

Consistency Analysis of Network Traffic Repositories

Elmer Lastdrager and Aiko Pras

University of Twente, the Netherlands

`e.e.h.lastdrager@student.utwente.nl`, `a.pras@utwente.nl`

Abstract. Traffic repositories with TCP/IP header information are very important for network analysis. Researchers often assume that such repositories reliably represent all traffic that has been flowing over the network; little thoughts are made regarding the consistency of these repositories. Still, for various reasons, the traffic capturing process may have missed packets. For certain kinds of analysis, for example loss measurements, such inconsistencies may lead to the wrong conclusions.

This paper proposes an algorithm to detect such inconsistencies, using the idea of “fake gaps”. A prototype has been developed, and used to test two well-known repositories: the WIDE and Simpleweb repositories. The paper shows that both repositories contain several inconsistencies.

1 Introduction

A network traffic repository contains network traffic gathered from one or more location(s), often a router or backbone. Captured traffic is stored in data files in a repository; typically such files contain data captured over a longer period, for example minutes, hours or even days. Repositories can store different types of network data, for example TCP/IP header files, netflow records or SNMP packets. In this paper the focus is on the most common type: TCP/IP header data.

Using a repository can be very convenient for a researcher, as gathering data yourself can be very time-consuming, or even impossible. A potential issue in using a repository, is the consistency of the traffic inside the repository. When traffic inside a repository does not completely correspond with the actual traffic that was transmitted and received, this can influence measurements, analysis and therefore also conclusions that researchers draw. Hence, it is critical to have information about the consistency of the network traffic repository, so it can be taken into account when analysing the data.

Issues with consistency of the data in repositories have been reported by M. Timmer in [1]. While using a repository, it appeared that not all data was recorded properly. Timmer introduces the term “fake gap” to represent those

parts of a TCP flow that are absent in the repository, although they were acknowledged at the TCP level. In [2] a relatively simple algorithm has been developed to find a sudden decrease in data in small intervals, which may indicate a problem with the repository. However it may very well be a temporary network problem and therefore does not necessarily affect the consistency of the repository. Although [2] has performed some initial research, the consistency analysis never exceeded a few data files. However, as researcher, knowledge about inconsistency in a repository is essential. At this moment, there are no statistics about possible inconsistency of repositories available. In this paper we will therefore analyse two well-known repositories: the WIDE and Simpleweb repositories. A tool has been developed that analyses TCP flows. We focus on TCP, because its state-full nature allows detection of fake gaps; for UDP this is, due to its stateless nature, not possible.

In this paper, the main question that will be answered is: *How can inconsistency be detected in a TCP traffic repository?*

To answer the main question, we first have a look at possible inconsistencies by answering a sub question: *What could cause inconsistencies in a TCP traffic repository?*

Next, we will focus on detecting fake gaps by introducing an algorithm. The sub question we will answer is: *How can we detect fake gaps?*

Next, we built a prototype of the algorithm to test existing repositories in order to answer the second sub question: *How consistent are today's repositories?*

It should be noted that a short version, covering roughly half of this paper, has already been published at AIMS 2009 [3]. Furthermore, an earlier version of this paper has been presented at the tenth Twente Student Conference on Information Technology [4]. That conference is an internal conference of the University of Twente, of which the proceedings have not officially been published by a real publisher.

The structure of this paper is as follows. In Section 2 we will briefly identify possible causes that could lead to inconsistent repositories. In Section 3 we will propose an algorithm to detect inconsistency and introduce the prototype we built. In Section 4 this prototype will be evaluated. In Section 5 we will discuss the results of testing two existing repositories using the prototype and finally in Section 6 we will answer our research questions, draw conclusions and discuss possible future work.

2 Causes for Inconsistency

In this Section, we will think of ways a repository can become inconsistent. We do not intend to be complete and provide an in-depth analysis. Instead, we intend to show the reader possible issues that could lead to inconsistency. Inconsistency in a repository occurs only when the recording device failed to record the actual traffic that was sent. We distinguish issues at the switch or router side (where the traffic is copied) and issues at the recording device itself.

