Efficient zero-copy mechanism for intelligent video surveillance networks

Shiyan Chen¹, and Dagang Li^{2,*}

¹School of Electronic and Computer Engineering, Peking University, China

²The Shenzhen Key Lab of Advanced Electron Device and Integration, ECE, PKUSZ, China

Abstract. Most of today's intelligent video surveillance systems are based on Linux core and rely on the kernel's socket mechanism for data transportation. In this paper, we propose APRO, a novel framework with optimized zero-copy capability customized for video surveillance networks. Without the help of special hardware support such as RNIC or NetFPGA, the software-based APRO can effectively reduce the CPU overhead and decrease the transmission latency, both of which are much appreciated for resource-limited video surveillance networks. Furthermore, unlike other software-based zero-copy mechanisms, in APRO zero-copied data from network packets are already reassembled and page aligned for user-space applications to utilize, which makes it a 'true' zero-copy solution for localhost applications. The proposed mechanism is compared with standard TCP and netmap, a typical zero-copy framework. Simulation results show that APRO outperforms both TCP and localhost optimized netmap implementation with the smallest transmission delay and lowest CPU consumption.

1 Introduction

Video surveillance networks (VSN) has a wide variety of applications both in public and private environments such as homeland security, crime prevention, traffic control, accident prediction and detection, and monitoring patients, elderly and children at home. It can collect and extract useful information from huge amount of videos generated by intelligent cameras that automatically detect, track and recognize objects of interest, understand and analyze their activities [1]. VSN can be regarded as a form of sensor network but with its own characteristics: 1) higher computing, communication, and power resources per node to support more local tasks; 2) largely stationary nodes with an established backbone network [2]. In order to reduce CPU load and decrease data transmission delay, some researchers work on improving image processing algorithm [3] and video compression technology [4], whereas some researchers focus on optimizing network transmission performance, which is more straightforward and effective. This paper focuses on new light-weighting and efficient networking mechanism to save precious CPU resource and reduce network delays in video surveillance system.

* Corresponding author: dgli@pkusz.edu.cn

At present day, the most popular transport protocol used in networked systems is TCP, which can ensure reliable data transportation. However it is too heavy for the scenario of VSN, because of the following reasons: 1) TCP connection setup involves extra packet exchange; 2) TCP congestion control mechanism requires state maintenance and the rate adaptation is not suitable for VSN; 3) data buffering in TCP is too costly [5]. All these cause much CPU overhead and transmission delay in VSN. Some VSN systems use other protocols but they are still based on the socket implementation that provides a unified interface of network stack in kernel, so they still have the same disadvantages of the heavy data structures and complicated processing such as multiple data copy between buffers. Zero-copy can be of much help in this scenario to save CPU resource and reduce network delays, but most existing zero-copy solutions targeted at packet caching or forwarding scenarios and are not optimized to serve local applications. RDMA allows nodes to read or write remote data directly for local user applications without the involvement of CPU, but it requires reliable transport implemented in the network interface cards (called RNICs) and the support for the full set of Infiniband verbs in the kernel, which makes it more befitting in high performance sever clusters but not IOT environment[6][7][8]. There are also complete software-based RDMA implementations such as softROCE and softIWarp, but the performance is nowhere near expected [9]. In addition, some studies tried other ways to extend traditional network protocol stack with zero-copy features, but still require special support of NIC in hardware such as descriptor managing to classify received packets. These hardware-dependent solutions will greatly complicate application development, management and compatibility. Other researchers have been trying to improve traditional protocol stack on general hardware, focusing on simplifying connection setup and removing data copy, both of which could be optimized in the case of VSN[10][11], but they does not have ‘true’ zero-copy solutions for local applications, which seriously affects the performance of local applications when accessing data [12][13].

