Analysis and Enhancement of Pipelining the Protocol Overheads for a High Throughput

Hyun-Wook Jin and Chuck Yoo
Department of Computer Science and Engineering, Korea University
{hwjin, hxy}@os.korea.ac.kr

Abstract

This paper investigates the protocol overhead pipelining between the host and network interface card (NIC). Existing researches into the protocol overhead pipelining assume that protocol overheads in the host and NIC can be naturally pipelined. Our architecture-aware investigation, however, finds a new fact that the host and NIC compete against each other to access the host memory, system bus, and I/O bus, so that the overhead pipelining is seriously hindered, which leads to a sub-optimal performance. We suggest several methods to avoid such competitions for the hardware resources and implement a pipelining UDP named π-UDP on Myrinet. As a result, π-UDP achieves over 97% of the theoretical maximum throughput of Myrinet.

Keywords: Overhead Pipelining, Myrinet, Gigabit LAN, UDP/IP

1. Introduction

Low overhead and high throughput are very important requisites of network protocols in clustering systems and storage area networks. Accordingly, many research groups are trying to minimize the protocol overheads of an end-node [1-4].

The protocol overheads of an end-node can be divided into the host, network interface card (NIC), and link overheads. The host overhead is the time that the host processor spends to perform the protocol stack in the kernel or communication library. On the other hand, the NIC overhead is generated by the NIC itself including the overhead for DMA between the host and NIC memories. And the link overhead is the time to send/receive a packet to/from the physical network link. The protocol overhead pipelining attempts to hide all these overheads behind the largest overhead so that the protocol fully utilizes the hardware resources and achieves a near theoretical maximum performance.

In order to optimize the network performance, the protocol overhead pipelining has been applied to many gigabit network protocols [5-10]. However, they have focused only on NIC to attain the pipelining between the NIC and link overheads. In contrast, the pipelining between the host and NIC overheads has not been studied at all because it is generally believed that, since they are produced by different processors, they can be naturally pipelined. Although Wang et al. [7] reported the only research that considers the host overhead, they also implicitly assume that the host and NIC overheads are pipelined as a matter of course.

If the host and NIC overheads are truly pipelined, it is expected that even traditional kernel-level protocols, such as UDP and TCP, can utilize the maximum physical network bandwidth because of the low host overhead. In the past, the host overhead of traditional protocols was larger than NIC and link overheads,

This Research was supported by University Software Research Center Supporting Project from Korea Ministry Information and Communication.

but now it is not true any more - thanks to the giga-hertz processor [11].

This paper questions the assumption:

- Can the host and NIC overheads be naturally pipelined?
- If so, can a traditional protocol such as UDP utilize the maximum bandwidth of I/O bus or physical network link by adopting the overhead pipelining?

In order to find answers to these questions, this paper starts with intensive measurements on UDP. Contrary to our expectation, the measurement results reveal that overheads of the host and NIC are not naturally pipelined. The architecture-aware measurements show that the overhead pipelining is considerably degraded by the competition between the host and NIC for hardware resources. To ease the competition, we implement several methods into UDP on Myrinet [12] and name it π-UDP. As a result, we can achieve the overheads fully-pipelined with π-UDP.

Achieving a high throughput by a kernel-level protocol is much meaningful than that by a user-level protocol because kernel-level protocols based on IP are applicable to various network infrastructures and can work on a secure network protocol such as IPsec.

Overall, the contribution of this paper includes:

- Architecture-aware measurements and detailed analysis of the protocol overhead pipelining, especially focusing on the host and NIC overheads. The measurement results elucidate what factors obstruct the protocol overhead pipelining and convey the importance of understanding the architecture in designing a gigabit network protocol.
- The implementation of π-UDP, which is the first implementation that allows the practical pipelining between the host and NIC overheads.
- The measurement results and implemented mechanisms, which are applicable to another I/O system. For example, in designing a high performance file I/O system, our experience tells that the most important consideration is to allow the overhead pipelining between the host and SCSI adapter.

The remainder of this paper is organized as follows: In Section 2, we perform in-depth measurements in order to analyze the pipelining of protocol overheads on UDP. Section 3 describes several methods to achieve the full overhead pipelining and the implementation of π-UDP. Finally, Section 4 concludes this paper.

