Adaptive Carving Method for Live FLV Streaming

Haidong Ge[1], Ning Zheng[1], Lin Cai[1], Ming Xu[1](✉), Tong Qiao[2], Tao Yang[3](✉), Jinkai Sun[1], and Sudeng Hu[4]

[1] Internet and Network Security Laboratory, School of Computer Science and Technology, Hangzhou Dianzi University, Hangzhou, China
{151050149,nzheng,mxu,152050160}@hdu.edu.cn, cail@zj.gov.cn

[2] School of Cyberspace, Hangzhou Dianzi University, Hangzhou, China
tong.qiao@hdu.edu.cn

[3] Key Lab of the Third Research Institute of the Ministry of Public Security, Shanghai, China
yangtao@stars.org.cn

[4] Ming Hsieh Department of Electrical Engineering, University of Southern California, Los Angeles, CA, USA
sudenghu@usc.edu

Abstract. Currently, most video carving methods are to recover video files from disk file system, but these methods often do not work well for video form network streams, especially for live streaming video. In this paper, an adaptive video carving method is proposed to recover the live FLV (Flash Video) streaming video from network traffic. Firstly, to recover videos when there is no packet loss during data capture, a method based on network data structure is proposed. Secondly, to solve the problem of packet loss or corruption during data capture, another video carving method is proposed based on both the FLV structure and network data structure. Finally, to achieve good balance between computational complexity and recovery accuracy, an adaptive method based above two methods is proposed. The experimental results show that the proposed methods achieve good performance both in consuming time and recovery rate.

Keywords: Live streaming video · Recovery · Forensic Network traffic

1 Introduction

Digital forensics is the process of preserving, acquiring, checking, analyzing and presenting digital evidence from any digital devices. Recovering videos from digital data fragments is an important work in digital forensics. With the rapid development of computer network, live streaming videos have been popular in our daily life. Meanwhile live video related crimes have occurred more and more frequently and it has aroused great concerns in society [1–4]. In addition, since

© ICST Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2018

criminal evidence in storage media can be easily accessed by criminals and suffer irreversible damages, the traditional digital forensics based on storage media becomes less reliable in the case of providing direct and effective evidence in criminal investigations [5–7]. Criminal evidence [8] sent to networks with live streaming video could be hardly damaged by criminals. Therefore live streaming video is safer to store criminal scenes and evidence, and carving methods for live streaming videos are highly desired.

In the literature, most video carving methods use file structures to restore video files from disk file system, but such methods do not perform well for videos from network packet data. A few number of carving methods can restore videos from network streams, like the open source FLV download tool FLVCD, which downloads the FLV video from servers. However FLVCD does not work for live streaming videos, because it requires source's IP addresses, which are often encrypted or hidden to prevent video stealing. Wireshark [9] is one of the excellent network security and monitoring tools, which can extract parts of the video in the network stream. The Flash Video (FLV) format is widely used in live streaming video and usually transmitted with Hypertext Transfer Protocol (HTTP)[10] chunked transfer encoding method. Inspired by the working principle of Wireshark and structure of FLV, the HTTP chunk based carving (HCBC) is proposed to extract video data.

Furthermore, network data used for live streaming video recovering is usually captured at certain gateways, where video data flows through with large volume of other data. Sometimes we cannot capture all the needed video data, but inevitably have some video packets lost, corrupted or disordered during capture process [11,12]. Recovering from partial live streaming data is quite challenge work. For example, HCBC does not perform well in such cases. To solve this problem, we propose the FLV tag based carving (FTBC) method, which is based on both FLV tag structure and network data structure, rather than network data structure only. Since FLV data is only the subset of network data, we can recover videos in the absence of some irrelevant network data. Moreover, the redundancy in FLV data can be used to further recover videos precisely.

FTBC is better than HCBC in terms of video recovery rate, however it consumes much more time than HCBC due to the complicated retrieving scheme it adopts. In the case of losing no packet or a few packet, HCBC could also achieves good performance in recovery rate. To make a good balance between consuming time and recovery rate, an adaptive method based on FTBC and HCBC is finally proposed in this paper.

