An Improved Cache Mechanism for a Cache-based Network Processor

Hayato Yamaki¹, Hiroaki Nishi¹
¹Graduate school of Science and Technology, Keio University, Yokohama, Japan

Abstract – Internet traffic has increased due to the development of richer web content and services. In particular, IP telephony, Messengers, and Twitter are composed of a large number of small packets, and these services are expected to increase. Thus, routers need to handle large-bandwidth fine-grain communication. We proposed a network processor called P-Gear, which has a special cache mechanism that reduces the processing load by taking advantage of the localizations of network traffic. In this paper, we describe cache architecture for P-Gear and we propose an enhanced cache mechanism, Multi-Context-Aware Cache. Multi-Context-Aware Cache can improve the cache-hit ratio because it can control cache entry by analyzing the packet’s header. We implemented P-Gear as software and we simulated this mechanism using real network traces. The simulation showed that Multi-Context-Aware Cache is effective in reducing the processing cost of processing arrays on a network processor.

Keywords: Cache-based Network Processor, Layer-7 Analysis, Network Processor, Network Traffic Engineering

1 Introduction

Internet traffic has increased due to the popularization of personal computers, smartphones, and growth of richer web contents and services. Furthermore, services that are composed of a large number of small packets such as video traffic, IP telephony, and Messengers are expected grow in popularity. If these enormous fine-grain packets are concentrated in a router, its processing capacity might exceed its limit. Future backbone network router need to handle large-bandwidth fine-grain communication.

Conventional network processors utilize the characteristic independency of packets, where each packet can be processed independently by a Processing Unit (PU) or in parallel by increasing the number of Processing Units (PUs) or by implementing a multithreading mechanism to achieve high throughput packet processing [1][2]. However, the leakage of current or electrical power consumption makes it difficult to improve the number of PUs accumulated due to the scaling rule of semiconductors. Therefore, it is necessary for network processors to reduce PU processing costs.

In earlier research, a mechanism was proposed that the reduced costs of packet processing when retrieving routing tables [3][4][5]. This mechanism resolved the processing when retrieving routing table by using cache architecture. Thus, we proposed an architecture for a network processor, called P-Gear, which was an extension of this mechanism [6][7]. Eliminating the cache-miss ratio is significant for P-Gear and all cache-based network processors. P-Gear takes advantage of the temporal locality of network traffic to reduce processing costs like general network processors. It caches to process the packet, which improves the potential of cache-based network processor. However, the cache-hit ratio depends on the number of processing units required and the cache memory size. Reducing the cache-miss ratio is an important issue for P-Gear type network processors. In this paper, we propose and evaluate a mechanism that allows us to control cache entry by focusing on trends in network traffic to produce a lower cache-miss ratio using the same cache entry size.

2 Related Work

Various cache-based network processors have been proposed in earlier research. These processors exploit the temporal locality of network traffic, where many sequential packets arrive during a short period of time. In previous studies [3][4], a cache architecture for the high-speed processing of routing table lookup on layer 3 was proposed. This architecture utilized a quarter of the L1 cache to store the destination IP address and the results of routing table lookups. Part of the destination IP address was used as the virtual address of a cache. The ID of an output port was obtained when the cache was hits. Moreover, they considered that the cache entry size, block size, and associativity size were important and they proposed a method for compressing the entry size of the routing table [8]. In paper [5], another cache mechanism was proposed for routing table lookup, which was evaluated using real network traces. This study showed that it was effective to use the upper half of an IP address as a cache tag and the lower half of an IP address as an index in the cache.In another study [9], the cache architecture was partitioned according to a prefix length for routing table lookup. They demonstrated that IP traffic had temporal and spatial locality.
Thus, cache-based network processors are used for resolving the lookup of routing table entry. However, because of the emergence of rich services, recent network processors are required to execute a lot of complex processing in addition to routing table lookup, e.g., packet encapsulation, and these processes may be a bottleneck in packet processing. Thus, we proposed a cache-based network processor called P-Gear, which facilitates high-speed processing for various processes by caching the IP address, port, and protocol.

