

## Boosting Ethernet using Regular Switching Hubs

CHIH-WEN HSUEH, HSIN-HUNG LIN\* AND GUO-CHIUAN HUANG\*

*Graduate Institute of Networking and Multimedia and  
Department of Computer Science and Information Engineering  
National Taiwan University  
Taipei, 106 Taiwan*

*E-mail: cwhsueh@csie.ntu.edu.tw*

*\*Real-Time Systems Laboratory*

*Department of Computer Science and Information Engineering  
National Chung Cheng University  
Chiayi, 621 Taiwan*

*E-mail: {lsh; hgc89}@cs.ccu.edu.tw*

Ethernet is the most pervasive communication technology in use today. It is a standard for connecting computers to form a local area network and provides a common method for the exchange of data. In this paper, we propose a Linux solution, Bonding-Plus, to support real-time control and boost bandwidth using multiple network interface cards connected to regular switching hubs in an Ethernet environment. BondingPlus schedules packets in the data link layer without modification to the hardware or operating system in the host machine. Real-time packets can be transmitted via one or several dedicated bonding Ethernet interfaces, and there is no competition with low priority packets. Transmission delay and network jitter for real-time packets can be dramatically reduced. Furthermore, bandwidth can be increased in proportion to the number of bonding Ethernet interfaces under both TCP and UDP transmission. Various types of systems, such as clustering and parallel systems, can be enhanced at minimal hardware cost.

**Keywords:** channel bonding, link aggregation, packet scheduling, bandwidth increasing, Linux LAN

### 1. INTRODUCTION

Bandwidth is critical in most network systems, especially those with large data transmission. These include clustering and parallel systems, such as weather forecasting systems and bioinformatics systems. If the network bandwidth is not sufficient, packets may be delayed, leading to poor network performance. High priority packets may even have to compete with low priority packets and fail to satisfy their timing constraints [2, 13]. Therefore, bandwidth management is very important. Furthermore, in many server applications, transactions with real-time constraints or priorities need to be processed and send results back as soon as possible. Although we can use several network interface cards (NICs) in a host to obtain higher bandwidth, one IP address is needed for each NIC, and this is not practical in large-scale systems.

For the IEEE 802.3 network specification [18], the Link Aggregation Standard has been proposed to merge bandwidth and specify many additional features. Many products

---

Received October 10, 2005; accepted April 11, 2006.  
Communicated by K. J. Lin and T. W. Kuo.

which support the Link Aggregation Standard have been developed to increase the host bandwidth in a LAN environment; these include CISCO EtherChannel [20], Intel Link Aggregation [5], Sun Trunking [15], Linux Bonding [8], etc. Although bandwidth can be increased by using these approaches, most of them need special hardware support, which means extra cost. For example, a switching hub with Link Aggregation Standard support is several times more expensive than a normal one.

Nowadays, low-end NICs and switching hubs are very cheap. They support 100BASE-TX and Full-Duplex and have very high filtering/forwarding rates. We seek to boost Ethernet performance by making use of such inexpensive NICs simultaneously between hosts that are connected to normal switching hubs. The bandwidth of multiple bonding NICs of a host is merged within a single IP address and is increased in proportion to the number of bonding NICs [11]. Furthermore, since packets are scheduled in the data link layer, the packets of high priority or specific applications can be transmitted via one or several dedicated NICs between hosts [10]. The Ethernet performance can, thus, be improved with very little additional cost. Transmission delay and network jitter for real-time packets can also be massively reduced [7, 14] without any modification to the hardware or operating systems in the host machines.

The proposed approach is implemented as two Linux kernel modules [1, 3, 6, 17] between the IP layer [4, 19, 21] and Ethernet interface drivers. The BondingPlus driver creates a pseudo Ethernet interface when loaded and bonds multiple Ethernet interfaces. It receives packets from the IP layer and dispatches them to the underlying interfaces for transmission. The ARP+ module maintains the mapping between each IP address and the hardware addresses of all the bonding interfaces within a LAN environment in an ARP+ table. When BondingPlus transmits a packet, it queries the ARP+ table and modifies the source and destination hardware addresses of the packet accordingly.

The rest of this paper is organized as follows. The next section introduces Linux Ethernet Bonding. Section 3 discusses in detail the design and implementation of BondingPlus in a Linux LAN environment. In section 4, we present and analyze the experiment results. Conclusion are drawn in section 5.

## 2. LINUX ETHERNET BONDING DRIVER

The Linux Ethernet bonding driver is a kernel driver that can aggregate traffic over several ports of switches which support Link Aggregation [8]. When the Linux Ethernet bonding driver is initialized, it creates a pseudo Ethernet device and registers itself in the Linux kernel. The Linux kernel then initializes the pseudo Ethernet device and creates a link list which is used to hold Ethernet interfaces (called slaves) that can be used by the pseudo device. To make the pseudo Ethernet device work, we have to assign an IP address and add a routing setting. The pseudo Ethernet device is set as the master device of the slaves, and it adds the slaves to its link list. The hardware address of the pseudo Ethernet device is set as the first Ethernet interface of its slave list. All the hardware addresses of the subsequent Ethernet interfaces are also set as the pseudo Ethernet device.