The remainder of the paper is structured as follows: In Sect. 2, related video carving work is described; In Sect. 3, the adaptive video carving method is introduced; in Sect. 4, the experimental results are given to compare the performance of the proposed methods and other benchmarks. Finally, Sect. 5 concludes this paper.

2 Related Work

The traditional recovery technique recover the video files from the storage device, we recover video from the captured network traffic based on the network data structures and video structure. Basic video recovery technique for video file restoration use the meta-information [13] of the file system to recover a video file stored in a storage medium. New attempts have been made to recover the files depending on the file structure or file content instead of the file system information. The Scalpel [14] is one of the file carving tools which do not rely on the file system meta-information to restore a video file. This carver uses header and footer of video files to restore the video file in a whole disc. Pal and Memon proposed [15] a general framework to restore a file. This method identifies fragments and reorders the fragments by uses three main steps: preprocessing, collation and reassembly. These files in the storage medium which was not allocated. Park and Lee proposed a procedure for use on video data fragments forensics related to DVRs [16]. They present a way to sort the extracted video frames based on the above properties. These frames will be converted to images and these images is used to construct a playable video file. Yang et al. [17] proposed a technique in their paper is an AVI carving method based on frame size and index. This method extracts video data from a storage medium using signatures and statistic information rather than file system information. However, to the best of our knowledge, there is only a little research of network flow solution that is available for video craving.

Most of the previous works focused on MPEG4 or some other formats and not much work has been proposed in video carving for network flows. With the consideration of wide use of the FLV format in live streaming videos, recovering FLV data in network flows remains important.

3 The Proposed Methods

In this section, two carving methods based on network data structure and FLV structure are developed to handle different packet loss rate. In order to handle various network situations, an adaptive carving method is proposed based on these two methods. These proposed carving methods mainly include three stage:

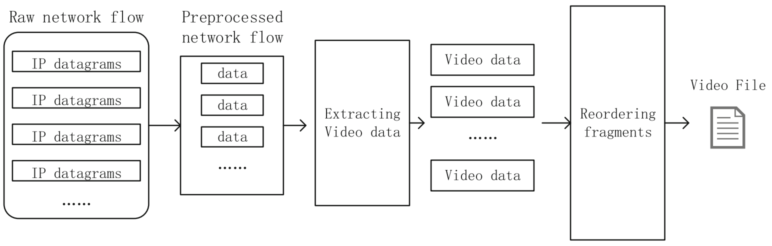


Fig. 1. Processing steps of the proposed video file restoration technique

preprocessing, extraction and connection, as shown in Fig. 1. Due to the huge amount of data flowing on the network, directly recovering video files this data may not be efficient. In the preprocessing stage, network work data is screened based on the source IP, destination IP and port and playloads of selected TCP packets are extracted. In the extraction stage, the FLV header data and the video data is extracted in the data blocks based on the characteristics of the network flow or the structural features of the FLV file. Finally, in the reordering stage, extracted data is sorted and merged into video files.

3.1 FLV File Structure

The FLV file consists of a file header and a number of tags that could be classified into three categories including audio tags, video tags and script tags based on the types of streams they represent. The overview structure of a FLV file is shown in Fig. 2. The FLV file starts with a file header consisting of 9 bytes, while the first three bytes are the signature of the FLV file "46 4C 56". 4 bytes followed by the 9 bytes are the first tag size which is always 0. After that, the FLV file is splitted into different tags. The first eleven bytes of a tag is tag header, where first byte indicates tag types, followed by three bytes indicating playload size of a tag. Other information like timestamp is indicated in the remaining bytes. Since we are only interested in video and audio files in this work, our discussion on FLV tags is limited to these two types. Table 1 shows typical values of different types of FLV tags, where we can see each type of FLV tag has a unique value in the first byte. For example, '0x08' represent audio, '0x09' for video, '0x12' for metadata which contains codec specifications. Furthermore, the first byte of tag playload, which indicates its type, should be restricted to certain values according to tag types. For instance, as shown in Table 2, inside a 'video' tag, the first byte of tag playload is either'0x17' denoting a keyframe of video data or '0x27' denoting interframe of video data. Other values are invalid.

