



Real-time pre-processing system with hardware accelerator for mobile core networks^{*#}

Mian CHENG, Jin-shu SU[‡], Jing XU

(College of Computer, National University of Defense Technology, Changsha 410073, China)

E-mail: cm@nudt.edu.cn; sjs@nudt.edu.cn; jing.xu@nudt.edu.cn

Received Aug. 2, 2017; Revision accepted Sept. 30, 2017; Crosschecked Nov. 23, 2017

Abstract: With the rapidly increasing number of mobile devices being used as essential terminals or platforms for communication, security threats now target the whole telecommunication infrastructure and become increasingly serious. Network probing tools, which are deployed as a bypass device at a mobile core network gateway, can collect and analyze all the traffic for security detection. However, due to the ever-increasing link speed, it is of vital importance to offload the processing pressure of the detection system. In this paper, we design and evaluate a real-time pre-processing system, which includes a hardware accelerator and a multi-core processor. The implemented prototype can quickly restore each encapsulated packet and effectively distribute traffic to multiple back-end detection systems. We demonstrate the prototype in a well-deployed network environment with large volumes of real data. Experimental results show that our system can achieve at least 18 Gb/s with no packet loss with all kinds of communication protocols.

Key words: Mobile network; Real-time processing; Hardware acceleration

<https://doi.org/10.1631/FITEE.1700507>

CLC number: TP309.2

1 Introduction

The past decade has witnessed huge growth in the popularity of mobile devices, but this creates significant network security issues. Security incidents that previously occurred only in conventional networks are now occurring in mobile communication networks, such as explosive worms, viruses, and distributed denial-of-service (DDoS) attacks. The openness of the mobile network makes application developers and interactive business more accessible to core networks and databases. Thus, effectively monitoring and processing security incidents that occur on the mobile Internet is an important challenge

for network managers.

Real-time collection of network traffic is important for network service providers. In addition to monitoring malicious behaviors, collected traffic can be used to predict possible traffic conditions and optimize the network environment through selection of optimal routing paths and configuration of load-balancing policies. When connecting to a high-speed link, most detection systems use server cluster technology that collaboratively analyzes a traffic stream without sacrificing detection accuracy and must resort to specialized hardware for front-end traffic distribution to a set of back-end servers (Vallentin *et al.*, 2007). However, the size of the cluster will be very large and incur greater costs when transmission speeds are improved. Although many researchers focus on the optimization of the packet capture engine through software-based solutions (Vasiliadis *et al.*, 2011; Rizzo, 2012; Cisco Systems, Inc., 2013) or hardware acceleration (Han *et al.*, 2010; Intel

[‡] Corresponding author

* Project supported by the National High-Tech R&D Program (863) of China (No. 2012AA013002)

A preliminary version was presented at the 6th International Conference on Instrumentation, Measurement, Computer, Communication and Control, July 21–23, 2016, China

ORCID: Jin-shu SU, <http://orcid.org/0000-0001-9273-616X>

© Zhejiang University and Springer-Verlag GmbH Germany 2017

Products, Inc., 2010; Peemen *et al.*, 2013; Kekely *et al.*, 2014), these approaches are still limited to only a few common protocols (e.g., HTTP) and cannot satisfy the requirements of mobile networks.

Different access networks are consolidated to connect the radio access network and the traditional Internet in the mobile core network. The 3GPP2 standard (China Communications Standards Association, Inc., 2006) sets several packet encapsulation protocols (e.g., GPRS tunneling protocol (GTP) and generic routing encapsulation (GRE)) and compression structures for packet transmission optimization. The packet structure determines the complexity of the process, and advanced analysis of the message content (e.g., deep packet inspection) consumes much of the processing capability. With the constantly increasing of network bandwidth, high-performance real-time collection of network traffic in a limited time and without loss is still a challenge.

In Cheng *et al.* (2016), we have presented a simplified system for mobile core network measurement, which sketchily introduced a two-level pre-processing mechanism for collecting and distributing packets. However, it was not well established. To further refine our research, in this paper, we propose a real-time pre-processing system with a hardware accelerator for mobile core networks. The implemented prototype can quickly process each encapsulated packet and effectively distribute the restored packet to back-end servers. Fig. 1 gives the overview of our implemented pre-processing system architecture. The packet processing procedure has two components. First, the hardware accelerator, based on a field-programmable gate array (FPGA), performs processing which has a simple calculation procedure but consumes plenty of computing resources, such as packet decapsulation and character transformation of PPP frames in the CDMA2000 core network. Then for those operations that require significant resource overhead, such as packet decompression and recombination, we use a multi-core processor to significantly improve the processing capacity in a high-speed mobile core network. The hardware accelerator uses two distribution approaches to satisfy different packet distribution requirements from back-end servers. In most cases, we use the five-tuple in each packet, and flows are split roughly equally across equal length paths. If a suspicious user must be tracked, a distribution approach based on the user's

terminal address is proposed to make ensure the integrity of the bidirectional data stream. The main idea is the cooperation between the hardware accelerator and the multi-core processor to identify the user terminal IP address of each data packet. Experiments on a real dataset demonstrate that our processing prototype can provide high throughput and can be applied to all kinds of International Telecommunication Union (ITU) standards, including CDMA2000, WCDMA, TD-SCMDA, and LTE.

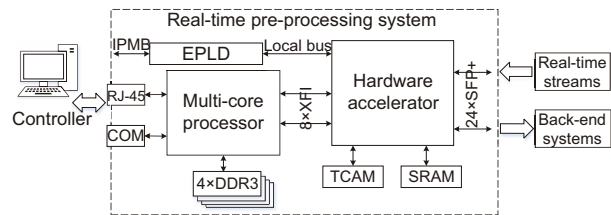


Fig. 1 Structure of the real-time pre-processing system

2 Related work

In this section, we list some packet processing engines. Section 2.1 introduces the software-based solutions, which have achieved significant performance improvements in packet capture compared with the Linux native approach. Section 2.2 summarizes the hardware acceleration approaches, which give us a better way to implement packet processing. Section 2.3 points out the advantages of the hardware method and proposes our research direction.