3 P-Gear

In this section, we provide an outline of P-Gear. P-Gear utilizes two features of packets. The first is that packets are processed by the same process in the same session. The second is that packets often arrive continuously in a short period. P-Gear caches before processing the packet and applies the cached process to later packets during the same session. The later packets in the same session can be processed without using PU. P-Gear defines a five-tuple (Source IP, Destination IP, Source Port, Destination Port, and Protocol) as one session. The architecture of P-Gear is shown in Figure 1.

![Fig. 1 Architecture of P-Gear](image)

When PLC hits, P-Gear can process the packet without using PU by applying a cached process to packet processing. If there is a cache-miss, the packet is sent to the Cache Miss Handler (CMH), and P-Gear processes the packet normally using PU. After processing the packet, the session information of the packet and the results of the packet processing are stored in PLC. As a PLC address, the hashing value of the session information is used to analyze the spatial locality of packets. However, if a later packet arrives in PLC while an earlier packet is still being processed by a PU in the same session, the later packet does not match the cache entry in PLC, so the packet will be processed by another PU. To reduce this inefficient allocation of PU, we implemented a Cache Miss Table (CMT) and Cache Miss Queue (CMQ) in CMH. CMT stores a packet that has been processed by PU. When a later packet arrives in CMH, CMH checks whether CMT has the table entry for this packet. If the packet matches a table entry, the packet is sent to a queue called CMQ. After the earlier packet in the same session is processed by PU, the later packet in CMQ is processed based on the processing results for the earlier packet without being allocated to PU. Thus, the total cache-hit ratio is the summation of the hit ratio in PLC and CMH, while the cache-miss ratio for P-Gear is the ratio of packets allocated to PU.

In previous research, we concluded that four methods for associativity and a cache. 4K entries cache were an appropriate structure of PLC. It was also appropriate to use ca. 1K entries CMT and ca. 16K entries CMQ, respectively, according to the scaling limitation. We now propose a method for increasing the cache-hit ratio by adding a small hardware or implementing a tiny routine into the processing units of P-Gear.

4 Multi-Context-Aware Cache

P-Gear can utilize the temporal locality of packets effectively and aims to achieve high throughput using a small number of PUs. In this section, we propose and describe Multi-Context-Aware Cache, which extends the cache mechanism of P-Gear to exploit the PLC cache more efficiently.

P-Gear captures the results of a process during one session into a PLC cache entry. When a cache hit occurs, P-Gear can process the packets of a stream using special wired logic. However, all the packets are cached to PLC with same priority. Because there are other trends of network traffic in addition to the temporal locality, we consider that it is effective for PLC to cache packets with different priorities instead of a uniform priority. For example, if a large number of packets with various headers are concentrated into P-Gear in a short period of time, degradation occurs in the cache-hit ratio because PLC stores useless entries. To prevent these useless entries, PLC is required to determine whether an incoming entry is reused based on implicit information in the IP layer and application layer. This context-based cache management is required to improve the efficiency of processing. Here, we propose a Multi-Context-Aware Cache, which facilitates controlled insertion and deletion of the cache entry by using several contexts simultaneously to achieve a higher cache-hit ratio without increasing the size of cache entries.

It is possible to use the specification of the services of an application layer as a context to determine whether the first packet from a service follows succeeding packets. For example, File Transfer Protocol (FTP) has a specification. It uses a control port and a data transfer port for communication. When a packet arrives at PLC and the source or destination port indicates that the packet is an FTP control packet, PLC can predict that it should not cache the control packet but that it should make another cache entry for following data packets. This is called an FTP-Aware Cache. As another
example, the Session Initiation Protocol (SIP) of the Voice-of-IP Protocol also has the same behavior and this is known as an SIP-Aware Cache. These cache architectures were proposed and evaluated in our previous studies, and these architectures improved the cache-hit ratio by several percent when used in an IP backbone router [7]. In this paper, we propose a DNS-Aware cache: Rare-DstPort-Aware Cache and P2P-Aware Cache as a new cache architecture based on this strategy.

