Building High-Performance and Reconfigurable Bandwidth Controllers with Adaptive Clustering

Chien-Hua Chiu, Chin-Yen Lee, Pan-Lung Tsai, Chun-Ying Huang, and Chin-Laung Lei
Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan
{william, hamblue, charles, huangant}@fractal.ee.ntu.edu.tw, lei@cc.ee.ntu.edu.tw

Abstract—**As quality of service gains more and more attention, bandwidth controllers gradually become one of the most important network systems used in modern Internet environment. The demand for high-performance in-line bandwidth controllers is driven by the growing bandwidth available in the last mile WAN links as well as the sophisticated packet processing functions that become essential in current computer networks. In this paper, we propose an adaptive clustering scheme to scale the throughput of in-line devices and implement the bandwidth control functions over a cluster of in-line devices. The proposed scheme aggregates the processing power of multiple in-line devices in the cluster by making incoming traffic self-dispatched in a transparent fashion, and incorporates a flow migration mechanism that keeps the load of each device balanced. The resulted cluster is also able to tolerate device failures and hence is run-time reconfigurable. Based on the proposed scheme, we successfully design a distributed policy adjustment algorithm, the proportional bandwidth allocation algorithm, and implement a clustered bandwidth controller over embedded Linux. The results of performance evaluation suggest that the proposed traffic redistribution mechanism and distributed policy adjustment algorithm can be used together to realize high-performance and reconfigurable bandwidth controllers.**

Keywords-traffic dispatching; in-line device; bandwidth controller; proportional bandwidth allocation; fault tolerance; load balance

I. INTRODUCTION

Since its invention, computer networks have gradually become an essential part in our daily lives. Not only applications including peer-to-peer (P2P) file sharing, voice over Internet protocol (VoIP), instant messaging (IM) are getting more and more popular and start to change our personal life styles, but companies and organizations also take advantages of computer networks to greatly reduce the cost of information exchange and resource management, and sometimes even rely on them to carry out critical business transactions.

However, the Internet is so designed that network resources are shared by all applications that generate packets. Under such circumstances, applications that send packets faster are served with higher probability, and applications that send packets at lower rates may experience starvation in a congested network. For example, an HTTP bulk file transfer may consume all available bandwidth and forbid another VoIP application from operating properly. This is why bandwidth management (BM) mechanisms are invented to help by providing guarantees to various quality-of-service (QoS) requirements needed by various applications. With BM mechanisms, network administrators are able to set up policies according to which network packets are processed so that mission critical applications can be protected from the over-consuming behavior of less critical but demanding applications.

Researchers [1,2] and vendors [3,4] have figured out different approaches to augment existing local area networks (LANs) with the advantageous BM functions mentioned above. A straightforward way is to replace each and every edge router with its new, BM-enabled generation, and this is surely less preferred, especially when the cost of model revisions and the complexity of router configuration migration in the installed base are considered. A much better way is to engineer standalone bandwidth controllers in the form of in-line devices that sits right in front of edge routers and works transparently to other nodes on individual networks. Thus, the standalone bandwidth controllers can be nicely and easily inserted into existing LANs with minimal disruptions to the original networks. Figure 1 shows a typical deployment of such transparent, in-line bandwidth controllers that may effectively shape the traffic passing through the link between a LAN and a wide area network (WAN).

Bandwidth controllers perform sophisticated tasks like queue management, statistics maintenance, and classification upon packet arrival. Such operations consume much of the processing power but still in an acceptable level with small scale networks and limited bandwidth. The performance requirement of bandwidth controllers primary depends on the bandwidth available at the last mile (i.e., the link between the

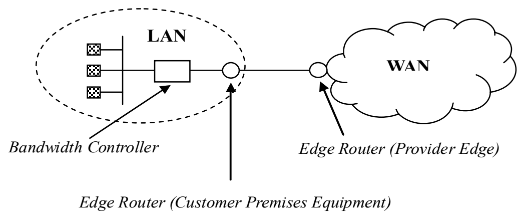
LAN
WAN
Bandwidth Controller
Edge Router (Provider Edge)
Edge Router (Customer Premises Equipment)

Figure 1. A typical deployment of transparent, in-line bandwidth controllers.

