Available online at www.sciencedirect.com
ScienceDirect
Computer Communications 29 (2006) 3170–3181
computer communications
www.elsevier.com/locate/comcom

Analysis and evaluation of a multiple gateway traffic-distribution scheme for gateway clusters

Pan-Lung Tsai *, Chin-Laung Lei

Department of Electrical Engineering, National Taiwan University, Taipei 106, Taiwan

Received 11 August 2005; received in revised form 20 April 2006; accepted 27 April 2006
Available online 24 May 2006

Abstract

Next-generation Internet gateways are expected to deal with higher volume of network traffic and also perform more sophisticated tasks besides packet forwarding. As the scale-up approach does not escape from the tradeoff between functionality and performance, architectural improvements such as clustering become necessary in the design of future Internet gateways. In this paper, we investigate different clustering architectures for high-performance, feature-rich Internet gateways and formally define the optimization problem behind these architectures as *Multiple Gateway Traffic-Distribution Problem*, both in a discrete and a continuous form. In addition to proposing various algorithms that solve the problem exactly and approximately, we also develop an on-line, self-adjusting scheme based on the solution algorithms. The numerical results of simulation suggest that the proposed approximate solution algorithms are effective and efficient, and the derived adaptive scheme is able to make the best decision on traffic distribution when dealing with the dynamic nature of network traffic in practice.

Keywords: Internet gateway; Approximation algorithm; Self-dispatching; Scalability; Cluster computing

1. Introduction

As the Internet keeps evolving, more and more network applications and services have been invented to provide a wide variety of advanced functions. In addition to deploying dedicated servers for each of these innovative applications and services, researchers and vendors have also spent their efforts on developing novel approaches to seamlessly integrate new functions with existing routing architectures. As a result, next-generation Internet routers, especially those deployed on the edges (i.e., the access routers mentioned in [1]), are expected to accomplish sophisticated tasks like URL filtering, anti-virus, anti-spam, and bandwidth control, rather than just routing.

In this paper, we use a more general term, gateways, to represent such versatile routers. Following this terminology, gateways need to implement some or all of the advanced traffic-processing functions including packet filtering, packet rewriting, packet scheduling, connection splicing, and even pattern matching within the payloads, in addition to the primitive packet forwarding, fragmentation, and reassembly. On the other hand, the advancement of networking technologies and optical components has resulted in a data rate of multiple gigabits per second or even higher in modern computers and communication networks. Consequently, though it is possible to build a standard router that performs basic packet forwarding at very high speed [2,3], the tradeoff between functionality and performance prevents current implementation of gateways from processing network packets in wire speed. Under such circumstances, a more scalable solution for the design and implementation of the gateways is highly desired.

Although hardware-based solutions (e.g., using ASICs) have been long proven in the field to be capable of delivering high throughput for well-defined operations [4], they are also infamous for their inflexibility. In particular, the revision of hardware designs in response to the invention

* Corresponding author. Tel.: +886 2 33663700x6604; fax: +886 2 23638247.
E-mail addresses: charles@fractal.ee.ntu.edu.tw (P.-L. Tsai), lei@cc.ee.ntu.edu.tw (C.-L. Lei).

0140-3664/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2006.04.015

of new protocols is both time-consuming and costly. Conversely, software-based network systems [5–8] are flexible enough but usually fail to achieve satisfactory performance for demanding applications. Since the scale-up approach [9] applying to a single network system is still governed by the tradeoff between functionality and performance, gateway clusters [10,11] that follow the scale-out approach [9] are proposed as the remedy.

In this paper, we investigate different clustering architectures and formally define the problem of traffic distribution over clustered gateways. Following the formal definition of the optimization problem, we propose a number of algorithms for solving the problem exactly and approximately. We also develop an on-line, self-adjusting scheme based on the solution algorithms. The rest of the paper is organized as follows. Section 2 reviews the two broad categories of clustering architectures as well as the models that help to construct the traffic-distribution scheme for gateway clusters. Section 3 formally defines the traffic-distribution problem and presents the solution algorithms, followed by the simulation results of the algorithms in Section 4. Section 5 describes the proposed adaptive scheme based on the solution algorithms, and Section 6 concludes the paper by summarizing our achievements.

2. Related work

Clustering can be viewed as an architectural improvement to the design of network systems. A cluster is inherently scalable under the condition that the input load is elegantly distributed over the clustered units. In this section, we first examine the two broad categories of clustering architectures, one with a dedicated traffic dispatcher and the other without any (as shown in Fig. 1), as well as their corresponding traffic-dispatching techniques. Then we outline the construction of the traffic-distribution scheme and also discuss the literatures relevant to our modeling.

2.1. Gateway clusters with dedicated traffic dispatchers

A straightforward approach to distribute processing load over multiple gateways within a cluster is to deploy a dedicated traffic dispatcher in front of the gateways so that the dispatcher may redirect individual network packets to different gateways. The decision of redirection can be made according to simple rules such as statically round robin, or it can be as flexible as evaluating complicated criteria like selecting the gateway with least load over the past 10 s. As long as the traffic dispatcher is carefully designed, it is possible to keep the load of the clustered gateways balanced over time and thus to maximize the throughput of the entire cluster.

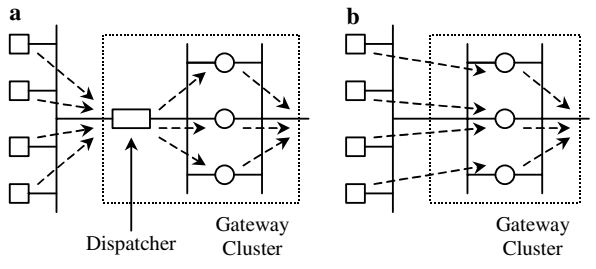

Fig. 1. Two categories of gateway clusters: (a) a gateway cluster with a dedicated traffic dispatcher, and (b) a fully decentralized gateway cluster. The squares represent hosts that generates input load and the circles represent individual gateways. The arrows with dashed lines depict the path of network traffic.

There are several options for the traffic-dispatching mechanisms adopted in this architecture. The dispatcher may make use of standard routing, network address translation (with network-layer headers being altered) [12], tunneling (e.g., GRE [13,14] and IP-within-IP [15]), etc., or hybrid mechanisms [16,17], depending on the performance requirements, granularity of traffic dispatching, and the complexity of the application running on the gateways behind the dispatcher.

The primary advantages of using a dedicated dispatcher are its simplicity and flexibility. Since the issue of load distribution is isolated and can be taken care of solely by the dispatcher, the gateways in the cluster may focus on application-specific processing and are virtually independent from one another and from the dispatcher. In addition, the dispatcher itself may serve as the representative entity of the entire gateway cluster to other hosts on the network and hence help the cluster maintain the property of transparency without additional mechanisms. The features of simplicity and flexibility together make such dispatcher-based clustering architecture applicable to many classes of network applications and services, ranging from layer 3 to layer 7 in OSI models, and inspire network-equipment manufacturers to create products such as server load balancers [18–20].

2.2. Gateway clusters with self-dispatching mechanisms

Despite the advantages mentioned above, the major drawback of the dispatcher-based clustering architecture is obvious. The centralized design adopted makes the dispatcher the potential performance bottleneck and a single point of failure. In contrast to the dispatcher-based clustering architecture, the second architecture described in this section completely eliminates the need for traffic dispatchers by taking advantage of various self-dispatching techniques, and can be fully decentralized.

To accomplish the task of traffic dispatching and maintain the transparency at the same time, gateway clusters adopting this second architecture usually perform special processing on ARP [21] requests (e.g., proxy ARP [22,23]) and frame filtering. For example, the clustered units in [24] are configured to answer ARP requests either with a nonexistent Ethernet address or with a layer-2 multicast address so that the switch residing between the cluster and other hosts will always flood the frames sent to the cluster. In [25], the clustered units are simply configured with the same layer-2 address to ensure that ARP replies will contain the correct answer. The gateway cluster

proposed in [11] first generates a number of virtual MAC addresses and uses them as the answers when replying ARP requests. These solutions are transparent in that all of them successfully create an undistinguishable illusion of a single node (i.e., a node with a single network-layer address) for the multiple units in a cluster. Refs. [10,26,27] adopt a dissimilar approach, in which no particular measures are taken to hide the real identities of the clustered units from hosts, and network packets first arrive at some clustered unit unaffectedly and then may be rerouted to other clustered units for further processing.