When a packet received from an upper network layer (usually is IP layer) needs to be transmitted by the Bonding driver, the kernel passes the socket buffer to it. The Bonding driver selects an active Ethernet interface from its slave list, changes the output

device of the socket buffer to the selected device, and then enqueues the packet into the queue of the selected interface driver. The selected Ethernet interface is then responsible for sending the packet when the *NET\_TX\_SOFTIRQ* softirq of the Linux kernel is activated.

When a packet is received by one of the slaves, the driver of this slave device creates a new socket buffer and copies the data of the received packet into the socket buffer. Then, the driver stores the socket buffer in an appropriate queue for subsequent handling. When *NET\_RX\_SOFTIRQ* softirq is activated, the Linux kernel processes the packet queue. The Bonding driver changes the input device of the socket buffer to the pseudo Ethernet device. Thus, when a packet is received from any of the slave devices, it will be regarded as having been received from the pseudo Ethernet device. This bonding mechanism operates under the TCP/IP layer, so it is fully compatible with upper layers. However, Linux bonding needs to work with switches which support Link Aggregation; thus, the hardware cost is high.

### 3. DESIGN AND IMPLEMENTATION

This section describes the design and implementation of the two components of BondingPlus, the BondingPlus driver, and the ARP+ module. We also prove the correctness of ARP+ using a state transition diagram.

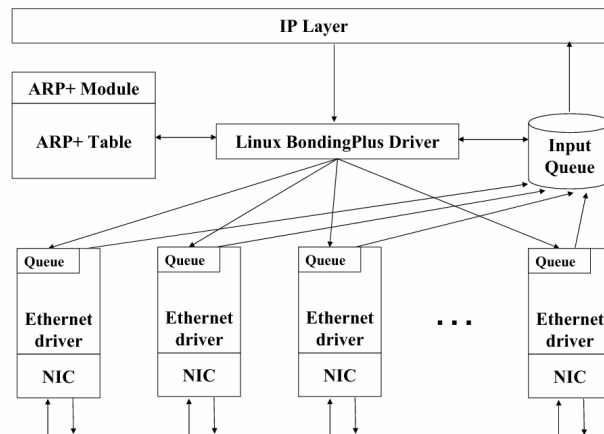


Fig. 1. BondingPlus architecture.

#### 3.1 BondingPlus

As shown in Fig. 1, BondingPlus is a pseudo Ethernet driver which resides between an IP layer and NIC drivers. It can bond and make use of multiple NICs simultaneously. When there is packet to be transmitted, BondingPlus receives the socket buffer from the upper IP layer and selects an adequate NIC from both sending and receiving hosts for transmission. Different scheduling policies can be applied for NIC selection. Currently, we implement the round robin and prioritized scheduling policies in BondingPlus. While

the round robin policy selects NICs alternately, the prioritized policy sends high priority or specific packets using dedicated NICs and sends other packets using the rest of the NICs. After changing the attributes of the socket buffer, including the source hardware address, destination hardware address, and output device, BondingPlus dispatches it to the selected underlying NIC driver.

BondingPlus is implemented as a Linux kernel module without any modification of the Linux kernel. After it is loaded into the Linux kernel, it creates a pseudo Ethernet interface and registers it to the kernel. Existing network configuration tools, like *ifconfig* and *route*, can be used to configure the pseudo BondingPlus interface as an ordinary Ethernet interface. After initialization and configuration is completed, BondingPlus broadcasts all of the bonding hardware addresses via the ARP+ protocol, which we will discuss in the next subsection. BondingPlus also creates a slave list and an ARP+ table. The slave list records in the system all of the Ethernet interfaces that are bonded to BondingPlus. The hardware address of the BondingPlus driver is set as its first bonding Ethernet interface. The ARP+ table contains the mapping between each IP address and all of its bonding hardware addresses of a LAN environment. Since the mapping between each IP address and its bonding hardware addresses in a LAN is recorded in the ARP+ table, BondingPlus needs no hardware support, and all the bonding interfaces can continue using their original hardware addresses. The IP address and all of the bonding hardware addresses of the BondingPlus driver are copied to the ARP+ table following IP address initialization.

Hosts may send address resolution requests via the address resolution protocol (ARP) if they are not loaded with BondingPlus. Only BondingPlus should reply to such requests, not its bonding Ethernet interfaces. Because one IP address with multiple hardware addresses will confuse other hosts and cause them to update their ARP tables frequently, BondingPlus sets a NOARP flag to all its bonding Ethernet interfaces to forbid them to respond to ARP requests. Moreover, packets received from bonding Ethernet interfaces should be considered as having been received from the BondingPlus pseudo interface. BondingPlus also sets a slave flag to its bonding Ethernet interfaces and manipulates the MAC header of received packets, such as the destination hardware address field.

When the Linux kernel allocates a new socket buffer, the priority of the buffer is set to the default value, 0. Every packet is put in the same queue of the Linux generic packet scheduler. If the load of a NIC is high, then high priority packets may be delayed by other packets. We can raise the priority of packets by using `setsockopt()` system call so that they can be processed first. Furthermore, to reduce the transmission delay of high priority packets, we can use the prioritized scheduling policy to send high priority or specific packets using one or more dedicated NICs and thus create a dedicated message channel between hosts with the minimum delay.