3.2 Video Carving with HCBC

In China, the technique of HTTP-FLV is widely used in live video streaming. Due to the real-time nature of live video, the size of the transmitted content

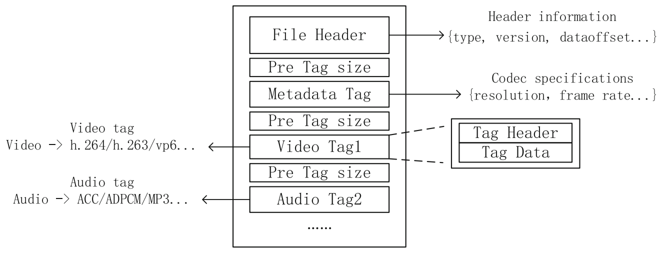


Fig. 2. The structure of video file

Table 1. FLV tag type. ‘x’ is hexadecimal data.

Tag code	Name	Description
0x08xxxxxxxxxxxxxxxxxxxx	Audio	Contains a audio packet similar to a SWF SoundStreamBlock plus codec information
0x09xxxxxxxxxxxxxxxxxxxx	Video	Contains a video packet similar to a SWF VideoFrame plus codec information
0x12xxxxxxxxxxxxxxxxxxxx	Meta	Contains two AMF packet, the name of the event and the data to gowith it

could not be determined when HTTP is used to transmit video data. Therefore most websites adopt HTTP chunked transfer encoding. HCBC is developed to recover video data from such network data.

Table 2. Tag data type.

Tag type	First byte of playload	Description
Meta	0x02	Metadata
Audio	0xAF	Audio data
Video	0x17	Keyframe data
	0x27	Interframe data

Firstly, HTTP chunks is scanned byte by byte to search for the FLV file signature “46 4C 56”. When the signature is identified, the data after it is extracted. Secondly, the header and data portion of the HTTP chunks are extracted. Finally, we stop scanning when meeting the end tag. In the reordering process, the fragments are merged into videos based on the extraction order of the chunk data.

In our experiments, it was found that the performance of HCBC is close to that of the file-based transfer method when no packet is missing or corrupted during the capture at gateways. This method can be realized on network data structure, referencing to as HTTP chunked data, without handling complicated file structures, it consumes less time as comparing to FTBC. But as the packet missing rate during capture goes high, it could not extract valid video data as effectively as FTBC.

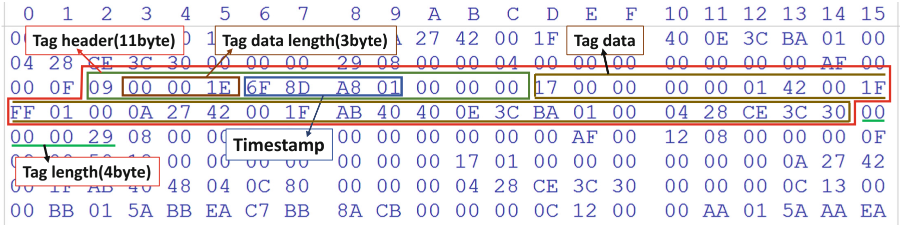
3.3 Video Carving with FTBC

In HCBC, video files are extracted from network streams in a coarse way that it requires the completeness of whole HTTP chunks. However due to the huge mount of data passing through gateways, sometimes it is impossible to capture all the related packets. When captured IP datagrams are incomplete or corrupted [11], HTTP chunks are more likely to become incomplete, and thus the possibility of successfully extracting data will be reduced significantly. Therefore instead of extracting at HTTP chunk level, we extract video at a lower and finer level: FLV file level.