This work is supported in part by the National Science Council under the Grants NSC 95-3114-P-001-001-Y02 and NSC 95-2213-E-002-009, and by the Taiwan Information Security Center (TWISC), National Science Council under the Grants No. NSC 94-3114-P-001-001Y and NSC 94-3114-P-011-001.

edge routers, as depicted in Figure 1). As communication technology keeps advancing, broadband services such as fiber to the building (FTTB)/fiber to the home (FTTH) [5] gradually become more and more popular, and the bandwidth available on the WAN link has grown to a rate of tens or even hundreds of megabits per second (Mbps). Therefore, the time available for bandwidth controllers to process a single packet has decreased to the level of few microseconds or less. On the other hand, as Internet applications keep evolving, bandwidth controllers need to recognize more and more difference protocols, and the time required to process each packet grows. These two factors have made bandwidth controllers, which have to deal with aggregate traffic transmitted to and from individual hosts, become the new performance bottleneck.

In this paper, we propose an adaptive clustering scheme to scale the throughput of in-line devices. The proposed scheme aggregates the processing power of multiple in-line devices in a cluster by transparently making incoming traffic self-dispatched, and it also incorporates the flow migration mechanism in our previous work [6] to keep the load of each device in the cluster balanced. In addition to load balancing, the proposed scheme also encompasses the ability of fault tolerance and hence makes the resulted cluster run-time reconfigurable. Based on this clustering scheme, we successfully design a distributed policy adjustment algorithm and implement a clustered bandwidth controller over embedded Linux.

The rest of the paper is organized as follows. Section II gives a brief review of the literatures related to improving the performance of BM devices. Section III elaborates the proposed clustering scheme, and describes additional details specific to the clustered bandwidth controller. Section IV then presents the results of performance evaluation. Finally, we conclude the paper by summarizing our achievements and discussing the future work in Section V.

II. Related Work

Each solution proposed to improving the performance of network devices generally follows one of the two approaches. The first one is the scale up approach [7], which aims at increasing the processing power of a single device by upgrading hardware components or optimizing algorithms, and the second one is the scale out approach [7], in which several devices collaborate to form a cluster and then together provide more processing power to handle higher volume of network traffic. Although the most common scale up solutions such as the hardware-based ones using application specific integrated circuits (ASICs) have been long proven in the field to be capable of delivering high throughput for well-defined operations, they are also infamous for their inflexibility in that the revision of hardware chips is both time consuming and costly. Besides, the enhancement of hardware may quickly hit various physical, electrical, and thermal limitations. Therefore, we follow the scale out approach and propose a scalable clustering scheme to improve the performance of bandwidth controllers by aggregating multiple devices to form a cluster. The proposed scheme does not suffer the limitations mentioned above, and it does not interfere with hardware enhancement and hence can be combined with the scale up approach to achieve even higher performance.

In order to fully utilize the capability of each device in the cluster, the incoming traffic need to be properly dispatched among the devices. In general, traffic dispatching mechanisms fall into two categories, the centralized and the decentralized ones. For the centralized approach, traffic dispatching mechanisms are usually implemented in a dedicated dispatcher. The dispatcher may redirect individual packets to different in-lines devices in simple ways such as round robin, or it can be as flexible as evaluating complicated criteria like selecting the device with least load over the past ten seconds. The major advantage of the centralized approach is its simplicity that the issue of load distribution is isolated and taken care of solely by the dispatcher. However, the shortcomings of the approach are also obvious in that the dispatcher itself is likely to become the performance bottleneck and a single point of failure.

As for the decentralized approach, the devices in the cluster may perform special processing on ARP [8] requests in order to distribute the input traffic among themselves. For example, the clustered units in [9,10] can be configured to answer ARP requests either with a nonsexist Ethernet address or with a layer-2 multicast address so that the switch residing between the cluster and the hosts will always flood the frames sent to the cluster. The clustering scheme proposed in [6] first generates a number of virtual MAC addresses and uses them as the answers when replying ARP requests. These mechanisms make the cluster transparent to other nodes on the same network and all member nodes in the cluster jointly create an undistinguishable illusion of a device. Section III presents the decentralized traffic dispatching mechanism we propose for the in-line device cluster, and also our implementation of the BM functions based on our traffic dispatching mechanism over embedded Linux.