Among these solutions, Refs. [11,25] explicitly incorporate adaptive load-distribution mechanisms, while Ref. [24] only makes use of static load-balancing algorithm, which must be configured manually and does not respond to the variation of input load and the status change of the clustered nodes. On the other hand, each clustered unit in [10,26,27] does not attempt to balance the load by influencing the way in which the incoming traffic arrives, but instead transfers the excess load to other units afterwards, and hence may cause network packets to make a detour before they get processed. Refs. [26,27] also only focus on the processing of Web traffic, while Ref. [24] recognizes more protocols including TCP, UDP, GRE, whereas Ref. [11] is applicable to virtually all applications and services at or above the network layer.

Take the clustering architecture adopted in [11] as an example. Fig. 2 depicts more details of the corresponding traffic-distribution mechanism. In the phase of initial setup, the hosts that generate input traffic to the gateway cluster are first partitioned into a number of logical host groups. The partitioning is controlled by a predefined mapping function, which takes the MAC addresses of the hosts as the input. An individual virtual gateway, which has its own virtual MAC address, is then assigned to each host group. Finally, the distribution of input traffic is accomplished by assigning the virtual gateways (or, equivalently, the flows generated by host groups) to the physical gateways in the cluster. As the assignment of the virtual gateways to the physical gateways is going to be recomputed periodically based on the measured volume of input traffic, the distribution of input traffic also changes dynamically in response to the variation of traffic volume and the resulting load differences of physical gateways. Section 3 formally defines the problem of determining the optimal assignments of the virtual gateways (i.e., the flows) to the physical gateways and also discusses the corresponding solution algorithms.

2.3. Modeling traffic distribution for gateway clusters

In order to increase the overall system utilization and hence scale the throughput of gateway clusters, we must find a way to optimally distribute the input traffic among the individual gateways in gateway clusters. The approach we take to constructing the traffic-distribution scheme for gateway clusters is outlined as follows.

We first put our focus on how to optimally distribute the input traffic with respect to a finite period of time. Suppose that the traffic sent to a gateway cluster for further processing in certain time interval can be divided into a number of individual flows with known sizes. Then we just come up with a variant of the deterministic task-scheduling problem [28,29] with respect to the given time interval. Since the sizes of individual flows are known, this problem belongs to the category of global static scheduling according to the widely accepted taxonomy in [30]. As a result, optimally distributing the input traffic among multiple gateways can be treated as solving a series of the varied deterministic task-scheduling problem instances.

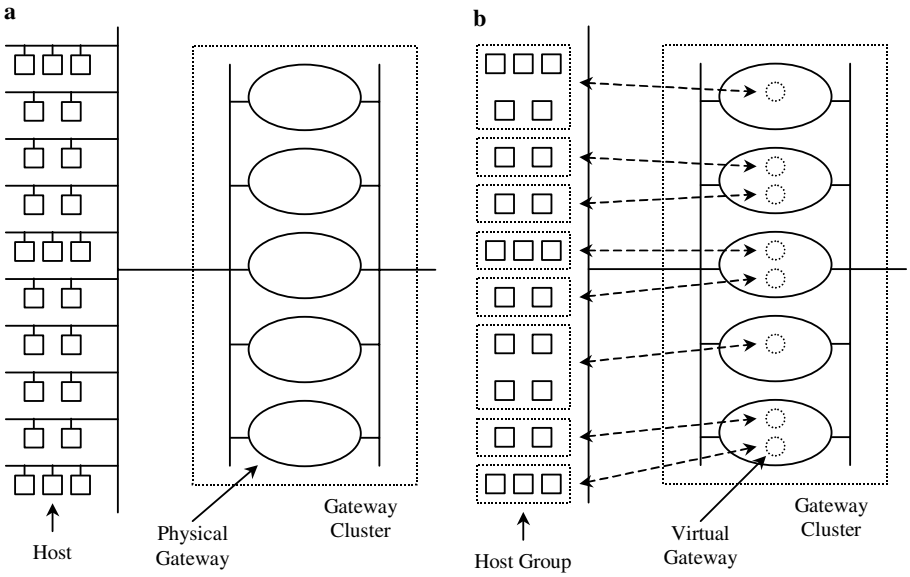

Fig. 2. The clustering architecture and the traffic-distribution mechanism of interest: (a) the network topology, and (b) the partitioning of the hosts into logical host groups (with some of the physical connections to the hosts being omitted), the one-to-one correspondence between the host groups and the virtual gateways, and the assignment of the virtual gateways to the physical gateways.

Taking a closer examination, we realize that we are not concerned with the processing order of individual flows (i.e., the schedule) but only care about the allocation of the processing power offered by the gateways. Therefore, we further abstract the traffic-distribution problem with respect to certain time interval into a form that is closely related to the multiple knapsack problem [31,32], which [33] also uses to model the resource allocation in a telecommunication network. Section 3 provides the formal definition of the problem and presents a number of corresponding solution algorithms.

The final step to constructing the traffic-distribution scheme is to deal with the dynamic nature of the input traffic. Based on the problem formulation as well as the solution algorithms we propose, we develop a round-based, self-adjusting traffic-distribution scheme that can be applied to gateway clusters and works in an on-line fashion. Further details are described in Section 5.

3. Traffic-distribution problem

Suppose that the input traffic consists of n individual flows and all these n flows are going to be processed by the m gateways in a cluster. Let $Load(f,t)$ denote the number of packets belonging to flow f with respect to time interval $[t-\Delta t, t)$. The goal of the traffic-distribution problem is to optimally assign the n flows, which are denoted by $\mathbf{F}_j$, $j=1,2,3,\ldots,n$, to the m gateways, which are denoted by $\mathbf{G}_i$, $i=1,2,3,\ldots,m$, so that the total number of packets processed by the gateway cluster is maximized. Section 3.1 gives the formal definition of the problem.

3.1. Traffic distribution over multiple gateways

If we use $Cap(g)$ to represent the processing capability of gateway g in packets per second, then the total number of packets that individual gateways are able to process in time interval $[t-\Delta t, t)$ can be denoted by $Cap(\mathbf{G}_i)\times\Delta t$, $i=1,2,3,\ldots,m$. By setting weights $w_j = Load(\mathbf{F}_j, t)$, $j=1,2,3,\ldots,n$, and capacities $c_i = Cap(\mathbf{G}_i)\times\Delta t$, $i=1,2,3,\ldots,m$, we can define *Multiple Gateway Traffic-Distribution Problem Version 1* (**MGTDPv1**) with respect to time interval $[t-\Delta t, t)$ as the following linear integer formulation.

Multiple Gateway Traffic-Distribution Problem Version 1 (**MGTDPv1**). Given a set of n flows, each of which contains w_j packets with respect to time interval $[t-\Delta t, t)$, $j=1,2,3,\ldots,n$, and a set of m gateways, each of which is capable of processing c_i packets in the same time interval, $i=1,2,3,\ldots,m$, the objective is to

$$\text{maximize} \quad \sum_{i=1}^{m} Min\left(\sum_{j=1}^{n} w_j x_{ij}, c_i\right)$$

$$\text{subject to} \quad \sum_{i=1}^{m} x_{ij} \leqslant 1, \quad j=1,2,3,\ldots,n, \tag{1}$$

$$x_{ij} \in \{0,1\}, \quad i=1,2,3,\ldots,m, \quad j=1,2,3,\ldots,n, \tag{2}$$

where $Min(x,y)$ denotes the smaller one of x and y.

Without loss of generality, it is assumed that $n \geqslant m$, $w_j > 0, j=1,2,3,\ldots,n, c_i > 0, i=1,2,3,\ldots,m, w_{\min} \leqslant c_{\min}$, $w_{\max} \leqslant c_{\max}$, and $\sum_{j=1}^{n} w_j > c_{\max}$, where $w_{\min}$ and $w_{\max}$ denote the minimum and the maximum of w_j's, $j=1,2,3,\ldots,n$, respectively, and $c_{\min}$ and $c_{\max}$ denote the minimum and the maximum of c_i's, $i=1,2,3,\ldots,m$, respectively. These assumptions are valid throughout the text.

