The Consistency Verification of Zebra BGP Data Collection

Hongwei Kong

1 Introduction

In spite of the wide deployment of BGP as the inter-domain protocol, the research on the BGP behavior and its influence on Internet performance is still undergoing. Due to the complexity of Internet, it is hard to model and analyze the behavior of BGP. To help understand the behavior of BGP and to help to monitor the performance of Internet better, several BGP looking glasses have been set up to provide useful BGP data collection. For instance, RIPE and Routeviews are two well-known looking glasses (refer to [13][15] for more details on what is a looking glass and how to retrieve BGP data from these looking glasses). Many of these looking glasses have been using Zebra [1] as BGP data collector. These sites collect both BGP update messages and instant routing tables. BGP updates and instant routing tables are generally written into binary files periodically. When BGP messages are dumped into the file, they are dumped according to some predefined packet dump format, which generally includes the information fields like time stamp, message length, from and to IP addresses, from and to AS (Autonomous System) numbers, message type, etc (refer to Appendix.2 for more details). These fields are provided both for the manipulating of dumped messages as well as analyzing BGP behavior. Since binary file is not very convenient and intuitive for analyzing, these binary files are generally decoded into readable ASCII format BGP messages for further analysis. There are several tools able to do such kind decoding. Among them, Route_BtoA coming along with MRTd [2] toolsets and Libbgpdump [11] on RIPE NCC, are two widely used tools for decoding dumped BGP data.

Currently, much BGP research is based on the BGP data collections on public looking glasses. Here we only list some of them as examples: [16][17][18][19]. Ensuring the consistency and reliability of these data collections is very important to guarantee the accuracy of these research results. It has been reported in [3] that some Zebra BGP data collections reported different number of BGP update messages when processed with Route_BtoA on Linux and Solaris platform separately. It's also mentioned that some BGP messages have their message body missed when decoded with Route_BtoA. This arouses suspicions on the reliability of both Zebra data collection and the processing tools, for example Route_BtoA and Libbgpdump. Unfortunately, besides [3], little work has been done to give a thorough exploration on the consistency of Zebra data collection and the related tools. Thus one main objective of this report is to verify the consistency of Zebra BGP data collections. To achieve this, we take an approach to verify the consistency by comparing the data captured and processed with different tools. In this way, we can not only find out whether Zebra data collection is consistent with the real BGP data, but also verify the consistency of the observing results obtained by different tools. It has been noticed that some of the related tools, such as Route_BtoA, behaves differently on Linux from on Solaris. Therefore another purpose of this report is to verify the behavior of these tools on different platforms.

Verifying the consistency of Zebra BGP data collection actually includes two different aspects: one is to verify that the related tools, which are used to process Zebra BGP data collection, behave as expected, and the other is to verify that the results obtained with different tools agree with each other in all cases (in some cases, some discrepancies are allowed provided these discrepancies are rational under the specific conditions). We have done a lot of tests to check whether the tools behave as expected. During these tests, some bugs of the tools have been found and solved, but there are still some problems not fixed at present. After finding the reasons that cause these problems and being sure that they won't compromise the reliability of the verification results, we isolate those unfixed problems so that the tools behave as expected.

Two different verification approaches can be adopted to verify the consistency of Zebra data collections. One way is to program the BGP speaker to send specific patterns and compare it with the data dumped with Zebra. The other is to compare the on-wire captured data with that from Zebra. The former approach is especially useful to pinpoint the bugs of related tools, and the latter approach has a better adaptability since it can be used with both the synthetic BGP source and the real BGP source. In this report both approaches have been adopted and the results show that these approaches are very effective.

The arrangement of the following sections is: Firstly, the verification methodologies and test-beds are presented. Then short descriptions on the related tools in the verification are given. Next, the experimental results together with the detected problems of the tools used in the test are presented. After the analysis of the experimental results, we draw some conclusions of the work in this report.

2 Verification Approaches and Tools involved

2.1 Verification Approaches

In this report, two different test-beds are used. Their configurations are shown in Fig.1 and Fig.2, respectively. The test-bed in Fig.1 is for tests on Linux platform while the other in Fig.2 is to test the behaviors of the tools on Solaris platform.

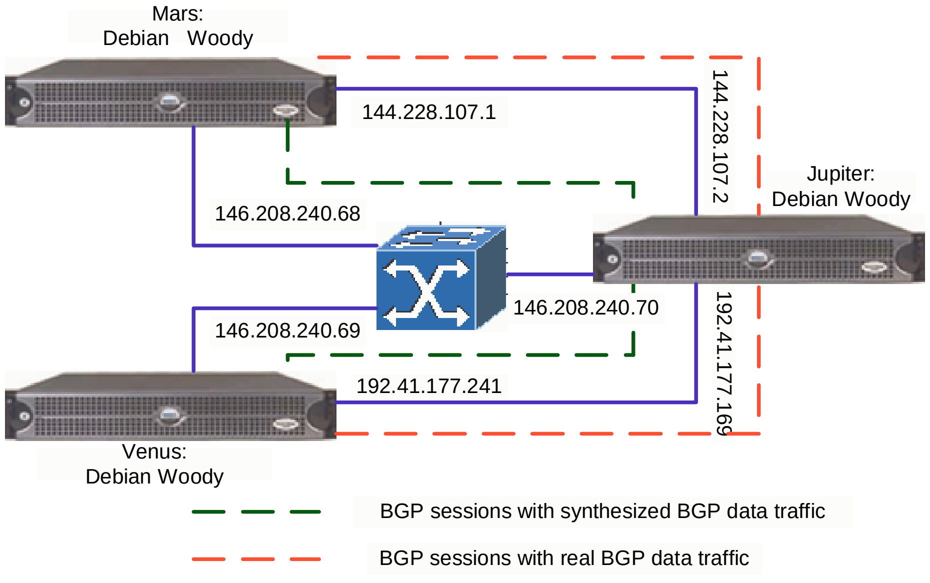


Fig.1 Test-bed on Linux platform

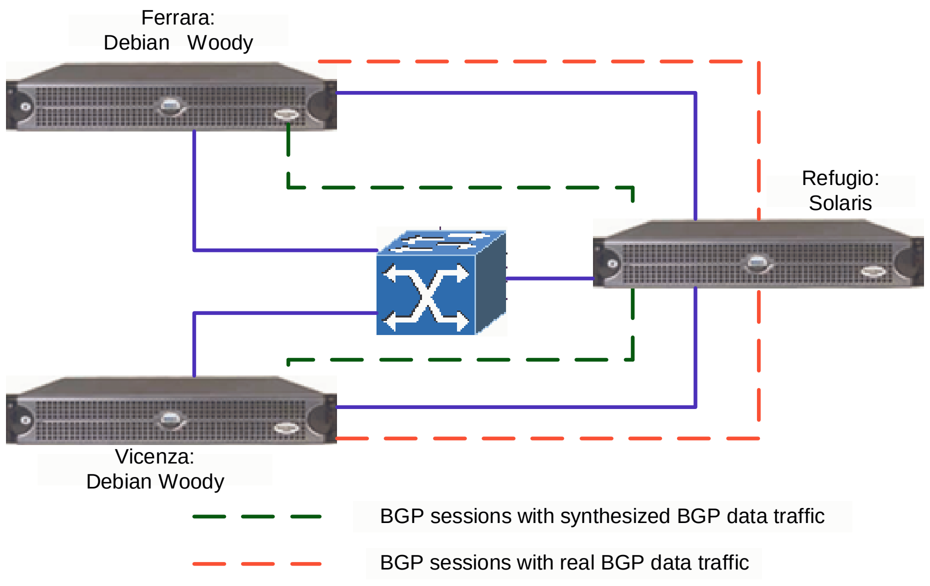

Ferrara:
Debian Woody
Refugio:
Solaris
Vicenza:
Debian Woody
BGP sessions with synthesized BGP data traffic
BGP sessions with real BGP data traffic

Fig.2 Test-bed to test tools on Solaris platform

The verification methods as well as the tools used in the tests are further illustrated in Fig.3. As shown in Fig.1, Fig.2 and Fig.3, both test-beds are very simple and similar. They only consist of a BGP source, a BGP router, and a sink of BGP messages. In the tests, BGPsim and SBGP are used to generate and replay BGP source data, and Zebra acts both as BGP router and BGP collector. As shown in Fig.3, there are 4 different comparisons. The purposes of these comparisons are explained in the following: 1) Comparison C1. It represents the comparison between the on-wire captured data and Zebra dumped data. This approach is mainly used to verify the consistency of Zebra dumped data through on-wire captured data. 2) Comparison C2. It represents the comparison between the source data and Zebra dumped data. The main purpose of comparison C2 is to find the problems of Zebra and the related tools. When doing such test, specific source patterns are used to ease the pinpointing of the problems. For example, in the tests, the announcements consist of continuous, monotonically increasing prefixes like 124.0.1.0/24, 124.0.2.0/24, 124.0.3.0/24 … X.X.X.0/24. Thus if Zebra loses some BGP messages, it can be easily found out when these BGP messages are lost by exploring the pattern of the received prefixes. If further using KEEPALIVE messages as the “synchronization marker” between the source data and Zebra dumped data, then we can easily find out whether there are KEEPALIVE messages lost by check whether there are some “out of synchronization” intervals. 3) Comparison C3. It represents the comparison between the source data and on-wire captured data. The same source patterns mentioned above are used in the tests. The major purpose of C3 is to find the problems of on-wire captured tools. 4) Comparison C4. It is to test whether Ethereal and Tcpdump behave the same and how different versions of Libpcap influence the capture results.

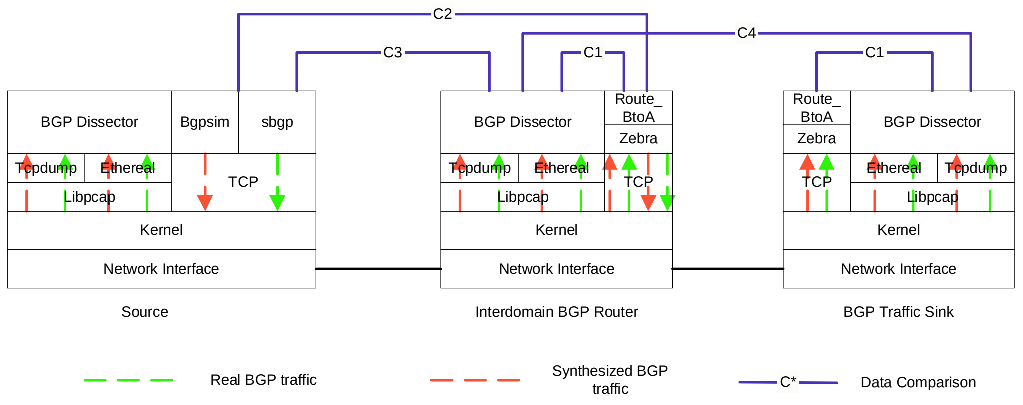


Fig.3 Illustration of verification methods

From Fig.3, it can be seen that all these comparisons require the comparison of the data files obtained from different tools. However, the binary source data file, Zebra dumped data and the on-wire captured data are usually in different formats, and BGP messages in these files have different packet encapsulations. Directly comparing these files is inconvenient and inefficient. These binary files are usually converted to ASCII format, and the BGP messages are decoded into a readable format and then are compared to verify the consistency of Zebra dumped BGP data. In this report, to avoid the time complexity of comparing BGP message bodies, the number of announcements and the number of withdrawals are compared instead. The question here is how to choose the appropriate comparison intervals. Specifying the time intervals and comparing the number of announcements and withdrawals in those intervals won't work in most of the cases. This is because the processing delay of BGP messages in Zebra may make the set of BGP messages in Zebra dumped file in the specified interval differ from that in the source data file and that in the on-wire captured data file in the same time interval. Since the processing delay of BGP messages in Zebra is unpredictable and varying with respect to the traffic load, it is hard to compensate with a fixed delay. To solve this problem, there should be some "synchronization" mechanism between the two data files to be compared. It is observed that ideally, the number of BGP messages and the order of these messages between two successive KEEPALIVE messages of one BGP session do not change when these messages arrive at the Zebra BGP message queue of the peer. Thus in this report, KEEPALIVE messages are used as "synchronization markers". Any two successive KEEPALIVE messages determine a comparison interval. By aligning these KEEPALIVE messages in the two files to be compared, the numbers of announcements and withdrawals in the corresponding interval can be compared. The rationale is very simple: suppose that the data dumped with Zebra is compared with on-wire capture data, and if on-wire captured data is guaranteed to be consistent with the source data, and the number of announcements and withdrawals in any comparison interval are the same for the two files, then based on the errorless transmission of TCP, we can conclude that the BGP data dumped with Zebra is consistent with on-wire captured data and thus consistent with the source data. In this report, we tried to use large BGP routing tables (both synthesized and real BGP data) to test whether Zebra dumped data collection is consistent.