### 3.2 ARP+

The ARP [16] protocol is used by the Internet Protocol to map IP addresses to hardware addresses which are used by a data link protocol. It is used when IP is adopted over Ethernet. ARP maintains only a one hardware address to be bounded to an IP address in a LAN environment. Since BondingPlus makes use of multiple NICs with only a

single IP address, ARP is not applicable. Therefore, we designed ARP+ to map an IP address to all its bonding hardware addresses in a LAN environment. ARP+ records the mapping in an ARP+ table. The ARP+ table contains 256 entries and uses the suffix of an IP address as an index. For example, the mapping of IP address 192.168.102.185 is recorded in the 185th entry of the ARP+ table.

As shown in Fig. 2, we implement a proprietary packet which can only be interpreted by ARP+ without interfering with other existing protocols. The *HLEN* field specifies the length of the hardware addresses in this message. For IEEE 802 MAC addresses, the value is 6. The *PLEN* field specifies the length of the layer three protocol address in this message. For IP(v4) addresses, this value is 4. ARP+ fills in the number of bonding hardware addresses in the *Number of HA* field. There are four types of ARP+ messages which may be sent by the ARP+ protocol. They are identified by the value in the *ARP+ Packet Type* field of an ARP+ message. Finally, ARP+ fills in all the bonding hardware addresses following the IP address of the residing host.

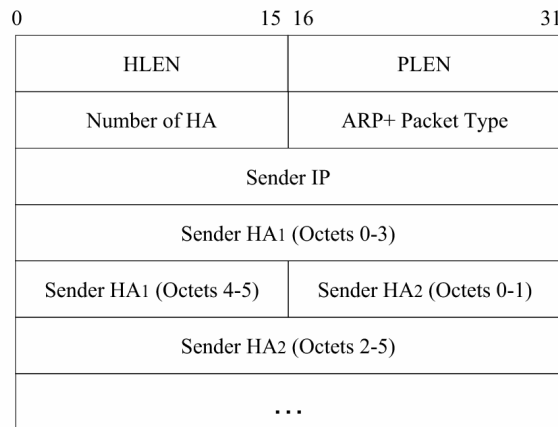


Fig. 2. Format of an ARP+ message.

We define four different types of ARP+ messages: *ARPP\_BROADCAST*, *ARPP\_REPLY*, *ARPP\_CHANGE*, and *ARPP\_CLEAR*. For example, when a host  $H_A$  newly joins a LAN, it sends an *ARPP\_BROADCAST* message which contains its IP address and all of the bonding hardware addresses using the Ethernet broadcast address. Since it is broadcasted, the message is received by all of the hosts in the same LAN. Therefore, all of the hosts become aware that there is a new host in the LAN and record the mapping between its IP address and bonding hardware addresses in the ARP+ tables. Afterwards every host receiving the *ARPP\_BROADCAST* message in the LAN unicasts an *ARPP\_REPLY* message which contains its own mapping between the IP address and bonding hardware addresses to host  $H_A$ . Host  $H_A$  constructs its ARP+ table accordingly.

If a host  $H_B$  changes its hardware address list, such as by adding or removing one or more NICs, it broadcasts an *ARPP\_CHANGE* message. Other hosts in the LAN that receive the message update their ARP+ tables accordingly. When a host  $H_C$  is going to unload BondingPlus or is ready to shut down, it broadcasts an *ARPP\_CLEAR* message to

notify other hosts. Hosts that receive this message remove the corresponding entry from their ARP+ tables.

BondingPlus is transparent to users and is backward compatible to hosts without loading it. If a host is not loaded with BondingPlus, which means it can not interpret ARP+ packets and will not reply, then there will be a null entry in the ARP+ tables of the other hosts that have BondingPlus. However, the hardware address will still be obtainable via the original ARP protocol, so hosts with BondingPlus will be able to send packets to hosts without loading it and vice versa. Consequently, there is no real-time support, and the bandwidth of a host without BondingPlus is not increased.

### 3.3 Correctness of ARP+

We can use a state transition diagram in Fig. 3 to verify the correctness of ARP+ [9]. There are four nodes, which represent all of the possible reachable states of the protocol in the diagram. Every transition in the diagram stands for a legitimate action and is labelled with the message event that is executed during the transition. Each transition has a label of the form  $\langle event\ type \rangle : \langle message\ type \rangle$ , where  $\langle event\ type \rangle$  is one of the following:

- B stands for broadcasting a message of the specified type;
- U stands for unicasting a message of the specified type;
- R stands for receiving a message of the specified type.

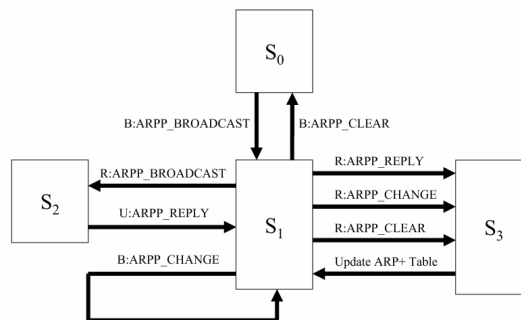


Fig. 3. State transition diagram of ARP+.