Each binary variable x_{ij}, $i=1,2,3,\ldots,m$, $j=1,2,3,\ldots,n$, denotes whether flow $\mathbf{F}_j$ is assigned to gateway $\mathbf{G}_i$ (represented by $x_{ij}=1$) or not (represented by $x_{ij}=0$), and constraint (1) prevents a single flow from being assigned to more than one gateway. Note that when $\mathbf{F}_j$ is assigned to $\mathbf{G}_i$ (i.e., $x_{ij}=1$), the packets belonging to flow $\mathbf{F}_j$ are directed to $\mathbf{G}_i$ and consume some processing power of $\mathbf{G}_i$. By definition, $\mathbf{G}_i$ cannot process more than c_i packets in time interval $[t-\Delta t, t)$ even if more packets are directed to it (in this case, the excess packets are simply discarded), and this is the reason why $Min(x,y)$ is involved in the evaluation of the objective function.

The problem of traffic distribution over multiple gateways can also be written in a continuous form, resulting in *Multiple Gateway Traffic-Distribution Problem Version 2* (**MGTDPv2**). This second version eliminates the use of $Min(x,y)$ in the calculation of the objective values and is defined as follows.

Multiple Gateway Traffic-Distribution Problem Version 2 (**MGTDPv2**).[1] Given a set of n flows, each of which contains w_j packets with respect to time interval $[t-\Delta t, t)$, $j=1,2,3,\ldots,n$, and a set of m gateways, each of which is capable of processing c_i packets in the same time interval, $i=1,2,3,\ldots,m$, the objective is to

$$\text{maximize} \quad \sum_{i=1}^{m}\sum_{j=1}^{n} w_j x_{ij}$$

$$\text{subject to} \quad \sum_{j=1}^{n} w_j x_{ij} \leqslant c_i, \quad i=1,2,3,\ldots,m, \tag{3}$$

$$|\mathbf{X}_j| \leqslant 1, \quad j=1,2,3,\ldots,n, \tag{4}$$

$$0 \leqslant x_{ij} \leqslant 1, \quad i=1,2,3,\ldots,m, \quad j=1,2,3,\ldots,n, \tag{5}$$

where $\mathbf{X}_j = \{x_{ij} | x_{ij} > 0,\ i=1,2,3,\ldots,m\}$.

The interpretation of each variable x_{ij}, $i=1,2,3,\ldots,m$, $j=1,2,3,\ldots,n$, has changed in **MGTDPv2**. When flow $\mathbf{F}_j$ is assigned to gateway $\mathbf{G}_i$

[1] Note that **MGTDPv2** is very much similar to the linear programming relaxation of a well-studied problem, *Multiple Subset Sum Problem* (**MSSP**) [31,32,34]. In fact, constraint (4) is the only difference between **MGTDPv2** and the linear programming relaxation of **MSSP**. The linear programming relaxation of **MSSP** is often denoted by **C**(**MSSP**), and **C**(**MSSP**) is known to be solvable in linear time. Despite the high similarity of **MGTDPv2** and **C**(**MSSP**), no polynomial-time algorithms that generate exact solutions to **MGTDPv2** have been reported yet.

(now represented by $x_{ij} > 0$), it is now permissible for $\mathbf{G}_i$ to process only a fraction of all packets belonging to flow $\mathbf{F}_j$, as stated in constraint (5). On the other hand, constraint (4) still prevents a single flow from being assigned to more than one gateway. The newly introduced constraint (3) effectively limits the total number of packets to be processed by each gateway and hence helps to simplify the objective function.

At the first glance, the original problem of **MGTDPv1** seems to be transformed into an easier one by relaxing constraint (2) to constraint (5). However, the existence of constraint (3) in **MGTDPv2** actually comes to balance. To prove that the two problems share the same optimal solution values, we first introduce the following lemmas.

Lemma 1[2]. *For every feasible solution of* **MGTDPv1**, *there exists a feasible solution of* **MGTDPv2** *such that the solution value of the latter is equal to that of the former.*

Lemma 2[3]. *For every feasible solution of* **MGTDPv2**, *there exists a feasible solution of* **MGTDPv1** *such that the solution value of the latter is greater than or equal to that of the former.*

Let $\bar{z}^{(1)}$ and $\bar{z}^{(2)}$ denote the optimal solution values of **MGTDPv1** and **MGTDPv2**, respectively. The following theorem asserts the equality of optimal solution values.

Theorem 1. **MGTDPv1** *and* **MGTDPv2** *share the same optimal solution values. That is,* $\bar{z}^{(1)} = \bar{z}^{(2)}$.

Proof. Since the optimal solution of **MGTDPv1** is also a feasible solution of the same problem, according to Lemma 1, there exists a feasible solution of **MGTDPv2** such that the corresponding solution value, which is denoted by $z^{(2)}$, is equal to $\bar{z}^{(1)}$. By definition, $z^{(2)} \leqslant \bar{z}^{(2)}$. Therefore, we have $\bar{z}^{(1)} = z^{(2)} \leqslant \bar{z}^{(2)}$.

Similarly, since the optimal solution of **MGTDPv2** is also a feasible solution of the same problem, according to Lemma 2, there exists a feasible solution of **MGTDPv1** such that the corresponding solution value, which is denoted by $z^{(1)}$, is greater than or equal to $\bar{z}^{(2)}$. By definition, $z^{(1)} \leqslant \bar{z}^{(1)}$. Therefore, we have $\bar{z}^{(2)} \leqslant z^{(1)} \leqslant \bar{z}^{(1)}$.

As $\bar{z}^{(1)} \leqslant \bar{z}^{(2)}$ and $\bar{z}^{(2)} \leqslant \bar{z}^{(1)}$ both hold, we conclude that $\bar{z}^{(1)} = \bar{z}^{(1)}$ and hence prove the theorem. □

Two key findings regarding **MGTDPv2** are stated by the following theorem and corollary. They provide some hints about what the optimal solutions of **MGTDPv2** look like and may help to find better algorithms that solve the problem.

Theorem 2[4]. *In an optimal solution of* **MGTDPv2**, *if there exist one or more unassigned flows, all gateways must be fully loaded.*

Corollary 1. *In an optimal solution of* **MGTDPv2**, *if there exist some unassigned flows, the optimal solution value must be maximal* (*i.e.*, $\sum_{i=1}^{m} c_i$).

3.2. Solutions to the traffic-distribution problems

The *brute-force* algorithm can be used to solve the traffic-distribution problems by examining the solution values of all possible assignments of each flow to every gateway. However, the running time of such algorithm is $O((m+1)^n)$, which is exponential and hardly acceptable even with small values of m and n. Fortunately, we may improve the running time with *branch-and-bound* techniques. The improved algorithm examines one flow after another, and the branching part involves generating all possibilities in each round by assigning the flow of interest to every gateway. The bounding part eliminates some of these possibilities that are determined to be incapable of producing better solution values than the best solution value ever produced. We simply take the smaller of the sum of the remaining capacities and the sum of the numbers of packets associated with all unassigned flows in each round as the filtering criterion (i.e., the upper bound) used in the bounding part, and evaluate the surviving possibilities by the *depth-first search*. Note that theoretically the branch-and-bound techniques do not change the worst-case running time of the algorithm, but the time required for solving general cases encountered in real-world applications can be effectively reduced to an acceptable level.

Though an algorithm that solves all instances of the traffic-distribution problems in polynomial running time is not known yet, we have developed three different polynomial-time approximation algorithms that produce reasonably good solutions for real-world applications. The description of the algorithms follows the notations used to describe **MGTDPv1** and the solution produced is denoted by the disjoint subsets $\mathbf{W}_i$'s, $i = 0, 1, 2, \ldots, m$, of set $\mathbf{W} = \{w_j | j = 1, 2, 3, \ldots, n\}$,[5] where $\mathbf{W}_0^{(1)}$ comprises the numbers of the packets which are not going to be processed by any gateway, each of the remaining $\mathbf{W}_i^{(1)}$, $i = 1, 2, 3, \ldots, m$, comprises the numbers of the packets that are going to be processed by gateway $\mathbf{G}_i$, and $\cup_{i=0}^{m} \mathbf{W}_i^{(1)} = \mathbf{W}$. Before each of the following algorithms begins, an extra step is taken to ensure that the elements in $\mathbf{W}$ are sorted in a descending order (i.e., $w_1 \geqslant w_2 \geqslant w_3 \geqslant \cdots \geqslant w_n$).