2.1 Software-based solutions

Software-based solutions often focus on parallel processing or optimization of processing architecture based on the traditional CPU. In Rizzo (2012), NETMAP was proposed to optimize the packet capture engine. It adopts batch processing and pre-allocates a fixed cache space (2048 bytes) in the initialization phase. NETMAP implements memory-mapped technology to allow applications to directly access the metadata structures in the kernel package, which is called NETMAP-RING. Note that the receipt and delivery of each receive sliding (RSS) queue can use the NETMAP-RING to directly achieve parallel data channels. On the other hand, conventional computing platforms usually use an inefficient raw socket or a packet capture (PCAP)

interface to transmit packets to the user application level. MIDeA (Vasiliadis *et al.*, 2011) adopts PF_RING to quickly pass the packet to the user layer space and runs the snort intrusion detection system on a common multi-core computing platform. Experimental results show that the system can achieve 5.2 Gb/s network packet processing capability by using the multi-queue and GPU accelerator technologies. After receiving the packet, MIDeA invokes snort's decoding engine to process the content of the packet, and then passes packets to the GPU for intrusion detection. The data packet distribution tool in MIDeA is a hardware component. Compared with the pipelined model composed of traditional processing cores, the hardware solution is more efficient and the whole system has better scalability.

In a mobile core network, some access servers are implemented by software. For instance, the Cisco ASR 500 Series packet data serving node (PDSN) product's (Cisco Systems, Inc., 2013) packet processing card is implemented by a 2.5-Hz, quad-core x86 architecture processor and 16 GB of RAM, supporting a maximum of 2 million PDSN sessions, with a maximum processing capacity of 5 Gb/s. The multi-core processor can be seen as an integrated multiple core system on chip (SoC). Each core is a separate operation unit, and has its own separate instruction, data cache, and operating system scheduler. We can execute each core business concurrently without disturbing other units and set up multiple threads on each core.

2.2 Hardware acceleration

With the rapid growth of network bandwidth, the performance of software-based solutions was unable to satisfy line-speed packet collection requirements. Hardware acceleration based on FPGAs, application-specific integrated circuits (ASICs), and graphics processing units (GPUs) are of interest to system developers. The main idea of these commodity components is to offload a part of CPU packet processing functions to the hardware to improve performance.

In the field of custom chips, Intel proposed a high-speed packet processing platform, Crystal Forest (Intel Products, Inc., 2010). The structure of the 'multi-core CPU + dedicated accelerator' uses an ASIC to implement the acceleration function. It can effectively undertake some complicated packet

processing functions, offload the pressure of the multi-core processor, and improve overall processing efficiency. However, ASIC's shortcomings include expensive customization and lack of scalability. On the other hand, many studies have recently experimented with the GPU to accelerate packet processing in network applications. This provides a significant performance boost when compared to the CPU-only solution. Han *et al.* (2010) proposed PacketShader, which adopts a heterogeneous packet processing architecture of 'CPU+GPU' to accelerate packet processing. This approach offloads the packet protocol processing functions to the GPU, including IPv4 and IPv6. The CPU is responsible for only receiving and sending the packets. Cavigelli *et al.* (2015) presented a convolutional network accelerator that is scalable to network sizes that are currently handled by only workstation GPUs, but remains within the power envelope of embedded systems. It can significantly improve the external memory bottleneck of previous architectures, is more area efficient than previously reported results, and comes with the lowest-ever reported power consumption when including I/O power and external memory. Recently, Go *et al.* (2017) discussed eight popular algorithms widely used in network applications and suggested employing integrated GPU in recent accelerated processing unit (APU) platforms as a cost-effective packet processing accelerator. Results demonstrate that network applications based on APUs can achieve high performance (over 10 Gb/s) for many computation- and memory-intensive algorithms. However, the consumption of the GPU-based accelerator is too high, causing unnecessary waste.

FPGA is more flexible and scalable than other network acceleration. It can be adjusted according to different platform requirements, not only to satisfy high-speed Internet traffic processing needs, but also to reduce energy consumption. Kekely *et al.* (2014) proposed an optimization method called software defined monitoring (SDM), which is based on a configurable hardware accelerator implemented in FPGA and some smart monitoring tasks running as software on a general CPU. SDM is an optimization of a flexible flow-based network traffic monitor that supports application protocol analysis. Lavasani *et al.* (2014) presented a method for accelerating server applications using a hybrid 'CPU+FPGA' architecture.

It processes request packets directly from the network and avoids the CPU in most cases. Fast-path and slow-path techniques were proposed to speculatively execute a hot path by slicing the application and generating the fast-path hardware accelerator. Neil and Liu (2014) introduced an event-driven neural network accelerator. It can be integrated into existing robotics or it can offload computationally expensive neural network tasks from the CPU. Researchers have also proposed FPGA-based accelerators to compute over a billion operations per input image (Peemen *et al.*, 2013; Zhang *et al.*, 2015).

2.3 Comparison

Subject to the constraints of CPU processing capacity, a software-only solution no longer satisfies the requirement of core network processing. In the above hardware acceleration approaches, FPGAs are less costly and more scalable than ASICs. In the meantime, FPGAs have better flexibility than GPUs. According to the packet characteristics in the mobile core network, we propose an FPGA+multi-core processor architecture to effectively process encapsulated and compressed packets.