Initially, a host that has just joined starts at state  $S_0$  when the BondingPlus driver is loaded into the Linux kernel. At state  $S_0$ , there is only one legitimate action. When the host broadcasts an  $ARPP\_BROADCAST$  message, the host moves to state  $S_1$ .

At state  $S_1$ , six legitimate actions are enabled for execution. If the host receives an  $ARPP\_BROADCAST$  message, then it executes this action and moves to state  $S_2$ . If the host receives an  $ARPP\_REPLY$ ,  $ARPP\_CHANGE$  or  $ARPP\_CLEAR$  message, it moves to state  $S_3$ . The state remains unchanged when the bonding configuration is changed and an  $ARPP\_CHANGE$  message is broadcasted. When the host is going to stop bonding, it broadcasts the  $ARPP\_CLEAR$  message and moves to state  $S_0$ .

At state  $S_2$ , only one legitimate action is enabled for execution. After receiving the *ARPP\_BROADCAST* message, the host creates this entry in its ARP+ table. It then sends back an *ARPP\_REPLY* message. Executing this action leads the host back to state  $S_1$ .

At state  $S_3$ , after receiving an *ARPP\_REPLY*, *ARPP\_CHANGE*, or *ARPP\_CLEAR* message, the host updates its ARP+ table accordingly and then moves back to state  $S_1$ .

From the state transition diagram, it is clear that each action will eventually lead the host back to state  $S_1$  as long as the BondingPlus driver is loaded. The host can make progress in the cycle, hence, the correctness of ARP+ is established.

## 4. EXPERIMENTAL RESULTS

The main objective of this section is to evaluate the performance of BondingPlus. We performed the following experiments on two Intel Celeron 1.2GHz machines with 256MB memory each. The operating system on both machines was Mandrake 8.1 with the Linux kernel version 2.4.18. On each machine, we installed four NICs, which were directly connected to a DLink DES-1024R+ switching hub.

We used a widely used networking performance benchmark, *netperf* [12], to measure the performance of BondingPlus. Netperf is designed for use in a client/server architecture. In our experiments, while one machine executed its client, *netperf*, the other executed its server, *netserver*. The Netperf client continuously generated packets and sent them to the server; and the latency and throughput of the network could be measured.

### 4.1 Increasing Bandwidth

We measured the increase in bandwidth when multiple NICs were bonded into the system. Two types of NICs, Intel 21143 and Intel 82559, were tested for both TCP and UDP transmission, and, round robin scheduling policy was used in this experiment. The drivers of the Intel 21143 and Intel 82559 NICs in the experiment were “tulip” and “eepro100,” respectively, and can be found in the Linux kernel source tree. The results are shown in Fig. 4. The bandwidth for both TCP and UDP transmission increased in proportion to the number of bonding NICs. The bandwidth for UDP transmission was slightly higher, because it had less overhead as compared to TCP transmission. When the number of bonding NICs doubled, the network bandwidth nearly doubled.

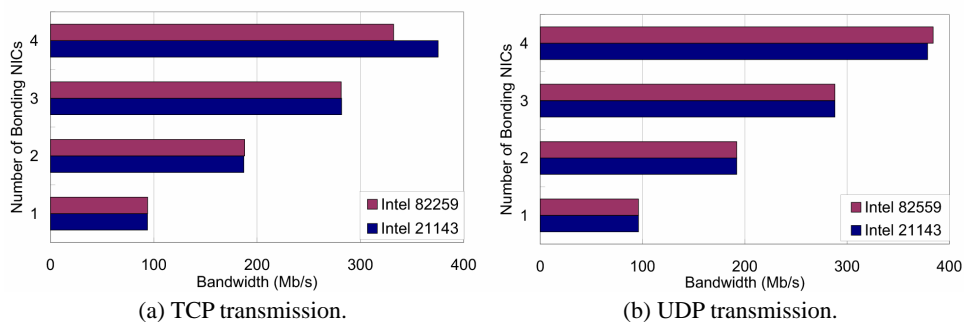


Fig. 4. Bandwidth increasing using BondingPlus.

However, some NICs consume lots of CPU resources when transmitting packets at full speed. Due to the computing power bottleneck, network bandwidth stops increasing, even if more NICs are bonded. This was verified by the following experiment. We measured the bandwidth of three different processors with the same system and network configuration. All the hosts were bonded with two Realtek 8139 NICs, which consume considerable amounts of CPU resources when transmitting at full speed. The “8139too” driver was used for the NICs in this experiment and can also be found in the Linux kernel source tree. As shown in Table 1, this phenomenon was eliminated when we used a more powerful machine or NICs which consumed few CPU resources when transmitting.