Algorithm 1: Sequential Assignment

1. **for** i **from** 0 **to** m **do**
2. $\quad \mathbf{W}_i \leftarrow \phi$
3. $j \leftarrow 1$, $i \leftarrow 1$, $w \leftarrow 0$

[2] For the complete proof of Lemma 1, please refer to Appendix A.
[3] For the complete proof of Lemma 2, please refer to Appendix A.
[4] For the complete proof of Theorem 2, please refer to Appendix A.

[5] Throughout this paper, all sets are treated as *multisets*. That is, the multiplicity of elements in each set is respected.

```
4. while i ⩽ m do
5.    while w < c_i do
6.       w ← w + w_j, W_i ← W_i ∪ w_j, j ← j + 1
7.       if j > n then
8.          return W_i's, i = 0,1,2,...,m, as the solution
9.    i ← i + 1, w ← 0
10. while j ⩽ n do
11.    W_0 ← W_0 ∪ w_j, j ← j + 1
12. return W_i's, i = 0,1,2,...,m, as the solution
```

Algorithm 2: Greedy Assignment

```
1. for i from 0 to m do
2.    W_i ← φ, v_i ← c_i
3. for j from 1 to n do
4.    k ← 1
5.    for i from 2 to m do
6.       if v_i > v_k then
7.          k ← i
8.    if v_k ⩽ 0 then
9.       while j ⩽ n do
10.         W_0 ← W_0 ∪ w_j, j ← j + 1
11.      return W_i's, i = 0,1,2,...,m, as the solution
12.   else
13.      W_k ← W_k ∪ w_j, v_k ← v_k − w_j
14. return W_i's, i = 0,1,2,...,m, as the solution
```

Algorithm 3: Best-Fit Assignment

```
1. for i from 0 to m do
2.    W_i ← φ, v_i ← c_i
3. for j from 1 to n do
4.    k ← 0
5.    for i from 1 to m do
6.       if v_i > w_j then
7.          if k = 0 then
8.             k ← i
9.          else
10.            if v_i < v_k then
11.               k ← i
12.   if k = 0 then
13.      k ← 1
14.      for i from 2 to n do
15.         if v_i > v_k then
16.            k ← i
17.   if v_k ⩽ 0 then
18.      while j ⩽ n do
19.         W_0 ← W_0 ∪ w_j, j ← j + 1
20.      return W_i's, i = 0, 1, 2, ..., m, as the solution
21.   else
22.      W_k ← W_k ∪ w_j, v_k ← v_k − w_j
23. return W_i's, i = 0,1,2,...,m, as the solution
```

In Algorithm 1, all w_j's, $j = 1,2,3,\ldots,n$, are processed sequentially, resulting in a running time of $O(m+n)$. Algorithm 2 follows the rule of always assigning the flow with the largest associated number of packets to the gateway with the most remaining capacity, and the running time is $O(n \cdot m)$. Algorithm 3 adopts the *best-fit* strategy and also runs in $O(n \cdot m)$ time. As a matter of fact, the running time of Algorithm 2 and Algorithm 3 can be further reduced to $O(n \cdot lg(m))$ by making use of appropriate data structures such as priority queues.

Note that these relatively fast algorithms are just approximation algorithms and they do not guarantee to derive the optimal solution values for all problem instances. For example, given a problem instance of $m = 3$, $n = 8$, and $\mathbf{W} = \{5, 5, 4, 4, 3, 3, 3, 3\}$, Algorithm 1 outputs $\mathbf{W}_0 = \phi$, $\mathbf{W}_1 = \{5, 5\}$, $\mathbf{W}_2 = \{4, 4, 3\}$, $\mathbf{W}_3 = \{3, 3, 3\}$, resulting in a solution value of 29, Algorithm 2 outputs $\mathbf{W}_0 = \phi$, $\mathbf{W}_1 = \{5, 3, 3\}$, $\mathbf{W}_2 = \{5, 3, 3\}$, $\mathbf{W}_3 = \{4, 4\}$, resulting in a solution value of 28, and Algorithm 3 also gives the same output as Algorithm 1 does, whereas the optimal solution value is actually 30, with one possible solution being $\mathbf{W}_0 = \phi$, $\mathbf{W}_1 = \{5, 5\}$, $\mathbf{W}_2 = \{4, 3, 3\}$, $\mathbf{W}_3 = \{4, 3, 3\}$, which can be derived by the aforementioned branch-and-bound algorithm.

4. Evaluation of traffic-distribution algorithms

Fig. 3 shows three different scenarios in which the gateways of interest may become the bottlenecks and hence limit the efficiency of the communication networks.

The topology shown in Fig. 3(a) usually appears in campus networks, and Fig. 3(b) depicts the typical settings for networks that connect to the Internet. In Fig. 3(c), there exists a mutual agreement between the gateways, and the direct network traffic between the local area networks receives special treatments when passing through the gateways. A commonly referred example of the topology in Fig. 3(c) is the deployment of virtual private networks.

In the first scenario, the gateway has to process the aggregated traffic from multiple local area networks and may become the performance bottleneck because of the large volume of network traffic. In the second case, the owner of the local area network may want to enforce access-control policies on the edge of the network. As a result, the gateway often incorporates various advanced packet-processing functions in addition to basic packet forwarding and hence needs much more computation power than ordinary edge routers do. The gateways in Fig. 3(c) are usually required to carry out application-specific computation-intensive operations such as encryption, decryption, and authentication. All of these scenarios may benefit from the clustering solutions. In this section, we evaluate the performance of the proposed traffic-distribution algorithms following the scenario shown in Fig. 1(a). A real-world application that corresponds to the scenarios shown in Fig. 1(c) is described in [11].

Fig. 4 depicts a clustering solution of the gateway in Fig. 3(a). Suppose that traffic generated by the hosts consists of n individual flows and there are m gateways in the cluster. In this section, we will examine the effectiveness

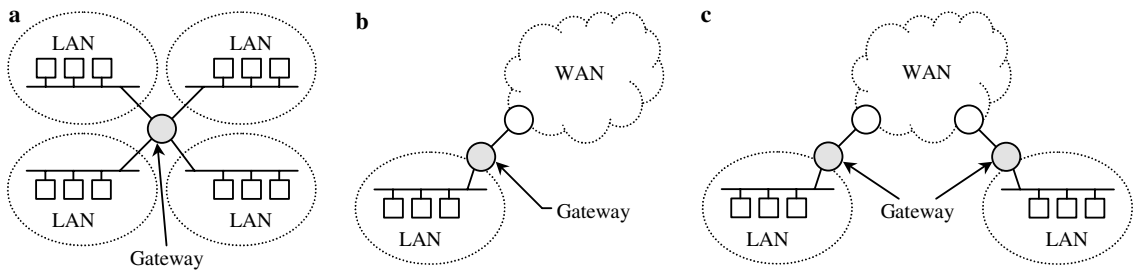

Fig. 3. Three different scenarios in which the gateways are likely to become performance bottlenecks: (a) multiple local area networks sharing a common gateway, (b) a local area network connected to a wide area network via an edge gateway, and (c) two local area networks communicating with each other through special-purpose gateways.

and the efficiency of the algorithms described in Section 3.2 with respect to different values of m and n.

In our simulation, the gateway units are assumed to be identical and each of them is able to process 10 packets in the given time interval. Without loss of generality, we also assume that the numbers of packets belonging to individual flows are sorted in a descending order. In other words, the capacities $c_1 = c_2 = c_3 = \cdots = c_m = 10$ and the weights $w_1 \geqslant w_2 \geqslant w_3 \geqslant \cdots \geqslant w_n$. Recall other assumptions made previously in Section 3.1. Putting everything together, now we have $n \geqslant m$, $1 \leqslant w_j \leqslant 10$, $j = 1, 2, 3, \ldots, n$, $\sum_{j=1}^{n} w_j > 10$, and $w_1 \geqslant w_2 \geqslant w_3 \geqslant \cdots \geqslant w_n$.

