REDUCE, a Python [24] script, operates on raw NetFlows as provided by the `flow-tools` package [16]. FLOW-REDUCE expects them to be sorted by *export* time. This at least in principle allows us to generate flow connection summaries in real time. The drawback is that it requires us to buffer flow connections for a certain amount of time (the router's `active_timeout`).

One problem we encountered is that NetFlows, as actually exported by the router, differ from Cisco's documentation. It appears that the output interface or next hop fields are included in the definition of a NetFlow. As a consequence two NetFlows from the same TCP connection may span overlapping time periods. We have not yet extended FLOW-REDUCE to handle this aspect. The TOS field poses another problem. It is not symmetric. Flows in one direction may have a different TOS field value than those of the other direction. Therefore we currently ignore this field.

FLOW-REDUCE uses two parameters. We determined a reasonable value for `flow_inact_tout` experimentally: we started with a large value of 10 minutes and then examined the distribution of the flow connections' maximum inactivity time. Manual inspection lead us to choose a value of 215 seconds for both traces. Our choice of `flow_inact_tout_f`, 30 seconds, is motivated by common values of maximum segment lifetimes and BRO's default behavior.

D. Quality assessment of flow connections

In order to evaluate our flow connection algorithm, we compare the summaries produced by BRO and FLOW-REDUCE first in general and then on a per connection basis. Ideally, for each connection found in one of the data sets there should be exactly one in the other. There are four reasons why this may not be the case, assuming that a packet connection correctly represents a TCP connection:

1. Two TCP connections violate assumption one of Section IV-C and are aggregated into the same NetFlow. This results in two packet connections but only one flow connection.
2. Two TCP connections violate assumption one and FLOW-REDUCE aggregates them into the same flow connection, due to a too large value for `flow_inact_tout`. Again, this results in two packet connections but only one flow connection.
3. A TCP connection violates assumption two and FLOW-REDUCE splits it into two flow connections due to a too small value for `flow_inact_tout`. This results in one packet connection vs. two flow connections.
4. Two NetFlows violate FLOW-REDUCE's assumption about concurrency of NetFlows due to Cisco's undocumented flow definition features. This results in one packet connection vs. two flow connections.

The first is a general NetFlow problem while the others point out possible limitations of our heuristic. In the reminder of this section we show that the first limitation is the most bothersome. But overall the results are rather promising.

D.1 General characteristics of flow connections

We start with some overall characteristics of flow connections. On average FLOW-REDUCE combines 2.188 NetFlows into one flow connection. While the maximum is 5499 the median is 2. This implies that most (65.9%) flow connections consist of only two NetFlows. This can be easily explained by the large number of short-lived TCP connections (84.1% of the packet connections are shorter than 15 seconds). These will, most of the time, result in exactly one NetFlow for each direction. Only 21.9% consist of one[4] NetFlow and 12.2% of more than two NetFlows.

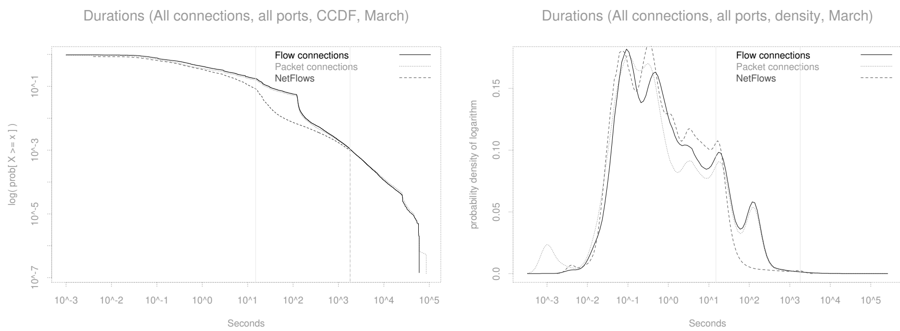
Figure 5 (a) shows the duration of all NetFlows, packet connections and flow connections[5] in a plot of the complementary cumulative distribution (CCDF) on a log-log scale for the March data set. We observe similar results for the April data set. The timeouts of the router are marked by vertical lines. We clearly see their effect on the duration of the NetFlows: the `inactive_timeout` allows the router to expire a significant number of NetFlows, and the `active_timeout` bounds their durations to an upper limit. On the other hand, we do not see any influence of the timeouts on the flow connections. Instead, flow and packet connections show very similar durations in this plot.

To further investigate the differences between flow and packet connections, we plot (Figure 5 (b)) the *density of the logarithm*[6] of their durations. For durations larger than the router's `inactive_timeout` we still see a very good match: While flow connections and packet connections show the same characteristics, the probability of NetFlows decreases rapidly until `active_timeout` is reached. But for smaller durations flow and packet connections differ considerably. Although the flow connections resemble the packet connections more closely than the NetFlows, there are quite a few incorrectly identified connections. There are two issues to consider. On the one hand the log scale forces us to exclude connections of zero duration and therefore changes the relative proportions. On the other hand there are the four reasons outlined above. We find that only one of them can lead to these inaccuracies. Since roughly 86% of the flow connections which are shorter than `inactive_timeout` seconds contain at most one NetFlow for each direction the seconds reason is eliminated. The third reason cannot cause the inaccuracies either since only 0.7% of all

[4]70.7% of this is Web traffic and probably due to the intercepting proxy

[5]In order to compare uni-directional NetFlows with bi-directional connections we have included each connection twice.