3 System architecture

Fig. 2 depicts the architecture of the proposed real-time pre-processing system with a hardware accelerator. The system consists of a multi-core processor, a hardware accelerator, and peripheral units such as ternary content-addressable memory (TCAM) and static random access memory (SRAM). The hardware accelerator and multi-core

processor are implemented on an Alterla Stratix V GX FPGA and a Broadcom XLP432 processor, respectively. To reduce the processing time, we use a two-stage structure to handle packets from the mobile core network. Upon receiving a packet, the hardware accelerator performs the processing which has a simple calculation procedure but consumes plenty of computing resources, such as escape character decoding and packet decapsulation. Then the pre-processed packet is transferred to the multi-core processor for in-depth processing. The multi-core processor performs the complicated calculation procedure, such as fragment reassembly, decompression of Van Jacobson (VJ) compressed packets, and internal frame reassembly. A multi-threading approach and an MIPS64-based architecture in the processor guarantee high performance. When the packet feeds back to the hardware accelerator, the distribution based on pre-set rules is done by the hardware accelerator, TCAM, and SRAM. A standard RJ-45 and a COM connector are deployed to allow the administrator to control the system for rule issuing, system monitoring, and debugging.

3.1 Deployment scenario

The key components of the 3G and LTE networks and the deployment of our pre-processing system are illustrated in Fig. 3. In WCDMA and TD-SCDMA packet core networks, we deploy our system to process packets through all Gn links between the serving GPRS support node (SGSN) and the gateway GPRS support node (GGSN). From the Gn interface, we can analyze a large volume of user business data and control signaling such as PDP activate

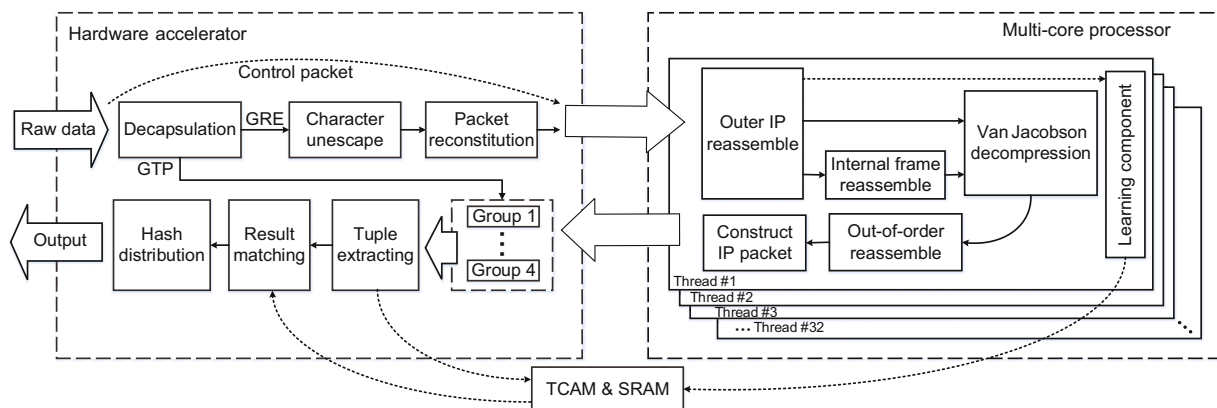


Fig. 2 Block diagram of the proposed real-time pre-processing system with a hardware accelerator

signaling, route update information, location area update information, and mobile management information. In the CDMA2000 packet core network, we do the same thing through all A10/A11 links between the packet control function (PCF) and the PDSN. There are many differences between LTE and 3G in both network structure and wireless technology. We receive packets from interfaces S1-MME, S11, and S1-U.

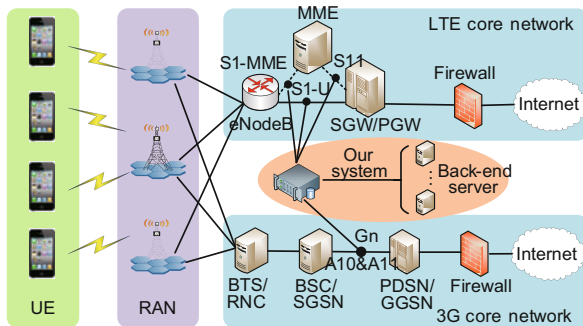


Fig. 3 Deployment in the 3G and LTE core network

3.2 Hardware accelerator

To accelerate processing, the hardware accelerator performs processing which has a simple calculation procedure but consumes plenty of computing resources. The parallel architecture of the FPGA means it can act as an extremely effective offload engine to relieve CPU bottlenecks. Shifting critical code to the FPGA and running those algorithms using multiple streaming processes in the FPGA can provide overall acceleration of 10 times or even more over CPU-only solutions. As shown in Fig. 2, we first assign packet decapsulation and character transformation to the hardware accelerator for pre-processing and reconstitute a custom packet structure for direct interaction between the hardware accelerator and multi-core processor. After receiving the restored packet, several distribution strategies are ready to allocate each packet to the back-end servers.

3.2.1 Packet processing acceleration

Processing acceleration is the most critical function of our system. First, to filter the packet that contains no user information, we discard all non-GRE and non-GTP packets, which may account for nearly 50% of the total traffic before process-

ing. Then we process different communication standards in different approaches. In the WCDMA, TD-SCDMA, and LTE core networks, the user packet is directly encapsulated in the GTP header and it is simple to decapsulate and extract the user information. Hence, the hardware accelerator will process all GTP data packets and directly forward the internal user packet to the back-end servers through a distribution strategy without any processing in the multi-core processor. In practice, we always need to manage a lot of traffic from different operators at the same time. Thus, the complete treatment of the GTP packet in the hardware accelerator can remove significant processing pressure from the multi-core processor, and provide better performance in processing complicated packets in CDMA2000.