2.2 Related tools

One purpose of this report is to have a thorough exploration of the tools used by verifying the consistency of Zebra BGP data. Short descriptions of the tools used in the tests together with their version information are given in this section.

1) MRTd (Ver. 2.2.2a). MRTd (Multi-threaded Routing Toolkit) is widely used in the field of network performance measurement and BGP data processing. It includes the following tools: MRTd daemon, BGPsim, SBGP, Route_AtoB and Route_BtoA. Except MRTd, the other four tools are all used in the tests. MRTd acts as a routing daemon, which supports RIPng, RIP1/2, multiple RIBs (route server) and BGP4+. However, in our tests, Zebra is used as the routing daemon instead of MRTd daemon. The tool BGPsim is a BGP4 traffic generator and simulator. It is used to generate BGP data with specific prefix patterns. The tool SBGP is a very simple implementation of BGP speaker and listener. It sends out the BGP messages in the data file and receives the BGP messages from the peer. However, it doesn't implement any routing policy and doesn't change the kernel's routing table. SBGP only supports the data file in MRTd format. It doesn't support replaying the data file in Zebra format. Route_BtoA and Route_AtoB are tools used to convert binary BGP messages to ASCII format and vice versa. They also support data files in Zebra format. In [3], it's reported that some BGP messages lose their message bodies and the missing of BGP message bodies was attributed to the inability of processing IPv6 (IP Version 6) Messages. However, in our tests, we found that the conversion errors come not only from Route_BtoA but also from Zebra data file. A more thorough summary on the problems and the reasons for these problems are given later.

2) Zebra (Ver. 0.93b). Zebra is a routing software package that supports a lot of routing protocols including RIP1/2, RIPng, OSPFv2, OSPFv3 and BGP. It is widely used in the field of inter-domain routing. Its current BGP implementation BGPd supports RFC 1771 (A Border Gateway Protocol 4 (BGP-4)) [4] and RFC 2858 (Multiple Protocol Extensions for BGP-4) [5]. Many looking glasses have been using Zebra as the BGP collector. A lot of research results are based on the Zebra BGP data collections from those public looking glasses. Therefore verifying the consistency of Zebra BGP data collections is very important for ensuring the reliability of these research results. In the tests, it was found that inconsistency did exist for some BGP data collections on these looking glasses, and the inconsistency comes not only from the bugs of Zebra but also from the bugs of other related tools. These problems will be presented in the next section. We also found that after fixing the bugs of the tools, Zebra data collections were consistent when there is no session break. That is, it agrees with the data collected by other methods, for example, the on-wire captured data.

3) Ethereal [6] (Ver. 0.9.11 and above) and Tcpdump [7] (Ver. 3.7.2). Both Ethereal and Tcpdump can use the packet capture library to capture on-wire data. However, one main difference between Ethereal and Tcpdump is that Ethereal provides more and stronger protocol analyzing features to analyze the captured data than Tcpdump. Specially, Ethereal provides a full-featured BGP dissector to decode the captured BGP packets into human readable ASCII format. Both Ethereal and Tcpdump use the same file format. Thus in the tests, Ethereal's BGP dissector is used to decode data files captured by Tcpdump. For the capture file format, refer to Appendix.3.

4) Libpcap [7-8] (Ver. 0.7.2). Libpcap is a packet capture library, which captures the packets on

the specified interface into the kernel, and provides APIs (Application Programming Interface) for other applications to access these captured packets. It also provides filtering mechanisms to capture only those packets that the filtering rules are matched. Phil Wood provided a patch to the Libpcap library on Linux to improve its performance. That patch utilized the configuration option CONFIG_PACKET_MMAP of the current Linux kernel (above Version 2.2.x). By using the shared memory ring implementation in MMAP mode (Memory Mapping, which provides IO mechanisms for the applications to access the device memory directly. Refer to [9] for more detail), this new patch of Libpcap can allocate a queue of as many as 32768 frames for the interface device to directly write the captured packets into. This greatly reduces the possibility of capture loss due to buffer overflow and due to busy system. Besides the ring buffer, the new patch can also provide better live statistics about the number of drops, receiving errors, transferring errors, etc. refer to [9] for more details. The version of Libpcap patch used in our test is Ver. 0.8.030314.

5) NIST NET [10] (Ver. 2.0.12). NIST NET is a network emulator that runs on Linux. It allows a single Linux PC set up as a router to emulate a wide variety of network conditions. In the tests, it's mainly used to generate TCP segment losses to test the behavior of Ethereal BGP dissector.

6) Libbgpdump [11] (Ver. 1.1). Besides Route_BtoA, Libbgpdump is another tool, which was provided to convert binary BGP messages in Zebra format to ASCII messages. In the tests, its results are compared with Route_BtoA, and the comparison results will be presented in the next section.

3 Experiment results

In above sections, the short introductions on the verifying approaches and the related tools are given. In this section, the test results are presented.

3.1 Statistics on public Zebra BGP data collection

It's reported in [3] that due to the issues of Route_BtoA, some BGP messages were not decoded correctly on Linux platform. Before verifying the consistency of Zebra BGP data collections, the composition of incorrectly decoded BGP messages was firstly analyzed to find out the problems of the related tools. According to the analysis results, those incorrectly decoded BGP messages are classified into several groups. An overview on the binary dump packet format of Zebra will help to understand what have caused these incorrectly decoded BGP messages. The dump packet header of BGP messages is show in Fig.4

0											1									2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Time																															
Type																Subtype															
Length																															
Source AS Number																Destination AS Number															
Interface Index																Address Family															

Source IP Address
Destination IP Address

Fig.4 Binary dump packet header of BGP message

1) Unsupported attributes. The analysis results show that the BGP messages that contain Multiprotocol Reachable NLRI (MP_REACH_NLRI) or Multiprotocol Unreachable NLRI attribute (MP_UNREACH_NLRI) [5] will not be decoded correctly on Linux. For those messages containing such attributes, some of them can be decoded correctly on Solaris, and some not. Further exploration shows that if the MP_REACH_NLRI or MP_UNREACH_NLRI attribute is for IPv6 prefixes, then these messages can be correctly decoded on Solaris platform but incorrectly decoded on Linux platform, and that if the MP_REACH_NLRI or MP_UNREACH_NLRI attribute is for the multicast prefixes of IPv4, then they are decoded incorrectly on both Linux platform and Solaris platform. The messages containing Multiprotocol Extension attributes differ from other BGP messages in that they don’t have NLRI field in the message body.

2) NULL AS. When Zebra dumps BGP messages, it will set the source AS number and destination AS number fields as shown in Fig.4 correspondingly. However, it was found that for some dumped messages, the source AS number and destination AS number fields are set to all zeros and there are no source IP address and destination IP address fields for these packets. These messages are decoded as NULL messages when converted to ASCII messages. A further examination on the body of these BGP message shows that these messages are actually OPEN messages.

3) NO IP addresses. Every dumped BGP message should have the same dump header format as shown in Fig.4. However, we found some dumped messages didn’t have the fields of source and destination IP in their dump headers, and that these messages were decoded as NULL message on both Linux and Solaris.

4) Incompletely dumped messages. In the tests, there are still some other incorrectly decoded messages, which don’t belong to the above 3 groups. These messages are all very large BGP messages. Careful examination on these messages shows that these messages are not completely dumped. It’s found that the values of length field in the dump header are the same for all of these large messages (4096 bytes), even if the length of these BGP messages differs. From Fig.4, it can be seen that the value of length field in the dump header should equal to the length of BGP message plus 16 bytes. Thus we suspected that some bugs of Zebra BGP dump program caused this problem. After some source code debugging, the bug was found and fixed. Although this bug compromised the consistency of Zebra BGP data collection, we think they won’t occur often. This is because most of the updates are very short messages. However, when there is burst BGP traffic, the problem of incompletely dumped BGP messages may occur much often. We did find burst occurrence of such incompletely dumped large BGP messages in Zebra BGP data collections.

After classifying the problems of incorrectly decoded BGP messages, we got the statistics of how frequently these problems occur in some Zebra BGP data collections. The Zebra BGP data collections we used are the BGP updates in January, 2003 on rrc0-rrc8 [12] of RIPE NCC [13] and the data collections on Routeview [14] for November and December of 2002. The statistics are given in the table.1.

Sites	# All	# State Change	#NULL_IP		#L	#Multicast BGP	#NULL_AS		#NO_IP	
			Total	#MPA6			Total	#O	Total	#O
rrc00	21773673	1391	0	0	86	0	314	314	0	0
rrc01	9794673	650776	0	0	11	0	0	0	0	0
rrc02	339188	137334	0	0	0	0	983	983	20158	3917
rrc03	19746099	339568	205777	203454	279	0	2156	2156	0	0
rrc04	3201253	129388	0	0	0	0	0	0	95476	2
rrc05	6889976	33616	0	0	54	0	1323	1323	0	0
rrc06	1811243	46088	0	0	0	0	162	162	416881	43
rrc07	6726286	71197	0	0	31	0	0	0	0	0
rrc08	52378607	48173	0	0	4	0	1	1	0	0
RV.11	34394689	0	0	0	276	661184	0	0	0	0
RV.12	38437887	0	0	0	5631	665198	0	0	0	0

Table.1 Statistics on the problems of Zebra BGP data collection on some looking glasses

Where #O represents the number of OPEN messages; #MPA6 represents the number of messages that contain MP_REACH_NLRI/ MP_UNREACH_NLRI attributes for IPv6 prefixes. #MPA4 represents the number of messages that contain MP_REACH_NLRI/ MP_UNREACH_NLRI attributes for IPv4 multicasting prefixes; and #L represents the number of large BGP messages which can't be decoded correctly.

Further explanations on the statistic results are given below:

1) From Table.1, we can see that all messages with zero AS numbers are OPEN messages. We also found that these messages have no source IP address and destination IP address field in the dump header. This indicates that these OPEN messages are not dumped in the format as shown in Fig.4. This causes Route_BtoA to decode these messages as NULL Message. We also noticed that there are correctly dumped OPEN messages, and the OPEN messages along the session between two Zebra BGPds are usually dumped correctly, and that the OPEN messages along the session between BGPsim and Zebra or SBGP and Zebra are usually dumped incorrectly. This should be a bug of Zebra BGPd dump program. Thus one suggestion for improving Zebra is to revise the BGP message dump part and dump OPEN messages in the format as shown in Fig.4. Another suggestion for improving Route_BtoA is to add the support for packets with irregular dump header (It's observed that BGP message bodies are correct for these NULL_AS messages).

2) For those messages without source and destination IP address fields (NO IP), they have non-zero AS numbers and most of them are not OPEN messages. These distinguish them from the NULL AS messages. These messages are found in the BGP data collections of January, 2003 on rrc02, rrc04 and rrc06. However, it is strange that for the month before January, 2003 and the month after January, 2003, we don't find any such messages on these sites. Thus it is not likely that this is merely due to a bug of Zebra (Otherwise, such malformed packets should persist). So we are interested in what has happened on rrc02, rrc04 and rrc06 during January, 2003, and what has resulted in those messages dumped without source and destination IP address fields. Although these messages don't have correct dump headers, they have correct BGP messages bodies. Thus adding the support for such messages in Route_BtoA will also relieve the problem of incorrectly decoded BGP messages.