2. Measurements and Analysis of Protocol Overhead Pipelining

Measurements were performed using an Intel Pentium III 1GHz processor on an ASUS motherboard (Intel 815EP chipset). Each machine has a Myrinet NIC (LANai 9.0) set to a 32bit 33MHz PCI slot, and NICs are directly connected to each other. The kernel is Linux (kernel version 2.2), and we adopt GM (version 1.4) [13] for the device driver and the firmware of Myrinet NIC. The MTU size is set to 32KB.

For the measurement of protocol overheads, we utilize the measurement methodology that is proposed by Jin et al. [11]. The key feature of this measurement methodology is to use the clock on NIC for the measurement of both host and NIC sides. The most important factor that should be considered in measuring the overhead pipelining between the host and NIC is that the timestamps of the host and NIC overheads have to be generated by the same clock.

Moreover, in order to understand the architectural behavior, we measure several performance parameters of the Intel processor by reading the performance-monitoring counters [14]. We can select a performance-monitoring event and read it with `wrmsr` and `rdmsr` instructions, respectively.

Figure 1 presents the measurement results of overhead pipelining on UDP. The figure shows the time chart of the host, NIC, and link overheads of both sender and receiver when 10 UDP packets (of 32KB) are transmitted. The notation used in the figure is very similar to those as in the graphs by the reference [8], but the figure in this paper includes all protocol layers of end nodes without limiting to the NIC side. The time is set to zero when the first packet is sent by

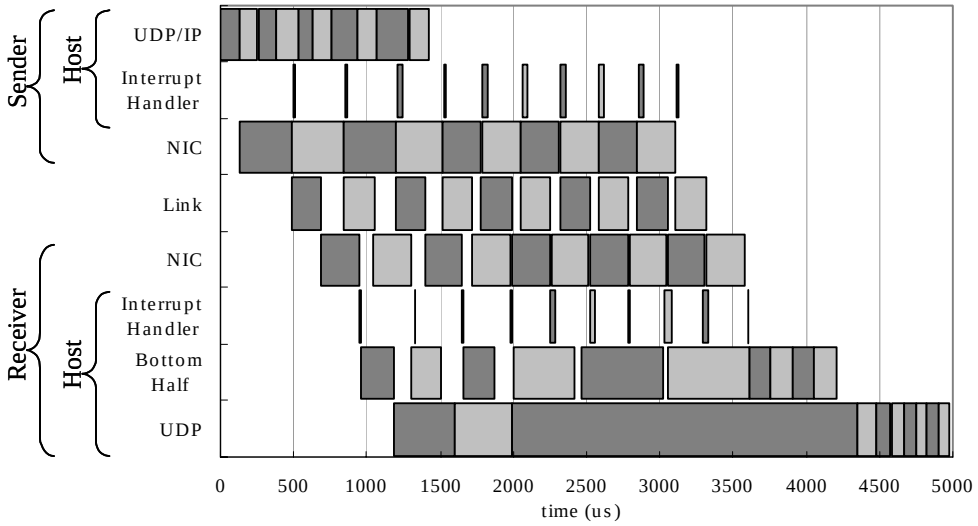


Figure 1: Overhead pipelining of traditional UDP/IP.

the UDP application. The *y*-axis represents the layers that each packet goes through in order to be processed. A rectangle is the time spent in the layer to process a packet. The rectangles are colored alternatively for better readability.

2.1. Analysis of Sender Side

Figure 1 shows that the first four rectangles in the NIC layer of the sender are larger than the rest in the same layer. The reason is that the host and NIC compete against each other for hardware resources: the host memory, system bus, and I/O bus (i.e., PCI bus).

First, we analyze the cause of the host memory contention. The UDP layer copies the data from the user buffer into the kernel buffer, where the kernel buffer is not cached because the kernel buffer is not referenced yet after the buffer allocation. Therefore, when the host processor tries to write the data into the kernel buffer, cache misses occur and they invoke cache line fills [14]. After the completion of a cache line fill, the host processor copies the data into the filled cache line. Here, it is very important to note that the operation for the cache line fill requires accessing the host memory. Since NIC is transmitting packets, NIC accesses the host memory to perform DMA that copies packets of previously-requested send as shown in Figure 2(a). The gray area in Figure 2(a) depicts the contention between the host and NIC for accessing the host memory concurrently. As a result, although the copy and DMA operations are performed by different processors, the operations are not pipelined.