**Table 1. Bottleneck of computing power.**

Host CPU	PII 300MHz	PIII 700MHz	P4 1.6GHz
Bandwidth	133Mb/s	171.5Mb/s	180Mb/s

We also measured the bandwidth of each bonding NIC to evaluate the load balancing achieved with BondingPlus. Fig. 5 shows the results of the experiments. The contribution to bandwidth of each NIC was almost the same, both in TCP and UDP transmission. Since all of the packets were set to the default priority and round robin scheduling policy was used, outgoing and incoming network packets were equally dispatched to all of the bonding NICs, which, thus, could all be fully utilized by BondingPlus.

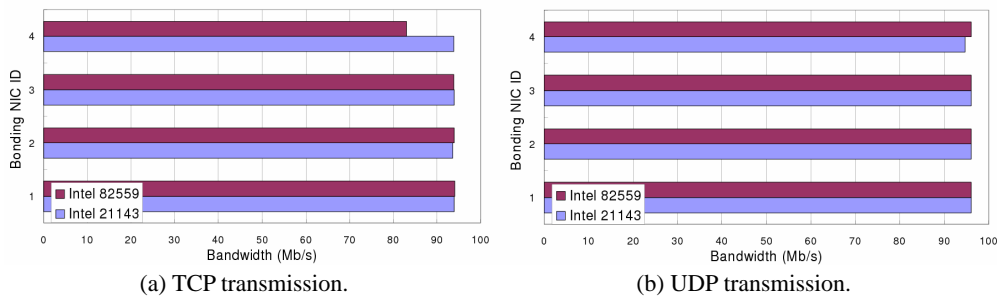


Fig. 5. Load balancing of BondingPlus.

#### 4.2 Prioritized Transmission

In this experiment, two programs were executed simultaneously on both hosts. One program generated and sent high priority packets to the other host periodically, and the other, which was considered to be a source of interference in the system, continuously generated and sent a large buffer of low priority packets to the other host. The prioritized scheduling policy was used, and three different scenarios were tested:

- One Channel Priority 0: In this scenario, high priority packets and interference packets had the same priority: 0. They were put in the same queue of the Linux Generic Packet Scheduler.

- One Channel Priority 6: The priority of the high priority packets was set to a higher value, 6, and the priority of the interference packets was still set to 0. High priority packets were assigned higher priority and were processed first.
- Two Channel: In this scenario, high priority packets were sent and received via dedicated NICs on each host and so were low priority packets.

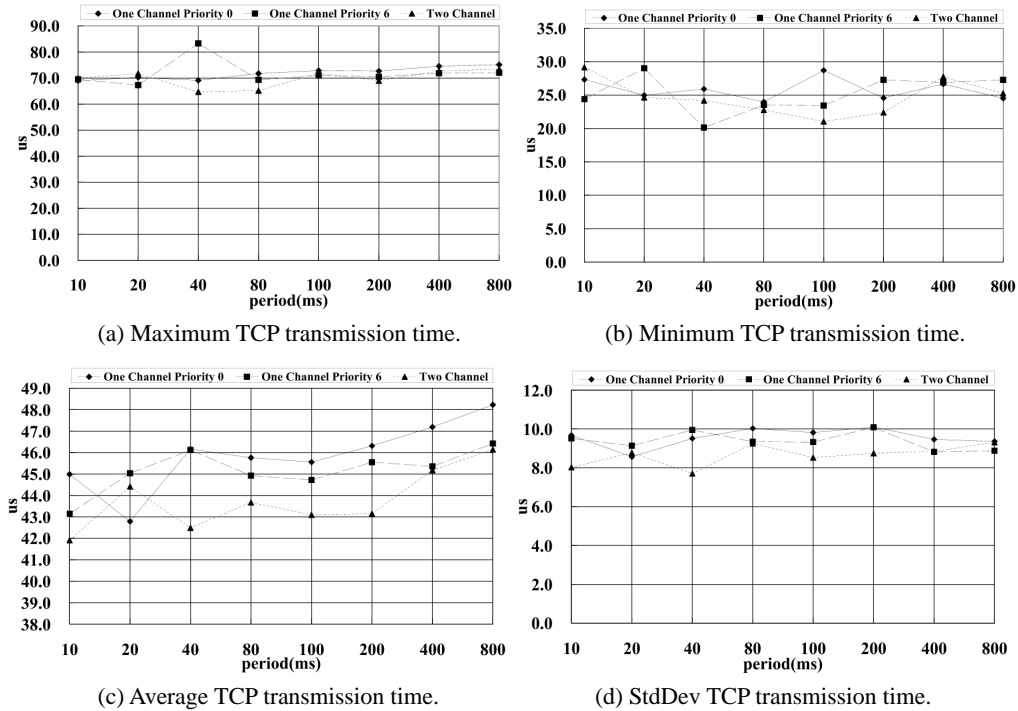


Fig. 6. TCP transmission time from applications to BondingPlus.

Fig. 6 shows the maximum, minimum, average, and standard deviation of the TCP transmission time from applications to the BondingPlus driver for high priority packets. The time was mainly spent in the TCP and IP layer, which is not controlled by BondingPlus. The average transmission time was between 42us and 49us, and there was almost no difference in average transmission time for the 3 scenarios.