[6]Coupled with a logarithmic scale on the x-axis, plotting the density of the logarithm of the data facilitates direct comparisons between different parts of the graphs based on the area under the curve.



(a) (b)

Fig. 5. Distribution of durations for packet and flow connections and NetFlows (March data)

packet connections last longer than `flow_inact_tout` seconds. The fourth reason contributes extra flow connections, but only a small fraction (1.4%). We therefore conclude that the vast majority of the differences are due to multiple TCP connections aggregated into the same NetFlow.

D.2 Per-connection comparison

To enable a pair-wise comparison of flow and packet connections we identify pairs of connections which correspond to the same TCP connection. Each TCP connection is identified by the combination of $conn_{id}$ and lifetime. Accordingly, we associate a flow and packet connection with each other if they use the same $conn_{id}$ and start simultaneously[7]. The result of the correlation step is a set of connection pairs, both flow and packet connection, a set of flow-only connections and a set of packet-only connections. The total number of connections is the sum of the sizes of all sets.

Table V summarizes the result of the correlation step. The correlated connection set contains 90.5% of all connections. Given that the total number of packet connections is 6.2% larger than the number of flow connections it is not surprising that the set of packet-only connections is bigger than the set of flow-only connections. Closer examination of the unmatched packet-only connections reveals that most are in the state `REJ` which indicates that no connection ever got established. One explanation for why there is no matching flow connection for these packet connections is the first observation of Section IV-D. NetFlow tends to combine packets of several TCP connections together if they use the same $conn_{id}$. Packet-only connections with state `REJ` are especially subject to this, if, as, e.g., seen for some file-sharing applications, the client uses the same socket for multiple connection attempts to another client. Closer inspection of the set of unmatched flow-only connections reveals that it contains a substantial fraction of long connections: Over $20.1\%/11.2\%$ last more than $30/100$ seconds. This suggests that due to inactivity periods long TCP connections may have been split into several flow connections.

[7]Due to the issue of clock skews we add a generous tolerance window of 3 minutes.

The various applications contribute to the sets in different ways. For example the success rate for Web traffic is very high with 97.8%. On the other hand the three file-sharing tools – `Gnutella`, `Kaaza` and `E-Donkey` – are contributing a lot of unmatched packet-only connections. In Section IV-E we examine which application protocol artifacts contribute to these unmatched connections.

Another reason for why we observe some packet-only and flow-only connections is our choice of data sets. The packet trace starts and ends exactly at midnight. This implies that quite a few packet connections start/end shortly after/before midnight even though their TCP connection started/ended much earlier/later. For these there will be no matching flow connection. Similar artifacts apply to the NetFlow trace explaining some flow-only connections.

TABLE V
SUMMARY OF CORRELATED CONNECTIONS (% OF APPLICATION'S TOTAL)

Application	Total	Correlated	Flow-only	Packet-only
All	100.0	90.5	1.4	8.1
`HTTP`	67.9	97.8	0.5	1.8
`Gnutella`	12.3	64.7	3.7	31.6
`eDonkey2000`	4.3	86.0	4.5	9.5
`FTP`	0.1	86.9	1.4	11.6
`FTP data`	0.2	98.9	0.8	0.3
`HTTPS`	1.0	97.6	0.3	2.1
`POP3`	1.3	93.1	0.4	6.6
`Kazaa`	1.7	61.6	6.7	31.7
`IRC`	1.1	99.0	0.9	0.1
`SMTP`	1.4	93.1	2.0	6.6

The next step is to analyze the quality of the fit within the set of correlated connections. We choose to consider the packet connections as our basis since they presumable represent TCP connections better than flow connections. Accordingly we look for explanations of why the values of the flow connections differ from the values of their matching packet connection. Note that errors from FLOW-REDUCE will percolate into this comparison. We compare the matched NetFlows along the following fields: byte counts, duration, originator, and state.

Byte count: Figure 6 (a) plots the *density of the logarithm* of the total number of bytes[8] transfered by packet and flow connections. The comparison of the curves indicates that overall both connection types match well. But there are some mismatches: notable for small byte counts. In general byte count mismatches can be due to:

- NetFlows aggregates all packets not just those which are eventually delivered to an applications. In particular, it counts unacknowledged and retransmitted packets [25].
- While BRO counts payload bytes NetFlow counts full IP packets. To compensate for this we adjust the NetFlow counts by subtracting a minimum size TCP header, 40 bytes, for each packet from NetFlow. To account for TCP options in `SYN` packets we subtract eight bytes for each direction. Should this yield a payload less than zero we set the payload to zero for this direction. While imperfect (it does not account for other options) this heuristic does quite well.
- An imperfect flow connection, one that does not include all NetFlows or one that includes more NetFlows than it should, is likely to have a significantly different, smaller or larger, byte count than the TCP connection.