Note that this type of contention can take place in any protocols (not just UDP) as long as they also copy data from a user buffer into the DMAable buffer.

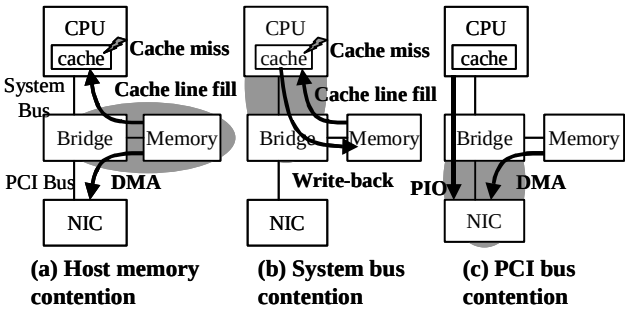


Figure 2: Competitions between the host and NIC on sender for hardware resources.

Now, let us look at the system bus contention. The data copied to the kernel buffer is located in the cache (after the cache line fill), so the data should be written-back to the host memory for cache coherency prior to the DMA from the kernel buffer. While the data is written back to host memory, another cache line fill operation is in progress because the copy from the user buffer to the kernel buffer is still being performed for the following sending requests. This causes the

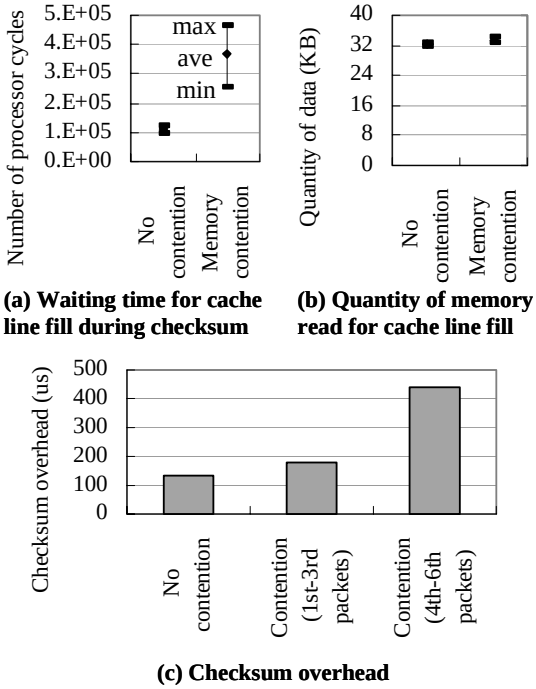
contention in the system bus, which is depicted in Figure 2(b).

Finally, the contention in the I/O bus is caused by the concurrent access of NIC. The device driver passes a descriptor that describes the data in the kernel buffer to NIC through PIO. Also DMA is copying data to NIC. Therefore, the PIO operation competes against the DMA operation in order to access PCI bus, as shown in Figure 2(c).

Again, note that this contention is true for any protocols. Especially, in the case of a lightweight user-level protocol, the competition by PIO becomes more serious because the very light host overhead makes send requests very frequently.

2.2. Analysis of Receiver Side

Figure 1 shows us that, in the case of the receiver, the host overheads of the bottom half and UDP are not fully pipelined with the NIC overhead. We analyze the reasons for this unexpected result.



(a) Waiting time for cache line fill during checksum

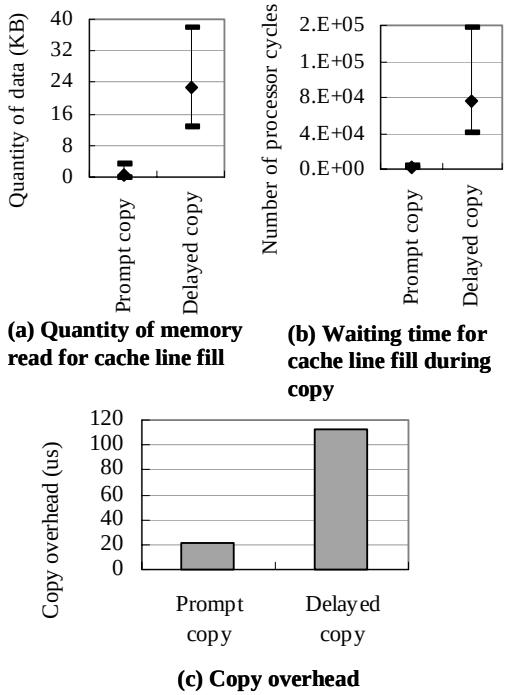
(b) Quantity of memory read for cache line fill

(c) Checksum overhead

Figure 3: Dependence of checksum computation overhead of the receiver on the host memory contention.