The traffic control framework designed for Linux integrates various components needed for traffic policing, shaping, and so on. The framework consists of three basic elements, which are queueing disciplines, classes, and filters [11,12]. Queueing disciplines are algorithms that schedule the packet transmission of a network interface, classes are subsidiaries of queueing disciplines that can be defined to specify different QoS requirements, and filters are used for classifying packets into classes. Many queueing disciplines, including the famous class-based queueing (CBQ) [13], the well-known token bucket filter (TBF), are already implemented in the mainstream Linux kernel [14], and also widely used by Linux-based bandwidth controllers. We choose hierarchical token bucket (HTB) [15] as the primary queueing discipline in our implementation for its flexibility, stability, and accuracy.

III. The Proposed Scheme

In this section, we first present the system architecture, and then describe the proposed clustering scheme that follows the scale out approach and makes use of a decentralized traffic dispatching mechanism in great detail. We also extend the flow migration techniques proposed in [6] to achieve the goals of both load balancing and fault tolerance. In the end of this section, an adaptive algorithm for dynamic and distributed policy adjustment, the proportional bandwidth allocation (PBA) algorithm, is proposed.

A. The System Architecture

Figure 2(a) depicts a typical deployment of a bandwidth controller implemented in the form of an in-line device, which is located between the hosts and the edge router (i.e., the gateway to the Internet). In order to achieve higher throughput while enforcing BM policies, the single in-line device is replaced with a cluster of multiple devices as illustrated in Figure 2(b). The cluster is constructed by surrounding all member devices with two Ethernet switches and also adding a third network interface to each device for the inter-device communication. Note that the third interfaces, the switch used for inter-device communication, and the cables are not explicitly shown in Figure 2(b).

For the reason of keeping the in-line-device cluster working transparently, each device is deployed as a bridge. In order to fully utilize the capability of each device in the cluster, we expect the frames sent by different hosts be forwarded to different devices so that the traffic can be dispatched evenly to each device as shown in Figure 2(b). Following the scale out approach, such design of the system architecture above is straightforward, but the system architecture alone cannot achieve the design goals because of the following reasons. First, the loop topology of the designed architecture does not allow broadcast frames send to it, or infinite frame forwarding loops may occur. Second, due to the self-learning behavior [16] of the Ethernet switches, the port mapping of a MAC address corresponds to only one of the interfaces of the switch. Therefore, while all hosts send frames to a same MAC address, the Ethernet switch forward all the frames to a same interface. In other words, all frames are forwarded to a single device of the cluster, and the behavior like this violates the desired objectives.

B. Design Fundamentals and Notations

The proposed traffic dispatching scheme is adapted from the proxy ARP scheme [17] and incorporates additional mechanisms that create the illusion of a single, high-performance in-line device with multiple underlying in-line devices. Each device in the cluster acts as a proxy between the hosts and the edge router by means of the frame rewriting techniques, and avoids infinite frame forwarding loops with the help of frame filtering. In addition, the scheme also actively influences the self-learning behavior of the surrounding switches so that frames sent by individual hosts can be assigned to one of the clustered devices in a controlled manner. As for fault tolerance, a heartbeat mechanism is proposed so that devices in the cluster communicate with one another through a private inter-communication channel. Based on the information exchanged, devices are able to detect node failure and then activate traffic redistribution. As a result, the cluster has the features of load balancing and fault tolerance and is hence run-time reconfigurable.

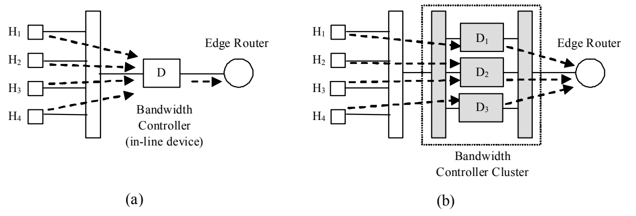

Figure 2. (a) A typical deployment of a bandwidth controller in LAN environment, and (b) System architecture of a clustered bandwidth controller. H_1, H_2, H_3, and H_4 denote individual hosts in the LAN, and D_1, D_2, and D_3 denote the in-line devices that comprise the cluster.