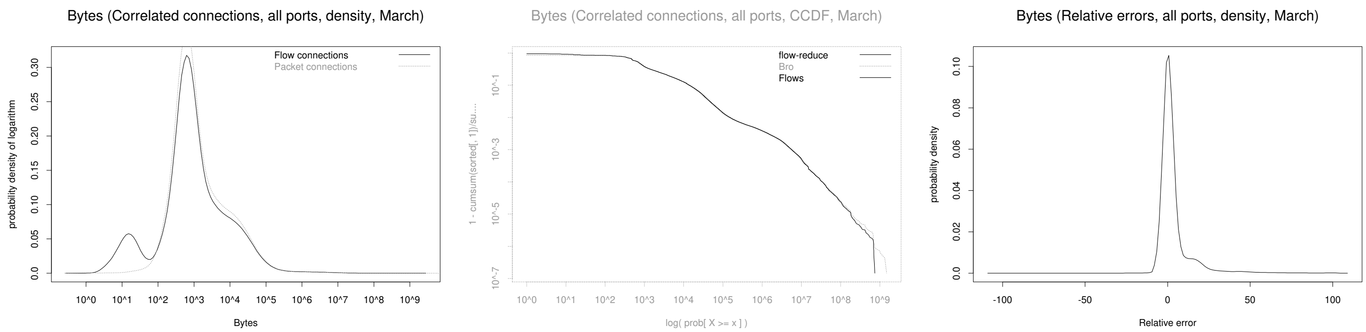
One notable disagreement in the densities of the two connection types are connections with a very small transfer volume (five to thirty bytes). The first impression from the plot is that FLOW-REDUCE sometimes underestimates the volume considerable. This is indeed the case for roughly 1% of the flow connections with byte counts in the range of $5 - 30$ bytes. Examining these flow connections more closely reveals that around 85.6% of them have been associated with packet connections in states `REJ`, `RSTR`, or `S0` and without payload. We assume that the router aggregates several connection attempts together. Correlating a flow connection containing such a NetFlow to a packet connection summarizing a single connection attempt leads, among other mismatches, to byte count mismatches. Our second impression is that FLOW-REDUCE sometimes overestimates the volume due to obmitting connections with zero byte counts. Since our plot uses a log scale these are obmitted. There are 13.6% correlated packet connections and 4.5% correlated flow connections with a byte count of zero. This further supports the conjecture that many connection attempts are aggregated into common NetFlows. The fourth reason of Section IV-D, time overlapping NetFlows using the same sockets, can lead to underestimating the bytes in a flow connection. One of the parallel NetFlows usually (72%) consists of a single packet while the other consists of several packets. Therefore "incorrectly" choosing the single packet NetFlow during the associating phase will lead to the mismatch.

Plotting the density of the logarithm is an excellent tool for understanding differences in the body of the distribution, but it does not allow us a closer inspection of the tail. Plotting the *CCDF on a log-log scale of the byte counts*, Figure 6 (b), lets us confirm that the curves match rather well. Notable differences only occur for connections transferring more than 10^8 MBytes. Note that there are a "grand total" of 13 connections of this type. Therefore it is questionable to conclude general statements about these. We presume that some of these outliers are due to the sequence number effects discussed in Section IV-A.

So far we have only compared the overall distributions of byte counts from flow and packet connections. But how good are the individual matches. Figure 6 (c) plots the *density of the relative errors of the byte counts*. The relative error is the percentage difference between the larger value and the smaller value. If the byte count of the packet connection is bigger we force the value to be negative otherwise the value is positive.[9] The advantage of using the relative error is that is captures both bigger and smaller deviations in a similar way centered at zero. If either the packet or the flow connection has zero bytes the relative error is infinite and we excluded it from the plot. This means that we exclude $1000K$ connections which corresponds to 15% of the connection pairs. For these we find that 30% of the connection pairs where both connections have a payload of zero. In 99% of the cases the packet connections have a payload of zero. The later are most likely rejected connection attempts that may have been incorrectly mapped to a NetFlow summarizing multiple such connection attempts. Furthermore we find that only a very small percentage, $0.08\%/0.75\%/11.2\%$ which corresponds to $5.4K/51.4K/764K$ connection pairs, have an relative error smaller/bigger than $\pm 1000/100/10$ percent. Since the number of connection pairs with large errors is so small Figure 6 (c) plots the *density of the relative errors* only for the range of ± 100. We see that most of er-

[8]The total number of bytes is the sum of the number of bytes transfered in each direction.

[9]For example a relative error of 100% means that the bytes derived of the flow connection is twice the bytes of the packet connection. A relative error of -50% means that the bytes of the flow connection is a factor of 1.5 bigger than the bytes of the packet connection. A relative error of 0 means that the byte counts are equal.



(a) (b) (c)

Fig. 6. Distribution of byte counts for packet and flow connections (March data)

rors are small and centered around zero. But there is a notable number of flow connections which contain considerably more bytes than their packet counter part. This is to be expected since flow connections include unacknowledged and retransmitted packets. Another reason is that our heuristic for removing header bytes from flow connections is imprecise.

Duration: In we Figure 5 we have shown the probability distributions of the durations of the uncorrelated packet and flow connections. The probability distribution for the correlated connection pairs (not shown) are similar in spirit. We observe an excellent match between the two curves for connections that last longer than the `inactive_timeout`. The relative percentage of very short packet connections has been reduced indicating that we cannot find matching flow connections for all of them since the packets of these connections have been aggregated into longer NetFlows. All in all there is a sizable disagreement in the range up to the value of `inactive_timeout`.

To compare the durations of the individual matches, Figure 7 plots the relative errors as percentage differences between larger duration and smaller duration. Connection pairs containing a connection of zero duration are removed (2%). In addition, we have omitted connections pairs with an absolute relative error larger than 100 (4.6%). From the plot we see, that most of the pairs match well. However, for 74.4% of the pairs the flow connection has a duration longer than the packet connection.