We first notice that the bottom half consumes an excessive amount of time for the first six packets. It is because the checksum computation competes against the DMA operation of NIC to access the host memory as shown in Figure 2(a) with the only exception of the direction of DMA. Figure 3 presents that, during the DMA operation, the memory read for the checksum computation takes much time because of the host memory contention (Figure 3(a)) even though the same quantity of data is read from the host memory (Figure 3(b)). Consequently, it takes a large amount of time to perform the checksum computation (Figure 3(c)).

In Figure 3(c), the relatively small processing overhead for the first three (1st – 3rd) packets comparing with the overheads of the following three (4th – 6th) packets is due to the fact that the arrival intervals of the first four packets are larger than those of the others as shown in Figure 1. That is, the checksum computation can access the host memory during the interval without competing against the DMA operation. The large intervals are due to the high NIC overhead of the sender side for the first four packets as described in the previous subsection.



(a) Quantity of memory read for cache line fill

(b) Waiting time for cache line fill during copy

(c) Copy overhead

Figure 4: Variation of data copy overhead of the receiver depending on whether the received data is cached or not.

We also notice that the enlarged overhead of the bottom half layer causes the dormancy in UDP layer because of the higher priority of the bottom half. UDP of the receiver in Figure 1 has to wait until the bottom half has completed processing for all packets received. This explains the very "big" rectangle in Figure 1. Consequently, when UDP tries to copy the data from kernel to user buffer, the cached data during the checksum computation has already been removed from the cache by other checksum computations. Thus, the delayed copy operation leads to cache misses and reads a great quantity of data from the host memory (Figure 4(a)). Accordingly, the overhead of memory read is enlarged (Figure 4(b)), which results in a large copy overhead (Figure 4(c)).

2.3. Summary

On the sender side, the data copy of UDP and DMA of NIC incur the resource contentions on the host memory and system bus. On the other hand, the I/O bus contention is caused by the frequent PIOs of the device driver.

On the receiver side, the checksum computation of UDP competes against DMA of NIC for the host memory. This contention causes an additional side effect that the copy operation of UDP cannot take the advantage of cache effect.

In summary, the pipelining between the host and NIC overheads are obstructed by the contention for the host memory, system bus, and I/O bus, and the factors causing these contentions are data copy, checksum computation, and frequent PIOs. These contentions are not reduced by the giga-hertz processor.

3. Enhancement of Protocol Overhead Pipelining

3.1. Host Memory and System Bus Contentions

The copy operation induces the host memory and system bus contentions with DMA and write-back operations, respectively.

In this paper, we eliminate the copy operation as follows: When an application calls the sending system call, it just saves packet information, such as the pointer to the user buffer and its length, without copying the data into the kernel buffer. Then, the data in the user buffer are directly moved to NIC by DMA when NIC is available to process the packet. In the receiver side, the receiving system call records the pointer to the user buffer like in the sender side and returns immediately. When the data arrives from the network, NIC moves it directly to the user buffer. The application can know the completion of sending/receiving for the buffer through a newly added system call.

This zero-copy mechanism, however, brings about additional overheads. The mechanism needs to traverse the page directory and table in order to translate the address of user buffer into the physical address. To reduce this overhead, we provide the cache of address translation. In addition, the data in the user buffer incurs the multiple DMA initializations because the user buffer is not physically linear when it crosses the page boundary. We prevent the multiple DMA initializations by allocating a physically linear user buffer.

The other fact that leads to the host memory contention is the checksum computation. We remove the checksum computation from the host by utilizing the DMA control block [15] on Myrinet NIC. Upon completion of the DMA operation, the DMA control block contains the checksum result.

We do not claim that the elimination mechanisms of copy operation and checksum computation are novel. Instead, we newly elucidate that the elimination of per-byte operations is indispensable for pipelining the host and NIC overheads.