‘True’ zero-copy on general hardware is very difficult to achieve because of the limited capability of general network interface cards (NICs). General NICs normally can only DMA received packets as they are to local memory, thus for local users it normally involves multiple memory copies to strip off the protocol headers, reassemble data fragments back to a complete chunk, and return to user-space. If we want to avoid costly memory copies and access the scattered and header-included data directly, much effort will be needed from the user applications to lookup and identify their data and access each fragment individually, while traditionally they can just use the assembled data as one piece. It is easy to see that this zero-copy approach has the following problems: 1) The applications need help from auxiliary data structures to manage the scattered data in the network buffers, which increases complexity and accessing delay, and it will get worse when the data volume is large as in the case of videos. 2) The efficiency of the TLB (Translation Lookaside Buffer) will be reduced because of the scattered data. Furthermore, it will decrease data access speed and seriously affect the performance of intelligent applications [14].

We propose an improved zero-copy mechanism called APRO, which includes buffer management and data transport optimized for the VSN scenarios to provide a ‘true’ zero-copy solution. APRO achieves its high performance through the following techniques:

- Instead of DMAing all received packets as they are into a common buffer, the DMA destination is carefully arranged so that data in the packet payload are reassembled into a page-aligned and application-access-friendly way as they are DMAed.
- Combine transport control and buffer management to make sure that data from different senders can be separated and zero-copied into different designated locations of the memory.
- Using overlapped reverse cascading (ORC) or scatter-gather (S/G) DMA to concatenate

segmented payload back into the original header-free data chunk in a zero-copy fashion.

2 The Zero-copy Design of APRO

2.1 The zero-copy model

In traditional systems, the kernel processes network messages and involves content transforming, multiple data copies and context switching, resulting in the high delay between network point-to-points. But in Zero-Copy model, as figure 1 shows, it by-passes the protocol stack. When NIC is doing DMA operation, it directly transmits the data to user application space. The Zero-Copy implementation contains two steps (here we use data packet receiving as an example): The first step is to pre-allocate packet buffers for receiving data. The second step is the address mapping. In the kernel, it allocates sequential space as packet pool, and then it can map this space into user application space by the system procedure mmap. In user application space, it can allocate sequential space as packet pool by the procedure sharemem. The link table (physical address mapping table) is stored in kernel, and is used to implement the user application space to physical address mapping, and then the packet address of descriptor received by network card can be read directly from this table.

Fig. 1. Zero-copy model.

Fig. 2. Data structures of managing the packet pool in netmap.

Fig. 3. Data structures of managing the packet pool in APRO.

2.2 The zero-copy design on VSN

For zero-copy implementation, different procedures of receiving data and sending data add the difficulties to eliminate data copying overheads. The key component in the zero-copy architecture are the data structures managing the packet pool. Netmap, as a typical zero-copy mechanism well-received in the research community, gives user space applications very fast access to network packets. As figure 2 shows, it uses some complicated data structure to provide the following features: 1) amortized per-packet processing overhead; 2) forwarding between interfaces; 3) communication between the NIC and the host stack; and 4) support for multi-queue adapters and multi core systems. But it still follow the common mentality and focus more on the optimization of fast packet forwarding processing. For

data destined to local applications it only provides a sharing pool (red square in figure 2) between the kernel and the user space, and relies on the applications themselves to find the right packets and strip out the useful data from the payload. Besides, it will increase the pressure of memory usage because the scattered data need more physical space to be mapped to the virtual space.

Because memory mapping is normally performed in unit of pages and has alignment requirement but the Ethernet packets have MTU limit of 1500 bytes, so if we want to get continuous received data after memory mapping we should reassemble several packets from the same user in a page. But considering the scenario of VSN in which camera nodes are in general managed in groups and connected to a common server to upload video streams (the servers might again send the videos upstream to a regional server or the control center), we can apply the idea of APRO to solve the problem and get rid of the complex data structures. Because each camera node constantly produces video data, so the server can use polling to pull video data in a round-robin fashion from its connected intelligent camera nodes, so the streams from different cameras won't be intertwined and cut into each other's packet groups, which can also solve the connection problem. Furthermore, with polling we can easily separate streams from different cameras even before they are DMAed from the NIC, so we can prepare separate buffers for each individual camera node, making it easier for the upper layer applications to use them. In each round of polling, the server sends control instruction messages to each camera nodes at intervals to pull video data requesting data size in a range, and the interval will be set according to the delay requirement on transmission. Upon receiving the message, each node starts sending its existing video data produced at a given bit rate, followed by an acknowledge in the last packet group with the info of the remaining data, so the sever knows what to request in the next polling round. As figure 3 shows, the video data from each camera node will be DMAed into the corresponding buffer of that node pre-prepared by the server and the received data is always a big block of continuous user-ready data, without any extra space wasted.