As already noted a NetFlow can contain packets from several TCP connections. This extends the duration of the NetFlow and of the corresponding flow connection. The resolution of the time stamps are different: while time stamps of the packet traces have microsecond resolution, the NetFlows' time stamps have a only millisecond resolution. Independent of the question about accuracy, these different resolutions will lead to differences for very short connections. In addition FLOW-REDUCE may incorrectly combine NetFlows which do not correspond to the same TCP connections, or may fail to combine some, which does create mismatches in the connection durations. The most important reason is the first one: For 89.0% of the connection pairs with a relative error larger than zero, the flow connection only contains at most one NetFlows for each direction. This implies that the NetFlows themselves already last longer than the corresponding packet connection.

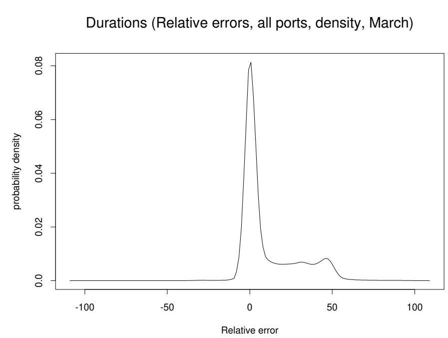


Fig. 7. Relative errors of durations for packet and flow connections (March data)

Connection originator: The comparison of the originator fields of the flow connections and the packet connections confirms that our simple heuristic works well for 99.5% of the cases. It appears that the reasons from Section IV-D do not impact the accuracy of FLOW-REDUCE's connection originator heuristic in any substantial fashion. This is confirmed if we examine the distribution of the errors across the various applications, see Table VI (a). All of them show values in the same order of magnitude.

Connection state: We find that in 98.4% the corresponding connections have matching states. Overall considering that FLOW-REDUCE only uses aggregated information this percentage is remarkable high. Table VI (a) shows how the percentages differ across the various applications. For some applications the agreement is better than for others. Its again the file-sharing applications for which we observe the "highest" error rates but still less than 3%.

To answer the question of the origin of the mismatches we examine which of the BRO states are most commonly misclassified by FLOW-REDUCE, see Table VI (b). We find that `REJ` account for 29.3% of the errors and also the largest absolute number of errors we explore which states FLOW-REDUCE generated instead. We find that 70.07% of the mismatching `REJ` packet connections are classified by FLOW-REDUCE as responder resets (`RSTR`). According to Figure IV this implies that the flow connections contain a responder-side `RST` packet and one `SYN` packet from each side (note that we match from top to bottom). On the other hand the packet connection contained at least one `RST` packet but no `SYN` from the responder. Therefore we have to assume that the flow connections combined several rejected TCP connection requests and a successful TCP connection establishment (all involving the same $conn_{id}$). We observed this kind of behavior with file-sharing applications. Indeed 70.0% of the (`REJ`,`RSTR`) pairs are `Gnutella` connections. Similar explanations hold for the other state mismatches.

E. *Quality assessment of application specific flow connections*

So far, the outcome of the comparison of flow connections and packet connections has been rather positive. FLOW-REDUCE performs very well. Nevertheless we have noticed that there are application characteristics that create certain kinds of TCP connection patterns that complicate the reconstruction from NetFlows. Now we consider two applications in more detail: Web traffic and `Gnutella` which contribute 67.9%/12.3% of the connections in March, see Table V.

Web traffic: Figure 9 plots the *densities of the logarithms* of the durations of HTTP connections in March and April. First note the different distributions of Web traffic within the two data sets. The intercepting proxy was imposed on a sizable fraction of the local subnets in March (see Section III). This creates a large number of half-open connections for the client requests. Accordingly 16.0%/15.96% of packet/flow connections are in state `SH`. For April the values are negligible (0.2%/0.4%).

Comparing the plots of the durations of all connections (Figure 5 (a)) with those for only Web traffic we observe a much better match for the latter. In particular, FLOW-REDUCE performs very well on the April data. Figure 8 plots the *relative errors* of correlated Web connections. Therefore it might be surprising to observe that the relative errors of the durations are larger for Web traffic (Figure 8 (a)) than for all traffic (Figure 7). But consider the large number of very short Web connections. We find that 81.9% of all correlated Web connections have a relative error between 0 and 50. For 61.0% of these the duration of the packet connection is less than 0.5 seconds. Since all the other fields including connection states (98.7% agreement) match rather well we believe that the differences are due to clock inaccuracies and not due to NetFlow or FLOW-REDUCE combining multiple HTTP connections to one flow connection. This is supported by the observation that the number of flow-only and packet-only connections, Table VI (c), is rather small. Figure 8 (b) shows the relative errors of byte counts. In this case the byte counts are closely centered around zero and the overall relative errors are comparable to the case where all connections are included. This supports the clock inaccuracy explanation for the duration differences. Overall we conclude that FLOW-REDUCE performs very well for Web traffic.

```
5354.386 A.B.C.D.1283 > E.F.G.H.6346: S 7:7(0)
5354.496 E.F.G.H.6346 > A.B.C.D.1283: R 0:0(0) ack 8
5354.957 A.B.C.D.1283 > E.F.G.H.6346: S 7:7(0)
5355.074 E.F.G.H.6346 > A.B.C.D.1283: R 0:0(0) ack 8
5355.558 A.B.C.D.1283 > E.F.G.H.6346: S 7:7(0)
5355.711 E.F.G.H.6346 > A.B.C.D.1283: S 148:148(0)
[...]
```