Firstly, video or audio tags are detected and verified in the captured network data. We obtain data parts of HTTP chucks by removing headers. In this way, no header information from HTTP chunks is utilized. Then FLV header is located in the data parts of HTTP chunks by searching for the FLV signature. Video and audio FLV tags could be identified by searching for tag signatures in Table 1 after the FLV header. For instance, if a tag starts with ‘08’, it is audio tag and the following bit stream is audio stream; if it starts with ‘09’, it is a video tag; if it starts with ‘12’, it is a metadata tag. Moreover, according to the FLV structure described in Sect. 3.1, we could access the first byte of tag playload and use it to validate the detected tags. As discussed in Sect. 3.1 and shown in Table 2, for a video tag, the first byte should be either ‘0x17’ or ‘0x27’, for a audio tag, it should be ‘AF’ and for a metadata tag, it should be ‘02’, otherwise the detected tags are invalid. For clarity, an example of video tag is given in Fig. 3, which is highlighted in the red box. It starts with video tag signature ‘09’ and the first eleven bytes within the red box is the tag header. The first byte following the tag header is ‘17’, which confirms the validity of the video tag.

Secondly, video and audio file information is extracted from tag header. It is straightforward to access the size of tag playload once tag is found. For example, as shown in Fig. 3, the three bytes right after the tag signature, *i.e.*,“00 00 1E” indicates the size of tag playload, which is 30 in decimal. Timestamps, which is another important information that will be used in the reordering stage, could be extracted from the four bytes after size bytes of tag data. It is “6F 8D A8 01” in this example.

Finally, the size of tag data is verified. The last four bytes before the next tag denotes the size of whole tag, denoted by LH. At the same time, the size of tag is recorded. Sometimes, video tags could cross multiple network packets as shown in Fig. 4, where block X is the first network packet that a new tag comes up, and block X+n is the last packet it ends with. L_1 and L_n are the bytes that the tag takes in the first block and the last block, respectively. Li,



```
 0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F  10 11 12 13 14 15
[illegible]                  27 42 00 1F [illegible] 40 0E 3C BA 01 00
04 28 CE 3C 30 00 00 00  29 08 00 00 04 00 00 00  00 00 00 00 AF 00
00 0F 09 00 00 1E 6F 8D  A8 01 00 00 00 17 00 00  00 00 01 42 00 1F
FF 01 00 0A 27 42 00 1F  AB 40 40 0E 3C BA 01 00  04 28 CE 3C 30 00
00 00 29 08 00 00 [illegible] 00 00 00 00 00 00 AF 00  12 08 00 00 00 0F
[illegible]  00 00 [illegible] 00 00 00 17 01 00 00 00  00 00 00 0A 27 42
[illegible]  48 04 0C 80  00 00 00 04 28 CE 3C 30  00 00 00 0C 13 00
00 BB 01 5A BB EA C7 BB  8A CB 00 00 00 0C 12 00  00 AA 01 5A AA EA
```

Fig. 3. The structure of video tag

$i = 2, \cdots, n-1$ is the size of block i between the first and last blocks. if $\sum L_i$ equals LH, the detected tag data is correct. Otherwise, the data is corrupted and should be dropped.

When a network packet containing a tag header is lost, two nearby tag data streams could be incorrectly concatenate into a single tag in FTBC. To deal with that case, M is set as the maximum size of a tag. If the size of the inspected tags exceed M, we will drop them. M is critical to the performance of our algorithm. If M is too small, it could falsely drop valid tags. If M is too large, it could be ineffective in detecting invalid tags. Through intensive experiments, our algorithm FTBC could achieve the optimal results when it is set as

$$M = 19 \times \bar{L}_{TCP} \tag{1}$$

where $\bar{L}_{TCP}$ is the average size of TCP packets.