3 The Data splicing Design of APRO

3.1 Data transmission model

Comparing to other zero-copy technologies, the greatest improvement of APRO is that it can ensure the received data to be user-ready without much accessing cost. In order to achieve this, we first figure out how to deal with the headers of network packets in a zero-copy fashion, so that the received data are less fragmented in the receiving buffer.

Figure 4 shows the transmission model of APRO, which presents how it splices data from sending node to receiving node, ensuring it header-free for user-space applications to utilize. Because a network packet always has a header section before the payload, when a train of network packets carrying a big chunk of data are received, even when they are stored next to each other in the buffer, the raw data in the payload will still be separated by the headers. Normal copying can move data fragments into a new buffer to reassemble the data chunk, but to achieve zero-copy we cannot do that anymore. Here in this paper we propose a technique called overlapped reverse cascading or ORC, which provides overlapped packet buffer descriptors to the DMA controller on the network card, so that the tail of the next buffer is overlapped with the head of the previous one, as shown in figure 4. When we send data fragments also in the reversed order, the headers of a packet will be overwritten by the payload of the next packet, so the data fragments of these two packets will be seamlessly concatenated as they should be. If we take the page size to be 4 KB, and only try to maintain the data completeness within a page, then the proposed ORC only

needs to be applied to groups of 3 successive packet, which is much easier to practice - considering Ethernet frames have MTU limit of 1500 bytes. For each group of 3 packets we leave room in the head area of the last packet to carry the state and control info of the whole group. In a very simple environment where packet order can be maintained and packet loss is very rare, the method in figure 4 is very efficient.

Fig. 4. Data path from sender to receiver.

Fig. 5. Organization of received buffers and frames.

Fig. 6. Mapping data from device to user.

3.2 Buffer organization and packets splicing

In this part we will introduce how we organize the receive buffers and splice the received packets so that they can be mapped to a continuous virtual user space. In a zero-copy mechanism, applications will only access their data in the receiving buffer using memory mapping so no data copy is involved. Memory mapping is normally performed in unit of pages and has alignment requirement. To make the receiving buffer ‘mapping friendly’, we divide the receiving buffer in sections of 8KB while each section serves one page of data, as shown in figure 6, if ORC is used to concatenate the payload inside the 3-packet groups. When providing buffer descriptors to the DMA controller, we want to make sure that the start of the data segment in the last packet (that is actually the first segment of the page) is precisely at the middle of the 8KB section, thus 4KB aligned. In this way the second half from the 4KB offset contains exactly one page of data, which is now page-aligned and fully mapping ready; the space of the first half of the 8KB section is used to hold everything else before the data page, containing the packet header and the small part of payload in front of the data fragment in the last packet (of the packet group).

For the example in figure 5, we can divide a page of data into fragments of 1104 bytes, 1496 bytes and 1496 bytes and encapsulate them into the 3 packets of a group in the reversed order. We choose these special sizes because in practice the size of DMAed memory blocks should be multiples of 4, or else many DMA controllers will pad the DMAed block to be 4-byte aligned; on the other hand, the DMA target address should also be 4-byte aligned. If the size is not 4-byte aligned, the automatically added padding bytes will change the size of the DMAed block and make seamless concatenation fail. However, since the size of the Ethernet frame header is 14 bytes and not a multiple of 4, we cannot meet both the 4-byte alignment requirement when performing ORC. This problem is solved