3.2. I/O Bus Contention

The bursty PIO by the device driver competes for the I/O bus against the DMA operation. In order to solve this problem, we suggest a two-level queue structure. The level-1 (L1) queue is large and locates in the host memory, whereas the level-2 (L2) queue is small and locates in the NIC memory.

When an application issues a sending request, if the L2 send queue is not full, the device driver puts the descriptor for the sending request to the L2 send queue. Otherwise, the device driver puts the descriptor to the L1 send queue. When the NIC firmware completes a send, an interrupt is triggered, which means that L2 is not full. Accordingly, the interrupt handler gets a descriptor from the L1 send queue and puts it to the L2 send queue. This means that the L1 send queue behaves in a self-clocking manner, that is, it uses the interrupt of send completion as a clock to put a new descriptor into the L2 send queue. The two-level queue structure is also useful to the receiver when an application performs an asynchronous receiving request.

Consequently, we can mitigate the rate of PIO less than

$$Rate_{\max} = \frac{n + sizeof(L2)}{t}$$

where n is the maximum number of requests that NIC can process in time t. And *sizeof(L2)* is the number of requests that L2 queue can hold. The kernel and NIC do not need to know any details of the queue structure because the device driver allocates and manages the L1 queue.

The *Qdisc* structure of Linux is similar with the two-level queue structure, but implemented in the kernel. Consequently, the device driver requires knowing the details of *Qdisc*. In addition, *Qdisc* is applied to the sender side only.

3.3. π-UDP

We implement the three methods suggested in the previous subsections into UDP configured as Section 2, and name it π-UDP. The time chart of π-UDP is shown in Figure 5, which shows us that the overhead of each layer can fully pipeline with others from the sender side to the receiver side. Consequently, the largest overhead (i.e., NIC overhead) hides smaller overheads.

Figure 6 shows the throughput of π-UDP. We measure the throughput using *ttcp* that calculates the throughput in the receiver side with the difference of the first packet arrival time and the last packet arrival time. A notable result is that π-UDP achieves over 97% of the theoretical maximum throughput for all data sizes larger than 512B. The theoretical maximum throughput can be evaluated as follows:

$$Throughput_{\max} = \frac{PacketSize}{MAX(O_1, O_2, \cdots, O_s)}$$

where O_n denotes the processing overhead for a *PacketSize* packet at the nth layer and s is the number of layers.

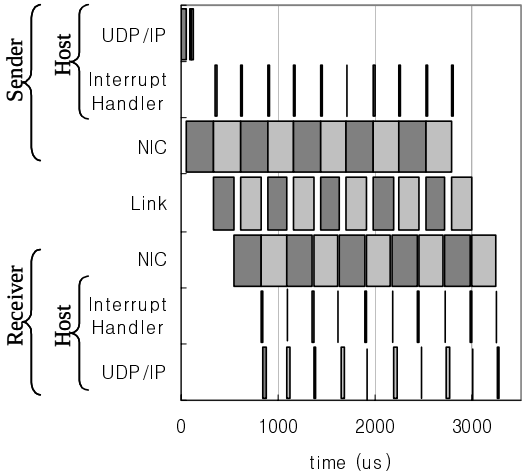


Figure 5: Overhead pipelining of π-UDP.

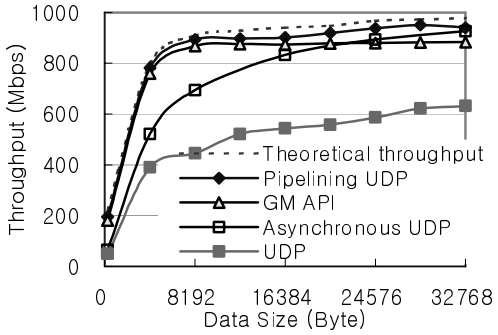


Figure 6: Comparison of throughput of π-UDP, GM API, Asynchronous UDP, and UDP with the theoretical maximum values.

The peak throughput of π-UDP is 951Mbps. Moreover, we observed that π-UDP achieved 1.22Gbps on a 64bit 66MHz PCI platform.