Some notations that are going to be used in the following sections are described as follows.

- Each in-line device of the cluster is assigned a unique integer as its **ID**.
- Each device maintains its own $\mathbf{L_V}$, a set of virtual MAC addresses.
- Two sets, $\mathbf{V_L}$ and $\mathbf{V_R}$, of predefined unique virtual MAC addresses, respectively, where $|\mathbf{V_L}| = N_\mathbf{L}$ and $|\mathbf{V_R}| = N_\mathbf{R}$. The values of $N_\mathbf{L}$ and $N_\mathbf{R}$ should be sufficiently large numbers.
- Two hash functions, $h_\mathbf{L}$ and $h_\mathbf{R}$, taking a MAC address, m, as input and map it to respective virtual addresses in $\mathbf{V_L}$ and $\mathbf{V_R}$. That is, $h_\mathbf{L}$: $\mathbf{M} \rightarrow \mathbf{V_L}$ and $h_\mathbf{R}$: $\mathbf{M} \rightarrow \mathbf{V_R}$, where $\mathbf{M} = \{m \mid m \in \mathbf{Z}, 0 \leq m \leq 2^{48} - 1\}$.

C. The Traffic Dispatching Scheme

The proposed traffic dispatching scheme comprises six steps depicted by Figure 3. ARP request packets are sent at the *1st* step when the hosts want to send packets to the Internet. Since the ARP request packet in the *2nd* step is broadcast to all the devices in the cluster, a frame filtering mechanism, which we name as responsibility check, is designed to prevent the infinite frame forwarding loops. The responsible device of a particular ARP request is determined in the responsibility check by taking the source MAC address of the ARP request as input and mapping it to one of the **ID**s. Once an **ID** is computed, only will the device with that **ID** forward the frame, while others simply ignore the request.

As a transparent proxy sitting between hosts and the edge device, in the *3rd* step, the source MAC address of the forwarded frame is replaced with a virtual MAC address $\boldsymbol{v}_\mathbf{L}$, which is generated by $h_\mathbf{L}$ and is an element of $\mathbf{V_L}$. In the *4th* step, when the ARP reply sent to $\boldsymbol{v}_\mathbf{L}$ is received by the same responsible in-line device, the device records the mapping of the IP and the MAC address of the responder. In the *5th* step, a proxy-ARP reply is sent back to the host. The destination and source MAC addresses of the reply are set to the MAC address of the host and a virtual MAC address $\boldsymbol{v}_\mathbf{R}$, respectively. The $\boldsymbol{v}_\mathbf{R}$ is generated by $h_\mathbf{R}$ and is an element of $\mathbf{V_R}$. At the same time, the $\boldsymbol{v}_\mathbf{R}$ is also added to the $\mathbf{L_V}$ of the responsible device. Throughout the first five steps, the network traffic sent from different hosts can be

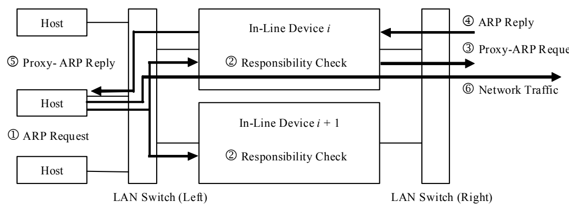

Figure 3. Six steps of the proposed traffic dispatching scheme.

dispatched to corresponding devices according to MAC addresses of hosts. In the final step, the source MAC address of an outbound data packet is replaced with one of the virtual MAC addresses in $\mathbf{V_L}$. In addition, the destination MAC address of the packet is replaced with the real MAC address of the destination IP address, which is stored in the *4*th step. On the other hand, the source MAC address of an inbound packet is replaced with one of the virtual MAC addresses in $\mathbf{V_R}$ and the destination MAC address of it is set to the MAC address of the sending host.