3) For data collections on Routeviews, it's observed that a lot of BGP UPDATE messages can't

be correctly decoded. The first glance shows that these messages all have no NLRI fields. Our statistics further indicates that all of them contain at least one MP_REACH_NLRI /MP_UNREACH_NLRI attribute and that all of them have correct dump headers. To check whether these messages are correct BGP message, we replay these messages with SBGP. Then on-wire BGP messages are captured with Ethereal. After decoding them with Ethereal BGP dissector, we find that these messages are correct BGP messages, and that they all contain MP_REACH_NLRI/MPUNREACH_NLRI attribute for multicasting IPv4 prefixes. As an example, a decoded BGP message containing MP_REACH_NLRI/MPUNREACH_NLRI attribute for multicasting IPv4 prefixes is shown in Fig.5. These suggest that Route_BtoA should be enhanced to support the MP_REACH_NLRI/MPUNREACH_NLRI attribute for multicasting IPv4 prefixes. Messages containing MP_REACH_NLRI /MPUNREACH_NLRI attribute for Multicast IPv4 prefixes are not found on the looking glasses rrc00 to rrc08.

```
Frame 16 (113 bytes on wire, 113 bytes captured)……
Ethernet II, Src: 00:90:27:a7:86:b2, Dst: 00:90:27:13:a9:d4……
Internet Protocol, Src Addr: 146.208.240.68 (146.208.240.68), Dst Addr: 146.208.240.70 ……
Transmission Control Protocol, Src Port: bgp (179), Dst Port: 32792 (32792), Seq: 49,     ……
Border Gateway Protocol
    UPDATE Message
        Marker: 16 bytes
        Length: 47 bytes
        Type: UPDATE Message (2)
        Unfeasible routes length: 0 bytes
        Total path attribute length: 24 bytes
        Path attributes
            MP_UNREACH_NLRI (24 bytes)
                Flags: 0x90 (Optional, Non-transitive, Complete, Extended Length)
                    1... .... = Optional
                    .0.. .... = Non-transitive
                    ..0. .... = Complete
                    ...1 .... = Extended length
                Type code: MP_UNREACH_NLRI (15)
                Length: 20 bytes
                Address family: IPv4 (1)
                Subsequent address family identifier: Multicast (2)
                Withdrawn routes (17 bytes)
                    130.184.0.0/16
                    144.167.0.0/16
                    150.208.97.0/24
                    150.208.116.0/24
                    159.150.0.0/16
```

Fig.5 Ethereal decoded BGP message containing MP attributes for multicast IPv4 prefixes

Our further exploration on MRTd source reveals that MRTd software package actually has implemented the support for MP_REACH_NLRI or MP_UNREACH_NLRI attribute for the multicast prefixes of IPv4. However, when it's built, the configure script will automatically detect whether the kernel has multicast routing support. If the kernel doesn't support multicast routing, this feature is turned off automatically. From this, it can be seen that the features of Route_BtoA is entangled with the kernel configurations. This is not reasonable and leads to the difference when running on different kernels. Thus we suggest improving Route_BtoA by turning on the support for these attributes.

4) On rrc03, a lot of dumped messages with all-zero source and destination IP addresses are found. These messages are not decoded correctly on Linux platform, but decoded correctly on Solaris platform. A further exploration on the message bodies shows that most of them contains MP_REACH_NLRI/MPUNREACH_NLRI attribute for IPv6 prefixes. For the left messages with all-zero source and destination IP addresses, we find many of them are KEEPALIVE messages. These imply that when Zebra dumps BGP messages for IPv6 messages, it sets the IP fields in the dump header of these messages to zero. The correct decoding of the messages on Solaris, which have MP_REACH_NLRI or MP_UNREACH_NLRI attribute for IPv6 prefixes, shows that the support of MP_REACH_NLRI or MP_UNREACH_NLRI attribute for IPv6 prefixes is also implemented in MRTd package. Further exploration on the source codes of MRTd reveals that when it's built, the configuration script automatically detect whether the kernel has the support for IPv6. If the kernel has no support for IPv6, then the feature that supports the decoding of MP_REACH_NLRI/MPUNREACH_NLRI attribute for IPv6 prefixes is turned off automatically. This has result in the observed different behavior of Route_BtoA when running on Linux platform and Solaris platform respectively. One possible improvement on Route_BtoA to decode these messages on Linux correctly is to turn on the support for IPv6 even when Linux kernel is not configured to support IPv6. These NULL IP messages are only found on rrc03.

5) Incompletely dumped large BGP messages are found on several sites, and the number of incompletely dumped large BGP messages is very small compared with the total number of messages dumped. This is just what we expected. However, this doesn't mean the incompletely dumped large BGP messages have neglectable effect on the consistency of Zebra BGP data. We did find for some data files, the number of incompletely dumped BGP messages is not neglectable compared with the total number of messages dumped. Several examples are given in table.2.

Site	Date	Time	# Total	#L	Percent (%)
Routeview	19/12/2002	20:22-20:37	42663	117	0.27
Routeview	19/12/2002	20:37-20:52	27707	114	0.4
Routeview	19/12/2002	20:52-21:07	28336	20	0.07
Routeview	19/12/2002	21:07-21:22	44393	100	0.23
rrc03	06/01/2003	09:30-09:45	22707	59	0.26
rrc03	31/01/2003	02:30-02:45	28636	103	0.36

Table.2 Some data files that contain incompletely dumped large BGP messages

In table.2, it can be seen that for these data files, the frequency of incompletely dumped large BGP messages is about once every thousand BGP messages. However, taking the number of prefixes in a large BGP message into account, the number of missing prefixed due to incompletely dumped large BGP message may compose a large percent of the total number of updates. To illustrate this, the correctly dumped parts of these incompletely dumped BGP messages are decoded and the numbers of prefixes before and after considering the incompletely dumped messages are compared in table.3. The results show that the number of prefixes lost due to incompletely dumped messages may be very large compared with the total number of prefixes in the same file. These incompletely dumped large BGP messages are suspected to result from some bug of Zebra BGP dump program. Details on the bug and how to fix it are presented later.

File	# Updates		# Updates		Loss Percent (%)		
	# A	# W	#A	#W	A	W	Total
RV-20021219.2022	220543	3370	225174	119284	2%	97.2%	35%
RV-20021219.2037	125850	1821	128343	117669	2%	98.5%	48.1%
RV-20021219.2052	259489	5806	265288	121775	2%	95.2%	31.5%
RV-20021219.2107	129341	1396	131777	19045	2%	92.7%	13.3%
RP-20030106.0930	107730	3278	107730	87850	0%	96.3%	43.2%
RP-20030131.0230	134127	6180	134127	112542	0%	94.5%	43.1%

Table.3 The number of prefixes influenced by incompletely dumped BGP packets.

In this section, the incorrectly decoded BGP messages are classified into different groups. It can be seen that some of these messages are due to the bugs of Zebra, and some of them are due to the problem of Route_BtoA. The reason of some (NO IP) other messages is different from other incorrectly decoded messages, since we observed different behaviors on RRC02 in Dec. 2002, Jan. 2003, Feb. 2003, respectively. The feedback from RIPE NCC is that this problem may be due to the upgrade of Zebra at that time. For all the problems we found, proposed improvements are given to solve them. After all the problems being fixed, there should be no incorrectly decoded BGP messages when using Route_BtoA.

Besides Route_BtoA, there are also other tools for converting binary dump BGP messages to ASCII. Libbgpdump [11] is one of them, for example. We also perform tests to see whether the problems mentioned above exist for Libbgpdump and to see whether Libbgpdump is superior to Route_BtoA. The test results are summarized as follows:

1) For Messages that contain MP_REACH_NLRI/MPUNREACH_NLRI attribute for multicasting IPv4 prefixes, libbgpdump can decode the attributes that it supports, but can't decode MP_REACH_NLRI/MPUNREACH_NLRI attribute for multicasting IPv4 prefixes. An example is given in Fig.6 to illustrate this.

2) For Messages that contain MP_REACH_NLRI/MPUNREACH_NLRI attribute for IPv6 prefixes, libbgpdump can decode the attributes that it supports, but can't decode MP_REACH_NLRI/MPUNREACH_NLRI attribute for IPv6 prefixes. An example is given in Fig.7 to illustrate this.

3) For the messages with NULL AS (OPEN message), libbgpdump doesn't decode and gives no messages.

4) For the messages without source and destination IP fields (NO IP), libbgpdump doesn't decode and gives no messages.

5) For the incompletely dumped large BGP messages, libbgpdump can decode the part which was dumped correctly, and decode the missing part as if they are all zeros. For such messages, Route_BtoA will not decode at all. So libbgpdump is superior to Route_BtoA in this point.

However, we also found that if there are several successive incompletely dumped BGP messages, libbgpdump will exit with a segmentation fault. The decoded message of incompletely dumped BGP message is given in Fig.7 as an example.

```
TIME                    : Fri Nov   1 08:03:33 2002
LENGTH                  : 115
TYPE                    : Zebra BGP
SUBTYPE                 : Zebra BGP Message
        SOURCE_AS       : 2914
        DEST_AS         : 6447
        INTERFACE       : 0
        SOURCE_IP       : 129.250.0.11
        DEST_IP         : 198.32.162.102
MESSAGE TYPE            : Update/Withdraw
WITHDRAW                :
ANNOUNCE                :
ATTRIBUTES              :
      ATTR_LEN          : 76
      ORIGIN            : 0
      ASPATH            : 2914
      NEXT_HOP          : 129.250.0.11
      MED               : 56
      LOCAL_PREF        : N/A
      ATOMIC_AGREG : N/A
      AGGREGATOR        : N/A
      COMMUNITIES   :   2914:410 2914:2000 2914:3000
```

Fig.6 Decoded message, which contains MP_REACH_NLRI/MPUNREACH_NLRI attribute for multicasting IPv4 prefixes

```
TIME                    : Tue Jan 28 03:46:26 2003
LENGTH                  : 100
TYPE                    : Zebra BGP
SUBTYPE                 : Zebra BGP Message
        SOURCE_AS       : 12859
        DEST_AS         : 12654
        INTERFACE       : 0
        SOURCE_IP       : 0.0.0.0
        DEST_IP         : 0.0.0.0
MESSAGE TYPE            : Update/Withdraw
WITHDRAW                :
ANNOUNCE                :
ATTRIBUTES              :
      ATTR_LEN          : 61
      ORIGIN            : 0
      ASPATH            : 12859 3265 2914 4685
      NEXT_HOP          : N/A
      MED               : 1
      LOCAL_PREF        : N/A
      ATOMIC_AGREG : N/A
      AGGREGATOR        : N/A
      COMMUNITIES   :   3265:4001
```

Fig.7 Decoded message, which contains MP_REACH_NLRI/MPUNREACH_NLRI attribute for IPv6 prefixes

```
TIME                    : Fri Dec 20 05:23:07 2002
LENGTH                  : 4096
TYPE                    : Zebra BGP
SUBTYPE                 : Zebra BGP Message
        SOURCE_AS       : 16150
        DEST_AS         : 6447
        INTERFACE       : 0
        SOURCE_IP       : 217.75.96.60
        DEST_IP           : 198.32.162.102
MESSAGE TYPE        : Update/Withdraw
WITHDRAW                :
ANNOUNCE                :
            17.255.240.0/23
            24.121.16.0/23
            24.121.18.0/23
            24.121.20.0/23
              .
              .
              .
            204.44.208.0/20
            204.52.242.0/24
            204.0.0.0/18         Not completely dumped part
            0.0.0.0/0             Use Zero to stuff
            0.0.0.0/0
            0.0.0.0/0
ATTRIBUTES             :
      ATTR_LEN          : 39
      ORIGIN             : 2
      ASPATH             : 16150 20757 1239 7018
      NEXT_HOP          : 217.75.96.60
      MED                 : N/A
      LOCAL_PREF       : N/A
      ATOMIC_AGREG : N/A
      AGGREGATOR       : N/A
      COMMUNITIES    :    16150:65303 16150:65304 16150:65321
```
Fig.8 Decoded message, which is incompletely dumped large BGP message

3.2 Inconsistency between Zebra BGP data collection and on-wire captured data

The statistic results on Zebra BGP data collections help us understand what has caused the problem of incorrectly decoded messages and how they influence the consistency of Zebra BGP data collection. We can see in the above section that all except the incompletely dumped BGP messages can be correctly decoded as long as some improvements are made on Route_BtoA since the bodies of these BGP messages are correct. For those incompletely dumped BGP messages, to ensure a correct decoding, the bug of Zebra must be fixed. To give an example on how serious this problem may be, we did tests using synthesized BGP data as the source. The content of BGPsim configuration file is given below. The test-bed is shown in Fig.1. On-wire captured data is compared with that dumped with Zebra on the same Linux machine, Jupiter. The numbers of announcements and withdrawals in corresponding comparison intervals are compared. The comparison results are given in Fig.9 and Fig.10, respectively.

```
network-list 1
      range 124.1.1.0/24 126.128.1.0
      stability    40
      change       40
      map    1    2
route-map 1
        set origin IGP
        set as-path 182 23 23 15
        set next-hop 146.208.240.68
route-map 2
        set origin IGP
        set as-path 1185 112 10
        set next-hop 146.208.240.68
```