1. Character transformation

In the CDMA2000 EV-DO core network, the user packet is encapsulated into a PPP frame and transmitted through the GRE tunnel. One GRE packet may encapsulate multiple PPP frames and each frame may contain incomplete IP packets. To distinguish each individual PPP frame, RFC 1661 (Internet Society, Inc., 2014) defines that when 0x7e and 0x7d occur in the information field, they are converted to 2-byte sequences 0x7d5e and 0x7d5d, respectively. Therefore, it is necessary to scan the entire PPP frame to complete the character transformation to restore the inside user packet. The conventional algorithm, which usually runs in software, is to process PPP packets sequentially and detect if there is a special character 0x7e or 0x7d. Apparently, the overall scanning will generate large amounts of overhead (the time complexity is $O(n)$, where n is the length of the PPP packet). Specifically, when the server receives many data streams, the processor has to spend great effort on scanning every byte, which is the biggest bottleneck restricting CPU processing performance. In addition, in a 2^n -bit CPU platform ($n \geq 3$), the range of data width is $\{1, 2, \dots, 2^{n-3}\}$ bytes. For example, a 32-bit or a 64-bit CPU can read-write only four or eight bytes, respectively. Therefore, we use the parallel feature of the FPGA to improve processing efficiency. In the hardware environment, the data width can be defined according to the availability of resources. Algorithm 1 proposes a 16-byte parallel approach on FPGA to improve the efficiency of the overall scanning.

As shown in Algorithm 1, we define $B_c = (B_0, B_1, \dots, B_{15})$ as the current 16-byte stream from the input first in, first out (FIFO) to the register, and B_p and B_n as the previous and next 16-byte streams. The FIFO in the hardware accelerator can hold at least one packet (assuming a maximum of 2048 bytes). In lines 1–11, the following conditions are processed. In line 1, if there is no 0x7d in B_c , write the bytes into data alignment component and read B_n into the register. If B_1 ends at 0x7d (line 4), then the first byte needs to be transformed before writing to the data alignment component. Finally, if any other byte is 0x7d (line 7), we write the current byte b to the $(b + 1)$ byte after transforming, and shift each byte from 0 to $(b - 1)$ by one bit. To reduce the logic complexity, each cycle deals with only one 0x7d, and if there are multiple 0x7d in B_c , we process sequentially from right to left. After transformation, we need to remove the invalid bytes before writing to the FIFO. As shown in Fig. 4, the data alignment component is composed mainly of a padding state machine, a read buffer state machine, and two sets of registers (Buffer0 and Buffer1). The padding state machine receives data containing a ‘blank’ from the character transformation component, and then completes data aligned with Buffer0 and Buffer1. The read buffer state machine sequentially reads the padded data and writes it to the FIFO. Throughout this process, we configure FIFO as 256×134 bits, of which two bits are used as flag

bits and another four bits represent the number of invalid data bytes; the data width is 128 bits.

The performance of the algorithm is analyzed below. From the above analysis, we can see that the performance bottleneck is the character transformation component. For one channel of the FPGA, we calculate the theoretical processing throughput P as

$$P = \frac{B_r}{N \cdot (1/M)}, \quad (1)$$

where M is the frequency of the FPGA, N is the number of clock cycles, and B_r is the maximum number of bytes that can be processed in parallel in each cycle (i.e., 16 bytes). Because we can process only one character at a time, $N = n + 1$, where n is the number of 0x7d. In other words, the frequency of the appearance of 0x7d determines the processing performance. In practice, we assume 0x7d and 0x7e appear randomly, then the average expected value E of the number of 0x7d is 0.125. In general, when $n = 0.125$, P is about 20 Gb/s. Altogether we configure eight channels to run the above task. So, it is easy to implement line-speed processing.

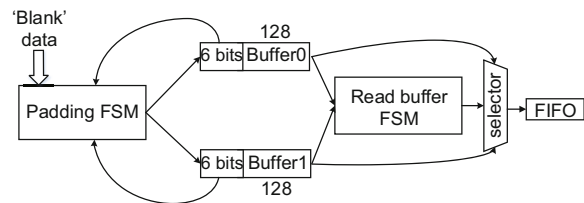


Fig. 4 Data alignment component block diagram

Algorithm 1 Parallel character transformation

Input: addr_ppp: the starting address of PPP packet;
invalid_len: length of invalid byte.

Output: len_ppp: the length of PPP packet after decapsulating.

- 1: if $0x7d \notin B_c$ then
 - 2: $B_c \rightarrow$ data alignment
 - 3: end if
 - 4: if $B_{p,16} = 0x7d$ then
 - 5: $B_{c,1} \oplus 0x20$ and $invalid_len + 1$
 - 6: end if
 - 7: if $B_{c,b} = 0x7d$, $1 \leq b \leq 14$ then
 - 8: $B_{c,b+1} \oplus 0x20 \rightarrow B_{c,n}$
 - 9: $(B_{c,b+2}, \dots, B_{c,16}) \ll 1$
 - 10: $invalid_len + 1$
 - 11: end if
 - 12: if $B_{c,15} = 0x7d$ then
 - 13: $B_{c,16} \oplus 0x20 \rightarrow B_{c,15}$
 - 14: $invalid_len + 1$
 - 15: end if
-

2. Packet reconstitution

As we described above, we perform the decapsulation of GRE and GTP tunnel packets and character transformation of the PPP packet. However, the encapsulation header actually contains some important information for association of the user session, such as the source and destination IP addresses, the GRE key, and the tunnel endpoint identifier (TEID) number. As depicted in Fig. 5, we add an extra header to transfer information between the hardware accelerator and the multi-core processor. It reduces the processing pressure by decreasing the length of the packet while ensuring information integrity. Then the control packet will directly transfer to the multi-core processor for gateway address learning, which we will describe in Section 3.3.2.