2.1 At the Side of the Switch

For recording data, the *port mirroring* feature of a switch can be used. This is a method of recording data without interfering with the regular operation [5]. Port mirroring is used to copy all traffic from one, or more, port(s) at the switch to another port of the switch, called the *mirror port*. When the traffic from only one port is copied to the mirror port, there shouldn't be any problem as long as the bandwidth on the mirror port is greater than, or equal to, the bandwidth of the source port. However, the situation in which all traffic is copied to the mirror port is more interesting. In this case, the mirror port could be overloaded when the bandwidth used by all source ports combined is too large for it to handle. For example, a common 'mistake' is when traffic from a full-duplex 1 Gbit/sec port is mirrored to a 1 Gbit/sec monitoring port. Since the original traffic can be 1 Gbit/sec for each direction, the total amount of data the mirror port may have to forward is 2 Gbit/sec.

When traffic is dropped due to limited bandwidth of the mirror port, the recording device will record incomplete traffic and thus introduce inconsistencies. Avoiding this is relatively easy in theory, by making sure the bandwidth of the mirror port is sufficient. In practice, however, the costs of high-speed interfaces may prevent adequate dimensioning of the mirror port.

2.2 At the Recording Device

A second way a repository could become inconsistent, is when there are issues at the side of the recording device. A recording device can be a (desktop)computer or server gathering data. Commonly a recording device is connected to a router or switch using port mirroring. When it receives more packets than it can handle, the device will start dropping packets.

A study by Deri in 2004 showed that packet capture with standard (libpcap) software may result in heavy packet loss [7]. For example, Windows 2000 showed a packet loss of 32% and Linux 2.4 even over 99%. The problem, in general, are interrupts, since packet handling is performed by the kernel. For 100 Mbit/sec Ethernet cards, Linux 2.4 raised an interrupt for each received packet, putting a heavy load on the system. To overcome this problem, Deri proposed a so-called *ring buffer*, which is now part of Linux 2.6. This ring buffer contains a number of packet descriptors. In the initial state, all packet descriptors are marked as 'ready'. When a packet arrives at the network interface card (NIC), it is copied into a packet descriptor marked as ready, after which the descriptor is marked as 'used'. Instead of raising an interrupt for each received packet, an interrupt is only raised when the buffer contains a certain number of packets (or after a time-out). As a result, packet capturing performance is improved dramatically [8], since the lower number of interrupts lowers the load on the CPU, leading to less packets being dropped.

It should be noted that modern, 1 Gbit/sec Ethernet cards, have already implemented similar rings in hardware. Still new hardware developments, like for example multi-core CPUs, raise new problems. Again, software modifications, like multiple receive queues, may solve potential problems [9].

3 Detecting Inconsistency

In this Section, we will describe inconsistency called fake gaps and introduce an algorithm for detecting inconsistency. Although there may be different ways a repository can be inconsistent, we will only consider fake gaps.

3.1 Fake Gaps

In this paper’s introduction, we said a fake gap to be representing those parts of a TCP flow that are absent in the repository, although they were acknowledged at the TCP level. To explain this more precisely we first have to consider a gap. To start with an example: when Alice wants to send some data to Bob, she sends ordered packets named A B C D E F. Bob may receive this as A B C E F **D**, but knows about the correct order and can therefore recreate the original message. This is called packet reordering. At the side of Bob, after having received C, there is a gap until D is received. In this example, the gap is filled when D is received.

We call a gap a fake gap, when one or more packets in a sequence of packets are not present, but are also not retransmitted; hence the original TCP flow is not affected. Consider Alice and Bob: Alice sends the sequence A B C D E F to Bob using TCP. Bob’s network administrator records all traffic sent to Bob. According to the data recorded, Bob received A B C E F. When the connection between Alice and Bob is closed, we know Bob must have received D. We can then say that the recorded TCP flow between Alice and Bob contains a fake gap. All data was transmitted and received correctly, but the recorded data does not reflect this. Hence the recorded data is inconsistent. From now on, if we are referring to the flow of data packets within a TCP connection, we will abbreviate it to a flow.

In recorded traffic data, it is likely that there are flows that start and/or end outside the recorded time period. Therefore, detecting fake gaps in these flows is difficult. To avoid this, we only take flows into account that are sufficiently recorded. All packets of a single flow, from the first SYN-packet up to a FIN- or RST-packet should be recorded. If this is the case, we call the TCP flow a *usable flow*. Note that the final FIN-handshake can be partly outside the recorded time period.