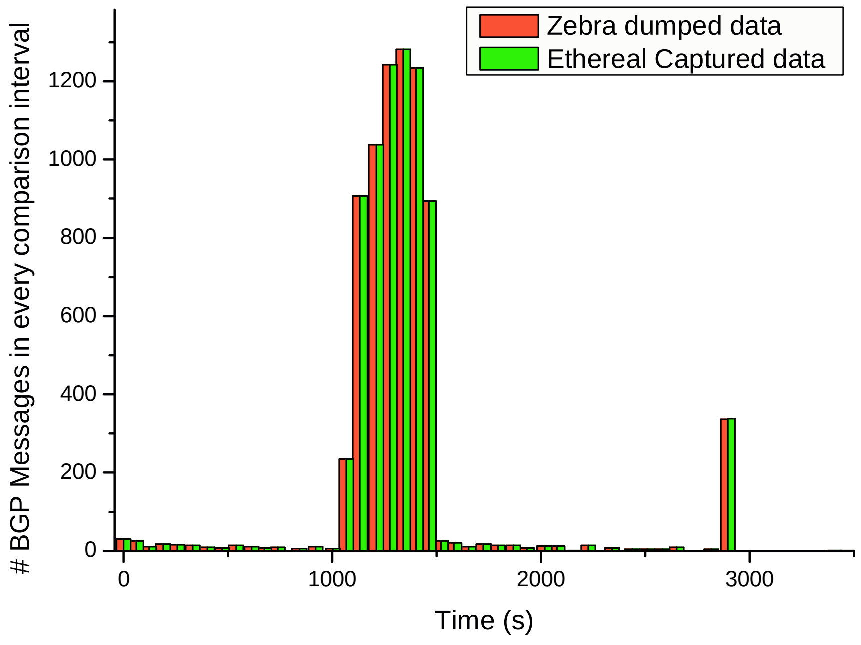


Fig.9 The numbers of BGP messages in every comparison interval

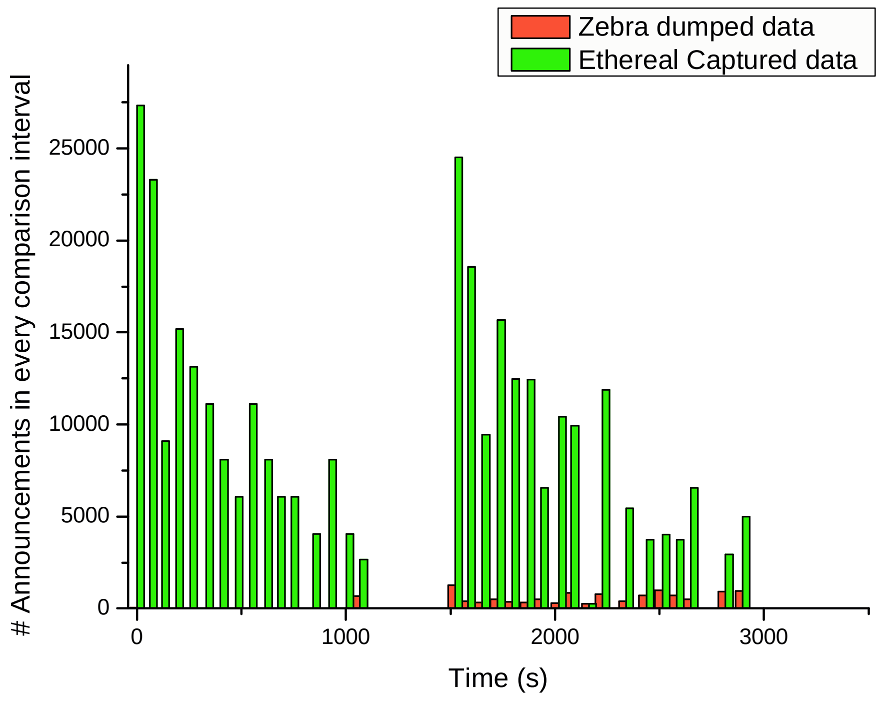


Fig.10 The numbers of announcements in every comparison interval

In Fig.9 and Fig.10, it can be seen that the number BGP messages in the corresponding comparison intervals is the same for both on-wire captured data and Zebra dumped data. However, the number of announcements in the corresponding comparison interval differs a lot. Further exploration shows that this difference is due to the incompletely dumped BGP messages. To solve this problem, after debugging BGP dump program of Zebra, we located the bug and fixed it.

Incompletely dumped BGP messages are mainly due to the limited size of dump buffer. When initializing, BGP dump module allocates a buffer with fixed space and uses it to hold the dumped messages before writing them into the data file. The code is shown below:

```
Bgp_dump_obuf = stream_new (BGP_MAX_PACKET_SIZE + BGP_DUMP_HEADER_SIZE);
/*
        BGP_MAX_PACKET_SIZE=4096
        BGP_DUMP_HEADER_SIZE=12
*/
```

However, from Fig.4, we can see that the dump header includes not only the common header, which occupies 12 bytes, but also the dump message header, whose length is different for different message type and subtype. Taking BGP update messages for an instance, the dump message header will include source and destination as numbers, Interface Index, Address Family, and IP addresses. This part alone will occupy 16 bytes. However, it can be seen the above code doesn't take the size of this part of dump header. Thus if the BGP message is a very large BGP message, the buffer will overflow and result in incompletely dumped BGP message. To solve this problem, allocating a larger buffer is adequate. Taking IPv6 into account, additional 40 bytes are allocated to the dump buffer and the code turns to be the following:

```
bgp_dump_obuf = stream_new (BGP_MAX_PACKET_SIZE + BGP_DUMP_MSG_HEADER+
BGP_DUMP_HEADER_SIZE);
/*
        BGP_MAX_PACKET_SIZE=4096
        BGP_DUMP_HEADER_SIZE=12
        BGP_DUMP_MSG_HEADER=40
*/
```

After fixing this bug of Zebra BGP, we did similar tests to check whether Zebra BGP data collection agrees with on-wire captured data. Both synthesized data and real BGP data from IPMA [20] are used. The comparison results with synthesized data are given in Fig.11. In Fig.11, it can be seen that the problem of incompletely dumped BGP messages is solved and that the counts in many of the comparison intervals are equal. However, there are some comparison intervals, in which although the counts are very close, the counts of Zebra dumped data are a little bigger than those of Ethereal captured data. Since the BGP source is configured to send specific prefix patterns, an examination on the prefix patterns of Ethereal captured data reveals that there are some prefixes lost. We further found that the losses appear in a rather periodic way, and that every time there are prefixes lost, it seems that the prefixes in one BGP messages are lost completely. It's also noticed in the test that one TCP segment usually contains tens of such short BGP messages. Thus such kind of prefix losses is not possible due to TCP segments. It is more possible that such prefix losses are due to bugs of BGP dissector.

To check whether such prefix losses occur for real BGP data, real BGP data (real BGP data [20] in January, 2000, on Mae-east) are replayed and on-wire captured data is compared with Zebra dumped data. The result is shown in Fig.11. In Fig.11, it can be seen that when replaying real BGP data, the counts of Ethereal captured data differ from those of Zebra dumped data for the same comparison interval due to prefix losses.

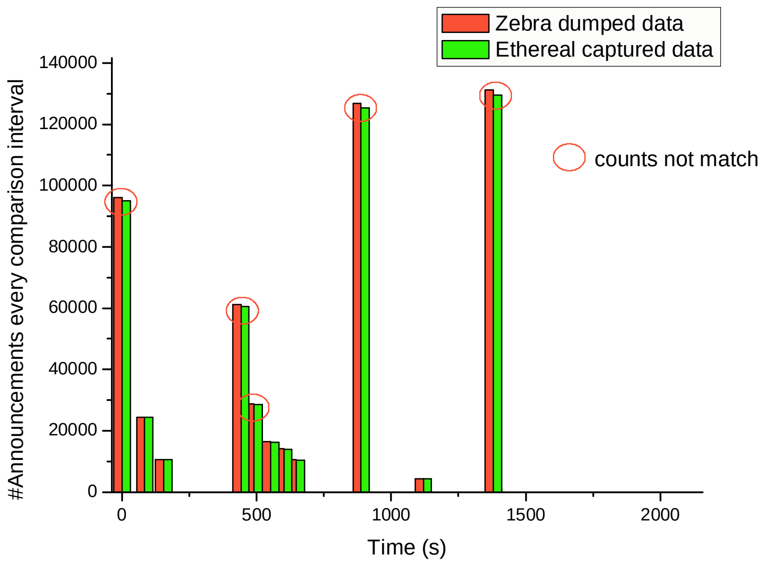


Fig.11 Comparison results when using synthesized BGP data

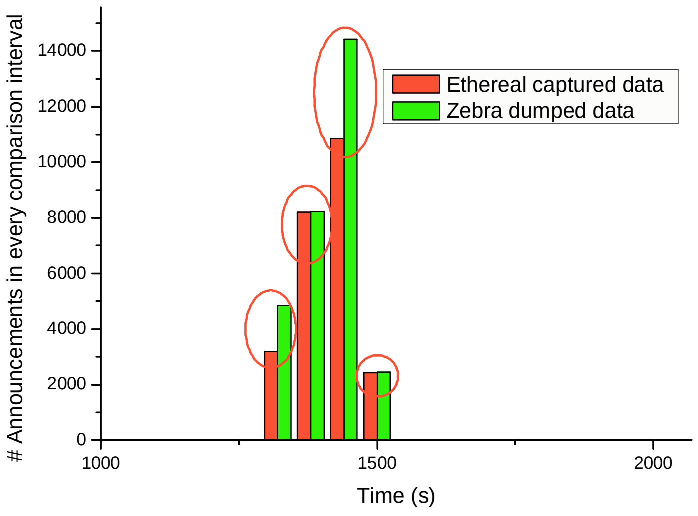


Fig.12 Comparison when using real BGP data from Mae-east

By debugging Ethereal BGP dissector code, we find that the bug is due to incorrect processing of BGP messages whose message head is over the edge of TCP segments. Ethereal BGP dissector works in the following way: BGP data in every TCP segment is cached according to the sequence they are captured. Then BGP dissector searches for the BGP message marker to delimit the BGP messages and passes the BGP messages to BGP message decoder. The codes, which cause the problem, are shown in Fig.13. When the BGP message header is over the edge of TCP segments, that message won't be processed since the message is not complete. However, the message header is not kept for later decoding of this message. This causes the message missed in the decoded ASCII file.

```
l = tvb_reported_length(tvb); /*l is the length of BGP PDU in TCP segments*/
```

```
    i = 0;
    found = -1;
/* run through the TCP packet looking for BGP headers                    */
        while (i + BGP_HEADER_SIZE <= l) {
         tvb_memcpy(tvb, bgp_marker, i, BGP_MARKER_SIZE);
         bgp_len = tvb_get_ntohs(tvb, i + BGP_MARKER_SIZE);
         bgp_type = tvb_get_guint8(tvb, i + BGP_MARKER_SIZE + 2);
         /* look for bgp header */
         if (memcmp(bgp_marker, marker, sizeof(marker)) != 0) {
                i++;
                continue;
         }
         found++;
         ……
         …….
  }/*end of while*/
}/*end of function dissect_bgp*/
```

Fig.13 The bug of BGP dissector

We reported this bug to the mail-list of Ethereal and submitted an incomplete patch for this bug. After our report, the new version of Ethereal (Ver. 0.9.12) has fixed this bug with a patch.

After fixing the bug of BGP dissector, we do the comparisons again and the results are given in Fig.14 and Fig.15, respectively.