It is also possible to use request-and-reply transactions as a context, which is generally used in network services. The headers of reply packets can be predicted by interchanging the source and destination information of a request packet, which will be effective for making a cache entry that is generated according to this rule beforehand. We propose a Look-Ahead Cache based on this strategy.

Furthermore, it is also possible to use the characteristics as a context in a network attack. For example, when the packets of a port scan attack are concentrated in P-Gear, P-Gear generates cache entries for a large number of packets that have different destination IP addresses or ports in a short period. These entries will not be reused and they disturb the registration of PLC entries that will be used. Because the packets in a network attack are sent from a source to various destinations and they will not be sent to the same address again, it is effective for the cache not to make entries for packets that are generated by a source node and that are distributed to many different destinations. Similarly, attacks from a worm and DoS attacks often use specific source or destination ports, so it would be effective not to store packets with specific port numbers. We propose an Attack-Aware Cache based on these strategies.

In this paper, we propose and evaluate DNS-Aware Cache, Rare-DstPort-Aware Cache, P2P-Aware Cache, Look-Ahead Cache, and Attack-Aware Cache as subsets of Multi-Context-Aware Cache.

### 4.1 DNS-Aware Cache

Sessions on a Domain Name System (DNS) have no temporal locality because DNS communication is composed of one request packet and one reply packet, while the session is composed of only one packet. DNS transactions are cached by domain name server and they are rarely reused. Thus, we propose a DNS-Aware Cache, which is a mechanism that does not store the packets from DNS port #53.

We analyzed network traffic to find the ratio of DNS sessions traffic. The analysis showed that over 50% of all sessions were composed of only one packet. These types of sessions are known as one packet sessions. Table 1 and Table 2 show the frequent port numbers and the percentages of >1 packet sessions and one packet sessions. Packets from port #53 mainly appeared in one packet sessions when comparing

<table>
<thead>
<tr>
<th>Port number</th>
<th>Source Quantity</th>
<th>Source Rate[%]</th>
<th>Destination Quantity</th>
<th>Destination Rate[%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>53</td>
<td>89,998</td>
<td>4.9</td>
<td>149,415</td>
<td>8.1</td>
</tr>
<tr>
<td>80</td>
<td>267,924</td>
<td>15</td>
<td>292,905</td>
<td>16</td>
</tr>
<tr>
<td>445</td>
<td>169,822</td>
<td>9.2</td>
<td>3,969</td>
<td>0.22</td>
</tr>
<tr>
<td>1433</td>
<td>430</td>
<td>0.023</td>
<td>211</td>
<td>0.011</td>
</tr>
<tr>
<td>6000</td>
<td>0</td>
<td>0</td>
<td>369</td>
<td>0.020</td>
</tr>
<tr>
<td>6881</td>
<td>17,398</td>
<td>0.94</td>
<td>16,725</td>
<td>0.91</td>
</tr>
</tbody>
</table>

Table 1 and Table 2. Furthermore, Table 2 shows that 73% of one packet sessions were DNS packets. Therefore, DNS-Aware Cache allows us to reject most of the one packet sessions and to utilize the cache entry more effectively.

#### 4.2 Rare-DstPort-Aware Cache

Sessions with rarely used destination ports have no temporal locality, so there is no need to store them. We propose a Rare-DstPort-Aware Cache, which finds rarely used destination ports in a time window and does not store them in the next time window. The mechanism is shown below.