The simulation program first generates every possible problem instance of **MGTDPv1** by enumerating all possibilities of w_j's, $j = 1, 2, 3, \ldots, n$, satisfying the constraints mentioned above, with respect to the given value of n, and computes the optimal solution value of every problem instance with respect to the given values of m and n by executing the branch-and-bound algorithm. Then it continues to run the algorithms proposed in Section 3.2 and computes the respective differences between the approximate solution values and the optimal solution values. Tables 1 and 2 show the results of the simulation.

In Table 1, the first two columns contain the values of m and n, respectively, and the third column shows the total number of problem instances with respect to the given values of m and n. Each data cell in the remaining columns of Table 1 contains two numbers. The first one is the number of problem instances with respect to which an algorithm fails to derive the optimal solution values, and the second one (surrounded by a pair parentheses) is the maximum of the differences between the respective optimal solution values and the respective solution values derived by the algorithm.

Taking the case where $m = 4$ and $n = 16$ as an example, there are 2,042,975 possibilities of w_j's, $j = 1, 2, 3, \ldots, n$, resulting in 2,042,975 distinct problem instances in total. Among these problem instances, Algorithm 1 (i.e., the algorithm of sequential assignment) fails to derive the optimal solution values in 14,010 cases, Algorithm 2 (i.e., the algorithm of greedy assignment) fails in 46 cases, Algorithm 3 (i.e., the algorithm of best-fit assignment) fails in 18 cases, and the algorithm of random assignment, which is expected to perform the worst, fails in 958,796 cases. Among these failed cases, the solution values computed by Algorithm 1 can be as bad as 10 less than the optimal solution values for some problem instances (comparing to the full capacity of $4 \times 10 = 40$, a deviation of 10 is relatively large), while the solution values computed by Algorithm 2 and Algorithm 3 are very close to the respective optimal solution values in all problem instances, with the deviations no more than 2 and 1, respectively. On the other hand, the solution values computed by the algorithm of random assignment can be arbitrarily bad and may change from one test to another. The measured maximum deviation in this particular test is 27, which is (not surprisingly) the largest among the four algorithms.

In Table 2, the first two columns also contain the values of m and n, respectively, and the third column shows the number of different assignments to be evaluated when solving a single problem instance using the brute-force algorithm. Each data cell in the fourth column of Table 2 again contains two numbers. This time the first one is the maximum among the numbers of different assignments tried by the branch-and-bound algorithm when computing the optimal solution values for individual problem instances, and the second one is the average of the same numbers over all problem instances with respect to the given values of m and n.

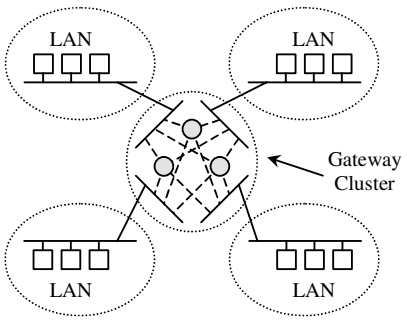

Fig. 4. Improving the throughput of the network in Fig. 3(a) by replacing the gateway with a cluster of collaborating gateway units.

Table 1
Effectiveness of the algorithms

m	n	Number of problem instances	Approximate solution: number of suboptimal assignments (maximum deviation)			
			Sequential assignment	Greedy assignment	Best-fit assignment	Random assignment
2	2	30	20 (8)	0 (0)	0 (0)	16 (10)
	4	688	308 (8)	0 (0)	0 (0)	446 (10)
	8	24,306	238 (4)	1 (1)	1 (1)	4,802 (10)
	16	2,042,975	0 (0)	0 (0)	0 (0)	12,761 (10)
4	4	688	559 (16)	0 (0)	0 (0)	411 (20)
	8	24,306	16,201 (16)	242 (2)	18 (1)	22,833 (29)
	16	2,042,975	14,010 (10)	46 (2)	18 (1)	958,796 (27)
8	8	24,306	23,465 (32)	0 (0)	0 (0)	20,664 (43)

Table 2
Efficiency of the algorithms

m	n	Exact solution		Approximate solution
		Number of different assignments evaluated by brute force	Number of different assignments evaluated by branch-and-bound: maximum (average)	Number of different assignments evaluated by each of the proposed algorithms and the random one
2	2	9	2 (1.67)	1
	4	81	8 (1.74)	1
	8	6,561	9 (1.01)	1
	16	43,046,721	1 (1.00)	1
4	4	625	10 (3.40)	1
	8	390,625	8,525 (35.12)	1
	16	152,587,890,625	7,783,779 (6.49)	1
8	8	43,046,721	78 (11.35)	1

For example, when $m = 4$ and $n = 16$, the brute-force algorithm needs to scan over all $(4+1)^{16} = 152{,}587{,}890{,}625$ possible assignments only to determine the optimal solution value for a single problem instance. On the contrary, the branch-and-bound algorithm is able to derive the optimal solution value by evaluating no more than 7,783,779 different assignments, and the average number of assignments evaluated by the branch-and-bound algorithm for a single problem instance is 6.49. (However, it generally takes longer to generate a worth-evaluating assignment in the branch-and-bound algorithm, so the worst-case speed-up is actually not as high as $152{,}587{,}890{,}625/7{,}783{,}779 \approx 19{,}603$.) Our simulation also shows that it takes only 64.09 s for the branch-and-bound algorithm to solve all of the 2,042,975 distinct problem instances on a machine with an Intel Pentium 4 2.4 GHz processor. That is, a single problem instance can be solved in 31.37 μs on average in this particular case.

The last column of Table 2 simply points out the fact that each of the algorithms of sequential assignment, greedy assignment, best-fit assignment, and random assignment attempts to generate and evaluate only one possible assignment. Although the time to generate the assignment may vary slightly across the four algorithms, they are much more efficient than the branch-and-bound algorithm and the brute-force algorithm are.

Tables 1 and 2 together demonstrate the effectiveness and the efficiency of the algorithms. As indicated by the results of simulation, the branch-and-bound algorithm is capable of reducing the time required to exactly solve given problem instances to a great extent. In addition, though Algorithm 2 and Algorithm 3 may fail to derive optimal solution values in a few cases, they run much faster than the branch-and-bound algorithm and the deviations are quite small. The algorithms of sequential assignment and random assignment run even faster and are also easy to implement, but the quality of the derived solution values are relatively poor.

5. Adaptive traffic-distribution scheme

With the help of the problem formulation, the analysis, and the solution algorithms presented previously, we are ready to construct an on-line, self-adjusting traffic-distribution scheme that is able to deal with the dynamic nature of the input traffic.

As mentioned in Section 2.3, the workings of the traffic-distribution scheme can be viewed as solving a series of **MGTDP** problem instances. However, as the input traffic changes over time and the flow sizes are generally not known in advanced, we must first figure out the input parameters for each problem instance by means of certain

predictive techniques. Let vector $\mathbf{U}_k = (u_{k,1}, u_{k,2}, u_{k,3}, \ldots, u_{k,n})$ represent the flow sizes observed by the gateway cluster of interest in kth time interval $[(k-1)\cdot\Delta t, k\cdot\Delta t)$, and let vector $\mathbf{V}_{k+1} = (v_{k+1,1}, v_{k+1,\ 2}, v_{k+1,3}, \ldots, v_{k+1,n})$ denote the derived values to be used as an estimation of the flow sizes with respect to next time interval $[k\cdot\Delta t, (k+1)\cdot\Delta t)$, where $k = 1, 2, 3, \ldots, \infty$. We adopt the method of exponentially weighted moving average to compute the input parameters for next problem instance based on the latest observation and the historical data of flow sizes.

Initially we randomly assign the n flows to the m individual gateways. At the end of the first time interval, we set $\mathbf{V}_2 = \mathbf{U}_1$. For $k > 1$, we derive $\mathbf{V}_{k+1}$ at the end of kth time interval as follows.

$$v_{k+1,j} = \alpha \cdot u_{k,j} + (1-\alpha)\cdot v_{k,j}, \quad j = 1, 2, 3, \ldots, n,$$

where coefficient α is adjustable within range $[0, 1]$ and controls the influence of the recently observed flow sizes on the calculation of $\mathbf{V}_{k+1}$. Then we take the elements of $\mathbf{V}_{k+1}$ along with other input parameters to run one of the solution algorithms presented in Section 3.2, and reassign the flows accordingly.