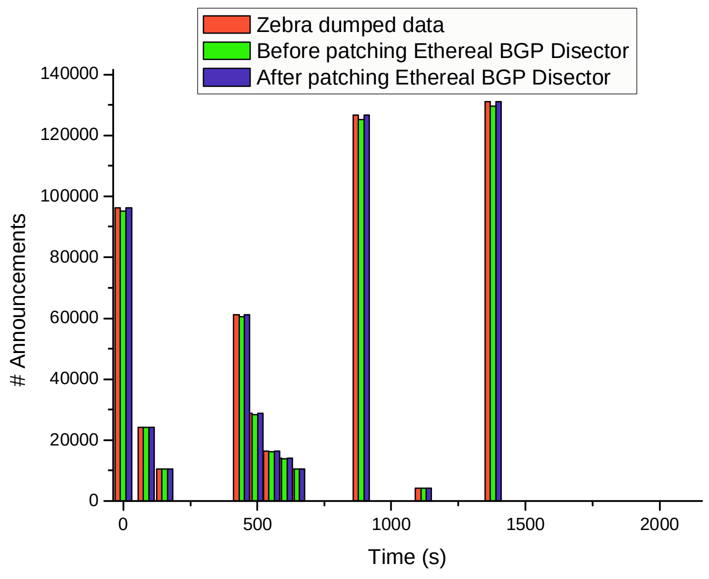


Fig.14 Comparison result before and after patching the bug of Ethereal

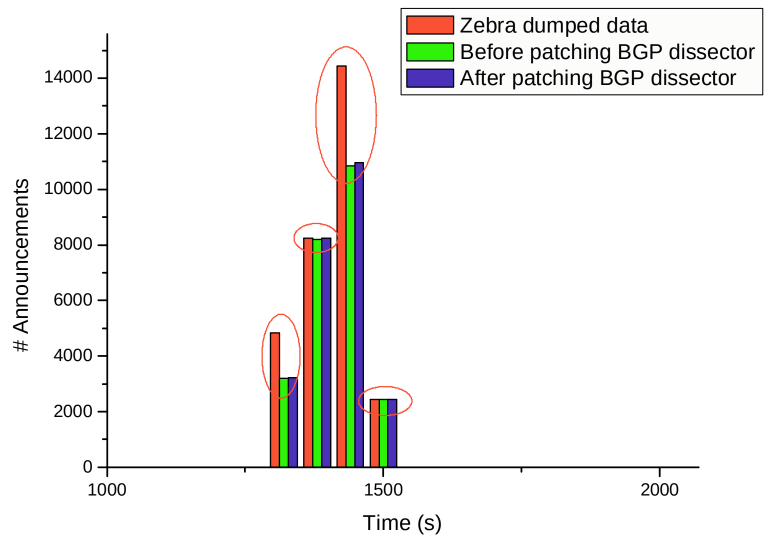


Fig.15 Comparison result before and after patching the bug of Ethereal

As shown In Fig.14, after patching the bug of BGP dissector, the discrepancies between Zebra dumped data and Ethereal captured data disappears. However, in Fig.15, it can be seen that although patching the bug of Ethereal BGP dissector eliminates some discrepancies between Zebra dumped data and Ethereal captured data (the 2nd and 4th bar), there are still some discrepancies remaining (the 1st and 3rd bar). These remaining discrepancies make us suspect that there may be some capture losses for the Ethereal captured data due to the heavy processing load, since we run Zebra dump and Ethereal on the same Linux machine.

To verify our suspicion, the synthesized BGP data with large BGP table (about 160k prefixes) is used. The prefixes are sent with the pattern mentioned in section.2 to help to find when there are capture losses. Data captured with Ethereal and data dumped with Zebra are compared. The comparison results are given in Fig.16. It can be seen that when the processing load is heavy, Ethereal loses some captures. To see the correlation between processing load and Ethereal capture losses, the numbers of updates every 15 minutes are shown in Fig.17.

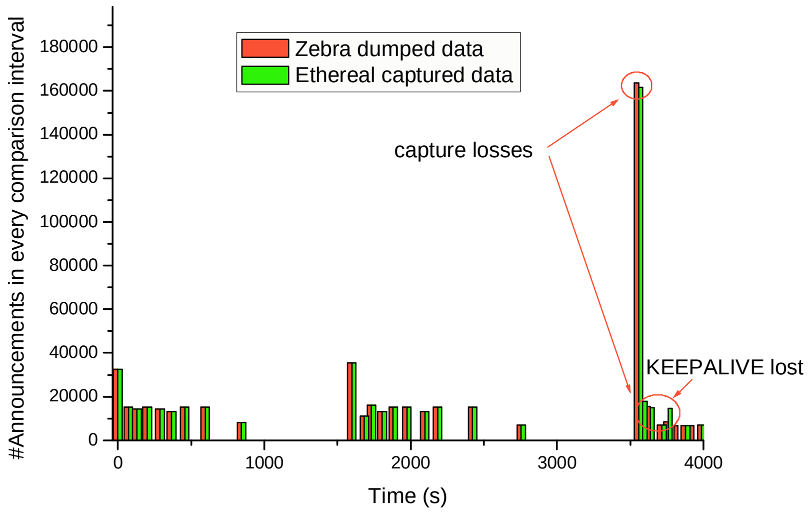


Fig.16 Capture losses of Ethereal

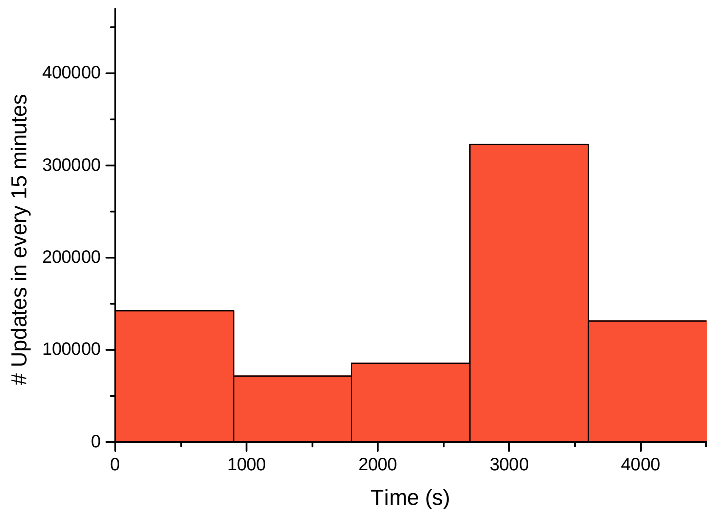


Fig.17 Number of updates in every 15 minutes

From Fig.16 and Fig.17, we can see that when there are a huge number of updates and the processing load is very heavy, Ethereal will lose some captures. The loss of KEEPALIVE message even causes some troubles when aligning the comparison intervals, as shown in Fig.16.

Observing the results show in Fig.16 and Fig.17 together, we can find some correlation between the capture losses and the processing load. Thus we speculate that the capture losses happen in Libpcap (ver.0.7.2) library. Maybe heavy processing load makes the kernel have insufficient time to fetch the captured packet from the capture queue of Libpcap, and thus the overflow of the capture queue of Libpcap leads to capture losses. As mentioned in section.2, Phil Wood provided a patch of Libpcap that utilizes the mechanism of MMAP. MMAP can allow the network adapter to directly capture the packets into system memory. With a large ring queue in system memory, this mechanism can greatly reduce the probability of queue overflow and capture losses. In the tests, Tcpdump (Ver. 3.7.2) is rebuilt and linked with the new patch of Libpcap (Ver 0.8.030314). Then the new Tcpdump is used to capture on-wire data. We did some tests and found that the new Libpcap library does eliminate the problem of capture loss of Ethereal. The hardware configurations of the PC, on which we run both Zebra and Tcpdump, consist of a Pentium III 800Mhz CPU, 256 MB memory. In the tests, we switched the speed of the link interface between 10Mb/s and 100Mb/s, used heavy BGP traffic, and turned on Zebra BGP dump on the PC where Tcpdump ran. All the results showed that there were no capture losses after applying the patch of Libpcap. As an example, the same real BGP data from mae-east is used as the source. The results are given in Fig.18. For the same source data, the results in Fig.18 are different from those shown in Fig.15. The discrepancies due to capture losses are eliminated.

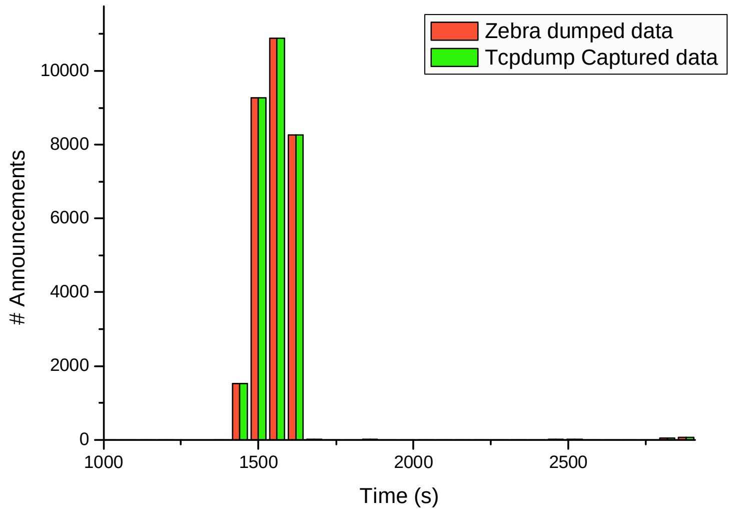
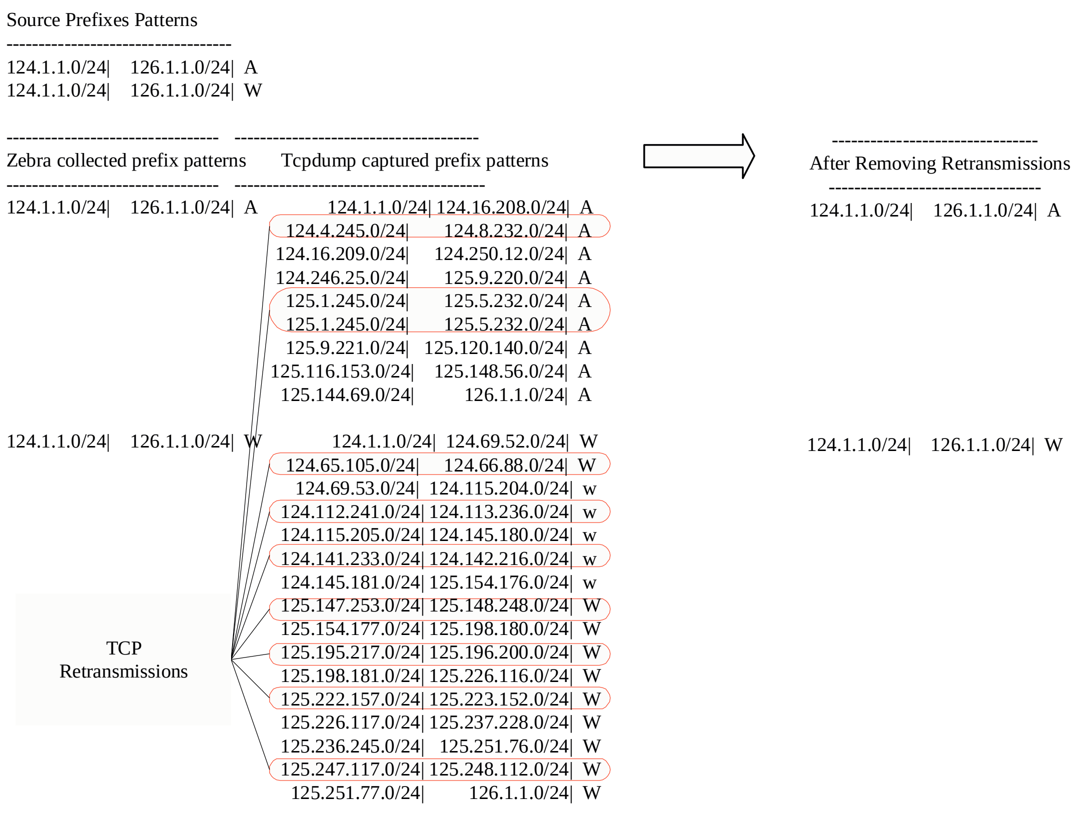


Fig.18 Comparison results when using new Libpcap library

Since Ethereal and Tcpdump capture data without being aware of what the possible protocol is used for the captured session while correctly decoding captured BGP packets requires the correct reconstruction of the BGP session. Thus we are curious about the behavior of Ethereal BGP dissector when there are TCP segment losses and retransmissions. To simulate TCP segment losses and retransmissions more efficiently, NIST Net [10] is used to drop the TCP segments at preset drop probability. Then BGP dissector is used to decode the on-wire captured BGP packets to see whether BGP dissector can correctly reconstruct the BGP session and decode the BGP messages correctly when there are lots of losses and retransmissions. The BGP dissector has taken the session reconstruction into account. To enable the session reconstruction option, Ethereal should be used with the following options on:

```
Tethereal –o bgp.desegment:true –o tcp.desegment_tcp_streams:true
```