1. We set a window size and threshold, while a counter counts how many packets with the same destination port arrive in the window per destination port.
2. When the counter reaches the window size, Rare-DstPort-Aware Cache determines the destination ports that are not over threshold as rare destination ports.
3. In the next window, Rare-DstPort-Aware Cache does not store the packets with rare destination ports.

#### 4.3 P2P-Aware Cache

A P2P protocol such as BitTorrent connects with extremely many nodes and the communication of the P2P protocol often finishes with only one packet. These situations will cause a disturbance in the cache. For this reason, we propose a cache mechanism that does not store packets with the specific ports used in the P2P protocol. This is known as a P2P-Aware Cache. It is known that BitTorrent uses 6881 to 6999 as a destination port and it mainly uses 6881. However, some P2P sessions are composed of >1 packets. Because of
this, the P2P-Aware Cache determines whether a BitTorrent session is finished by a packet by utilizing the packet length values of the packet headers, so it does not create a new cache entry if the packet lengths of the packets are less than certain values.

4.4 Look-Ahead Cache

When a packet arrives in a router, reply packets will often arrive soon after, while the difference between the arriving and replying packet header shows the source and destination IP and port that are exchanged. We propose a Look-Ahead Cache, which is a mechanism that creates a cache entry for reply packets based on the incoming packets by exchanging the source and destination information in advance. Figure 2 shows the architecture of the Look-Ahead Cache. In this figure, a packet arrives at the router, which comes from line card A and goes to line card B. At the ingress of Line card A, a cache entry of its ingress is created at the same time during the ingress processing. This information is forwarded to the Line card B. Line card B creates a cache entry based on the header information of the outgoing packet or the forwarded information from Line Card A. The cache entry of egress of Line card B is also created at the same time. The ingress and egress network processor in line card A and B can process the concerning packet without allocating a PU.

Fig. 2 Architecture of Look-Ahead Cache

4.5 Attack-Aware Cache

As described earlier, a port scan attack disturbs effective cache entries because P-Gear tries to store port scan packets with different destination ports. Therefore, it causes degradation of the cache-hit ratio. To reduce this degradation, we propose an Attack-Aware Cache that identifies network attack packets in network traffic and does not create a cache entry for these packets. In this paper, we propose two dynamic methods and one static method for determining network attacks.

Dynamic methods focus on the fact some types of attacks often send a large number of packets from a specific source node to a large number of destination nodes. To determine whether attacks are waged or not, it is effective to analyze the trend of traffic in increments of a certain amount of network transactions. If the number of connections exceeds a threshold in a window, a new entry is not created in the following window. We set the window size to 512 packets and the method applies results for each window to the next window to prevent entry creation. Dynamic method 1 processes network streams by two processing phases. In phase 1, a dominant source IP at the point of the number of receiving packets is marked. Figure 3 shows the mechanism of this method. When a packet arrives, PLC calculates the hashed value of its source IP address using CRC. A 7-bit value is used as the hashed value. Next, A 5-bit counters count the number of packets arriving with the same hash value. If the counter causes an overflow, the IP addresses group that has the same hashed value is a candidate of dominant source in this window. In phase 2, the group is determined whether it contains an attack source or not. Figure 4 shows the mechanism of this method. The method uses a memory that has three bits addresses. When a packet that has the dominant source arrives, 3-bit counters are incremented according to the destination address. As this counters, the 3-bit hash value of its destination IP address is used as an address in the memory. If the number of the counters which is not zero exceeds a threshold exist, this source IP group of the same hashed value is regarded as an attack source. PLC does not cache the packet from this IP group in the following window. Next, we describe dynamic method 2. Figure 5 shows the mechanism for this method. This method generates a hashed value for the source and destination IP address based on the arriving packet respectively using CRC. It calculates a 7-bit value as the

Fig. 3 Mechanism of dynamic method 1 in phase 1

Fig. 4 Mechanism of dynamic method 1 in phase 2

Fig. 5 Mechanism of dynamic method 2
hashed value and it stores the hash value of the destination IP address in memory. At this point, it uses the hash value of the source IP address as an address in its memory. When the same address is accessed again, it compares the hash values with the destination IP address. If the values are different, it increments a 3-bit counter. If the number of the counter exceeds a threshold, a new cache entry is not created until the counter is reset.