3.2 Algorithm

In Section 3.1 we concluded fake gaps prove inconsistency. To detect this the algorithm we introduce, listed as Fig. 1, first extracts all usable flows from the complete set of packets. The second part of the algorithm loops over all usable flows. For every usable flow, it checks if there is an acknowledging packet that acknowledges a packet that has not yet been seen. If so, this indicates a fake gap and an identifier of the flow together with the packet that is used to detect the fake gap, is added to a list. So the final result of this algorithm is a list of flows combined with packets directly after the fake gaps.

```
INIT usableFlows to {}           # all usable flows found
INIT testFlows to {}            # all flows found so far
FOR each packet in data file
  IF packet is SYN
    CREATE flow from packet
    ADD flow to testFlows
  ELSE IF packet belongs to flow in testFlows
    SET flow to flowOf(packet)
    ADD packet to flow
    IF packet is FIN or RST
      REMOVE flow from testFlows
      ADD flow to usableFlows
END FOR
INIT fakeGaps to {}
FOR each flow in usableFlows
  FOR each packet in flow
    IF packet is ACK
      IF acked packets are not in this flow
        ADD tuple (flow, packet) to fakeGaps
  END FOR
END FOR
```

Fig. 1. Pseudo-code of the fake gap detection algorithm

It is important to see that this algorithm alone cannot detect the exact amount of fake gaps within the traffic dump. It can give an indication and show which usable flows are affected. If, for example, during a small interval no packets were recorded at the recording device, multiple flows can be affected by this. Our algorithm would detect a fake gap for each affected flow. We do not consider this a limitation. The results of our algorithm should provide a starting point for deep inspection of packets and flows.

3.3 Prototype

To be able to detect inconsistencies in existing repositories, we implemented the algorithm in a prototype.

Our prototype implements the algorithm in Fig. 1, but uses speed and memory optimisations. The prototype can be downloaded at [11].

As the algorithm detects fake gaps once per affected usable flow, the prototype does also. The prototype returns, once finished, a set with affected flows and a list with per fake gap the time this fake gap occurred. This can be used for further analysing of the exact location of fake gaps. Our prototype tries to estimate the location of every fake gap by finding patterns in the detection times. As this is just a prototype, we consider such estimation sufficient.

Furthermore, our prototype tries to ‘fix’ fake gaps found, in order to continue analysing. If a fake gap is detected (e.g. a packet is expected but not seen in the rest of the flow) the prototype will report a fake gap and insert a dummy-packet.

In this way, it can continue to analyse the rest of the flow. If there is one (large) fake gap detected, there could be multiple smaller fake gaps in the data set. Therefore, the amount of detected fake gaps could be lower than the actual number. Also, the number of missing packets could be higher, since two or more packets could be missing due to a single fake gap.

The prototype has a few other limitations, which we will discuss. First of all, TCP sequence numbers are finite. If any sequence number reaches $2^{32} - 1$, it will continue with '0' [12]. The prototype does not deal with this limitation. Instead, it detects this situation, gives the user an error message and ignores the affected flow. During our research, no flows were dropped because of this limitation. As our prototype is using Java as implementation language, one may need to adjust the heap size of the Java virtual machine (JVM) to avoid crashes caused by the inability to allocate memory on the heap.

4 Evaluating the Prototype

Before the prototype that was built can be used to analyse gathered traffic from repositories, it has to be evaluated first. We have taken real life recorded traffic from the Simpleweb repository [6] and extracted a subset of packets that reflects different situations. Packet analysing software was used to manually analyse flows and extract flows from a set of traffic to be used for evaluating the prototype.

The requirements for the prototype are basically that it should find all missing packets caused by fake gaps, in usable flows and not detect false positives. The prototype should:

1. not detect any fake gap in a regular TCP flow
2. correctly handle duplicate acknowledgements
3. correctly handle packet reordering
4. correctly handle regular gaps
5. ignore all non TCP packets

We have evaluated the prototype according to these criteria. We extracted seven flows from a data file with traffic and had the prototype process them. The selected flows contained flows with fake gaps, duplicate acknowledgements, lost packets (gaps), partial handshakes and a flow without any of the above.

After having tested prototype on the extracted flows, we tested it on a small data file that was still feasible to manually inspect. We then checked whether our prototype behaved as intended according to the requirements. All files used to test out prototype can be downloaded at [11].

The processing speed of the prototype varies depending on the complexity and size of the data file. For example, it processed 2GB of data in 377 seconds. While processing, the memory usage was constantly around 50MB.

A problem we came across was a crashing JVM. After updating our JVM to the most recent one offered by Sun, most problems were over. However, there are still a few data files where the JVM crashes while running our prototype. As mentioned in section 3.3, extending the heap size of the JVM also cleared out

some crashes, but unfortunately not all. The main issue seems to be that our prototype creates many classes (one for each packet), and that causes the JVM to exit with an error.