Fig. 7 shows the maximum, minimum, average, and standard deviation of the TCP transmission time from the BondingPlus pseudo driver to the NIC drivers. In One Channel Priority 0 scenario, the transmission time for high priority packets transmitted from the BondingPlus driver to the NIC drivers was very long, because they had to compete with low priority packets. The transmission time could be reduced dramatically in the One Channel Priority 6 scenario. But the large standard deviation shows that high priority packets were still interfered with by the low priority packets. Transmitting high priority packets and low priority packets in the Two Channel scenario resulted in the lowest

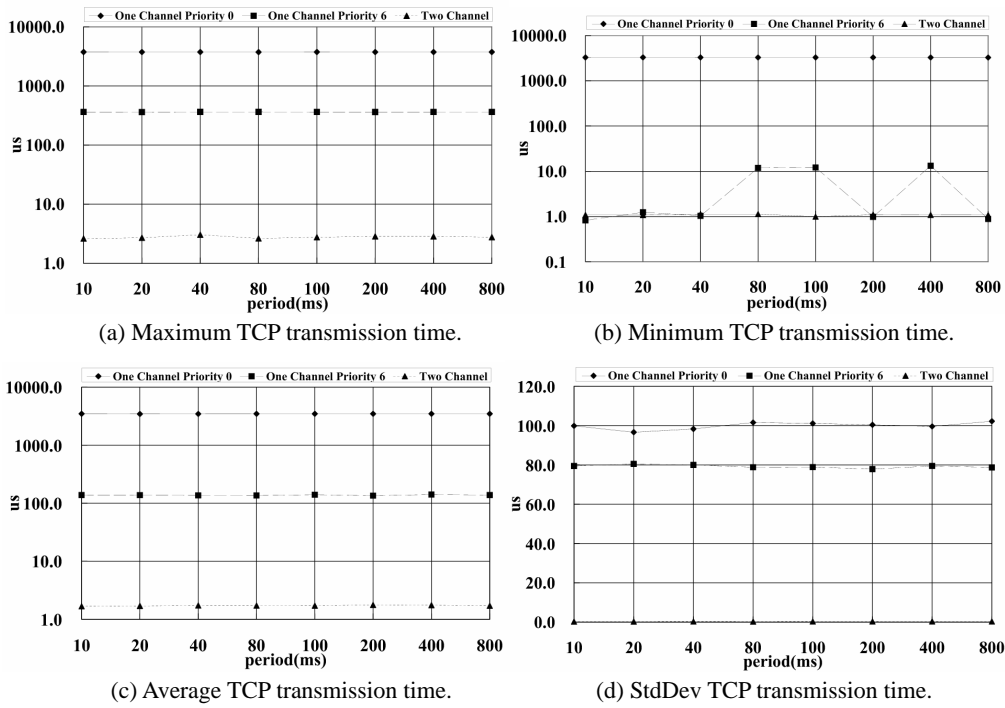


Fig. 7. TCP transmission time from BondingPlus to NIC drivers.

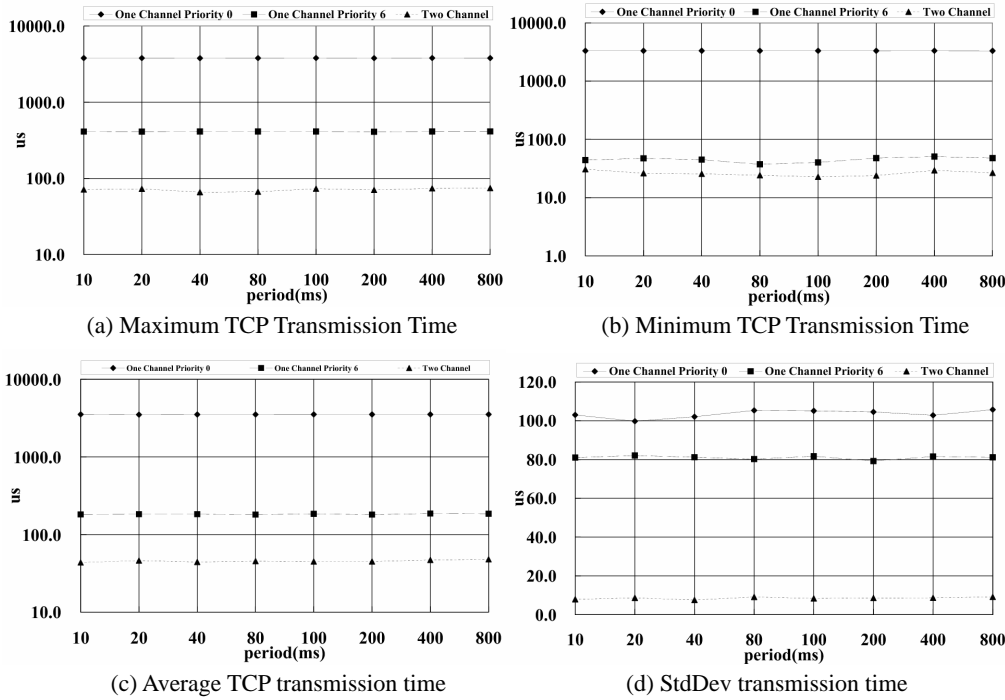


Fig. 8. TCP transmission time from applications to NIC drivers.

transmission overhead for high priority packets. BondingPlus can establish virtually real-time message channels for specific applications between hosts, thus reducing network jitter in a LAN environment.