The implementation of the adaptive traffic-distribution scheme mentioned above is straightforward in clustering architectures with dedicated dispatchers. In this case, the scheme simply runs on the dedicated dispatchers since the dispatchers themselves have all necessary information. In contrast, implementing the traffic-distribution scheme in clustering architectures without any dispatchers involves gathering the observed flow sizes before the derivation of the input parameters for next round and getting all gateways informed of the new flow assignment afterwards. An intuitive idea is to delegate such responsibility of information gathering, decision making, and result propagation to one of the gateways (e.g., the gateway with the smallest identification number). Alternatively, each gateway may also collect the information from one another and then derive the result by itself.

Since the worst-case running time of the brute-force solution algorithm is exponentially bounded, the approximate solution algorithms proposed in Section 3.2 are preferable in the cases of a larger number of flows (for finer granularity) or more than a few gateways (for higher limit of throughput). In fact, each of the approximate solution algorithms is able to make the flow-assignment decision very quickly, so it becomes both viable and beneficial for the decision makers (i.e., the dispatchers, the delegate gateways, or all gateways, depending on the clustering architectures and the implementation) to run more than one available approximate solution algorithms (either independently or collaboratively) and then select the best one from the results. Table 3 shows the benefit of running multiple approximate solution algorithms within a single round. The numbers are derived from the results of the simulation described in the previous section.

6. Conclusions

In this paper, we investigate different clustering architectures and dispatching techniques for scaling the throughput of gateways, and formally define the problem of *Multiple Gateway Traffic-Distribution Problem* both in a discrete form (**MGTDPv1**) and in a continuous form (**MGTDPv2**). We also prove that the two versions of the problem share the same optimal solution values. Since no polynomial-time algorithms that generate exact solutions to **MGTDPv1** or **MGTDPv2** have been reported yet, we propose three different approximation algorithms, one of which runs in $O(m+n)$ time and the other two run in $O(n\cdot m)$ time, where m is the number of gateways in the cluster and n is the number of flows, to solve the problem efficiently.

The numerical results of simulation show that the branch-and-bound algorithm with a properly selected upper-bound function and the depth-first strategy may greatly reduce the time required to derive the optimal solution values so that it becomes feasible to adopt the exact

Table 3
Benefit of using multiple approximate solution algorithms

m	n	Number of problem instances	Number of problem instances for which one or more approximate solution values are suboptimal (minimum/maximum nonzero deviation)	Number of problem instances for which two or more approximate solution values are suboptimal (minimum/maximum nonzero deviation)	Number of problem instances for which all three approximate solution values are suboptimal (minimum/maximum nonzero deviation)
2	2	30	20 (1/8)	0 (–/–)	0 (–/–)
	4	688	308 (1/8)	0 (–/–)	0 (–/–)
	8	24,306	239 (1/4)	1 (1/2)	0 (–/–)
	16	2,042,975	0 (–/–)	0 (–/–)	0 (–/–)
4	4	688	559 (1/16)	0 (–/–)	0 (–/–)
	8	24,306	16,211 (1/16)	250 (1/11)	0 (–/–)
	16	2,042,975	14,028 (1/10)	41 (1/5)	5 (1/4)
8	8	24,306	23,465 (1/32)	0 (–/–)	0 (–/–)

solution algorithm to gateway clusters with small cluster sizes (i.e., with small values of m) and moderate traffic-dispatching granularity (i.e., with moderate values of n). Better yet, the results also indicate that the algorithms of greedy assignment and best-fit assignment, both of which can be easily improved to run in $O(n \cdot lg(m))$ time, are able to produce approximate solution values that are close to the optimal solution values. At last, an on-line, self-adjusting scheme based on the solution algorithms is developed to scale the performance of gateway clusters by effectively and efficiently dealing with the imbalance caused by the dynamic nature of network traffic.

Appendix A. Proofs of Lemma 1, Lemma 2, and Theorem 2

A feasible solution of **MGTDPv1** can be denoted by the disjoint subsets $\mathbf{W}_i^{(1)}$'s, $i = 0, 1, 2, \ldots, m$, of set $\mathbf{W} = \{w_j | j = 1, 2, 3, \ldots, n\}$, where $\mathbf{W}_0^{(1)}$ comprises the numbers of the packets which are not going to be processed by any gateway, each of the remaining $\mathbf{W}_i^{(1)}$, $i = 1, 2, 3, \ldots, m$, comprises the numbers of the packets that are going to be processed by gateway $\mathbf{G}_i$, and $\cup_{i=0}^{m} \mathbf{W}_i^{(1)} = \mathbf{W}$. Based on the values of x_{ij}'s, $i = 1, 2, 3, \ldots, m$, $j = 1, 2, 3, \ldots, n$, in the feasible solution, $\mathbf{W}_i^{(1)}$'s, $i = 0, 1, 2, \ldots, m$, are derived by $\mathbf{W}_0^{(1)} = \{w_j | \forall i\, x_{ij} = 0, i = 1, 2, 3, \ldots, m, j = 1, 2, 3, \ldots, n\}$ and $\mathbf{W}_i^{(1)} = \{w_j | x_{ij} > 0, j = 1, 2, 3, \ldots, n\}$, $i = 1, 2, 3, \ldots, m$. Though the same criteria can also be applied to a feasible solution of **MGTDPv2** and help derive another disjoint subsets $\mathbf{W}_i^{(2)}$'s, $i = 0, 1, 2, \ldots, m$, of set $\mathbf{W}$, the feasible solution cannot be denoted by $\mathbf{W}_i^{(2)}$'s, $i = 0, 1, 2, \ldots, m$, alone because some of the elements in $\mathbf{W}_i^{(2)}$'s, $i = 0, 1, 2, \ldots, m$, may contribute only part of itself to the solution value (i.e., when $0 < x_{ij} < 1$ holds for an element's corresponding x_{ij}). Instead, a feasible solution of **MGTDPv2** is determined by considering $\mathbf{W}_i^{(2)}$'s, $i = 0, 1, 2, \ldots, m$, and the corresponding x_{ij}'s, $i = 1, 2, 3, \ldots, m$, $j = 1, 2, 3, \ldots, n$, at the same time. We use $x_{ij}^{(1)}$'s, $i = 1, 2, 3, \ldots, m$, $j = 1, 2, 3, \ldots, n$, and $x_{ij}^{(2)}$'s, $i = 1, 2, 3, \ldots, m$, $j = 1, 2, 3, \ldots, n$, to denote the corresponding x_{ij}'s, $i = 1, 2, 3, \ldots, m$, $j = 1, 2, 3, \ldots, n$, with respect to each elements of $\mathbf{W}_i^{(1)}$'s, $i = 0, 1, 2, \ldots, m$, and $\mathbf{W}_i^{(2)}$'s, $i = 0, 1, 2, \ldots, m$, respectively. These notations are useful when we prove the following lemmas.

Lemma 3. For every feasible solution of **MGTDPv1**, there exists a feasible solution of **MGTDPv2** such that the solution value of the latter is equal to that of the former.