To find the problems of Ethereal BGP dissector when there are TCP losses and retransmissions, again the synthesized BGP data with specific prefix patterns is used as the source. The results are given in Fig.19. When there are TCP retransmissions, Ethereal BGP dissector will produce some fake BGP messages, which is actually the decoding result of retransmitted TCP segment. Thus the on-wire captured BGP data doesn't agree with that dumped with Zebra. We have reported this problem to the mail-list of Ethereal developers; However, there has been no fix to this problem up to now (ver 0.9.12). After using some script to remove these fake BGP messages due to retransmitted TCP segments, we find that the on-wire captured BGP data agrees with Zebra dumped data again.



```
Source Prefixes Patterns
-----------------------------------
124.1.1.0/24|   126.1.1.0/24|  A
124.1.1.0/24|   126.1.1.0/24|  W

---------------------------------   ---------------------------------------        --------------------------------
Zebra collected prefix patterns     Tcpdump captured prefix patterns                After Removing Retransmissions
---------------------------------   ---------------------------------------        ---------------------------------
124.1.1.0/24|   126.1.1.0/24|  A         124.1.1.0/24| 124.16.208.0/24|  A          124.1.1.0/24|   126.1.1.0/24|  A
                                      124.4.245.0/24|      124.8.232.0/24|  A
                                     124.16.209.0/24|     124.250.12.0/24|  A
                                     124.246.25.0/24|      125.9.220.0/24|  A
                                      125.1.245.0/24|      125.5.232.0/24|  A
                                      125.1.245.0/24|      125.5.232.0/24|  A
                                      125.9.221.0/24|   125.120.140.0/24|  A
                                    125.116.153.0/24|     125.148.56.0/24|  A
                                     125.144.69.0/24|         126.1.1.0/24|  A

124.1.1.0/24|   126.1.1.0/24|  W         124.1.1.0/24|  124.69.52.0/24|  W          124.1.1.0/24|   126.1.1.0/24|  W
                                     124.65.105.0/24|     124.66.88.0/24|  W
                                       124.69.53.0/24|  124.115.204.0/24|  w
                                    124.112.241.0/24| 124.113.236.0/24|  w
                                    124.115.205.0/24| 124.145.180.0/24|  w
                                    124.141.233.0/24| 124.142.216.0/24|  w
                                    124.145.181.0/24| 125.154.176.0/24|  w
                                    125.147.253.0/24| 125.148.248.0/24|  W
                                    125.154.177.0/24| 125.198.180.0/24|  W
          TCP                       125.195.217.0/24| 125.196.200.0/24|  W
   Retransmissions                  125.198.181.0/24| 125.226.116.0/24|  W
                                    125.222.157.0/24| 125.223.152.0/24|  W
                                    125.226.117.0/24| 125.237.228.0/24|  W
                                    125.236.245.0/24|   125.251.76.0/24|  W
                                    125.247.117.0/24| 125.248.112.0/24|  W
                                      125.251.77.0/24|        126.1.1.0/24|  W
```

Fig.19 Discrepancies due to TCP retransmissions

After exploring the reasons that cause the discrepancies between Zebra dumped data and on-wire captured data and fixing these problems, we do more tests to verify the consistency of Zebra BGP data collection. The results are given in the following subsections.

3.3 Verification results when using synthetic BGP data

In the test, the synthesized BGP data with large prefix table (about 160k prefixes) is used as the source. The tests are done on both Linux and Solaris platform. Since no MP_REACH_NLRI /MP_UNREACH_NLRI attributes are used in the tests, both the test on Linux and the test on Solaris have the similar results. The results are shown in Fig.20. After all the problems mentioned in the above sections were solved, now the on-wire captured data agrees with Zebra BGP dumped data for most of the time except when there is a peering session break. However, the discrepancies during the session break are reasonable since at that time the state of BGP is reset abruptly. It will neglect all the following BGP messages. This may cause the discrepancies between on-wire captured data and data dumped with Zebra. The number of updates in every 15 minutes is shown in Fig.21 to show the relation between the peering session break and the heavy BGP load. It can be seen that the synthesized traffic used in the test is so heavy that Zebra will delay the sending of KEEPALIVE for so long a time that the peer on the other end (BGPsim in this test) tears down the BGP session. Thus the results shown in Fig.20 and Fig.21 verified that after solving the problems of the tools, when there is no session break, Zebra dumped data is consistent even under extremely heavy synthesized BGP traffic. However, when there is session break, Zebra dumped data may be inconsistent with what has been sent out from the source.

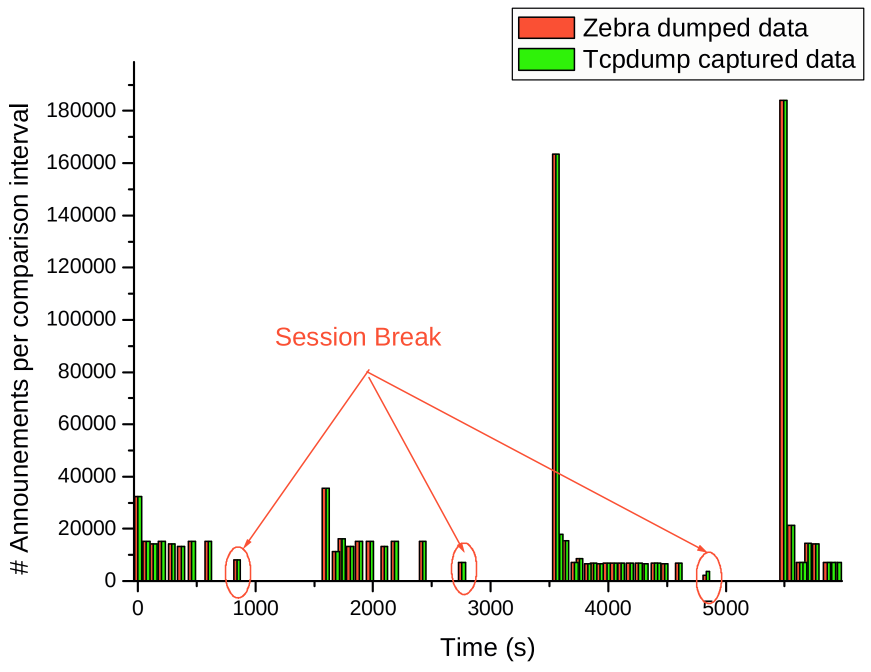


Fig.20 Comparison result with synthetic BGP data

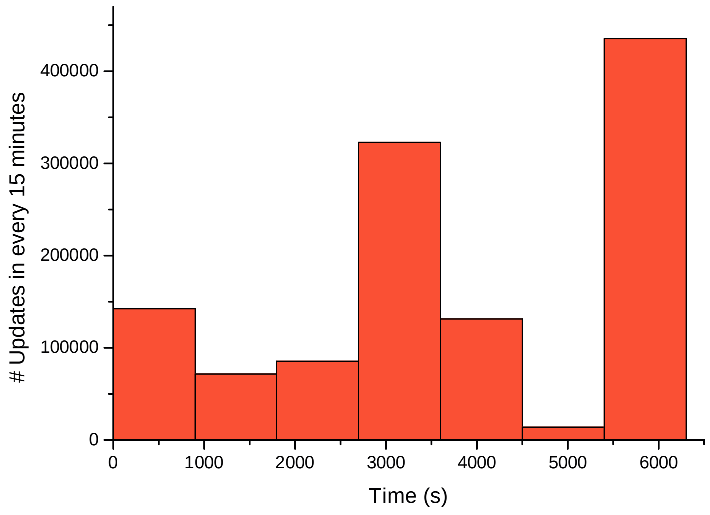


Fig.21 The number of updates in every 15 minutes

3.4 Verification results when using real BGP data

In this test, real burst BGP data sets are used to verify the consistency of Zebra data collection, where the burst BGP data sets are the dumped update data files on rrc03 from 5:30 to 9:45 on 25th, January, 2003. It can be seen that the size of these files is an order larger than that of other dumped files in magnitude. There are a lot of NOTIFICATION messages in these data files. However, in our tests, we simply ignore the session with NOTIFICATION messges. The number of updates of each BGP session in these files is counted, and the two with the largest number of updates are extracted out and replayed to do the tests. The sessions are AS13237 to AS12654 and AS12859 to AS12654. We find that over session AS12859 to AS12654 there are some BGP messages, which have MP_REACH_NLRI/MP_UNREACH_NLRI attributes for IPv6. Since Route_BtoA on Linux can't decode these messages correctly, we remove these messages before we do the test. What we want to mention here is that such removal won't compromise the reliability of the verification. This is based on two observations: 1) the replaying result of such messages on Solaris has shown that Zebra can dump such messages correctly. They are not different from other BGP messages in this point. 2) the number of such messages is very small compared with the total number of updates in these files. The comparison results are presented in Fig.22 together with the updates every 15 minutes of the source data (these are only for session AS13237 to AS12654).

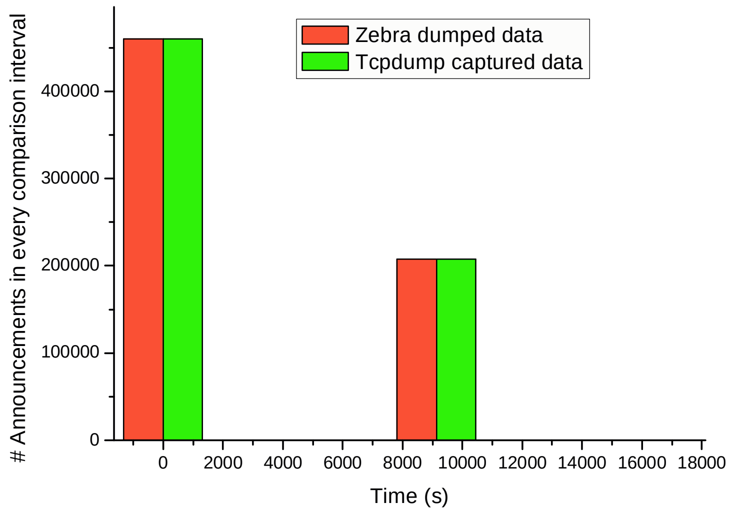


Fig.20 Comparison results when using real burst BGP data (AS13237 to AS12654).

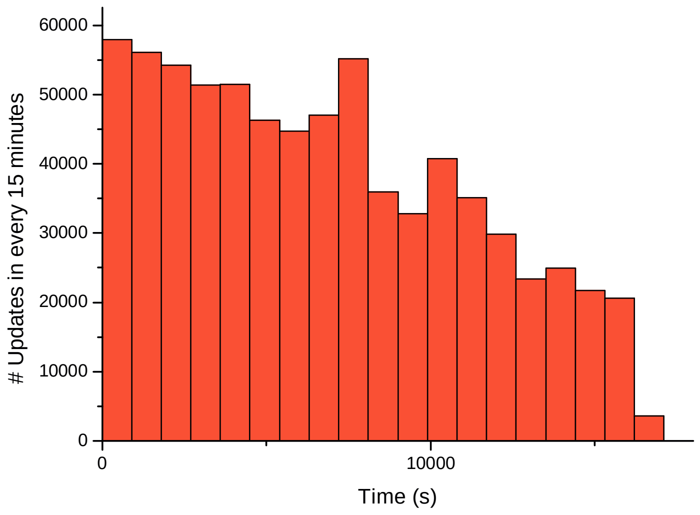


Fig.21 The number of updates in every 15 minutes for session AS13237 to AS12654

The comparison results show that even for real BGP data with severe burst, after the bugs of the tools were fixed, Zebra BGP dumped data is consistent with the source data if there is no session break, and that the results got using different tools agree with each other.

From all these results, we conclude that after fixing the bug of Zebra BGP dump program, Zebra dumped BGP data will be consistent when there is no session break, and after all the problems of the tools being fixed, the data got from different tools will agree with each other.

3.5 Results on the peering session breaks in real BGP data collections

In the above sections, the test results have shown that when there is no BGP session break, Zebra data collections are consistent and they can be verified with on-wire captured data. However, we also notice that when the BGP traffic is heavy, the BGP session between Zebra and its peer may be torn down due to expired hold timer. When this happens, Zebra data collection will be corrupted because of the updates due to Zebra BGP session down and up. To find out how serious the problem is, we do some statistics about the NOTIFICATION messages on the real data collections.