3.2.2 Distribution strategy

In addition to packet processing acceleration, we combine the hardware accelerator with TCAM and SRAM to build a quick packet distribution mechanism. In practice, our system receives packets from multiple operator networks. Thus, when the distribution step begins, we define four groups to distribute packets from different mobile operators to different back-end servers. Each group has a corresponding output port list, which can be pre-configured. Moreover, our system provides three distribution approaches to allocate each packet to back-end servers. In most cases, we use the five-tuple in each packet, and flows will be split roughly equally across paths of equal length. If a specific user must be tracked, we propose a distribution approach based on the user's terminal IP address to ensure the integrity of bidirectional data. The multi-core processor restores the IP packet and extracts the PDSN/GGSN address. By identifying the location of the PDSN in the IP address field, the transmission direction of the packet can be uniquely determined. Then we can identify the terminal IP address from the internal IP packet based on the transmission direction. We use the gateway IP address to allocate the control packet because the back-end server needs to extract the international mobile subscriber identification (IMSI) number and associate it with the user data. Based on the above approach, the hardware accelerator calculates the hash function to match the corresponding back-end server, and then rewrites the destination MAC address of each packet.

3.3 Multi-core processor

With the rapid increase in mobile core networks, the traditional general-purpose CPU has many performance bottlenecks in the packet processing procedure, which cannot meet the demands of a high-speed network environment. Thus, we implement

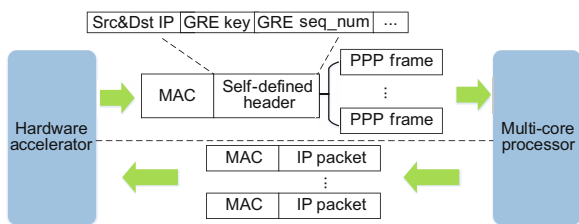


Fig. 5 Structure of self-defined packet transmission

some functions on a Broadcom XLP432, which is an SMP network processor consisting of 8 identical cores with 32 threads, and a shared 8-MB L3 cache. After the hardware accelerator performs packet pre-processing, the multi-core processor performs further in-depth processing. Our goal is to extract and restore the IP packet in each PPP frame and reorganize the possible IP fragmentation. The whole procedure consists of outer IP reassembly, decompression of the VJ compressed packet, internal frame reassembly, and out-of-order packet reassembly. These operations have complex processing logic, so it can hardly be implemented on an FPGA.

In the multi-core processor, there are several packet capture engines based on zero-copy and multi-core technology. The fast messaging network (FMN) is an important part of the multi-core processor. As depicted in Fig. 6, the FMN station is the connection point between the various functional units of the processor and the FMN message ring. The multi-core processor connects the functional units through the FMN station. Each physical core of the multi-core CPU has a corresponding station on the FMN (e.g., the XLP432 has eight physical cores, so there are eight stations and one-to-one correspondence). Each station has eight buckets; each bucket has a global bucket ID, which is unique in the FMN message ring. If the station sends a message to a bucket, it will be placed in the corresponding bucket according to the purpose of the message, and each thread or interface will read the message from the bucket and process it. Based on the above technology, we first complete the reorganization of the outer IP fragment. Then for a packet that has a PPP fragment, we do reorganization and then decompress the VJ packet to restore the user data. Finally, we deal with the out-of-order packet. In the real link, we find that the

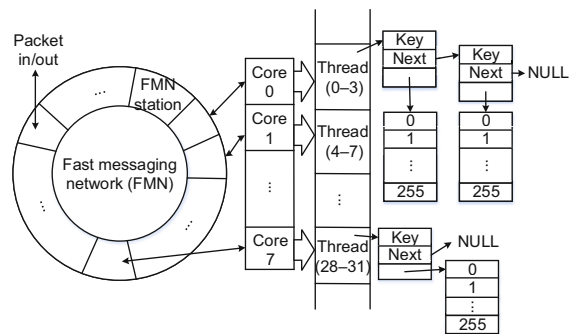


Fig. 6 Processing of the fast messaging network

outer IP fragment accounts for less than 3% of the total packet stream. However, the compressed VJ packet will account for more than 20%. Hence, the processing efficiency of VJ packets will greatly affect the processing performance of the whole system. In the next section, we introduce the procedure for decompressing the VJ packet.

3.3.1 Decompression strategy

Due to the limited bandwidth of the air link, information content is often transmitted using Van Jacobson TCP/IP header compression technology. With VJ compression, the TCP payload is encapsulated in the PPP protocol, and other information fields are omitted (including the source and destination IP addresses and the port number). However, without restoring the raw packet, it is difficult to monitor and analyze user traffic.

To take full advantage of the multi-core processor, we use 30 independent threads to process packets in parallel and maintain a TCP header flow table, which is divided into 30 equal partitions. As depicted in Fig. 6, the rightmost list contains the unique identity (source, destination IP, and GRE key) of the TCP connection, the next pointer, and the TCP header buffer pointer, pointing to the space for the different TCP connections, which contain 256 connection numbers. Due to these indexes, packets are split into each thread and do not access the table space of different threads, which means that the flow tables operate independently of each other. Meanwhile, different threads do not need to use a shared variable lock mechanism, so there is no delay between threads. Each thread independently completes the decompression without any communication between threads. Therefore, 30 message processing threads can be fully parallel.