Proof. Let $\mathbf{W}_i^{(1)}$'s, $i = 0, 1, 2, \ldots, m$, denote a feasible solution of **MGTDPv1**, and let $z^{(1)}$ denote the corresponding solution value.A feasible solution of **MGTDPv2** can be constructed from $\mathbf{W}_i^{(1)}$'s, $i = 0, 1, 2, \ldots, m$, as follows.Let $\mathbf{W}_i^{(2)}$'s, $i = 0, 1, 2, \ldots, m$, and $x_{ij}^{(2)}$'s, $i = 1, 2, 3, \ldots, m$, $j = 1, 2, 3, \ldots, n$, denote the feasible solution that we are going to construct, and let $z^{(2)}$ denote the corresponding solution value.For every $\mathbf{W}_i^{(1)}$ such that $\sum_{w \in \mathbf{W}_i^{(1)}} w \leqslant c_i$, $i = 0, 1, 2, \ldots, m$, we set $\mathbf{W}_i^{(2)} = \mathbf{W}_i^{(1)}$ and $x_{ij}^{(2)} = x_{ij}^{(1)}$ for every element $w \in \mathbf{W}_i^{(1)}$, where j is the index of w in $\mathbf{W}$. Doing so ensures that the contribution of $\mathbf{W}_i^{(2)}$ to $z^{(2)}$ is equal to the contribution of $\mathbf{W}_i^{(1)}$ to $z^{(1)}$.For every $\mathbf{W}_i^{(1)}$ such that $\sum_{w \in \mathbf{W}_i^{(1)}} w > c_i$, we first use $\mathbf{W}_i^{(1)} = \{w_{i_1}, w_{i_2}, w_{i_3}, \ldots, w_{i_k}\}$ to denote it, where $k = |\mathbf{W}_i^{(1)}|$.Next, we determine the value of positive integer s such that $\sum_{j=1}^{s-1} w_{i_j} < c_i$ and $\sum_{j=1}^{s} w_{i_j} \geqslant c_i$, in which case w_{i_s} is referred to as the *split element*.Then we set $\mathbf{W}_i^{(2)} = \{w_{i_1}, w_{i_2}, w_{i_3}, \ldots, w_{i_s}\}$, $x_{ij}^{(2)} = 1$ for every element $w \in \mathbf{W}_i^{(2)} \setminus \{w_{i_s}\}$, where j is the index of w in $\mathbf{W}$,$x_{ij}^{(2)} = \frac{1}{w_{i_s}}(c_i - \sum_{w \in \mathbf{W}_i^{(2)} \setminus \{w_{i_s}\}} w)$ for the split element, where j is the index of the split element in $\mathbf{W}$, and $x_{ij}^{(2)} = 0$ for every element $w \in \mathbf{W}_i^{(1)} \setminus \mathbf{W}_i^{(2)}$, where j is the index of w in $\mathbf{W}$. As a result, the contribution of $\mathbf{W}_i^{(2)}$ to $z^{(2)}$ is equal to c_i, which is also the same as the contribution of $\mathbf{W}_i^{(1)}$ to $z^{(1)}$. Finally, for every element $w \in \mathbf{W}_0^{(1)}$, we set the values of all $x_{ij}^{(2)}$'s, $i = 0, 1, 2, \ldots, m$, to 0, where j is the index of w in $\mathbf{W}$, and set $\mathbf{W}_0^{(2)} = \mathbf{W} \setminus \cup_{i=1}^{m} \mathbf{W}_i^{(2)}$.

Following the procedure mentioned above, we can guarantee that the contribution of $\mathbf{W}_i^{(2)}$ to $z^{(2)}$ is exactly the same as the contribution of $\mathbf{W}_i^{(1)}$ to $z^{(1)}$ for every $i = 1, 2, \ldots, m$. Therefore, we conclude that $z^{(2)} = \sum_{i=1}^{m} \sum_{w \in \mathbf{W}_i^{(2)}} w x_{i I_{\mathbf{W}}(w)}^{(2)} = \sum_{i=1}^{m} Min(\sum_{w \in \mathbf{W}_i^{(1)}} w, c_i) = z^{(1)}$, where $I_Y(x)$ denotes the index of element x in set $\mathbf{Y}$, and hence prove the lemma. □

Lemma 4. For every feasible solution of **MGTDPv2**, there exists a feasible solution of **MGTDPv1** such that the solution value of the latter is greater than or equal to that of the former.

Proof. Let $\mathbf{W}_i^{(2)}$'s, $i = 0, 1, 2, \ldots, m$, and $x_{ij}^{(2)}$'s, $i = 1, 2, 3, \ldots, m$, $j = 1, 2, 3, \ldots, n$, denote a feasible solution of **MGTDPv2**, and let $z^{(2)}$ denote the corresponding solution value. A feasible solution of **MGTDPv1** can be constructed from $\mathbf{W}_i^{(2)}$'s, $i = 0, 1, 2, \ldots, m$, and $x_{ij}^{(2)}$'s, $i = 1, 2, 3, \ldots, m$, $j = 1, 2, 3, \ldots, n$, as follows. Let $\mathbf{W}_i^{(1)}$'s, $i = 0, 1, 2, \ldots, m$, denote the feasible solution that we are going to construct, and let $z^{(1)}$ denote the corresponding solution value. We simply set $\mathbf{W}_i^{(1)} = \mathbf{W}_i^{(2)}$ for every $i = 1, 2, 3, \ldots, m$ and $x_{ij}^{(1)} = \lceil x_{ij}^{(2)} \rceil$ for every $i = 1, 2, 3, \ldots, m$ and every $j = 1, 2, 3, \ldots, n$. As a result, we conclude that $z^{(1)} = \sum_{i=1}^{m} \sum_{j=1}^{n} w_j x_{ij}^{(1)} = \sum_{i=1}^{m} \sum_{j=1}^{n} w_j \lceil x_{ij}^{(2)} \rceil \geqslant \sum_{i=1}^{m} \sum_{j=1}^{n} w_j x_{ij}^{(2)} = z^{(2)}$ and hence prove the lemma. □

Let $\bar{\mathbf{W}}_i^{(2)}$'s, $i = 0, 1, 2, \ldots, m$, denote the derived disjoint subsets of $\mathbf{W}$ with respect to an optimal solution of **MGTDPv2**. We prove Theorem 2 as follows.

Theorem 3. In an optimal solution of **MGTDPv2**, if there exist one or more unassigned flows, all gateways must be fully loaded. That is,

$$|\bar{\mathbf{W}}_0^{(2)}| > 0 \Rightarrow \sum_{w \in \bar{\mathbf{W}}_i^{(2)}} w = c_i, \quad i = 1, 2, 3, \ldots, m.$$

Proof. Let $\hat{z}^{(2)}$ denote the solution value of an optimal solution of **MGTDPv2** with one or more unassigned flows, and let $\mathbf{F}_q$ denote one of these unassigned flows. Suppose that there exist an under loaded gateway, which is denoted by $\mathbf{G}_p$. Following these notations, the number of packets belonging to $\mathbf{F}_q$ is w_q, the number of packets to be processed by $\mathbf{G}_p$ is $\sum_{w \in \bar{\mathbf{W}}_p^{(2)}} w$, and the remaining capacity of $\mathbf{G}_p$ is $c_i - \sum_{w \in \bar{\mathbf{W}}_p^{(2)}} w$. Since $\mathbf{G}_p$ is not fully loaded, we have $\sum_{w \in \bar{\mathbf{W}}_p^{(2)}} w < c_i$ and $c_i - \sum_{w \in \bar{\mathbf{W}}_p^{(2)}} w > 0$.

We may construct another feasible solution of **MGTDPv2** from the given optimal solution by assigning $\mathbf{F}_q$ to $\mathbf{G}_p$ as follows. Let $z^{(2)}$ denote the solution value of this newly constructed solution. By definition, $z^{(2)} \leqslant \hat{z}^{(2)}$. In the case of $w_q \leqslant c_i - \sum_{w \in \bar{\mathbf{W}}_p^{(2)}} w$, we change the value of the corresponding x_{pq} from 0 to 1. Then the number of packets to be processed by $\mathbf{G}_p$ becomes $\sum_{w \in \bar{\mathbf{W}}_p^{(2)}} w + w_q x_{pq} = \sum_{w \in \bar{\mathbf{W}}_p^{(2)}} w + w_q$, resulting in $z^{(2)} = \hat{z}^{(2)} + w_q$. Since $w_q > 0$, we have $z^{(2)} > \hat{z}^{(2)}$, which is a contradiction. In the case of $w_q > c_i - \sum_{w \in \bar{\mathbf{W}}_p^{(2)}} w$, we change the value of the corresponding x_{pq} from 0 to $\frac{1}{w_q}(c_i - \sum_{w \in \bar{\mathbf{W}}_p^{(2)}} w)$, which still satisfies $0 \leqslant x_{pq} \leqslant 1$. Then the number of packets to be processed by $\mathbf{G}_p$ becomes $\sum_{w \in \bar{\mathbf{W}}_p^{(2)}} w + w_q x_{pq} = \sum_{w \in \bar{\mathbf{W}}_p^{(2)}} w + (c_i - \sum_{w \in \bar{\mathbf{W}}_p^{(2)}} w) = c_i$, resulting in $z^{(2)} = \hat{z}^{(2)} + (c_i - \sum_{w \in \bar{\mathbf{W}}_p^{(2)}} w)$. Since $c_i - \sum_{w \in \bar{\mathbf{W}}_p^{(2)}} w > 0$, we have $z^{(2)} > \hat{z}^{(2)}$, which is also a contradiction. Therefore, we conclude that such gateway must not exist and hence prove the theorem. □

References

[1] Srinivasan Keshav, Rosen Sharma, Issues and trends in router design, IEEE Communications Magazine 36 (5) (1998) 144–151.