by adding a padding of 2 bytes at the beginning of the payload in front of the data fragment of a 4-byte aligned size, thus making the size of the frame also 4-byte aligned. Back to the example, 1496 is a multiple of 4, and with the 2-byte padding explained above, the total size of the payload is still within the MTU limit of 1500 bytes for a Ethernet frame. When performing the successive overwriting in the ORC, as shown in figure 5, now the overlapping size of the 16 bytes contains the 14-byte frame header and the 2-byte padding at the beginning of the payload, so the concatenation can be perfectly performed. If needed, the 2-byte padding can be used to carry other useful information. As figure 6 shows, because the mapping from physical address to virtual address takes one page as a unit, so any selection of the page-aligned second half from the 8KB sections can be easily mapped to a continuous virtual user space.

4 Performance Evaluation

We run our experiments on system of four camera nodes and one server equipped with a Pentium(R) Dual-Core CPU E5800 @ 3.2 GHz*2, memory of 3.6 GB and dual port gigabit card based on Realtek’s rtl8111 NIC. We use Xilinx zynq-7000 boards as intelligent camera nodes and they have a gigabit Ethernet controller. All equipments run on the latest Linux kernel version up-to-date. We implemented more than 10,000 lines of C code, including control devices and NIC drivers of nodes and server. We test the latency of data size from 4Mb to 256 Mb with different protocols. On APRO and netmap, the timer starts when the server begins to send a control instruction and ends after receiving all requested data. For TCP, we use the interface of socket to transmit data. The CPU load under different throughputs is also tested, comparing to TCP to see how much CPU load could be reduced by APRO. Finally, we test the accessing latency of different data size, and compare the accessing cost between APRO and netmap.

Fig. 7. Latency between camera node and server with different size(log increasing) of video.

Fig. 8. CPU load in camera node with different throughput.

Fig. 9. Comparison of accessing latency(including transmission latency) in server.

We test transmission latency between camera node and server with different sizes of video data. As figure 7 shows, all have almost linear increases in latency, but the latency in APRO is the smallest and about half of that in TCP, while the latency in netmap is only a bit bigger than that in APRO, because we also use polling to optimize and simplify netmap to make a fair comparison, so its time consumption of transmission is very close to APRO’s.

Fig. 10. Accessing latency(without transmission latency) in server of APRO and netmap.

Fig. 11. CPU load of APRO and netmap in server with different throughput.

Fig. 12. CPU load of APRO and netmap in server with different node number.

Suppose that the delay of 30 ms is acceptable (according to the video frame rate), then the largest polling size we can set is about 32 Mb in APRO. 4 Mb polling size have delay of 4 ms in APRO, so when we set polling size with 4 Mb it won't be too small to involve much cost of control instructions or too large with high delay. Figure 8 shows APRO's performance improvement in CPU load on camera nodes. The CPU load of optimized netmap is also very close to that of APRO (because in sending nodes, they have similar zero-copy method to transmit data), so we do not show it in the graph. We can see that the highest throughput TCP supports is less than 512 Mbps, and when the throughput gets higher than 256 Mbps, the CPU load will be more than 50%. But for APRO, even if the throughput gets more than 512 Mbps, the CPU load is still lower than 40%.

We focus on the comparison of data accessing latency of the local applications with APRO and the optimized netmap. Accessing latency is different from the transmission latency because it also includes the time used to allocate and access the data in the receiver memory. Figure 9 shows the comparison of accessing latency (including transmission latency) with the three protocols. The accessing latency in TCP is still the highest. By using TCP protocol, the data in user space is successive, thus the user has low accessing cost, but the delay of data receiving takes a large proportion so the whole latency is still very high. But in optimized netmap, the accessing latency (without transmission latency) increases faster than that in APRO as figure 10 shows. The tendency gets the same obvious in CPU load of server as figure 11 shows. And figure 12 shows with different number of camera nodes, the comparison of CPU load at the server between APRO and netmap, and the server's CPU load of netmap is much higher than that of APRO. In the server node the extra cost in netmap is brought by operations of extracting received data from memory pools and doing the records for user to read, besides, all the data with size of less than a page is scattered in physical space so it needs more mapping space and when reading the data it spends time to locate it. But for APRO, it ensure the received raw data page aligned, and all of them will be mapped to a successive virtual space for the user to access simply and fast.