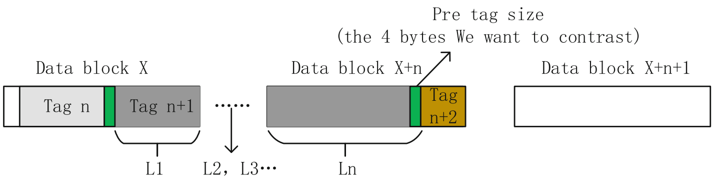


Fig. 4. Calculate the length of tag

In the extraction phase, timestamp is extracted for each video and audio tag. Since according to the timestamp format in FLV file, the first three bytes are low, the fourth byte is high, the correct timestamp could be calculated by recording these four bytes. In the case of Fig. 3, the right order of the timestamp information is "01 6E 8D A8". A packet may contains multiple tags and these tags within the same packet is in sequential order. Therefore instead of sorting tags, it is convenient to sort packets. For the packets containing multiple tags, only the timestamp from the first tag is used for sorting. Finally, tags and the fragments that contain these tags could be sorted according to the value of converted timestamp.

3.4 Adaptive Carving

Both HCBC and FTBC have their advantages and disadvantages. HCBC takes less time, but suffers less recovery rate, while FTBC can still achieve large recovery rate even at large packet loss rate, but consumes more time. That encourage us to take these two methods adaptively to achieve the balance between time and accuracy. A cost function is defined as

$$J = F + \lambda T \tag{2}$$

where F is failure rate that indicates the percentage of tags with failing to recover; T denotes the average time to process a packet. The algorithm that gives the lowest cost in Eq. (2), is the optimal solution we want to adopt. λ is a balance parameter to set the priority between accuracy and time. If λ is 0, accuracy is the only factor we care about when choosing between HCBC and FTBC. Based on our experimental results in Sect. 4, we assume failure rate of both method could be approximated by a linear model to packet loss rate as

$$F_H = \alpha R, \quad F_F = \beta R \tag{3}$$

where F_H and F_F is failure rate of HCBC and FTBC, respectively; R is the loss rate of the packets captured at gateways; α and β are the slopes of the linear models and $\alpha > \beta$.

Decision is made based on the cost of each method, and HCBC should be adopted when its cost is less than that of FTBC as

$$F_H + \lambda T_H < F_F + \lambda T_F \tag{4}$$

where T_H and T_F are the average processing time of a packet in HCBC and FTBC, respectively. By substituting Eq. (3) into Eq. (4), decision could be made based on

$$R < \phi(T_H - T_F) \tag{5}$$

where $\phi = \frac{\lambda}{\alpha - \beta}$ is a pre-setting parameter that sets preference between accuracy and consuming time. When Eq. (5) occurs, HCBC is adopted, otherwise FTBC is adopted. To calculate Eq. (5), R needs to be estimated. TCP packet could be sorted according to their sequence number and then a sliding window with size of n can be used to estimate R.

4 Experimental Results

4.1 Experimental Settings

To evaluate the performance of the proposed methods, an experimental network is set up to simulate the real environment as shown in Fig. 5, 10 virtual machines as users are linked to the Internet via the same gateway. The user machines are running on Microsoft Windows 10 (64-Bit) with IPs from 192.168.1.1 to 192.168.1.10 and the gateway is on Linux (CentOS 6.7) with IP 192.168.1.254. FengYunlive, which is one of the most popular live streaming video website in China, is used as our dataset source and video files from Google Chrome cache is used as original data. The network bandwidth at the gateway is set to 10 Mbps. The network packets use for video carving are captured at the gateway with open source software tcpdump while 3 user machines are watching live videos from FengYunlive, and the rest are browsing random websites. To simulate the packet loss during data capture, the captured network data are randomly deleted in 0%, 10%, 30% and 50% for different experiments. In order to estimate R in this paper, n is set to 1000. By extracting the sequence numbers of the first and the last TCP packets in the window, we are able to calculate the total number of TCP packets m between them if no packet is missing. Finally, the R is calculated