Site	Date	#NOTIFICATION
Routeview	Jan 15th-Jan 31st 2003	0
rrc00	Jan. 2003	11
rrc01	Jan. 2003	6
rrc02	Jan. 2003	1780
rrc03	Jan. 2003	45061
rrc04	Jan. 2003	0
rrc05	Jan. 2003	17
rrc07	Jan. 2003	1
rrc08	Jan. 2003	9314

Table.4 The number of Notifications in one-month data collections

From Table.4, we can see that for site rrc02, rrc03 and rrc08, there are a lot of NOTIFICATION messages. So many NOTIFICATION messages may cause corrupted data collections since there may be lots of updates due to Zebra session up and down. Further explorations on the NOTIFICATION messages reveals that there are several different kinds of NOTIFICATION message patterns:

1) Successive NOTIFICATION series (type 4/0) from the same peer due to hold timer expiration. We found that the NOTIFICATION series on rrc08 belong to this category. For rrc08, we observed consecutive NOTIFICATION messages from AS2914 with type 4/0. The data files from 9, 9th Jan. to 23:45 9th Jan are used. The NOTIFICATION series and the number of updates in every 5 minutes are shown in Fig.22 and Fig.23 to explore the possible relation between the hold timer expiration and the BGP traffic load on Zebra. From Fig.22 and Fig.23, we can coarsely see that the NOTIFICATION messages are related with the number of BGP updates. However, the explicit relation between the number of update messages and the NOTIFICATION messages is not found yet. It is very strange that the NOTIFICATION messages sometimes are so close to each other. Some interval between two NOTIFICATION messages even is smaller than 30 seconds.

Due the frequent NOTIFICATION messages, the peer sends almost no updates to the Zebra BGP daemon peering with it. Thus such kind of session break influence the consistency of Zebra BGP data collection very little, we can ignore the data from that session completely when there are successive NOTIFICATION series from the same peer to avoid the problem of corrupted data due to Zebra session up and down. Such NOTIFICATION patterns seem to imply there may be some bug in the BGP implementation of the routers. However, up to now we haven't verified this suspicion.

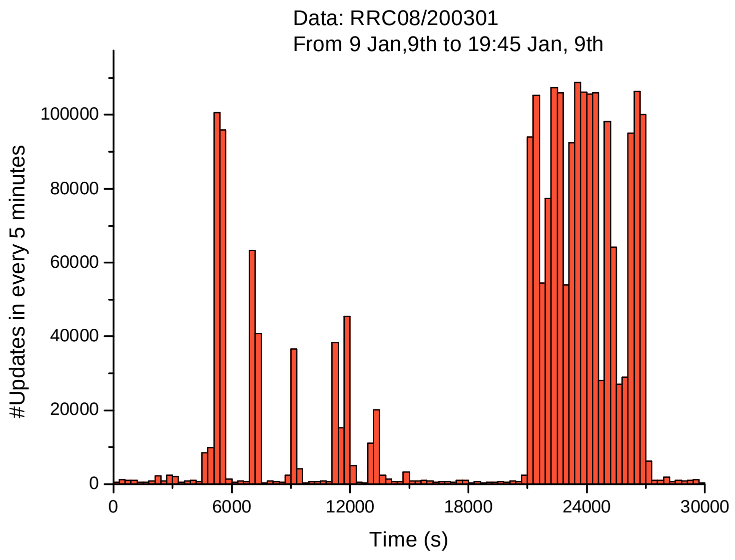


Fig.22 The number of update messages in every 5minutes

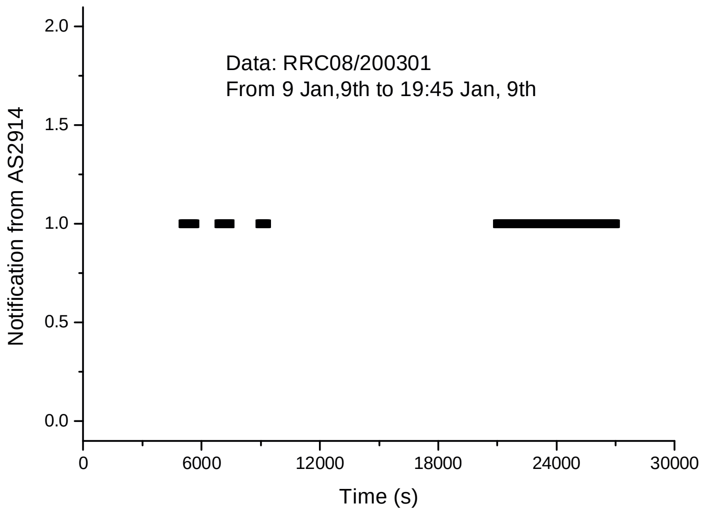


Fig.23 NOTIFICATION series

2) Successive NOTIFICATION messages (type 4/0) from several different peers due to hold timer expiration. We notice that when the BGP traffic is very heavy, the BGP sessions from several peers may break in a very short time interval. This can seriously corrupt the data collections. Since heavy BGP traffic often means a lot of network changes, we are extremely

interested in the data collections when the BGP traffic is very heavy. Thus peering session break under heavy BGP traffic will be problematic. To illustrate the possible relationship between the BGP traffic and the NOTIFICATION series from different peers, the data files for the time period from 10, 7th Jan to 14:45, 7th Jan are used. The results are shown in Fig.24 and Fig.25 respectively. Although sometimes the NOTIFICATION messages don't seem to correspond a heavy BGP traffic. We can see when there are heavy BGP traffic, some BGP session will experience session break due to hold timer expiration. To consistently collect BGP data at all time, Zebra need to be revised.

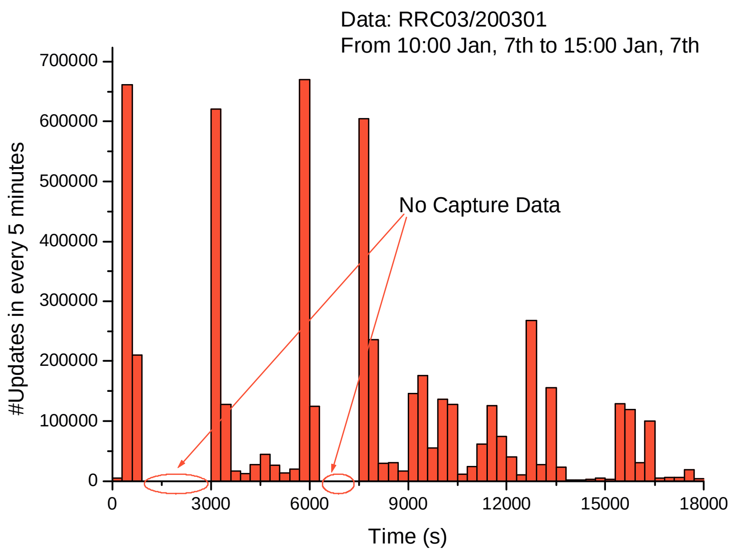


Fig.24 The number of updates messages in every 5 minutes

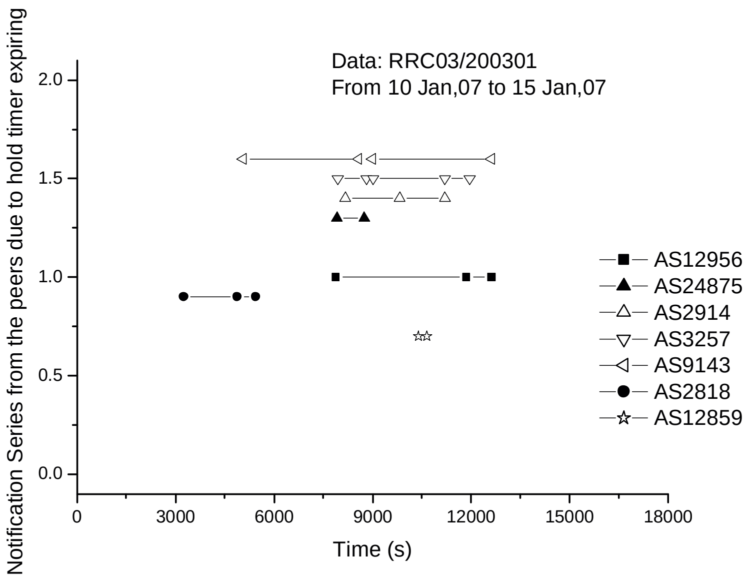


Fig.25 NOTIFICATION series (type 4/0) from different peers

3) Interleaved NOTIFICATION message series between type 4/0 and type 2/5. On rrc02, the successive NOTIFICATION messages from AS12956 belong to this category. This behavior is

also very strange, since the Zebra BGP daemon, which is collecting BGP data from AS12956, should be an authorized peer for AS12956. If we have the OPEN messages sent by the Zebra BGP daemon, we can find whether there are some errors at the beginning of BGP session establishment. Thus we propose to improve Zebra by adding the feature of dumping both in and out BGP messages.

4 Next step improvements on the tools

Ensuring that the tools behave as expected, and give consistent results is very important. According to the results in section.3, the next step improvements on the tools are given below:

1) Route_BtoA.
   - Add full support of MP_REACH_NLRI / MP_UNREACH_NLRI attributes for both IPv6 and multicasting IPv4 prefixes no matter whether the kernel has the supports for IPv6 and multicast routing.
   - Add support of decoding messages without correct dump header but with correct BGP messages
   - Add support of decoding incompletely dumped BGP messages
   - Add support of converting unsupported attributes directly into ASCII format
2) Ethereal
   - Introduce a new patch to correctly reconstruct BGP sessions when there are TCP losses, retransmissions and out of order TCP segments
3) Zebra
   - Revise the part of BGP dump to make sure that Zebra always dumps packets in the format given in Appendix.2.
   - Revise Zebra to make it more suitable for BGP data collecting, for example, removing the prefix update and out filter modules from BGPd implementation.
   - Dump both in and out BGP messages for research purpose, where the out messages are only OPEN, KEEPALIVE and NOTIFICATION messages.

5 Conclusions

In this report, the consistency of Zebra BGP data collection is verified. We thoroughly analyzed and classified the problems that can result in incorrectly decoded BGP messages, and presented statistic results of how frequently these problems may occur. Improvements on the tools are also proposed to solve these problems. Then we proposed two approaches to verify the consistency of Zebra data collection. Both verifying approaches are very effective and help to identify many problems that cause discrepancies between the results obtained by different tools. After these problems were fixed, the test results show that when there is no BGP session break, Zebra data collection is consistent and can be comparable with on-wire captured data. The results in this report also imply that although there may be some issues in public BGP data collection, the problems themselves are not very serious and most of the BGP data collections are reliable and consistent.

6 Acknowledgements

The author would like to express thanks to Alexander Tudor, Olaf Maennel and Xinyu Ma for their good suggestions on the report and their kind helps during this project. The discussion with Lance Tatman on the BGP session break patterns is very helpful for making the conclusions in this report complete and consistent. Matthew Williams and Arife Vural also provide helpful feedbacks on how to improve the report. The author is grateful for all of them for their helps on improving the report.

Reference

1. http://www.zebra.org
2. http://www.mrtd.net/
3. Vinay Kumar Aggarval, “Debugging ROUTE_BTOA”, March, 2003
4. http://www.ietf.org/rfc/rfc1771.txt
5. http://www.ietf.org/rfc/rfc2858.txt
6. http://www.ethereal.com
7. http://www.tcpdump.org/
8. http://public.lanl.gov/cpw/
9. http://www.xml.com/ldd/chapter/book/ch13.html
10. http://snad.ncsl.nist.gov/itg/nistnet/
11. http://www.ris.ripe.net/source/ (by Dan Ardelean with RIPE NCC)
12. http://data.ris.ripe.net
13. http://www.ripe.net
14. http://archive.routeviews.org
15. http://www.routeviews.org
16. R. Govindan, A. Reddy, An Analysis of Internet Inter-domain topology and Route Stability, IEEE INFOCOM 1997, Japan.
17. J. Rexford, J. Wang, Z. Xiao, Y. Zhang, BGP Routing Stability of Popular Destinations, ACM SIGCOMM Internet Measurement Workshop (IMW) 2002, http://www.icir.org/vern/imw-2001/imw2002-papers/160.ps.gz
18. C. Labovitz, A. Ahuja, A. Bose, Delayed Internet Routing Convergence. SIGCOMM 2000.
19. D. G. Anderson, N. Feamster, S. Bauer, H. Barakrishnan, Topology Inference from BGP Routing Dynamics, ACM SIGCOMM Internet Measurement Workshop (IMW), 2002. http://www.icir.org/vern/imw-2001/imw2002-papers/206.ps.gz
20. ftp://ftp.merit.edu/statistics/ipma/data/

Appendix.1 Packet binary dump format of MRTd

Followings are the binary packet formats used by most MRT tools.
1) MRT header (12 bytes)

0											1									2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Time																															
Type																Subtype															
Length																															