When processing the PPP frame, the multi-core processor calculates the index value of the header table according to the source IP, destination IP, and GRE key of the tunnel packet (the session uses the GRE key to identify different PPP connections), and then obtains a unique TCP connection according to the connection number. For uncompressed TCP packets, we update the header of the TCP connection, modify the protocol field, and directly forward the packet. For compressed packets, the header is restored according to the header of the TCP connection.

3.3.2 Gateway address learning

This is another function that involves cooperation between the hardware accelerator and the multi-core processor. In a CDMA2000 mobile core network, the PDSN serves as the bridge that connects the mobile core network and the TCP/IP Internet. Similarly, in a WCDMA/TD-SCDMA mobile core network, the above functional entity is called the GGSN. Each packet transferred through the mobile core network encapsulates the IP address of the PDSN/GGSN in its GRE/GTP header, and we call it the gateway address. In our system, we use the gateway address to achieve control packet distribution and identify the user terminal IP address as the input key value of the distribution.

As depicted in Fig. 7, by identifying the gateway address, the transmission direction of each packet can be uniquely determined. Then we modify the source MAC address of each data packet by adding a flag bit to indicate the transmission direction. When the hardware accelerator receives the restored packet, the flag bit indicates the user's terminal address. For instance, if the flag bit indicates that it is an uplink packet, the source IP address is the user's terminal address. Meanwhile, the control packet is transmitted back to the hardware accelerator and distributed through the gateway address.

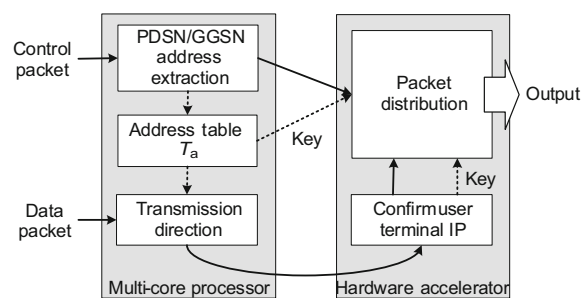


Fig. 7 Gateway address learning scenario

3.4 Available extension

Due to the flexibility and scalability of the multi-core processor and FPGA, our system can also provide serial access to the mobile core network for identifying, detecting, and forwarding multiple application protocols, such as VoIP, HTTP, and email. Our system helps discover unsafe content and malicious behaviors in the network, and plays an important role in spam detection, traffic statistics, and attack

a period of 5 h from 11:00 a.m. to 4:00 p.m. in Hunan Province on April 21, 2011. We collected 2.69 billion packets, corresponding to 1.5 TB of TD-SCDMA traffic. Similarly, we collected data from the A10/A11 interface of a CDMA2000 core network in Hebei Province and obtained 45.2 million packets, corresponding to 435 GB of CDMA2000 traffic. Table 1 shows the statistical analysis of different types of PPP frames in the CDMA2000 dataset. We can observe that more than 70% of the GRE encapsulation packets contain only one PPP frame, including 30.99% of the complete IP packets, which can be directly forwarded, 13.97% of the VJ compressed packets, and 37.43% of the IP fragments. Furthermore, according to our statistical result, the VJ compressed packets in the core network can account for 23%–25% of the total number of packets after recombination of the fragments. The remaining 30% indicates a GRE packet that contains multiple PPP frames, and most of them involve at least one fragment (14.72% of the total number of packets).

Table 1 Proportion of each type of packet

Packet type	Percentage	Packet type	Percentage
Single complete	30.99% (IP)	Single piece	37.43%
	13.97% (VJ)	Multi complete	2.53%
	0.36% (others)	Multi piece	14.72%

In practice, due to the low utilization of the real core network, a realistic environment cannot provide sufficiently large flow for our evaluation, and we used an evaluation tool to simulate the real mobile core network. Connected to a four-port optical splitter, we can generate a maximum input rate of 40 Gb/s. The experimental environment is shown in Fig. 10.

4.1 Performance evaluation among different types of packets

To more accurately assess, we chose three different types of packets from the CDMA2000 dataset to evaluate the performance bottleneck. Fig. 11 shows the packet loss of our system in case of decompression of the 64-byte VJ compressed packet, the reorganization process of multi-internal fragments in one GRE packet, and the reorganization process of internal fragments across two different packets. The main reason for choosing these packets is because of their frequent appearance in mobile core networks. According to our statistics, VJ compressed packets and

internal fragments account for about 26% and 52% of overall traffic, respectively. The decompression procedure requires caching of the VJ uncompressed packet while the subsequent VJ compressed packet is decompressed. Because the existing packets are out of order, multi-internal fragments and internal fragments cross two different packets and consume a large number of resources while waiting for reorganization. In experiments, we extracted three samples of the above packet types from the CDMA2000 dataset. Although each sample has only a few packets, we used IPRO to play back 10^6 times to evaluate the performance of our system with or without the hardware accelerator. As depicted in Fig. 11, because the hardware accelerator accomplishes the processing of packet decapsulation and character transformation, results indicate that the performance of our system improves by 27%, 57%, and 50% when dealing with the above three packet types.

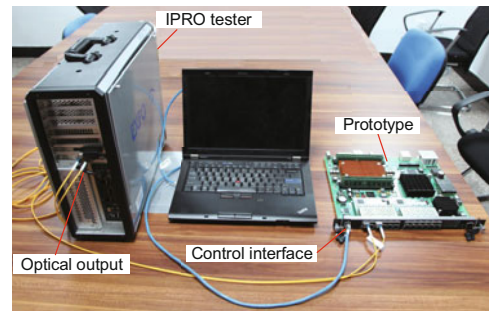


Fig. 10 Demonstration environment

4.2 Performance evaluation of the overall system

We used IPRO to play back the overall CDMA2000 and TD-SCDMA traffic at different rates. To estimate the performance in the CDMA2000 core network, we ran an experiment on a typical server running Ubuntu Linux with a 3.5-GHz Intel Core i7 processor and 32 GB of memory to compare the performance with our system. Fig. 12a indicates the distribution of packet length of the CDMA2000 dataset. We can observe that almost 54% of packets were longer than 512 bytes and the average length of a packet was 612 bytes. As illustrated in Fig. 12b, when the incoming packet rate is increasing, the typical server starts to drop packets at 450 Mb/s, and the packet loss rate reaches 65% when the incoming packet rate reaches 1 Gb/s.