[2] Craig Partridge, Philip P. Carvey, Ed Burgess, Isidro Castineyra, Tom Clarke, Lise Graham, Michael Hathaway, Phil Herman, Allen King, Steve Kohalmi, Tracy Ma, John Mcallen, Trevor Mendez, Walter C. Milliken, Ronald Pettyjohn, John Rokosz, Joshua Seeger, Michael Sollins, Steve Storch, Benjamin Tober, Gregory D. Troxel, David Waitzman, Scott Winterble, A 50-Gb/s IP router, IEEE/ACM Transactions on Networking 6 (3) (1998) 237–248.

[3] Yang Xu, Zhiwei Dai, Bin Liu, Wenjie Li, A scalable 10 Gb/s line-rate router with DiffServ support, in: Proceedings of the International Conference on Communication Technology (ICCT 2003), vol. 1, April 2003, pp. 407–411.

[4] Werner Bux, Wolfgang E. Denzel, Ton Engbersen, Andreas Herkersdorf, Ronald P. Luijten, Technologies and building blocks for fast packet forwarding, IEEE Communications Magazine 39 (1) (2001) 70–77.

[5] Dan Decasper, Zubin Dittia, Guru Parulkar, Bernhard Plattner, Router plugins: a software architecture for next-generation routers, IEEE/ACM Transactions on Networking 8 (1) (2000) 2–15.

[6] Scott Karlin, Larry Peterson, VERA: An extensible router architecture, Computer Networks 38 (3) (2002) 277–293.

[7] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, M. Frans Kaashoek, The click modular router, ACM Transactions on Computer Systems (TOCS) 18 (3) (2000) 263–297.

[8] Yitzchak Gottlieb, Larry Peterson, A comparative study of extensible routers, in: Proceedings of the 5th IEEE Conference on Open Architectures and Network Programming (OPENARCH 2002), June 2002, pp. 51–62.

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

[10] Young Bae Jang, Jung Wan Cho, A cluster-based router architecture for massive and various computations in active networks, in: Proceedings of the 17th International Conference on Information Networking (ICOIN 2003), Lecture Notes in Computer Science, vol. 2662, February 2003, pp. 326–335.

[11] Pan-Lung Tsai, Chun-Ying Huang, Yun-Yin Huang, Chia-Chang Hsu, Chin-Laung Lei, A clustering and traffic-redistribution scheme for high-performance IPsec VPNs, in: Proceedings of the 12th IEEE International Conference on High Performance Computing (HiPC 2005), Lecture Notes in Computer Science, vol. 3769, December 2005, pp. 432–443.

[12] Pyda Srisuresh, Kjeld Borch Egevang, Traditional IP network address translator (Traditional NAT) RFC 3022, 2001.

[13] Stan Hanks, Tony Li, Dino Farinacci, Paul Traina, Generic routing encapsulation (GRE) RFC 1701, 1994.

[14] Stan Hanks, Tony Li, Dino Farinacci, Paul Traina, Generic routing encapsulation over IPv4 networks RFC 1702, 1994.

[15] Charles Perkins, IP encapsulation within IP, RFC 2003, 1996.

[16] Om P. Damani, P. Emerald Chung, Yennun Huang, Chandra Kintala, Yi-Min Wang, ONE-IP: Techniques for hosting a service on a cluster of machines, Journal of Computer Networks and ISDN Systems 29 8-13 (1997) 1019–1027.

[17] The Linux Virtual Server Project – Linux Server Cluster for Load Balancing. <http://www.linuxvirtualserver.org//>.

[18] F5 Networks, Inc., F5 Networks – BIG-IP – Local Traffic Management. <http://www.f5.com/f5products/products/bigip/ltm//>.

[19] Foundry Networks, Inc., Foundry networks: products: scalable and high-performance application traffic management and web optimization switches. <http://www.foundrynet.com/products/webswitches/serveriron//>.

[20] Radware Ltd., Web Server Director for Server Farms: Load Balancing Web Server: Radware. <http://www.radware.com/content/products/wsd//>.

[21] David C. Plummer, An ethernet address resolution protocol, RFC 826 (1982).

[22] Jon Postel, Multi-LAN address resolution, RFC 925, 1984.

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

[24] Microsoft Corporation, Windows 2000 network load balancing technical overview. <http://www.microsoft.com/technet/prodtechnol/windows2000serv/deploy/confeat/nlbovw.mspx/>.

[25] Sujit Vaidya, Kenneth J. Christensen, A single system image server cluster using duplicated MAC and IP addresses, in: Proceedings of the 26th IEEE Conference on Local Computer Networks (LCN 2001).

[26] Luis Aversa, Azer Bestavros, Load balancing a cluster of web servers: using distributed packet rewriting, in: Proceedings of the 19th IEEE International Performance, Computing, and Communications Conference (IPCCC '00), 2000, pp. 24–29.

[27] Azer Bestavros, Mark Crovella, Jun Liu, David Martin, Distributed packet rewriting and its application to scalable server architectures, in: Proceedings of the 6th IEEE International Conference on Network Protocols (ICNP '98), 1998, pp. 290–297.

[28] Hesham El-Rewini, Hesham H. Ali, Ted Lewis, Task scheduling in multiprocessing systems, IEEE Computer 28 (12) (1995) 27–37.
[29] Mario J. Gonzalez Jr., Deterministic processor scheduling, ACM Computing Surveys 9 (3) (1977) 173–204.
[30] Thomas L. Casavant, Jon G. Kuhl, A taxonomy of scheduling in general-purpose distributed computing systems, IEEE Transactions on Software Engineering 14 (2) (1988) 141–154.
[31] Hans Kellerer, Ulrich Pferschy, David Pisinger, Knapsack Problems, Springer Verlag GmbH, Berlin, 2004, ISBN: 3-540-40286-1.
[32] Silvano Martello, Paolo Toth, Knapsack Problems Algorithms and Computer Implementations, Wiley, New York, 1990, ISBN: 0-471-92420-2.
[33] Shane Dye, Leen Stougie, Asgeir Tomasgard, Approximation algorithms and relaxations for a service provision problem on a telecommunication network, Discrete Applied Mathematics 129 (1) (2003) 63–81.
[34] Alberto Caprara, Hans Kellerer, Ulrich Pferschy, The multiple subset sum problem, SIAM Journal on Optimization 11 (2) (2000) 308–319.
[34] Alberto Caprara, Hans Kellerer, Ulrich Pferschy, The multiple subset sum problem, SIAM Journal on Optimization 11 (2) (2000) 308–319.

Pan-Lung Tsai received the B.S. degree in mechanical engineering and the M.S. degree in electrical engineering from National Taiwan University in 1997 and 1999, respectively.

Currently he is a Ph.D. candidate in the Department of Electrical Engineering, National Taiwan University. His research interests include software agents, mobile computing, network security, and distributed processing.

Mr. Tsai is a student member of the Institute of Electrical and Electronics Engineers and the Association for Computing Machinery.

Chin-Laung Lei received the B.S. degree in electrical engineering from National Taiwan University in 1980, and the Ph.D. degree in computer science from the University of Texas at Austin in 1986.

From 1986 to 1988, he was an assistant professor of the Computer and Information Science Department at the Ohio State University, Columbus, Ohio, U.S.A. In 1988 he joined the faculty of the Department of Electrical Engineering, National Taiwan University, where he is now a professor. His current research interests include computer and network security, cryptography, parallel and distributed processing, design and analysis of algorithms, and operating system design.

Dr. Lei is a member of the Institute of Electrical and Electronics Engineers and the Association for Computing Machinery.