All MRT messages have this common header, which includes Timestamp, message type and subtype and the length of the message. The length does not include the MRT header itself.

```
MRT_MSG_TYPES (Valid Type Values and Their Definitions)
0 MSG_NULL,
1 MSG_START,                 /* sender is starting up */
2 MSG_DIE,                   /* receiver should shut down */
3 MSG_I_AM_DEAD,             /* sender is shutting down */
4 MSG_PEER_DOWN,             /* sender's peer is down */
5 MSG_PROTOCOL_BGP,          /* msg is a BGP packet */
6 MSG_PROTOCOL_RIP,          /* msg is a RIP packet */
7 MSG_PROTOCOL_IDRP,         /* msg is an IDRP packet */
8 MSG_PROTOCOL_RIPNG,        /* msg is a RIPNG packet */
9 MSG_PROTOCOL_BGP4PLUS,     /* msg is a BGP4+ packet */
10 MSG_PROTOCOL_BGP4PLUS_01, /* msg is a BGP4+ (draft 01) packet */
```

2) Subtypes of BGP Message (`MSG_PROTOCOL_BGP, MSG_PROTOCOL_BGP4PLUS, MSG_PROTOCOL_BGP4PLUS_01`)

```
MRT_MSG_BGP_TYPES (Valid Subtype Values and Their Definitions)
0 MSG_BGP_NULL,
1 MSG_BGP_UPDATE,  /* raw update packet (contains both with and ann) */
2 MSG_BGP_PREF_UPDATE, /* tlv preferences followed by raw update */
3 MSG_BGP_STATE_CHANGE /* state change */
4 MSG_BGP_SYNC
```

- If the message type is `MSG_PROTOCOL_BGP` and the message subtype is `MSG_BGP_UPDATE,` the following applies:

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Source AS Number																Source IP Address															
Source IP Address																Destination AS Number															
Destination IP Address																															
BGP Update Packet																															

- If the message type is `MSG_PROTOCOL_BGP4PLUS` and the message subtype is `MSG_BGP_UPDATE`, the following applies:

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Source AS Number																Source IP Address															
Source IP Address continued																															
Source IP Address continued																															
Source IP Address continued																															
Source IP Address continued																Destination AS Number															
Destination IP Address																															
Destination IP Address continued																															
Destination IP Address continued																															
Destination IP Address continued																															
BGP Update Packet																															

- If the message subtype is `MSG_BGP_STATE_CHANGE`, the following format should be used for state change messages:

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Source AS Number																Source IP Address															
Source IP Address continued																															
Source IP Address continued																															
Source IP Address continued																															
Source IP Address continued																Old State															
New State																															

- If the message subtype is `MSG_BGP_SYNC`, the following format applies:

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
View #								File Name (Variable)																							

The filename ends with '\0' (null.)

3) The format for the routing table dump is:

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
View #																Prefix															
Prefix (continue)																Mask								Status							
Time Last change																															
Attribute Length																															
BGP Attribute (Variable Length)…																															

Appendix.2 Packet binary dump format of Zebra

The packet binary dump format of Zebra shares the same common header with that of MRTd. This is to provide backward compatibility of MRTd binary data file. The common header is in the following format:

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Time																															
Type																Subtype															
Length																															

The valid type for Zebra binary dump format is MSG_PROTOCOL_BGP4MP (16), and the valid subtype is:

BGP4MP_STATE_CHANGE	0
BGP4MP_MESSAGE	1
BGP4MP_ENTRY	2
BGP4MP_SNAPSHOT	3

- If `type' is PROTOCOL_BGP4MP, `subtype' is BGP4MP_STATE_CHANGE, and Address Family == IP (version 4)

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Source AS Number																Destination AS Number															
Interface Index																Address Family															
Source IP Address																															
Destination IP Address																															
Old State																New State															

Where State is the value defined in RFC1771.

- If `type' is PROTOCOL_BGP4MP, `subtype' is BGP4MP_STATE_CHANGE, and Address Family == IP (version 6)

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Source AS Number																Destination AS Number															
Interface Index																Address Family															
Source IP Address																															
Source IP Address Continued																															
Source IP Address Continued																															
Source IP Address Continued																															

Destination IP Address	
Source IP Address Continued	
Source IP Address Continued	
Source IP Address Continued	
Old State	New State

- If `type' is PROTOCOL_BGP4MP, `subtype' is BGP4MP_MESSAGE, and Address Family == IP (version 4)

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Source AS Number																Destination AS Number															
Interface Index																Address Family															
Source IP Address																															
Destination IP Address																															
BGP Message Packet																															

Where BGP Message Packet is the whole content of the BGP4 message including header portion.

- If `type' is PROTOCOL_BGP4MP, `subtype' is BGP4MP_MESSAGE, and Address Family == IP (version 6)

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Source AS Number																Destination AS Number															
Interface Index																Address Family															
Source IP Address																															
Source IP Address Continued																															
Source IP Address Continued																															
Source IP Address Continued																															
Destination IP Address																															
Destination IP Address Continued																															
Destination IP Address Continued																															
Destination IP Address Continued																															
BGP Message Packet																															

- If `type' is PROTOCOL_BGP4MP, `subtype' is BGP4MP_ENTRY, and Address Family == IP (version 4)

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
View #																Status															
Time Last Change																															

Address Family		SAFI	Next-Hop-Len
Next Hop Address			
Prefix Length	Address Prefix (variable)		
Attribute Length			
BGP Attributes (variable length)			

- If `type' is PROTOCOL_BGP4MP, `subtype' is BGP4MP_ENTRY, and Address Family == IP (version 6)

0 1 2 3 4 5 6 7	8 9 0 1 2 3 4 5	6 7 8 9 0 1 2 3	4 5 6 7 8 9 0 1
View #		Status	
Time Last Change			
Address Family		SAFI	Next-Hop-Len
Next Hop Address			
Next Hop Address Continued			
Next Hop Address Continued			
Next Hop Address Continued			
Prefix Length	Address Prefix (variable)		
Address Prefix continued (variable)			
Attribute Length			
BGP Attributes (variable length)			

BGP4 Attribute must not contain MP_UNREACH_NLRI. If BGP Attribute has MP_REACH_NLRI field, it must has zero length NLRI, e.g., MP_REACH_NLRI has only Address Family, SAFI and next-hop values.

- If `type' is PROTOCOL_BGP4MP and `subtype' is BGP4MP_SNAPSHOT, The file specified in "File Name" contains all routing entries, which are in the format of "subtype == BGP4MP_ENTRY".

0 1 2 3 4 5 6 7	8 9 0 1 2 3 4 5	6 7 8 9 0 1 2 3	4 5 6 7 8 9 0 1
View #		File Name (variable)	

Appendix.3 Ethereal data file format

Ethereal uses the same data file format as Tcpdump, which is the native format of WinPcap and Libpcap. At the beginning of the file, there is a header, whose format is given below:

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Magic Number																															
Major Version																Minor Version															
Time Zone Offset																															
Time stamp accuracy																															
File Length																															
Link Type																															

- The magic number 0xa1b2c3d4 is used to understand if the file was generated by a little endian architecture or by a big endian architecture. In the little endian case the bytes sequence is: 0xa1, 0xb2, 0xc3, 0xd4; in the big endian case: 0xd4, 0xc3, 0xb2, 0xa1.
- The Major Version field and Minor Version field are used to identify the version of the file format. Current Major Version is 2 and Minor Version is 4.
- Time Zone Offset: the time zone in relation with Greenwich.
- Time stamp accuracy: not actually used.
- The link type: describes on what kind of link the packets are captured. The map between the value and the link types is given below:

Value	Description
0	no link-layer encapsulation
1	Ethernet (10Mb)
2	Experimental Ethernet (3Mb)
3	Amateur Radio AX.25
4	Proteon ProNET Token Ring
5	Chaos
6	IEEE 802 Networks
7	ARCNET
8	Serial Line IP
9	Point-to-point Protocol
10	FDDI
11	LLC/SNAP encapsulated ATM
12	Raw IP
13	BSD/OS Serial Line IP
14	BSD/OS Point-to-point Protocol

Each packet has a capture header that contains the following information:

0											1									2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Seconds																															
Microseconds																															
Packet Length																															
The packet part length contained in the file																															

- Seconds: the number of seconds since January 1, 1970, 00:00:00 GMT.
- Microseconds: the number of microsecond since that second when the packet was captured.
- Packet Length: the number of bytes of packet data that were captured.
- The packet part length contained in the file: the actual length of the packet, which may be greater than the packet length if not a entire packet was saved.

Appendix.4 Libpcap8.0 patch for Linux

Libpcap8.0 patch for Linux utilizes the shared memory ring buffers, which were first implemented in the Linux kernel by Alexey Kuznetsov as a patch to the 2.2.x kernel and now a kernel configuration option (CONFIG_PACKET_MMAP) in the 2.4.x kernel. Up to approximately 32768 frames can be allocated and the frame size is a configurable parameter using environment variable.

"to_ms" is a configurable parameter, which was supposed to define a time in milliseconds when a "read" from the network would wait for incoming packets before returning to the pcap function that causes a read from the network. In the past, on Linux, and possibly other operating systems, this value was not deemed worthy of accommodating. However, with the advent of memory mapped ring buffers, in the new patch this function could be accommodated. In the patch, the meaning of "to_ms" was redefined as follows

1) to_ms is set to zero. This means that if no packet immediately available then return to calling program it will poll. This will be good for programs have other things than capture packets to do.

2) to_ms is set to be greater than zero. This means that when the timeout value (its initial value is the value of to_ms) has gone to zero since the beginning of the capture, then return.

3) to_ms is set to be less than zero. This means never returning from Libpcap library. Keep picking packet off the ring, doing callbacks and waiting, in that order until the program terminates via a signal, error or the PCAP_TIMEOUT time has been exceeded.

There are a number of ways a Libpcap based application can be coerced into using the shared ring buffer:

1) If shared libraries are being used by the application, then you can set an environment variable to clue pcap_open_live to go the ring route:

```
# PCAP_FRAMES=max tcpdump -i eth1 ...
```

Setting PCAP_FRAMES to max actually sets the size of the ring buffer to 32768 (0x8000) frames. For ring buffer of 32768 frames, with 1530 bytes per frame (ethernet mtu + frame overhead + 16), around 51 Mbytes is needed for the ring buffer alone.

Other environment variables include:

PCAP_SNAPLEN	think tcpdump -s
PCAP_PROMISC	-1 = promiscous -2 = not promiscuous
PCAP_TO_MS	variable meanings, think milliseconds(ms) to wait for pkt, or how long to loop reading packets and calling the user supplied callback.
PCAP_RAW	2 = cooked mode
PCAP_PROTO	ip,ipv6,arp,rarp,802.2,802.3,lat,dec,atalk,aarp,ipx,x25
PCAP_MADDR	requires hex string which will override PCAP_PROMISC
PCAP_FRAMES	greater than 0 up to 32768, "max" is equiv to 32768. Setting PCAP_FRAMES to 0 will disable the ring buffer.
PCAP_VERBOSE	print informative messages, since old app doesn't see them.
PCAP_STATS	print pcap statistics to stderr every PCAP_PERIOD ms.

```
                        Stats will also be printed whenever pcap_read, pcap_dispatch,
                        and pcap_loop return to the calling program.
         PCAP_TIMEOUT return errno "ETIMEDOUT" when packet time is greater than
                        value provided (eg PCAP_TIMEOUT=1044406300 will cause
                        tcpdump to quit on Mon Feb 4 17:51:40 MST 2003).
         PCAP_PERIOD    milliseconds between stats (will not cause pcap_dispatch
                        to return, will generate stats).
```

2) If static libraries are being used the application will need to be reloaded.

3) Or, one can write non-portable Libpcap based applications by calling a linux specific pcap routine called pcap_ring_args which initializes ebuf prior to calling pcap_open_live as a way of passing in extra ring related arguments (Not recommended).

The new patch can coerce live statistics using the environment variable 'PCAP_STATS'. The format for PCAP_STATS is a 32 bit mask. Bit 0 is defined to be 0x00000001. The meaning of the statistics and their corresponding mask bits are given below:

0	secs since 00:00:00, Jan 1, 1970, followed by a '.' fraction (us)
1	Packets processed
2	Packets dropped
3	Packets total
4	Packets ignored
5	Packets seen by device (in and out)
6	Bytes seen by device (in and out)
7	Bytes received
8	Number of times poll system call called
9	Current ring buffer index
10	Maximum number of frames pulled from ring before having to poll
11	Specious signal to pull frames from ring
12	Elapsed time between first and last packet seen during sample time
13	Received errors
14	Received drops
15	Transfer errors
16	Transfer drops
17	Multicast packet count (the Packets total includes this number)

Setting PCAP_STATS=0x021fff means that the ring statistics line would print items 0 through 12 and item 17 in that order.