Fig. 8 shows the maximum, minimum, average, and standard deviation of the TCP transmission time from the applications to the NIC drivers. The time needed for transmission from the applications to the BondingPlus driver remained almost constant due to the time needed for transmission from the BondingPlus driver to the NIC drivers was the main factor affecting the packet transmission time. Although One Channel Priority 6 transmission could greatly reduce the transmission time for high priority packets, Two Channel transmission resulted in the greatest improvement.

We performed the same experiments with packet transmission over UDP. The results were similar to those for transmission over TCP and are shown in Fig. 9. The UDP transmission time for high priority packets was slightly shorter than that for TCP transmission due to its lower overhead. The main factor affecting the packet transmission time was the time needed for transmission from the BondingPlus driver to the NIC drivers. The transmission time from the applications to the BondingPlus driver remained almost constant in all of the scenarios. Still, although One Channel Priority 6 transmission was able to greatly reduce the packet transmission time for high priority packets, Two Channel transmission again resulted in the greatest improvement.

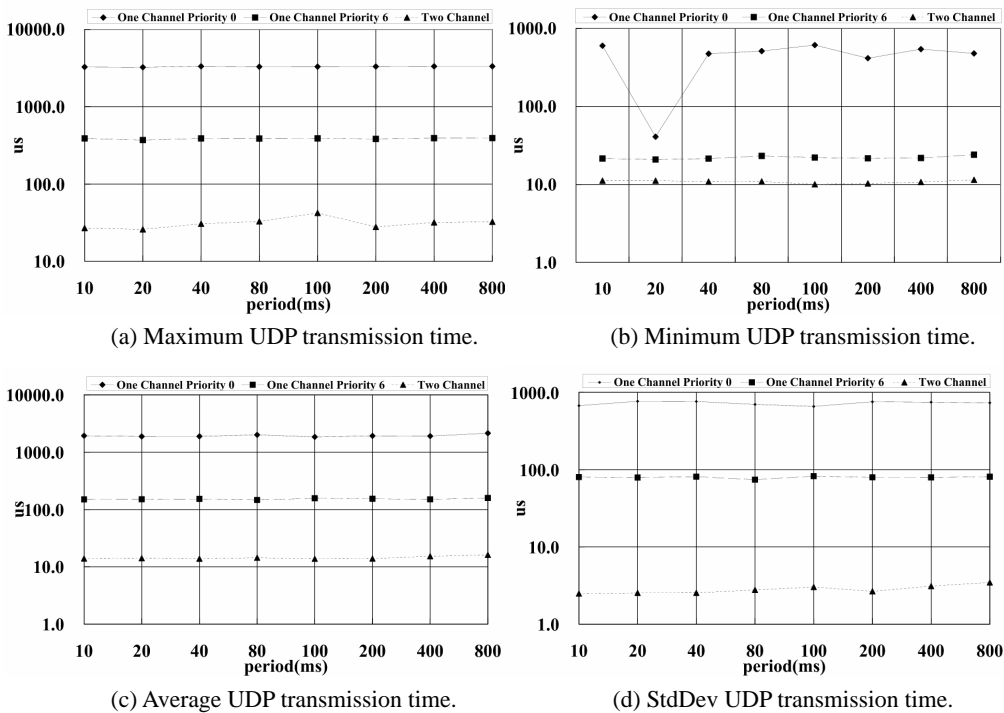


Fig. 9. UDP Transmission time from applications to NIC drivers.

### 4.3 Overhead Evaluation

We performed the following experiments to evaluate the effect of BondingPlus on a system. We measured network bandwidth and CPU utilization when one NIC was used with the default kernel driver on each host. The results were compared with those obtained when BondingPlus was used to bond one NIC to each host. As shown in Table 2, BondingPlus only decreased the bandwidth of a NIC by 0.05% and increased CPU time by 6.6% on average when packets were sent and received for 60 seconds. Therefore, the overhead of BondingPlus is manageable.

**Table 2. Overhead of BondingPlus.**

BondingPlus	Without	With
Bandwidth	94.05Mb/s	94.01Mb/s
Minimum User Time	0s	0s
Maximum User Time	0s	0s
Minimum System Time	2.11s	2.33s
Maximum System Time	2.32s	2.49s
Average System Time	2.25s	2.40s

## 5. CONCLUSION

We have proposed a new and inexpensive approach to boosting Ethernet performance using normal switching hubs. The ARP+ is designed to maintain the mapping of an IP address and all of the bonding hardware addresses of a host in a Linux LAN environment. We have also designed and implemented the BondingPlus pseudo Ethernet interface driver, which can schedule packets in the data link layer and make use of multiple NICs connected to regular switching hubs simultaneously. Packets can be sent and received via multiple bonding NICs with only one IP address. High priority packets of specific applications can be transmitted via one or several dedicated NICs; thus, the transmission delay and network jitter can be dramatically reduced. Furthermore, the bandwidth of a host can be increased in proportion to the number of bonding NICs. We believe BondingPlus is practical for use in network systems and that it can greatly reduce hardware costs. We will further implement BondingPlus Linux 2.6 driver due to the popular use of Linux 2.6 today.