Fig. 11. Example `tcpdump` output of `Gnutella` connections attempts (shortened)

Gnutella: For `Gnutella`[26] connections[10] Figure 10 plots (a) the *density of the logarithms of the durations*, (b) the *density of the relative errors of the durations*, and (c) *of the byte counts* while Table VI (c) summarizes the other performance criteria. On the one hand we observe better than overall results for the relative errors of the connection durations (Figure 10 (b)). On the other hand, the same cannot be said for byte counts and the other criteria. Rather these indicate that FLOW-REDUCE performs quite poorly for `Gnutella`. Over 30% of the packet connections have not been associated with a flow connection. The duration distributions of all flow and packet connections (not shown) show rather different characteristics and even those in the correlated set still show quite some differences both for durations as well as for byte counts (latter plot omitted). The density of the relative errors of the byte counts show spikes at approximately 44 and 100.

We assume that the specific characteristics of the peer-to-peer application `Gnutella` are the main contributing factor. `Gnutella` clients maintain a sizable list of peers to whom they had at some point established a connection. Upon start-up they try to establish connections to a subset of these. If a connection request is rejected there may be several reattempts. There are some clients that, reusing

[10]While `Gnutella` may be used on different ports we only consider port 6346 which is the most common one.

TABLE VI

COMPARISON OF CORRELATED PACKET AND FLOW CONNECTIONS

Application	Direction	States
All	99.5	98.4
HTTP	99.7	98.7
Gnutella	99.5	97.0
eDonkey2000	99.0	97.3
FTP	99.8	98.4
FTP data	99.6	98.5
HTTPS	99.8	99.2
POP3	99.5	99.1
Kazaa	99.6	97.5
IRC	99.7	99.7

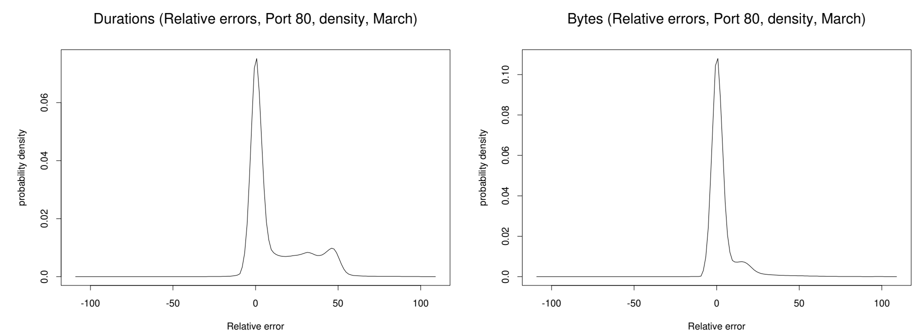
(a) Matching directions/states (% of application's total)

State	Percent
REJ	70.7
SF	76.3
RSTR	83.3
RSTO	89.3
SH	91.3
S0	95.4
SHR	96.8
RSTOS0	98.6
RSTRH	99.4
S3	99.6
S1	99.7
S2	99.8
OTH	99.9

(b) Matching per BRO state (% of state's total)

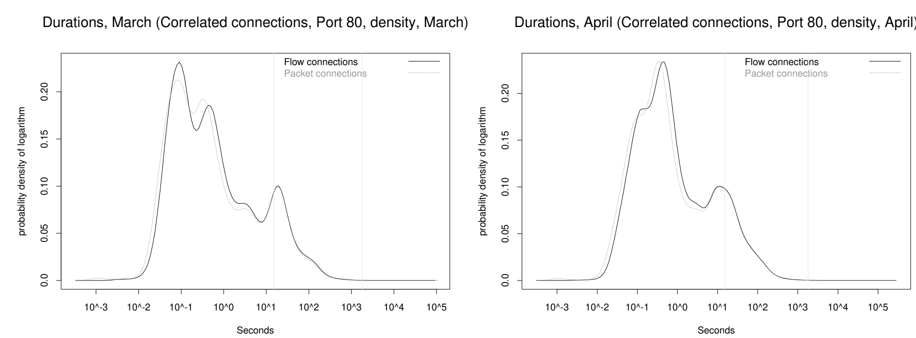
Name	Web	Gnutella
Correlated	97.8%	64.7%
Flow-only	0.5%	3.7%
Packet-only	1.8%	31.6%
Direction match	99.7%	95.5%
State match	98.7%	97.0%

(c) Comparison of Web and Gnutella (March data) % of total per application



(a) (b)

Fig. 8. Port 80: Relative errors of packet and flow connections (March data)