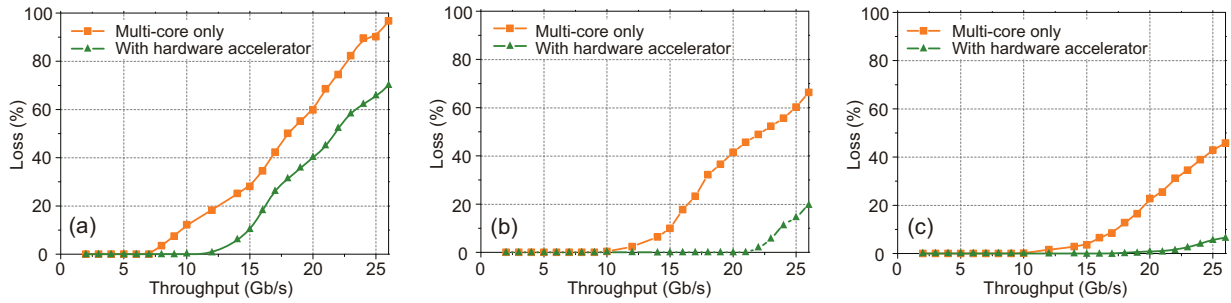


Fig. 11 Packet loss experiment on different types of packet: (a) 64-byte VJ; (b) multi-internal fragment; (c) cross fragment

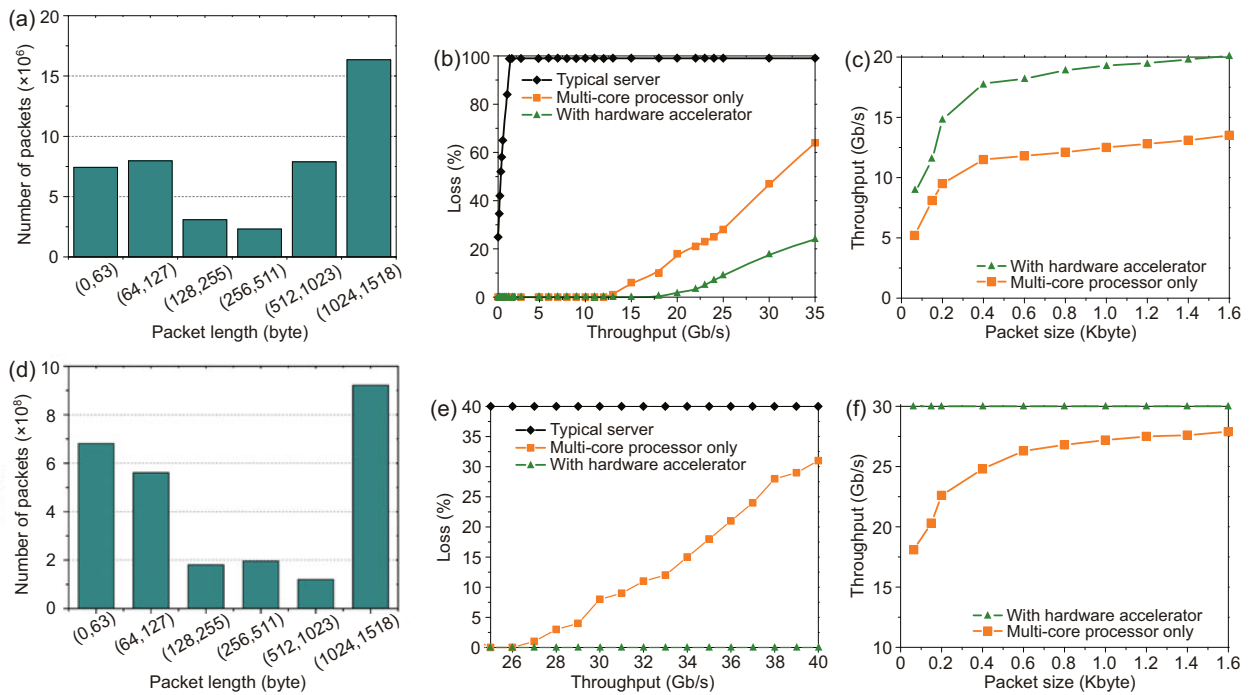


Fig. 12 Results of system performance: (a)–(c) are the packet size distribution, packet loss rate, and throughput in the CDMA2000 core network, respectively; (d)–(f) are the packet size distribution, packet loss rate, and throughput in the TD-SCDMA core network, respectively

Similarly, the multi-core processor drops packets at 13 Gb/s, which is a nice result for a common user network, but not enough for a mobile core network. When the incoming packet rate reaches 30 Gb/s, the multi-core processor has dropped more than 50% of the packets. Finally, because of the large workload processed in the FPGA-based hardware accelerator, it has no packet leakage, and even the incoming packet rate reaches 18 Gb/s. In Fig. 12c, we evaluated the throughput of different packet lengths. It is obvious that the throughput increases as the size of the packet increases and the processing performance

with the hardware accelerator is much better than that in the multi-core processor-only solution. In Figs. 12d–12f, we used the same evaluation scenario on the TD-SCDMA core network dataset. First, we provided the distribution of packet length in Fig. 12d and found that almost 46% of packets have a length of less than 128 bytes and the average packet length is 572 bytes. Then as shown in Figs. 12e and 12f, because the GTP packet was completely processed and forwarded by the hardware accelerator, our system was able to process line-speed traffic.