D. The Fault Tolerant Mechanism

Following the six steps in the previous scheme, network traffic is dispatched to devices according to results of the responsibility check. Since the responsibility check is a hash-based operation, the traffic may not be guaranteed to dispatch evenly. Therefore, we apply the flow migration technique proposed in our previous work [6] to balance the workload of the cluster by migrating virtual MAC addresses among all the collaborative devices.

In this work, we use a heartbeat mechanism to detect fault nodes in the cluster. Each node in the cluster periodically broadcasts a heartbeat message, which contains the **ID** and the $\mathbf{L_V}$ of the node, using the private inter-communication channel. By collecting all the heartbeat messages, a device hence knows the global information of the cluster, which includes the total number of active devices *N*, the health state of each node, and the virtual MAC addresses that an arbitrary node handles. The information is important for the construction of a fault tolerant mechanism. When a node fails, all the virtual MAC addresses originally handled by the failed one must be migrated to other active devices. To do this in a distributed manner, each node has to compute its *rank* before the redistribution of those orphaned virtual MAC addresses. A node's rank is defined as the ascendant order of its **ID** among all the active nodes. Therefore, the node with the smallest **ID** has a rank of zero and the one with the largest **ID** has a rank of $N-1$. An example of the rank computation is given as follows: Suppose that there are three active nodes in the cluster and the **ID**s of them are 2, 4, and 5, respectively. As a result, the rank of the three nodes will be 0, 1, and 2, respectively. With the rank, an active device can decide to handle an orphaned virtual MAC address if the remainder of the hashed address dividing *N* matches the rank of the device. Furthermore, those selected addresses can be really re-dispatched to the device using the previously introduced flow migration technique.

E. The Proportional Bandwidth Allocation Algorithm

Implementing a bandwidth management mechanism over our host-based in-line cluster is much different from over a single bandwidth controller. Since a cluster contains several collaborative devices, policies on each device should be configured carefully. Suppose that a total amount of *U* Mbps bandwidth is reserved for all hypertext-transfer-protocol (HTTP) connections in a cluster of *N* bandwidth controllers, a naïve configuration may be to let each device allow at most U/N Mbps bandwidth. However, the solution may not accurate enough. Since our clustering mechanism divides clients into several subgroups, it is obvious that the bandwidth consumptions of HTTP connections from different subgroups are diverse. As a result, the reserved bandwidth *U* Mbps cannot be fully utilized if and only if one of the subgroups does not consume the U/N Mbps bandwidth. The bandwidth utilization becomes even worse if all the HTTP clients are only dispatched to certain devices in the cluster.

Here we present an effective mechanism to solve the above problem. By dynamically adjusting policies on each device, we expect the reserved bandwidth can be better utilized. The mechanism is called the proportional bandwidth allocation (PBA) algorithm. With the algorithm, all devices have to monitor the received traffic statistics for all the policies and exchange the monitored results via the inter-communication channel periodically. The upper bound of the bandwidth limitation for each policy on all devices is tuned dynamically according to the proportional traffic received by each device. Given the number of the devices *N* and the monitored received traffic, $S_1, S_2, \ldots, S_N$ for a given policy on all devices, the PBA algorithm configures the upper bound bandwidth limitation of the policy on the *i*th device as $U \times S_i/(S_1+S_2+\ldots+S_N)$.

IV. Performance Evaluation

To evaluate the performance of our scheme, we implement all the proposed mechanisms on a Linux system, which runs a kernel of version 2.4.31. The traffic dispatching scheme and the fault tolerant mechanism are implemented as kernel modules. The bandwidth management mechanism leverages those functions provided by the iproute2 package with tc, the traffic control utility. The policy of the traffic controller is configured on each device according to the PBA algorithm described previously. In this section, the experiment environment is explained first and then the performance results are presented later.

A. The Experiment Environment

The network architecture of the experiment environment is depicted in Figure 4. The cluster is comprised by four Intel IXP425 platforms. The LAN side clients and the WAN side gateways are simulated using the SmartBits, which is one of the world-recognized test equipment. The SmartBits is configured to simulate 120 different clients sending traffic to the simulated gateway. Each client is assigned a unique IP address and a unique MAC address. All the configurations on the SmartBits are setup using the SmartFlow application (version 5.0).