5 Analysing Repositories

We analysed two existing repositories using our prototype. The first repository we used was Simpleweb [6], maintained by the University of Twente. It was included in this paper because it is publicly available and easily accessible. We analysed a subset of data covering all 6 locations the Simpleweb repository provides, totalling to 224 data files. The second repository we included in this paper is the WIDE traffic repository [10], maintained by the MAWI working group. It contains data files with traffic of several trans-Pacific lines. We used one data file from samplepoint A to test our prototype and we analysed 27 data files from samplepoint F, which were all over 1 GB in size. We chose samplepoint F since this one is daily updated while the other samplepoints are discontinued or were not accessible. The full table of results created by our prototype can be found at [11]. Our prototype calculates fake gap statistics using various intervals. That is, fake gaps from different flows within this interval are grouped together and reported as a single fake gap. It should be noted that further research should give an indication of the exact value for the intervals we should use.

Figure 2 shows the estimated number of missing packets per repository we tested and the estimated number of fake gaps within an interval of 0.05 seconds. The left part of the graph shows the extreme values. A first observation includes the presence of fake gaps in almost all tested data files from the WIDE repository. To make comparison as clear as possible, note that only a subset of the Simpleweb data files is shown. In fact, only the 28 highest values are plotted; the remaining 196 data files are skipped. It can be observed from the raw test data, that there

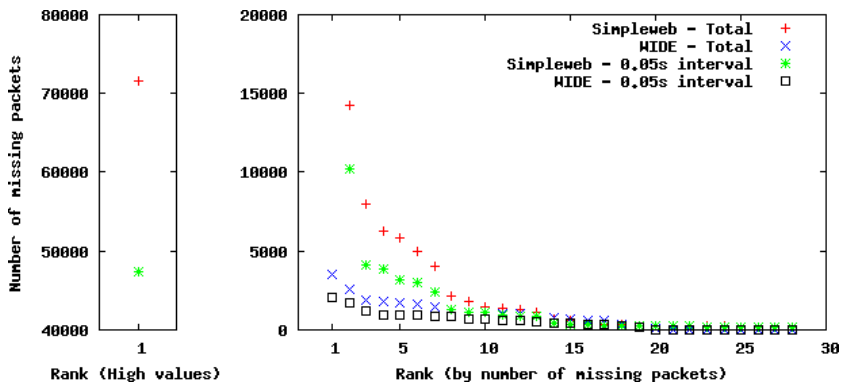


Fig. 2. Estimated number of missing packets and fake gaps. Each plotted value represents a data file.

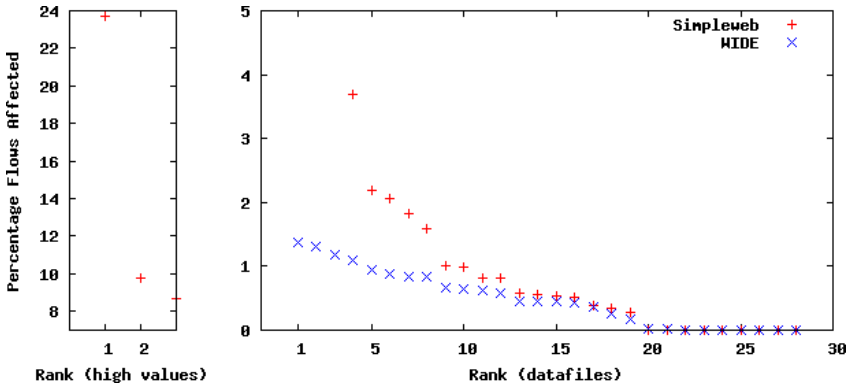


Fig. 3. Percentage of affected usable flows. Each plotted value represents a data file.

is an absence of fake gaps in 50.8% of the tested data files of the Simpleweb repository. Data files from location 2 are almost solely responsible for this. A possible explanation could be the low amount of traffic at this location.

Figure 3 plots the percentage of affected usable flows that have at least one fake gap. For the WIDE repository, on average 0.49% of the usable flows is affected by at least one fake gap. When ignoring data files without fake gaps, the average percentage of affected usable flows is 0.51%. Of the tested data files of the Simpleweb repository, an average of 0.27% of the usable flows in all data files was affected by at least one fake gap. When ignoring consistent files, the average is 0.55%. The highest value is in the file `loc2-20030718-1530`, where 23.72% of the flows is affected by at least one fake gap.