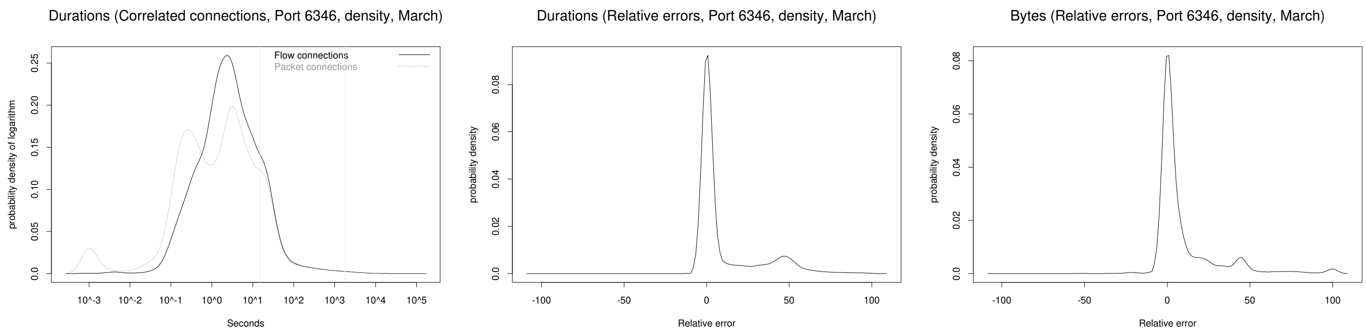


(a) (b)

Fig. 9. Port 80: Distribution of durations of packet and flow connections

the old socket, repeatedly try to reconnect within a very short time (a few seconds). Figure 11 shows a typical example from our traces. (Figure 2 is generated from this trace). From a clients viewpoint this is a sensible behavior: Gnutella clients allow connections to only a small number of peers and close old connections occasionally. Therefore, even if the first connection attempt does not, a retry might succeed. This connection setup phase violates assumption one of Section IV-C. Accordingly NetFlow will summarize all packets of the retried TCP connection attempts into the same NetFlows. The process of correlating packet and flow connections will pair the first packet connection with the flow connection. This may be the rejected connection which can cause extreme mismatches. 38.9% of all non-correlated packet connections are rejected Gnutella connections.

We conclude that Gnutella's setup phase mislead both FLOW-REDUCE and the correlation. Other file-sharers, e.g., Kazaa[27] and eDonkey2000[28], fare a bit but not that much better. We assume for similar rea-



(a) (b) (c)

Fig. 10. Port 6346: Distribution of durations and byte counts for packet and flow connections (March data)

sons.

V. Analysis of byte/packet counts

One common use of CISCO's NetFlows is to extract the application specific byte counts[11] of the traffic passing through a router. These values may, for example, be used for accounting, e.g., [18], [17] or visualization, e.g., [16]. While SNMP [1] data provides byte counts only on a per router and per interface basis, NetFlow contains additional information, see Table I. For accounting purposes the following NetFlow are particular interesting: the src/dst IP address and network masks, the src/dst autonomous system numbers and the next hop, the transport layer protocol and the src/dst port numbers, and the input/output interfaces.

Given this information tools like `FlowScan` [16] can be used to, e.g., aggregate byte counts per subnet, per application or per destination autonomous system. There is just one problem with this approach as compared to SNMP, granularity. This translates into a possible lack of accuracy. The data that is sampled via SNMP is a cumulative counter that counts every packet/byte. Querying the counter we can determine the exact number of bytes that have passed a certain interface since the last query. The granularity of NetFlows is much coarser: NetFlows already are packet/byte aggregates. If we want to derive counts at regular intervals from NetFlows, we have to distribute the bytes/packets from the NetFlows over time.

More formally, we define a NetFlow $flow_i$ to have a start time $start_i$, end time end_i, export time $export_i$, and byte count $bytes_i$. We have a time window $[0, T]$ divided into $n := \lceil T/I \rceil$ time slots T_j of size I. With each time slot T_j we associate two values: an actual count B_j of all bytes encountered within this slot; and a *flow byte count* F_j which is derived from all processed NetFlows. The goal is to approximate B_j by F_j for all time slots. To derive the set F_i of flow byte counts, we define a function $f : flow_i \to \mathcal{P}(\{0, \ldots, n-1\})$ which distributes $bytes_i$ to a subset of $\{B_j\}$. $\mathcal{P}(U)$ is the power set of U.

We consider four choices for f: given a NetFlow $flow_i$

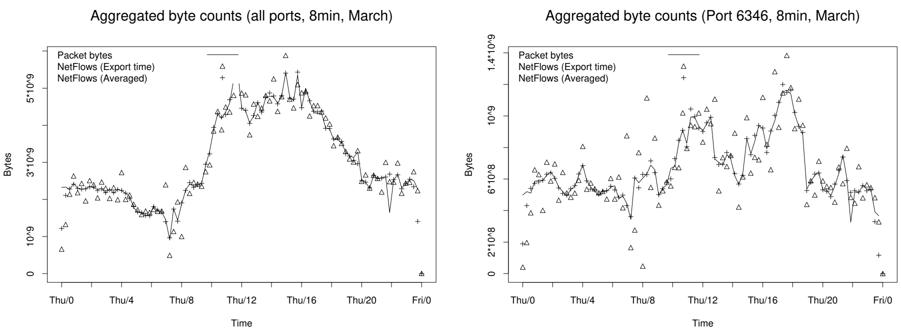
1. add $bytes_i$ to the time slot which contains $export_i$.
2. add $bytes_i$ to the time slot which contains $start_i$.
3. add $bytes_i$ to the time slot which contains end_i.
4. prorate $bytes_i$ across all time slots intersecting the interval $[start_i, end_i]$.

The first choice is commonly used for accounting and visualizing (for example by `FlowScan`[16]). Its main advantage is that it is well-suited for real-time processing: only one time slot is adjusted with every new flow and its number is monotonically increasing. This implies that counts of earlier time slots are preserved. On the other hand, remember Cisco's definition of NetFlow (see Section II). A devils advocate might consider the export time "independent" of the time when the packets actually passed the router. In the worst case an active NetFlow starting at time $start_i$ will be exported at time $start_i + \texttt{active\_timeout}$. In our case this will be 30 minutes later.