## REFERENCES

1. T. Aivazian, "Linux kernel 2.4 internals," <http://www.tldp.org/LDP/iki/index.html>.
2. R. Bettati, "End-to-end scheduling to meet deadlines in distributed systems," Ph.D. Dissertation, Technical Report, No. UIUCDCS-R-94-1840, Dept. of Computer Science, University of Illinois at Urbana-Champaign, U.S.A., 1994.
3. D. P. Bovet and M. Cesati, *Understanding the LINUX KERNEL*, O'Reilly, 2001.
4. D. Comer, *Internetworking with TCP/IP: Principles, Protocols, and Architecture*, Vol. 1, Prentice Hall, 2000.

5. Intel Corporation, *Intel Link Aggregation*, <http://www.intel.com/support/express/switches/53x/31460.htm>.
6. J. Crowcroft and I. Phillips, *TCP/IP and Linux Protocol Implementation: Systems Code for the Linux Internet*, Wiley, 2002.
7. M. D. Natale and J. A. Stankovic, "Scheduling distributed real-time tasks with minimum jitter," *IEEE Transactions on Computers*, Vol. 49, 2000, pp. 303-316.
8. T. Davis, "Linux bonding," <http://sourceforge.net/projects/bonding>.
9. M. G. Gouda and C. T. Huang, "A secure address resolution protocol," *Computer Networks: the International Journal of Computer and Telecommunications Networking*, Vol. 41, 2003, pp. 57-71.
10. H. H. Lin, C. W. Hsueh, and G. C. Huang, "Bondingplus: real-time message channel in Linux Ethernet environment using regular switching hub," in *Proceedings of 9th International Conference on Real-Time and Embedded Computing Systems and Applications (RTCSA)*, 2003, pp. 176-193.
11. H. H. Lin, C. W. Hsueh, and G. C. Huang, "Channel bonding in linux ethernet environment using regular switching hub," *Journal of Systemics, Cybernetics and Informatics*, Vol. 2, 2004.
12. R. Jones, *Netperf*, <http://www.netperf.org/>.
13. D. W. Leinbaugh, "Guaranteed response time in a hard real-time environment," *IEEE Transactions on Software Engineering*, Vol. SE-6, 1980, pp. 85-91.
14. K. J. Lin and A. Herkert, "Jitter control in time-triggered systems," in *Proceedings of 29th Hawaii International Conference on System Sciences*, 1996, pp. 451-459.
15. S. Microsystems, "Sun trunking," <http://www.sun.com/products-nsolutions/hw/networking/connectivity/suntrunking>.
16. D. C. Plummer, "An Ethernet address resolution protocol," RFC 826, <http://www.ietf.org/rfc/rfc826.txt>, 1982.
17. A. Rubini and J. Corbet, *Linux Device Drivers*, 2nd ed., O'Reilly, 2001.
18. IEEE 802.3 Std., *IEEE 802.3 CSMA/CD Access Method*, IEEE, 2000.
19. W. R. Stevens, *TCP/IP Illustrated: The Protocols*, Vol. 1, Addison-Wesley, 1994.
20. Cisco Systems, *ETHERCHANNEL*, <http://www.cisco.com/en/US/tech>.
21. G. R. Wright and W. R. Stevens, *TCP/IP Illustrated: The Implementation*, Vol. 2, Addison-Wesley, 1995.



**Chih-Wen Hsueh (薛智文)** is an Assistant Professor in the Department of Computer Science and Information Engineering at the National Taiwan University (NTUCSIE), R.O.C. since March 2006. His research interests include real-time systems, embedded systems, and operating systems. He received the B.S. degree from NTUCSIE in 1989, June. After two years of military service, he worked for one year as a research assistant in the Institute of Information Science at the Academia Sinica, R.O.C. (ASIIS). He received his M.S. degree in Computer Science from the University of Southern California in 1994, June, and Ph.D. in Information and Computer Science from the University of California at Irvine in 1997, December. Then, he worked for several startups in Bay Area. He joined as an Assistant Professor in

the Department of Computer Science and Information Engineering at the National Chung Cheng University, R.O.C., in 1999, August and an associate research engineer in ASIIS, in 2005, August.



**Hsin-Hung Lin** (林信宏) is a PhD student in the Department of Computer Science and Information Engineering, National Chung Cheng University, R.O.C. since 2002. His research interests include real-time scheduling, embedded systems, distributed systems and operating systems. He received his MS and BS degree in Computer Science and Information Engineering from the National Chung Cheng University in June, 2000 and 2002.



**Guo-Chiuan Huang** (黃國全) received his M.S. and B.S. degrees in Computer Science and Information Engineering from the National Chung Cheng University in June, 2000 and 2002. His research interests include operating systems, computer networks, and embedded systems.