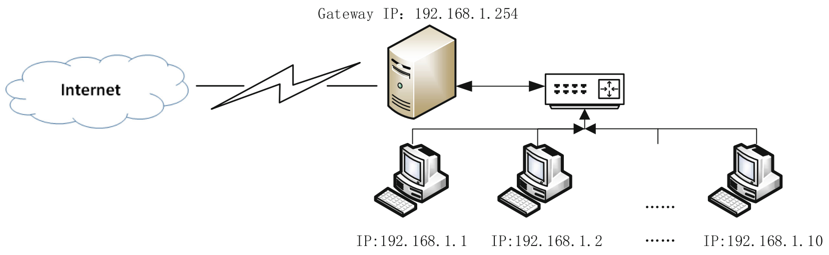


Fig. 5. The environment of experiment

as $\frac{m-n}{m}$. In our experiments, we found T_H and T_F are very stable and thus right side of Eq. (5), *i.e.*, $\phi(T_H - T_F)$ is set as a constant value, 0.07 in our experiments.

The performance of proposed methods was compared with one of similar video carving tools: Wireshark. It extracts video data in a way similar to HCBC. However when a IP datagram is lost or corrupted, it could only recover video data before the lost or corrupted network data and is unable to extract data after that.

4.2 Evaluation Metrics

The performance of the proposed methods are evaluated in three metrics: precision, recall and F-Measure calculate as

$$P(Precision) = \frac{A}{A+B} * 100\% \tag{6}$$

$$R(Recall) = \frac{A}{A+C} * 100\% \tag{7}$$

$$F - value = \frac{2 * P * R}{P + R} * 100\% \tag{8}$$

where A represents the number of true recovered FLV tags, B is the number of false recovered FLV tags, and C is the number of unrecovered FLV tags. F-value is the weighted average of precision and recall. To check if the tag data is restored, the hash values of the restored video tags and the original video tags is compared through the MD5 function.

4.3 Performance Comparison

Three FLV files are tested in our experiments, where each file contains 63991, 23678 and 31231 tags, respectively. Table 3 shows the recall of different methods when there is no packet missing or corrupted in the captured data. HCBC consumes less time than FTBC as expected and they achieves similar performance in recall, 93.73% and 95.66% on average by HCBC and FTBC, respectively. The performance of adaptive carving is exactly the same as HCBC, because HCBC is

Table 3. Experiment results of no packet losing or corrupted

Method	Recovered tags count						Average recall (%)	Time taken (hh:mm:ss)
	test1.flv (63991)	Recall	test2.flv (23678)	Recall (%)	test3.flv (31231)	Recall (%)		
Wireshark	63991	100.00	23678	100.00	31231	100.00	**100.00**	00:13:25
HCBC	61021	95.36	22054	93.14	28971	92.76	**93.75**	00:19:24
FTBC	61821	96.61	22354	94.41	29971	95.97	**95.66**	00:49:48
Adaptive	61021	95.36	22054	93.14	28971	92.76	**93.75**	00:19:24

Table 4. Experiment results of deleting 10% related IP datagrams

Method (Del-10%)	Recovered tags count						Average recall (%)	Time taken (hh:mm:ss)
	test1.flv (60191)	Recall (%)	test2.flv (21308)	Recall (%)	test3.flv (27893)	Recall (%)		
Wireshark	33915	56.35	16131	75.70	19231	68.95	**67.00**	00:13:25
HCBC	54637	90.77	18904	88.72	24601	88.20	**89.23**	00:18:34
FTBC	58318	96.89	20138	94.51	26971	96.69	**96.03**	00:48:18
Adaptive	57492	95.52	19904	93.41	26108	93.60	**94.18**	00:22:59

always preferred in the adaptive method when there is no packet loss. Wireshark achieves the highest recall 100% and consumes the least time.