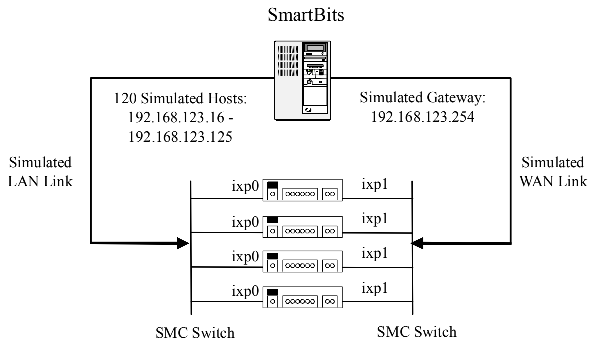

Figure 4. The network architecture for all experiments.

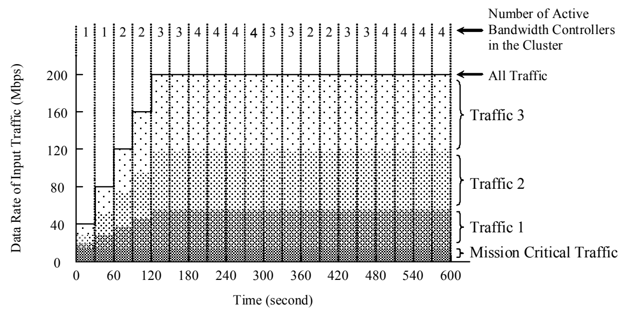

Figure 5. The scenario and the parameter settings for Experiment-II.

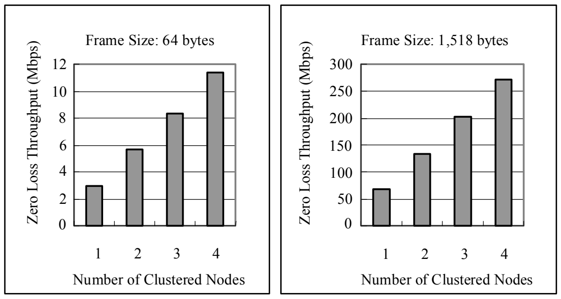

Figure 6. The zero loss throughput measured using different combinations of cluster sizes and frame sizes.

There are two experiments in our work. Experiment-I is used to verify the effectiveness of the traffic dispatching scheme. Experiment-II is used to confirm the effectiveness of the traffic redistribution scheme, the fault tolerant mechanism, and the PBA algorithm. In the former one, we configure the cluster to dispatch the received LAN-to-WAN traffic to each device in the cluster evenly and then measure the zero loss throughput on the SmartBits. The experiment is repeated with different combinations of cluster sizes (one, two, three, or four devices) and frame sizes (64 or 1,518 bytes). In the latter one, the link with a capacity of 100 Mbps is used to simulate the limited WAN link, which is only one-tenth in capacity of the simulated LAN.

The scenario and the parameter settings for Experiment-II are illustrated in Figure 5. The frame size is set as 1,024 bytes. The horizontal axis indicates the elapsed time of the experiment. We take a snapshot of measured results in a time interval of 30 seconds. The size of the cluster varies due to a node joining or a node leaving. The numbers shown on the top of every two time intervals are the size of the cluster (i.e., the number of collaborative devices at that time). The vertical axis indicates the amount of LAN-to-WAN traffic generated by the SmartBits. The bold line in the figure shows the total amount of the generated traffic. The total amount of the traffic is generated in an ascendant manner. It begins from 40 Mbps, steps 40 Mbps every time interval, and holds when it reaches 200 Mbps. The purpose of using such incremental strategy is to warm up the devices in the cluster.`

All the generated traffic is classified into four different classes, namely $\mathbf{T_1}$, $\mathbf{T_2}$, $\mathbf{T_3}$, and $\mathbf{T_C}$. The $\mathbf{T_C}$ class, which is simulated for mission critical use, is generated using a constant rate of 10Mbps. The other three classes, $\mathbf{T_1}$, $\mathbf{T_2}$, $\mathbf{T_3}$, are generated in a ratio of 2:3:4, respectively. The amounts of different traffic classes are expressed as the height of the bars in Figure 5.