5 Conclusion

APRO is a high-performance framework designed for the scenario of VSN. It exploits the simple behaviors of video surveillance system in network transmission to replace the heavy TCP with a light-weighted zero-copy mechanism that not only saves precious CPU resources on camera nodes but also reduces delays. APRO is customized for VSN, using polling to retrieve video data from different camera nodes together with a corresponding data splicing method to assure efficient zero-copy data access at the receiving server. APRO is compared to TCP and a VSN-optimized netmap, and results show that APRO has the lowest transmission latency and saves more than 50% of CPU resource, with better accessing performance than TCP and the optimized netmap.

Acknowledgement

This work was supported by Science, Technology and Innovation Commission of Shenzhen Municipality (JCYJ20160525154348175), and the Shenzhen Key Lab Project (ZDSYS20170303140513705).

References

1. X Wang, "Intelligent multi-camera video surveillance: A review," *Pattern Recognition Letters*, vol. **34**(1), pp. 3-19, January (2013)
2. B Rinner, W Wolf, "An Introduction to Distributed Smart Cameras," *Proceedings of the IEEE*, vol. **96**(10), pp. 3-19, October (2008)
3. AA Zarezadeh, C Bobda, F Yonga, M Mefenza, "Efficient network clustering for traffic reduction in embedded smart camera networks," *Journal of Real-Time Image Processing*, vol. **12**(4), pp. 813-826, December (2016)
4. JM Boyce, Y Ye, J Chen, AK Ramasubramonian, "Overview of shvc: scalable extensions of the high efficiency video coding standard," *IEEE Transactions on Circuits & Systems for Video Technology*, vol. **26**(1), pp. 20-34, January (2016)
5. S Li, LD Xu, S Zhao, "The Internet of Things: A survey," *Information Systems Frontiers*, vol. **17**(2), pp. 243-259, April (2015)
6. "Infiniband Trade Organization. Introduction to Infiniband for end users.," <https://cw.infinibandta.org/document/dl/7268>, April (2010)
7. JW Lockwood, N Mckeown, G Watson, G Gibb, P Hartke, J Naous, et al. "Netfpga-an open platform for gigabit-rate network switching and routing," *IEEE International Conference on Microelectronic Systems Education*, San Diego, CA, USA, pp. 160-161, June (2007)
8. S Peter, J Li, I Zhang, DRK Ports, D Woos, A Krishnamurthy, et al. "Arrakis: The Operating System is the Control Plane," *ACM Transactions on Computer Systems (TOCS)*, vol. **33**(4), pp. 11, (2016)
9. G Kaur, M Kumar, M Bala, "Comparing Ethernet & Soft RoCE over 1 Gigabit Ethernet," *International Journal of Computer Science & Information Technolo*, vol. **5**(1), pp. 323, January (2014)
10. L Rizzo, "netmap: a novel framework for fast packet I/O," *Usenix Conference on Technical Conference*. USENIX Association, Berkeley, CA, USA, pp. 9-9, June (2012)
11. Compaq Computer Corp., Intel Corporation, and Microsoft Corporation, "Virtual Interface Architecture Specification," version 1.0 edition, December (1997)
12. A Belay, G Prekas, A Klimovic, S Grossman, C Kozyrakis, E Bugnion, "IX: A protected dataplane operating system for high throughput and low latency," *Usenix Conference on Operating Systems Design and Implementation*, Broomfield, CO, pp. 49-65, October (2014)
13. EY Jeong, S Woo, M Jamshed, H Jeong, S Ihm, D Han, et al. "mTCP: A Highly Scalable User-level TCP Stack for Multicore Systems." *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, Seattle, WA, USA, April (2014)
14. B Pham, A Bhattacharjee, Y Eckert, G H Loh, "Increasing TLB reach by exploiting clustering in page translations," *International Symposium on High Performance Computer Architecture*, Orlando, FL, USA, pp. 558-567, February (2014)