The throughput of *GM API* [13], a lightweight user-level protocol, is slightly lower than that of π-UDP. The reason is that *GM API* splits a large packet into 4KB sized fragments. This fragmentation partly pipelines the overheads for a packet at a layer and the right upper layer, which leads to a low one-way latency [6, 7]. However, the fragmentation sacrifices the throughput

because the per-packet overhead increases in proportion to the number of fragments.

Comparing with an improved UDP named *Asynchronous UDP* [16], π-UDP reaches to a near theoretical throughput at a prominently smaller data size. It is because *Asynchronous UDP* focuses only on reducing the per-byte overhead doing not consider the overhead pipelining.

4. Conclusions

This paper discusses the protocol overhead pipelining between the host and NIC. It should be emphasized that, while existing researches are focusing only on NIC, we thoroughly analyze the overhead pipelining between the host and NIC. Through the in-depth measurements, we elucidate that the host and NIC compete against each other for the hardware resources such as the host memory, system bus, and I/O bus and that the host and NIC overheads are not pipelined. The host operations that obstruct the overhead pipelining are the data copy, checksum computation, and frequent PIO operations.

In order to remove the contentions for hardware resources, we get rid of the copy operation and checksum computation from UDP, and implement a two-level queue structure into the device driver. As a result, π-UDP performs perfectly the overhead pipelining on Myrinet, and reaches over 97% of the theoretical maximum throughput for data sizes larger than 512B.

Overall, our study for the improvement of overhead pipelining will be very valuable for designing highly optimized I/O systems.

References

[1] L. Prylli and B. Tourancheau, "BIP: a new protocol designed for high performance networking on myrinet," Proceedings of IPPS/SPDP98, 1998.

[2] D. Dunning, G. Regnier, G. McAlpine, D. Cameron, B. Shubert, A. M. Berry, E. Gronke, and C. Dodd, "The Virtual Interface Architecture," IEEE Micro, 8:66-76, March-April 1998.

[3] J. Chase, A. Gallatin, and K. Yocum, "End-System Optimizations for High-Speed TCP," IEEE Communications, special issue on TCP Performance in Future Networking Environments, 39(4), April 2001.

[4] P. Shivam, P. Wyckoff, and D. Panda, "EMP: Zero-copy OS-bypass NIC-driven Gigabit Ethernet Message Passing," Proceedings of SC2001, November 2001.

[5] H. A. Jamrozik, M. J. Feeley, G. M. Voelker, J. Evans II, A. R. Karlin, H. M. Levy, and M. K. Vernon, "Reducing Network Latency Using Subpages in a Global Memory Environment," Proceedings of ACM ASPLOS, October 1996.

[6] L. Prylli, R. Westerlin, and B. Tourancheau, "Modeling of a High Speed Network to Maximize Throughput Performance: the Experience of BPI over Myrinet," Proceedings of PDPTA'98, 1998.

[7] R. Y. Wang, A. Krishnamurthy, R. P. Martin, T. E. Anderson, and D. E. Culler, "Modeling and Optimizing Communication Pipelines," Proceedings of ACM SIGMETRICS'98, June 1998.

[8] J. Chase, D. Anderson, A. Gallatin, A. Lebeck, and K. Yocum, "Network I/O with Trapeze," Proceedings of 1999 Hot-I, August 1999.

[9] B. Tourancheau and R. Westerlin, "Study of the Medium Message Performance of BIP/Myrinet," Proceedings of IEEE Cluster2000, November 2000.

[10] R. Westerlin, "A New Software Architecture for the BIP/Myrinet Firmware," Proceedings of IEEE CCGrid2001, May 2001.

[11] H. –W. Jin, C. Yoo, and J. –Y. Choi, "Firmware-Level Latency Analysis on a Gigabit Network," to appear in The Journal of Supercomputing, 2003.

[12] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W. Su, "Myrinet--A Gigabit-per-Second Local-Area Network," IEEE Micro, 15(1): 29-36, February 1995.

[13] Myricom Inc., The GM Message Passing System, http://www.myri.com, January 2000.

[14] Intel Corporation, Intel Architecture Software Developer's Manual, Volume 3: System Programming, 1999.

[15] Myricom Inc., PCI64 Programmer's Documentation, http://www.myri.com, May 2001.

[16] C. Yoo, H. –W. Jin, and S. –C. Kwon, "Asynchronous UDP," IEICE Transactions on Communications, E84-B(12): 3243-3251, December 2001.