In Experiment-II, the bandwidth usage for mission critical traffic is not limited. On the contrast, the expected bandwidth consumptions for the traffic classes $\mathbf{T_1}$, $\mathbf{T_2}$, and $\mathbf{T_3}$ are limited to 20 Mbps, 30 Mbps, and 40 Mbps, respectively. With the different combinations of the cluster size and the offered traffic work load, we measure the throughput of each class of traffic, monitor the CPU load on each device, and observe the influence of the change of the cluster size.

B. *Experiment Results*

Figure 6 shows the results of Experiment-I. It is obvious that the overall throughput can be scaled linearly using our traffic dispatching scheme. Take the experiment using a frame size of 1,518-byte as an example. When the size of the cluster increased from one to two, the zero loss throughput also increases accordingly from 68.26 Mbps to 134.12 Mbps. When we activate all four devices in the cluster, the zero loss throughput increases to 271.63 Mbps, which is 3.98 times compared with the zero loss throughput of a single device. The results show that first, the effectiveness of the in-line traffic dispatching scheme really scales and second, the overhead incurred by the scheme are negligible.

The results of Experiment-II are shown in Figure 7 and Figure 8. The former shows the measured average throughput and the latter illustrates the monitored CPU load on each device. During the time interval from the *0*th second to the *300*th second, because the traffic load on node 1 can be migrated to later joined nodes, the CPU load of the node 1 gradually decreases. When the four nodes are all activated, the CPU loads

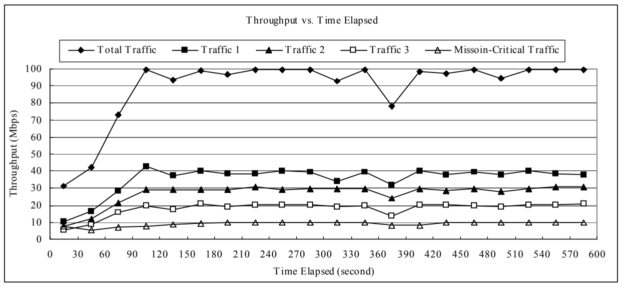

Figure 7. The throughput of each classes of traffic.

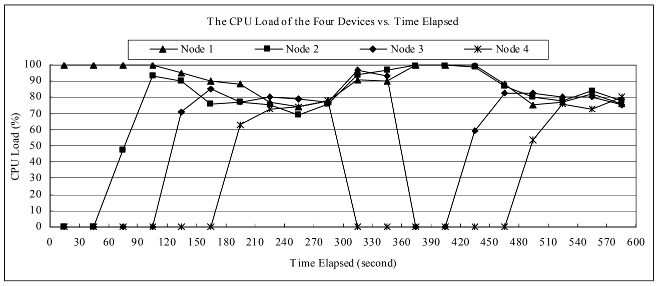

Figure 8. The CPU load information of the four clustered devices.

of these nodes are almost the same (roughly 80% CPU in use). This shows that our traffic redistribution scheme has successfully distributed workloads to all the nodes evenly. We can also find that the bandwidth usage of the four traffic classes during the traffic redistribution process is still kept stable, which means that the PBA algorithm also works properly.

The effectiveness of the fault tolerant mechanism can be observed from the *300*th second to the *600*th second. When node 3 and node 4 are inactive, the CPU loads of node 1 and node 2 increases because the original LAN-to-WAN traffic handled by the leaving nodes is gradually migrated to the two active nodes. When node 3 and node 4 later join the cluster, the workload is again evenly shared between all the nodes and hence the CPU loads of these nodes are re-balanced. It should be noted that the throughput of each traffic class decreases when a node in the cluster becomes active or inactive. Nevertheless, the decreased throughput can be recovered within the interval of 30 seconds.