6 Conclusions

This paper describes our research on the consistency of network traffic repositories. Before answering the main research question, we first look at three sub questions identified in Section 1.

The first sub question, “*What could cause inconsistencies in a TCP traffic repository?*”, was answered in Section 2. To minimize the loss of packets that lead to inconsistencies, sufficient bandwidth should be present. At the recording device, customizations to the operating system and hardware could be required to keep up with large amounts of packets arriving.

The second sub question was “*How can we detect fake gaps?*”. We proposed an algorithm in Section 3, which extracts TCP flows. Then, it tries to identify fake gaps, packets that are not recorded by the recording device but were sent. The algorithm checks whether all data in the TCP flow is present by analysing TCP headers.

The last sub question, “*How consistent are today’s repositories?*”, was answered in Section 5. We performed measurements on the Simpleweb [6] and

WIDE [10] repositories. We showed both repositories contain inconsistencies. In the Simpleweb repository an average of 0.27% of the investigated TCP flows was affected by at least one fake gap. For the WIDE repository, this average was 0.49%. The research covered a substantial subset of data from both repositories. We analysed 28 data files from the WIDE repository and 224 data files from the Simpleweb repository.

Going back to the main research question, “*How can inconsistency be detected in a TCP traffic repository?*”, we can now conclude detecting inconsistency is possible by using the proposed algorithm, which detects fake gaps. The knowledge that a repository is not always consistent is very important for research where it is critical to have all data recorded, like research on packet loss. For this kind of research, it is recommended to take possible inconsistency in the repository into account and, if no statistics are present, analyse the repository data before using it.

Future research could include extending the proposed algorithm to support TCP flows that are not completely present in the data file. This research can be used together with algorithms like the one described in [2], which checks for anomalies in traffic rate, to find the exact locations of fake gaps. This can, in turn, be used to draw conclusions about non-TCP traffic, thereby getting a better overview of the consistency of a network traffic repository.

Acknowledgments. This research work has been supported by the EC IST-EMANICS Network of Excellence (#26854).

References

1. Timmer, M.: How to identify the speed limiting factor of a TCP flow, http://dacs.ewi.utwente.nl/assignments/completed/bachelor/reports/B-assignment_Timmer.pdf (retrieved at October 5, 2008)
2. Slomp, G.: Consistency of repositories. Presented at 8th TSConIT, <http://referaat.cs.utwente.nl/new/paper.php?paperID=377> (retrieved at October 5, 2008)
3. Lastdrager, E.E.H.: Consistency of network traffic repositories - an overview. In: Proceedings of 3rd Conference on Autonomous Infrastructure, Management and Security, AIMS 2009 (2009)
4. Lastdrager, E.E.H.: Consistency analysis of network traffic repositories. Presented at 10th TSConIT, <http://referaat.cs.utwente.nl/new/paper.php?paperID=464> (retrieved at February 20, 2009)
5. Wessels, D., Fomenkov, M.: Wow, that’s a lot of packets. In: Proc. Passive and Active Measurements Workshop, PAM (2003)
6. van de Meent, R., Pras, A.: Simpleweb/University of Twente – Traffic Measurement Data Repository, <http://traces.simpleweb.org> (retrieved on October 5, 2008)
7. Deri, L.: Improving Passive Packet Capture: Beyond Device Polling. In: Proceedings of 4th International System Administration and Network Engineering Conference, SANE (October 2004)
8. Wu, W., Crawford, M., Bowden, M.: The performance analysis of linux networking - Packet receiving. *Computer Communications* 30(5), 1044–1057 (2007)

9. Deri, L.: Towards 10 Gbit NetFlow Monitoring Using Commodity Hardware. Presentation at the joint Emanics / IRTF-NMRG workshop on NetFlow/IPFIX for network management,
<http://www.ibr.cs.tu-bs.de/projects/nmrg/meetings/2008/munich/>
(retrieved on March 1, 2009)
10. Cho, K., Mitsuya, K., Kato, A.: Traffic data repository at the WIDE project. In: Proc. USENIX Annual Technical Conference, p. 51 (2000)
11. Lastdrager, E.E.H.: Prototype and results,
<http://www.vf.utwente.nl/~lastdragereeh/referaat>
12. Postel, J.: RFC 793: Transmission Control Protocol, Internet Engineering Task Force (1981)