Similar observations apply to the second and third aggregation strategy. Therefore, we expect that the first three aggregation strategies might perform rather badly if applied to a time slots of small size (say $< 2 \times \texttt{active\_timeout}$). For intervals that are sufficiently large we expect that the law of large numbers will hold and the errors will average out. Note that even with arbitrary large intervals there can be significant errors if one NetFlow contains the majority of the bytes and starts/ends/is_exported just before/after/after a time slot boundary.

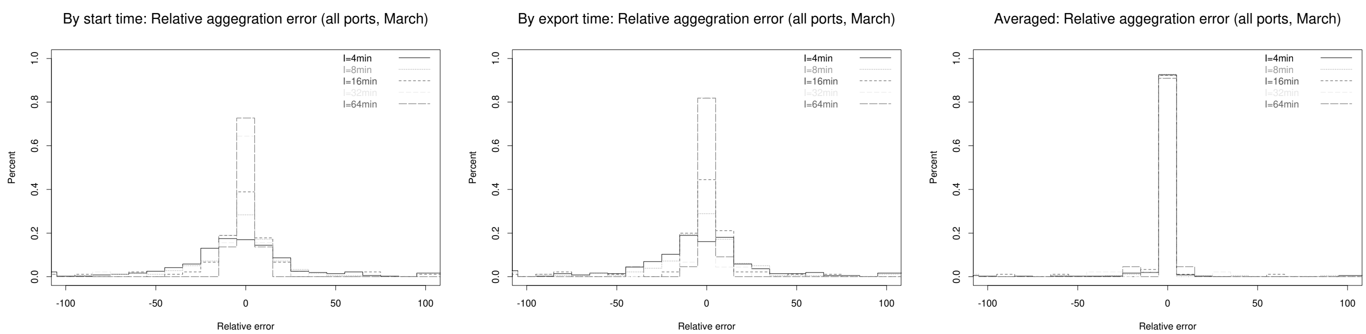
The fourth aggregation strategy is build upon the assumption that bytes within a NetFlow are evenly distributed over the duration of the NetFlow. Given that most NetFlows are aggregating TCP this is likely to be the case

[11] The following discussion applies to packet counts as well.



(a) (b)

Fig. 12. Byte counts aggregated into time slots of length $I = 8$ minutes for all and `Gnutella` traffic



(a) (b) (c)

Fig. 13. Aggregations: Densities of relative errors

but there is no guarantee either[12].

We have applied all four aggregation methods to our NetFlow data sets from March and April. For the evaluation we derive SNMP-like data from the packet traces, counting bytes per time slot (excluding link level headers). Figure 12 plots the distribution of packet bytes for an aggregation interval of $I = 8$ minutes (using the March data) for (a) all traffic and (b) `Gnutella` only traffic. In addition, it shows the corresponding values of NetFlow aggregation by export time as well as by averaging. We see that averaged NetFlows match the actual byte counts from packet traces very well. On the other hand aggregation by export time shows distinct deviations which indicate rather large errors. This plot also highlights that the cleaning of the two data sets (see Section III), packet traces and netflow data, is successful. While `Gnutella` created a major challenge for us in the analysis of TCP connection summaries here it does behave just as well as all the other applications. This highlights that NetFlow is an appropriate tool to derive application level summaries.

[12]Just presume a TCP connection lasting for 30 minutes and sending a packet every 15 seconds to keep the corresponding NetFlow active. If this flow sends at the maximum rate within the first x minutes it may transmit significantly more data during those minutes than during the rest of its duration, thereby fooling this aggregation strategy, e.g., consider a BGP session reset.

To see what effect the choice of aggregation interval has Figure 13 plots the relative errors for different intervals starting from 4 minutes to 64 minutes. Note that 4 minutes is a typical frequency for SNMP queries. Figure 13 plots aggregations based on (a) start time, (b) export time and (c) averaging (the plot for end time resembles the plot for export time). Note the different scales on the y-axis. In calculating the densities we have excluded all time slots within the first/last 30 minutes of the trace to avoid edge effects. For plotting, we excluded intervals with an relative error smaller/larger than ± 100 ($0\% - 1.4\%$ for (a), $0\% - 1.2\%$ for (b) and $0\% - 0.3\%$ for (c)). Averaging yields the best results for all time intervals. The error rates for aggregation by export time are maybe a bit better than those by start time. While both perform poorly for the first three intervals both improve considerably at $I = 64$ minutes. Additional tests with larger intervals indicate that they approach the quality of the averaging aggregation. Due to our limit data sets we cannot generalize this.

VI. Summary

In this paper we set out to gain insights into the capabilities of NetFlow. In what sense can we gain the same or additional information from NetFlow as compared to finer grained or coarser grained measurement techniques.

Obviously NetFlow cannot provide the packet level detail of packet traces. On the other hand we somewhat surprisingly do not have to abandon TCP connection summary information. We propose a methodology and present an implementation, FLOW-REDUCE, that is able to combine NetFlows into flow connections. Our evaluation shows that the derived flow connection summaries compare rather favorable to TCP connection summaries. This is the case even for such none-obvious fields as TCP connection state and end-point role. Therefore one can say that FLOW-REDUCE overcomes the limitations of Cisco's unspecific and resource dependent definition of NetFlows. Our detailed analysis of the mismatches reveals several problems: NetFlow may aggregate packets from several TCP connections into one NetFlow, e.g., if the same socket is used for several connection attempts as done by file-sharing applications, NetFlow does not allow us to differentiate between transmitted data and goodput, and NetFlow's time resolution is limited to milliseconds.