V. Conclusions and Future Work

In this paper, we proposed an adaptive clustering scheme to construct a scalable bandwidth management service. We take the advantage of the transparency of in-line devices and implement BM mechanisms as an in-line device cluster so that the standalone bandwidth controllers can be nicely and easily inserted into existing LANs with minimal disruptions to the original networks.

The results show that the traffic dispatching scheme can linearly improves the throughput according to the number of nodes in the cluster. The traffic redistribution scheme keeps the load of each device balanced. In addition, it handles the node failure problem and provides more flexibility to the use of the cluster. Finally, the proposed PBA algorithm keeps the accuracy of bandwidth management. The collaboration of the above components consequently achieves a high-performance and reconfigurable bandwidth-controller cluster. The in-line clustering scheme proposed in this paper is not limited for bandwidth management. Applications that require more computation power, such as intrusion detection/prevention systems (IDS/IPS), may also benefit from our scheme. In the future, we will try to investigate possible applications and integrate them with the proposed scheme.

References

[1] Bob Braden, David Clark, and Scott Shenker, "Integrated services in the Intenet architecture: an overview," RFC 1633, June 1994.

[2] Steven Blake, David L. Black, Mark A. Carlson, Elwyn Davies, Zheng Wang, and Walter Weiss, "An architecture for differentiated services," RFC 2475, December 1998.

[3] Packeteer, Inc., "Packetteer > WAN application traffic management," http://www.packeteer.com/.

[4] AscenVision Technology, Inc., "AscenVision – the intelligent network provider," http://www.ascenvision.com/.

[5] David Kettler, Hank Kafka, and Dan Spears, "Driving fiber to the home," IEEE Communications Magazine Vol. 38, No. 11, pp. 106-110, 2000.

[6] Pan-Lung Tsai, Chun-Ying Huang, Yun-Yin Huang, Chia-Chang Hsu, and Chin-Laung Lei, "A clustering and traffic-redistribution scheme for high-performance IPSec VPNs," Proceedings of 12th IEEE International Conference on High Performance Computing (HiPC 2005), LNCS 3769, pp. 432-443, December 2005.

[7] Bill Devlin, Jim Gray, Bill Laing, and George Spix, "Scalability terminology: farms, clones, partitions, and packs: RACS and RAPS," Technical Report MS-TR-99-85, Microsoft Research, December 1999.

[8] David C. Plummer, "An Ethernet address resolution," RFC 826, November 1982.

[9] "Windows 2000 network load balancing technical overview," Microsoft Corporation, http://www.microsoft.com/technet/prodtechnol/windows 2000serv/deploy/confeat/nlbovw.mspx.

[10] Sujit Vaidya and Kenneth J. Christensen, "A single system image server cluster using duplicated MAC and IP addresses," Proceedings of 26th IEEE Conference on Local Computer Networks (LCN 2001), pp. 206-214, November 2001.

[11] Bert Hubert, Thomas Graf, Greg Maxwell, Remco van Mook, Martijn van Oosterhout, Paul B. Schroeder, Jasper Spaans, and Pedro Larroy, "Linux advanced routing & traffic control HOWTO," http://lartc.org/.

[12] Klaus Wehrle, Frank Pahlke, Hartmut Ritter, Daniel Muller, Marc Bechler, "The Linux networking architecture: design and implementation of network protocols in the Linux kernel," Pearson Prentice Hall, Inc., ISBN: 0-13-177720-3, April 2004.

[13] Sally Floyd and Van Jocobson, "Link-sharing and resource management models for packet networks," IEEE/ACM Transactions on Networking Vol. 3, No. 4, pp. 365-386 , August 1995.

[14] Kernel.Org Organization, Inc., "The Linux kernel archives," http://www.kernel. org/.

[15] Martin Devera, "HTB home," http://luxik.cdi.cz/~devik/qos/htb.

[16] Rich Seifert, The Switch Book: The Complete Guide to LAN Switching Technology, John Wiley & Sons, Inc., ISBN: 0-471-34586-5, June 2000.

[17] Smoot Carl-Mitchell and John S. Quarterman, "Using ARP to implement transparent subnet gateways," RFC 1027, October 1987.