However when parts of the captured packets are missing, as shown in Tables 4, 5 and 6, the performance of Wireshark decreases dramatically. It only achieves 67.00%, 39.95% and 22.16% on average in recall when packet loss rate is 10%, 30%, 50%, respectively. HCBC also suffers less recovery rate, but its performance

Table 5. Experiment results of deleting 30% related IP datagrams

Method (Del-30%)	Recovered tags count						Average recall (%)	Time taken (hh:mm:ss)
	test1.flv (46703)	Recall (%)	test2.flv (18537)	Recall (%)	test3.flv (22501)	Recall (%)		
Wireshark	13815	29.58	9131	49.26	19231	41.02	**39.95**	00:13:25
HCBC	36021	77.13	12054	65.03	17083	75.92	**72.69**	00:16:43
FTBC	45802	98.07	17913	96.63	21285	94.60	**96.43**	00:42:57
Adaptive	42827	91.70	17231	92.95	20231	89.91	**91.52**	00:33:43

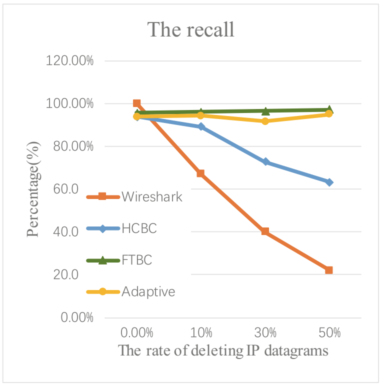
Table 6. Experiment results of deleting 50% related IP datagrams

Method (Del-50%)	Recovered tags count						Average recall (%)	Time taken (hh:mm:ss)
	test1.flv (38095)	Recall (%)	test2.flv (13349)	Recall (%)	test3.flv (17839)	Recall (%)		
Wireshark	5917	15.53	5131	38.44	2231	12.51	**22.16**	00:13:25
HCBC	25917	68.03	8019	60.07	10971	61.50	**63.20**	00:15:02
FTBC	37021	97.18	13054	97.79	17202	96.43	**97.13**	00:38:08
Adaptive	36561	95.97	12563	94.11	16941	94.97	**95.02**	00:28:54

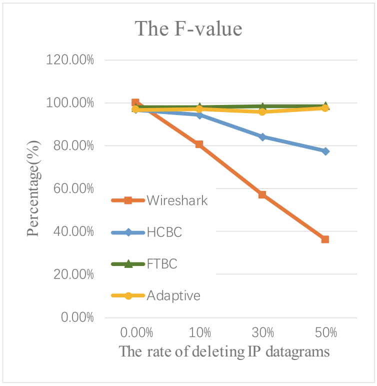
in recall is still better than that of Wireshark. Even though FTBC consumes the most time among all the methods, it could achieve recall higher than 95% in all conditions, because FTBC has traversed all the data as much as possible to extract the video data based on network data structure and FLV structure. As for the adaptive carving, benefiting from adaptively switching between HCBC and FTBC, it could always achieve comparable performance to FTBC in term of recall, but consumes less time, especially at low packet loss rate, *e.g.* 10%.

As measured in precision P in Eq. (6), all methods in all cases achieve 100%, which means there is no false recovery. This is because the redundant information in both network packet header and FLV tag header could effectively prevent false recovery.

Figure 6 shows the curves of recall and F-value versus packet loss rate for all methods, where we can see both FTBC and adaptive methods could maintain high recall and F-value in various the packet loss rates, while recall and F-value of HCBC and Wireshark sharply decrease as packet loss rate increases. In addition,



(a) Recall.



(b) F-Value.

Fig. 6. Recall and F-Value of different methods in various packet loss rate.

in Fig. 6(a), the relation between packet loss and recall of HCBC and FTBC can be both modeled linearly, which verifies our assumption of linear model on the failure rate in Eq. (3).

5 Conclusion

This paper presents a recovery technology for live streaming videos. FLV is a popular video container widely used in live streaming videos and the proposed carving methods target at recovering data from FLV based video network data. First, two different carving approaches are proposed to process network data captured at gateways in different situations. When captured data is complete or with a small scale of data loss, HCBC is able to recover video quickly and accurately. When captured data suffers severe packet loss, FTBC could obtain high recovery rate at cost of high consuming time. To achieve the balance between recovery rate and consuming time, an adaptive method is proposed based FTBC and HCBC. The experimental results verify the effectiveness of the proposed methods as comparing with the prior such as Wireshark.