While NetFlows provide more detail than SNMP statistics it is unclear with what accuracy one can derive statistics similar to SNMP statistics from them. Comparing NetFlow derived byte/packet counts against SNMP like counts we find that using a certain aggregation technique the relative errors are surprisingly small even for short aggregation periods. Otherwise we find it necessary to use larger aggregation periods in order to achieve similar quality. But using the most common aggregation technique based on export time together with small aggregation time periods can be misleading due to sizable deviations. Therefore while NetFlow looses some accuracy compared to SNMP the loss is not significant if weighted against the information gain. NetFlow enables us to compute a more differentiated view of the traffic, e.g., application specific traffic counts. We find that the same observations about quality are applicable. NetFlow does a good job if the right aggregation technique or/and sufficiently large aggregation time periods are used.

REFERENCES

[1] William Stallings, *SNMP, SNMPv2, SNMPv3 and RMON 1 and 2*, 1999.

[2] Cisco Systems Inc., "NetFlow Services and Applications - White paper," http://www.cisco.com/warp/public/cc/pd/iosw/ioft/neflct/tech/napps_wp.h%tm.

[3] Cisco IP accounting, ," http://www.cisco.com/warp/public/477/SNMP/mac_ip_snmp.shtml.

[4] CAIDA NGI Project: OC48mon, ," http://www.caida.org/projects/ngi/content/.

[5] "tcpdump," http://www.tcpdump.org.

[6] 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*, vol. 13, no. 8, pp. 1481–1494, 1995.

[7] Kevin Thompson, Gregory J. Miller, and Rick Wilder, "Wide-area internet traffic patterns and characteristics," *IEEE Network Magazine*, vol. 11, no. 6, pp. 10–23, November/December 1997.

[8] Steven Lin and Nick McKeown, "A simulation study of IP switching," in *Proceedings of ACM SIGCOMM 1997*, Sept. 1997, pp. 15–24.

[9] Anja Feldmann, Jennifer Rexford, and Ramon Caceres, "Efficient policies for carrying Web traffic over flow-switched net works," *IEEE/ACM Transactions on Networking*, pp. 673–685, Dec. 1998.

[10] Peter Newman, Greg Minshall, and Tom Lyon, "IP switching: ATM under IP," *IEEE/ACM Transactions on Networking*, vol. 6, no. 2, pp. 117–129, Apr. 1998.

[11] Paul Barford and Dave Plonka, "Characteristics of Network Traffic Flow Anomalies," in *Proceedings of ACM SIGCOMM Internet Measurement Workshop (IMW) 2001*, 2001.

[12] Anja Feldmann, Albert Greenberg, Carsten Lund, Nick Reingold, Jennifer Rexford, , and Fred True, "Deriving traffic demands for operational ip networks: Methodology and experience," Oct. 2000, pp. 257–2500.

[13] S. Handelman, S. Stibler, N. Brownlee, and G. Ruth, "RTFM: New attributes for traffic flow measurement," Request for Comments 2724, Oct. 1999.

[14] N. Brownlee, "Traffic flow measurement: Meter MIB," Request for Comments 2720, Oct. 1999.

[15] "Internet Protocol Flow Information eXport (IPFIX)," http://ipfix.doit.wisc.edu.

[16] Dave Plonka, "FlowScan: A network traffic flow reporting and visualization tool," in *Proceedings of the 14th Conference on Systems Administration (LISA) 2000*, Berkeley, CA, Dec. 2000, pp. 305–318, USENIX Association.

[17] Simon Leinen, "Flow-based Traffic Analysis at SWITCH," Poster at PAM2001.

[18] Mark Fullmer and Steve Romig, "The OSU flow-tools package and CISCO netflow logs," in *Proceedings of the 14th Conference on Systems Administration (LISA) 2000*, Berkeley, CA, Dec. 2000, pp. 291–304, USENIX Association.

[19] "cflowd: Traffic Flow Analysis Tool," http://www.caida.org/tools/measurement/cflowd.

[20] Vern Paxson, "On Calibrating Measurements of Packet Transit Times," in *Proceedings of ACM SIGMETRICS 1998*, jun 1998.

[21] "tcp-reduce," http://ita.ee.lbl.gov/html/contrib/tcp-reduce.html.

[22] Vern Paxson, "Bro: A system for detecting network intruders in real-time," *Computer Networks*, vol. 31, no. 23–24, pp. 2435–2463, 1999.

[23] Thomas H. Ptacek and Timothy N. Newsham, "Insertion, evasion, and denial of service: Eluding network intrusion detection," Tech. Rep., Secure Networks, Inc., jan 1998.

[24] "Python," http://www.python.org.

[25] Paul Barford and Dave Plonka, "Inferring Client Experience From Flow-based Measurements," http://www.cs.wisc.edu/~pb/pbdp1_imw_01.ps.

[26] "Gnutella," http://www.gnutelliums.com.

[27] "Kazaa," http://www.kazaa.com.

[28] "eDonkey2000," http://www.edonkey2000.org.