Static methods focus on the fact that some types of attacks use specific ports. For example, DoS attacks exploit the vulnerability of particular applications such as database services, so the attack packets used the specific ports for database services. Furthermore, the headers of these attack packets are often very variable. We analyzed the network traffic to find the ports used by attacks, and Table 1 and 2 show the results. We found that packets with destination port #1433 and source port #6000 were sent in bulk during one packet sessions. Packets with the destination port #1433 were sent to SQL as worm attacks, while packets with source port #6000 were sent as port scan attacks. By analyzing the network traffic, we found that packets with source port #1434 and source port #3306 could also be regarded as network attack packets. The static method identifies packets using these ports as attack packets and it does not create cache entries for them.

5 Evaluation

In this section, we show the simulation results, where we evaluate the five proposed cache mechanisms. We used P-Gear simulator and WIDE trace (150Mbps) as real network traffic for this evaluation. P-Gear simulator was written in C++ while PLC was implemented as a set associative cache using four methods. The size of cache entries was 1,024 because we considered a 394-bit PLC tag and 365-bit PLC data, so the capacity of the cache memory was estimated as 89 Kbytes. This size of the cache can be implemented as a high-speed on-chip L1 or L2 cache. The number of PUs was 32 and a delay when processing a packet was 2 µs in the simulations.

5.1 Results with the DNS-Aware Cache

We simulated the DNS-Aware Cache and compared it with a normal cache mechanism that caches all ports. Figure 6 shows the cache-miss ratio with the normal mechanism and the proposed mechanism over 12 hours (from 0:00AM to 12:00AM). The cache-miss ratio with DNS-Aware Cache was reduced by roughly 1-2 points, which led to a ca. 6% improvement in the packet processing costs of the PU. The DNS-Aware Cache eliminated most of the one packet sessions by preventing the creation of new entries for DNS packets, which was effective for reducing the cost of packet processing.

5.2 Results with the Rare-DstPort-Aware

Figure 7 shows the simulation results for the cache-miss ratio with each window size. We changed the window size (x-axis) and the threshold n for the Rare-DstPort-Aware Cache, as shown in the different graphs. With all thresholds, the cache-miss ratio was improved by increasing the window size. However, the cache-miss ratio with this mechanism was never less than the ratio when all destination ports were cached. This was because the destination port identified in the previous window appeared in the next window in some cases. When the window size was 5,000,000 packets, the number of unique destination ports was 8,426. However, one of these destination ports appeared in next window, which led to an increase in the cache-miss ratio. Furthermore, caching destination ports that never appeared in the window was sometimes ineffective because they may also appear in next window. We needed to consider a method that could identify rare destination ports more accurately.
5.3 Results with the P2P-Aware Cache

We simulated the P2P-Aware Cache using the same traffic and we compared the four mechanisms shown below.

A) Caching all ports (all ports cached)
B) Not caching port 6881 only, as mainly used by BitTorrent (P2P_6881)
C) Not caching port 6881 to port 6999, as used by BitTorrent (P2P_6881-6999)
D) Not caching port 6881 and less than 256-byte length packets (P2P_P-Length)

By analyzing packet length values of the packet headers, mechanism D determined whether the P2P sessions consisted of one packet or more. The threshold of the packet length was 256 bytes. If the packet length value was over the threshold, we identified the packet as a session with one packet.