Acknowledgment. This work is supported by the National Key R&D Plan of China under grant no. 2016YFB0800201, the Natural Science Foundation of China under grant no. 61070212, 61572165 and 61702150, the State Key Program of Zhejiang Province Natural Science Foundation of China under grant no. LZ15F020003, the Key research and development plan project of Zhejiang Province under grant no. 2017C01065, the Key Lab of Information Network Security, Ministry of Public Security, under grant no C16603.

References

1. Zhu, X., Chen, C.W.: A joint source-channel adaptive scheme for wireless H.264 video authentication. In: IEEE International Conference on Multimedia and Expo, pp. 13–18 (2010)
2. Dong, Y.N., Zhao, J.J., Jin, J.: Novel feature selection and classification of internet video traffic based on a hierarchical scheme. Comput. Netw. **119**, 102–111 (2017)
3. Li, J., Bhattacharyya, R., Paul, S., Shakkottai, S.: Incentivizing sharing in realtime D2D streaming networks: a mean field game perspective. In: Computer Communications, pp. 2119–2127 (2016)
4. Espina, F., Morato, D., Izal, M., Magaña, E.: Analytical model for MPEG video frame loss rates and playback interruptions on packet networks. Multimed. Tools Appl. **72**(1), 361–383 (2014)
5. Na, G.H., Shim, K.S., Moon, K.W., Kong, S.G., Kim, E.S., Lee, J.: Frame-based recovery of corrupted video files using video codec specifications. IEEE Trans. Image Process. Publ. IEEE Signal Process. Soc. **23**(2), 517–26 (2014)
6. Zheng, X., Liu, Y., Mei, L., Chuanping, H., Chen, L.: Semantic based representing and organizing surveillance big data using video structural description technology. J. Syst. Softw. **102**(C), 217–225 (2015)
7. Uzun, E., Sencar, H.T.: Carving orphaned JPEG file fragments. IEEE Trans. Inf. Forensics Secur. **10**(8), 1549–1563 (2015)

8. Chen, S., Pande, A., Zeng, K., Mohapatra, P.: Live video forensics: source identification in lossy wireless networks. IEEE Trans. Inf. Forensics Secur. **10**(1), 28–39 (2015)
9. Laing, S., Locasto, M.E., Aycock, J.: An experience report on extracting and viewing memory events via Wireshark. In: USENIX Conference on Offensive Technologies (2014)
10. Wu, T., Petrangeli, S., Huysegems, R., Bostoen, T., De Turck, F.: Network-based video freeze detection and prediction in HTTP adaptive streaming. Comput. Commun. **99**, 37–47 (2016)
11. Matoušek, P., et al.: Advanced techniques for reconstruction of incomplete network data. In: James, J.I., Breitinger, F. (eds.) ICDF2C 2015. LNICST, vol. 157, pp. 69–84. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25512-5_6
12. Ning, J., Singh, S., Pelechrinis, K., Liu, B., Krishnamurthy, S.V., Govindan, R.: Forensic analysis of packet losses in wireless networks. IEEE/ACM Trans. Netw. **24**(4), 1975–1988 (2016)
13. Sommer, P.: File system forensic analysis, B. carrier. Addison Wesley, ISBN 0-321-26817-2. Digit. Investig. **3**(2), 103 (2006)
14. Richard III, G.G., Roussev, V.: Scalpel: a frugal, high performance file carver. In: Refereed Proceedings of the Digital Forensic Research Workshop, DFRWS 2005, Astor Crowne Plaza, New Orleans, Louisiana, USA, August 2005
15. Pal, A., Memon, N.: The evolution of file carving. IEEE Signal Process. Mag. **26**(2), 59–71 (2009)
16. Park, J., Lee, S.: Data fragment forensics for embedded DVR systems. Digit. Investig. **11**(3), 187–200 (2014)
17. Yang, Y., Zheng, X., Liu, L., Sun, G.: A security carving approach for AVI video based on frame size and index. Multimedia Tools Appl. **76**, 3293–3312 (2016)