4.3 Performance evaluation of the extension

Because some functions have not been implemented, we evaluated only the performance of the protocol analysis component. We separated several POP3 sessions and video-on-demand (VOD) streams from the CDMA2000 dataset to build a new synthetic dataset. The whole dataset includes 13 POP3 sessions and 5 VOD streams, corresponding to 13 195 packets. The IPRO tester was used to perform 10^4 times cyclic playback with randomly changing IP addresses. For comparison, the protocol analysis component was also implemented in a typical server as mentioned above. Fig. 13 evaluates the performance of our system with or without protocol analysis. Results show that there is no significant reduction in performance when processing the POP3 protocol. In contrast, due to the implementation of regular expressions, the multi-core processor is less efficient, which will inevitably lead to a decline in the performance of our system. However, it still far exceeds the traditional network server. In the future, we intend to employ a separate accelerator chip to achieve high-speed regular expression matching, such as the Netlogic NLS2008.

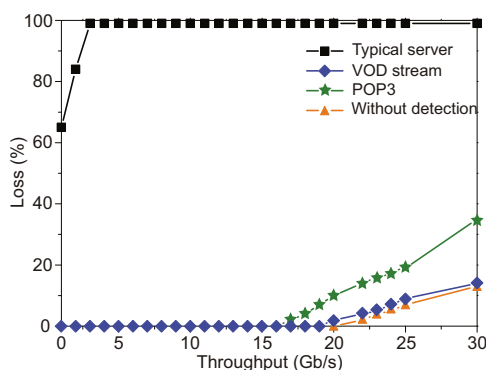


Fig. 13 Performance evaluation of the extension scheme

5 Conclusions

In this paper, we have proposed a real-time pre-processing system for mobile core network. For reducing the processing pressure on back-end detection server, an FPGA-based hardware accelerator and a multi-core processor were implemented to handle different stages of packet processing. Evaluation results showed that our system can achieve a speed of at least 18 Gb/s with no packet loss.

References

- Cavigelli, L., Gschwend, D., Mayer, C., et al., 2015. Origami: a convolutional network accelerator. Proc. 25th Edition on Great Lakes Symp. on VLSI, p.199-204. <https://doi.org/10.1145/2742060.2743766>
- Cheng, M., Sun, Y., Su, J., 2016. A real-time pre-processing system for mobile core network measurement. Proc. 6th Int. Conf. on Instrumentation, Measurement, Computer, Communication and Control, p.298-302. <https://doi.org/10.1109/IMCCC.2016.73>
- China Communications Standards Association, Inc., 2006. Third Generation Partnership Project 2.
- Cisco Systems, Inc., 2013. Cisco ASR 5000 Series Aggregation Services Router Installation and Administration Guide Version 10.0.
- Go, Y., Jamshed, M.A., Moon, Y., et al., 2017. APUNet: revitalizing GPU as packet processing accelerator. Proc. USENIX Symp. on Networked Systems Design and Implementation, p.83-96.
- Han, S., Jang, K., Park, K., et al., 2010. PacketShader: a GPU-accelerated software router. *ACM SIGCOMM Comput. Commun. Rev.*, **40**(4):195-206. <https://doi.org/10.1145/1851182.1851207>
- Intel Products, Inc., 2010. Crystal Forest Platform: Product Overview.
- Internet Society, Inc., 2014. Request For Comments No. 1661.
- Kekely, L., Puš, V., Benáček, P., et al., 2014. Trade-offs and progressive adoption of FPGA acceleration in network traffic monitoring. Proc. 24th Int. Conf. on Field Programmable Logic and Applications, p.1-4. <https://doi.org/10.1109/fpl.2014.6927443>
- Lavasani, M., Angepat, H., Chiou, D., 2014. An FPGA-based in-line accelerator for memcached. *IEEE Comput. Archit. Lett.*, **13**(2):57-60. <https://doi.org/10.1109/l-ca.2013.17>
- Neil, D., Liu, S.C., 2014. Minitaur, an event-driven FPGA-based spiking network accelerator. *IEEE Trans. VLSI Syst.*, **22**(12):2621-2628. <https://doi.org/10.1109/tvlsi.2013.2294916>
- Peemen, M., Setio, A.A., Mesman, B., et al., 2013. Memory-centric accelerator design for convolutional neural networks. Proc. IEEE 31st Int. Conf. on Computer Design, p.13-19. <https://doi.org/10.1109/iccd.2013.6657019>
- Rizzo, L., 2012. Netmap: a novel framework for fast packet I/O. Proc. 21st USENIX Security Symp., p.101-112.
- Vallentin, M., Sommer, R., Lee, J., et al., 2007. The NIDS cluster: scalable, stateful network intrusion detection on commodity hardware. Proc. Int. Workshop on Recent Advances in Intrusion Detection, p.107-126. https://doi.org/10.1007/978-3-540-74320-0_6
- Vasiliadis, G., Polychronakis, M., Ioannidis, S., 2011. MIDeA: a multi-parallel intrusion detection architecture. Proc. 18th ACM Conf. on Computer and Communications Security, p.297-308. <https://doi.org/10.1145/2046707.2046741>
- Zhang, C., Li, P., Sun, G., et al., 2015. Optimizing FPGA-based accelerator design for deep convolutional neural networks. Proc. ACM/SIGDA Int. Symp. on Field-Programmable Gate Arrays, p.161-170. <https://doi.org/10.1145/2684746.2689060>