We compared the cache-miss ratio with the P2P-Aware Cache and the cache-miss ratio using the all ports cached mechanism, and Figure 8 shows the results. All the P2P-Aware mechanisms improved the cache-miss ratio in many cases. However, P2P_6881 and P2P_6881-6999 sometimes increased the cache-miss ratio. This was because these mechanisms did not create entries for sessions composed of >1 packets, although packets for these sessions appeared in some cases. However, with P2P_P-Length, this increase in the cache-miss ratio was less than one-third, while the cache-miss ratio was improved by about 0.5%.

![Figure 8](image)

This demonstrates that the Look-Ahead Cache was effective and it improved the packet processing costs by about 9%.

5.5 Results with the Attack-Aware Cache

We simulated two dynamic mechanisms and one static mechanism with the Attack-Aware Cache. In the simulations of dynamic mechanisms, we changed the threshold from 4 to 32. Figure 10 and Figure 11 show the results of simulations for dynamic mechanisms each second. If the cache-miss ratio increased suddenly, both mechanisms reduced the cache-miss ratio by about 2 points. However, they increased the average of the cache-miss ratio because the source port determined by the dynamic mechanisms included sessions composed of >1 packets. As shown in figure 10 and Figure 11, dynamic mechanism 2 was more effective than dynamic mechanism 1.

In the simulations with the static mechanism, the mechanism did not cache packets that included destination port #1433, #1434, and #3306, and source port #6000. Figure 12 shows the shift in the cache-miss ratio during each second. As shown in Figure 12, when the cache-miss ratio increased suddenly, the static mechanism reduced the cache-miss ratio by 4%. The mechanism did not degrade the average cache-miss ratio in many cases. Therefore, this static mechanism was effective during network attacks and improved the packet processing costs by about 10% when the cache-miss ratio increased suddenly due to network attacks.

![Figure 9](image)

![Figure 10](image)

5.4 Results with the Look-Ahead Cache

We simulated the Look-Ahead Cache using the same traffic. This simulation assumed that request-and-reply transactions were processed on the same line card. Figure 9 shows the cache-miss ratio in the simulation. This shows that the mechanism reduced the cache-miss ratio by 2% compared with the normal mechanism while the cache-miss ratio using this mechanism was always lower than the normal mechanism.

![Figure 11](image)
6 Conclusions

We proposed P-Gear, which has a special cache architecture that efficiently utilizes the temporal locality of network traffic to achieve a higher throughput with a limited number of PUs. For P-Gear, the reduction of the cache-miss ratio is a significant requirement for reducing the processing costs of PUs. We proposed a cache mechanism known as Multi-Context-Aware Cache to improve the cache-miss ratio with P-Gear. Multi-Context-Aware Cache estimates trends in network traffic by continuously analyzing the packet headers and it optimally manages cache entry.

In this paper, we proposed and simulated five mechanisms that formed a subset of Multi-Context-Aware Cache. DNS-Aware Cache, Rare-DstPort-Aware Cache, and P2P-Aware Cache utilized port information to determine whether sessions had temporal locality. Look-Ahead Cache utilized the relationships of request-and-reply transactions. Attack-Aware Cache utilizes the trend in network attacks. In simulations, the DNS-Aware Cache eliminated about 70% of a session composed of one packets and improved the cache-miss ratio by 6%. However, it was difficult to estimate rare destination ports using Rare-DstPort-Aware Cache. This method did not improve the cache-miss ratio, so there is still a need to estimate rarely used ports more accurately. P2P-Aware Cache eliminated the packets of BitTorrent, which often communicates using one packets and it improved the cache-hit ratio by 0.5%. However, we need to find a more precise method for determining whether sessions of packets are composed of only one packet. Furthermore, the Look-Ahead Cache estimated the reply packet headers and improved the cache-hit ratio by 9%. Finally, the Attack-Aware Cache identified network attack packets and improved the cache-hit ratio by almost 10% when the cache-miss ratio increased suddenly.

7 Acknowledgment

This work was partially supported by National Institute of Information and Communications Technology (NICT) and a Grant-in-Aid for Scientific Research (C) (22500069).

8 References


