Achieving Load Balance by Separating IP Address Spaces

Sanqi Zhou[1], Jia Chen[1], Huachun Zhou[1] and Hongke Zhang[1]

[1] National Engineering Laboratory for Next Generation Internet Interconnection Infrastructure
Beijing JiaoTong University
Beijing, 100044, China

Abstract

In this paper, we propose a load balance approach by separating the host and router IP addresses into two spaces. In addition, in our approach, we propose a scheduling algorithm, named Edge Stream Balance (ESB), which is used by the proposed multipath routing scheme based on the address space separation. Each router can schedule each stream that is initiated by the connected host onto the proper path to the destination host by ESB dynamically. The multiple paths between any pair of hosts can be obtained by the connected routers by using the address separating mechanism. The merit of our approach is that: it balances the network traffic dynamically while being free of traffic demand assumption and offline flow optimization. The path of each stream is selected by each router individually other than using central system based on the address separation. The time complexity of ESB is much lower than the linear programming (LP) and integer linear programming (ILP) which are used in flow optimization. Simulation results show that on average of all simulated scenarios, compared to the existing single path routing which is based on the address separating, the unused link ratio (ULR) reduces by 82%. And in the relative sense, the traffic across the network is balanced 31%.

Keywords: *Load Balance, Address Space Separation, Multipath.*

1. Introduction

In the existing Internet, load balance is a critical issue which is not still solved elegantly [1]. This is caused by two reasons: one is that the hosts and routers which are growing rapidly are distributed by power-law in the topology [2, 3], the other is the routing scheme is executed independently in each router based on the Shortest Path First (SPF) policy which may overload certain paths while underloading some others [4].

Previous works have explored load balance in some different ways. Fortz et. al. [5] proposed a solution of optimizing OSPF weights. Sridharan et. al. [4] and Wang et. al. [6] exploited more effective ways to split traffic over the shortest paths. Xu et. al. [7] optimized the link loads by using only E link weights against $O(NE)$ parameters in [4] and [6], where N is the number of routers. These schemes are all depending on assuming having knowledge of the network traffic demands. Furthermore in [4] and [6], a central controller is used to compute and configure the flow splitting ratio instead of routers. Thus, it sacrifices the main benefit of running a distributed protocol. Both Antic et. al. [8] and Tsunoda et. al. [9] proposed the shortest path routing (SPR) solutions respectively, in which each source router selects different intermediate routers for forwarding each packet at a time. In the both solutions, the traffic demand bound of each router should be known by all routers. In [8], the high time complexity of the realtime linear programming (LP) may affect practicing in a large topology. Keller et. al. [10] optimized the traffic infiltrated into other autonomous systems (AS) by migrating the intradomain edge links. However, the realtime traffic (e.g., the Audio stream) cannot be kept in this approach. Both Saucez et. al. [11] and Paul et. al. [12] proposed traffic engineering solutions in the Identifier(ID)/Locator separating context. However, in these solutions, the path selection is totally based on a central controller in each AS. Meanwhile, the topology and traffic information must be gathered in time or periodically by the controller.

Therefore, although the research on load balance has made a great progress [4-12], to our best knowledge, the following question is still to be addressed: *Is it possible to achieve load balance in a wholly distributed system without traffic demand assumption and high time complexity?* To answer, we present a novel approach.

As [11] and [12], our approach employs the principle of separating IP address space. That is because a key merit of separating IP space is to make possible to associate multiple router addresses to a unique host address. Thus, it enables each router choose different routers (i.e., the paths) to route the packets between the same host pair for balancing the traffic. In this paper, we address the question presented above by the following two steps.

First, we propose a multipath routing scheme based on a generalized address space separating mechanism. By this scheme, each router can encapsulate a set of appropriate router addresses into the packets of the same stream when each packet initially enters the network from the router, and can also decapsulate the addresses from the packet when it leaves the network. These addresses are obtained from the response messages of address space separating mechanism (described in Section II). Second, we propose

a scheduling algorithm, which is called Edge Stream Balance (ESB). In each router, it is used to select the appropriate router addresses to encapsulate into packets.

This approach brings three main merits. 1) The network traffic is balanced without a central controller. Because each router gets all host-to-routers addresses mapping information by the address space separating mechanism, and some parts of the realtime traffic information by dumping the packets it received, each router can select the appropriate router addresses while without centralized service. 2) The time complexity of ESB is much lower than the LP and integer linear programming (ILP) such that the traffic can be balanced timely. 3) ESB runs without any transcendental knowledge of traffic demand.

We also simulate our approach in a large number of scenarios and analyze the average performance in each topology model.

The rest of the paper is organized as follows. Section II describes the generalized IP address space separating mechanism. Section III describes the multipath routing scheme and the ESB algorithm. In section IV, we perform simulations and analyze the results. Section IV concludes the paper.

2. Generalized IP Address Space Separation

Several address space separating solutions, which are generally in the principle of ID/Locator separation, were proposed in recent years [13-15]. In this paper, we consider a generalized IP address space separating mechanism (without specific packet format definition and etc.) named *One-hop Distributed Hash Table (One-hop DHT) based address separation* [16], which can support the proposed multipath routing scheme and ESB algorithm to achieve load balance. Fig. 1 [17] shows how it works.

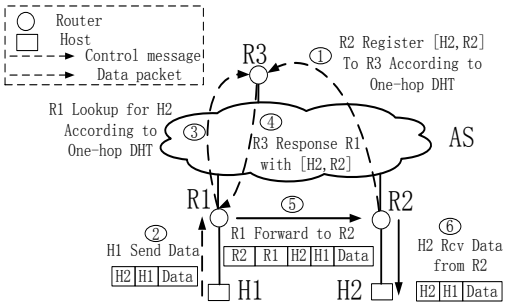


Fig. 1 One-hop DHT based address separation.

In Fig. 1, it shows the 6 primary steps of forwarding a packet from one host to another (i.e., H1 to H2) based on the address separation. Each router creates a hash ring with the specific addresses of all routers (e.g., the maximum address of each router obtained from Link State Database (LSDB) of OSPF [18]) by using an identical hash function (e.g., Secure Hash Algorithm 1, SHA-1 [19]). When a host accesses the network, the mapping entry of the host IP, its connected router's maximum IP (here we call it Router ID) and the router's other IPes are stored in a certain router according to the hash function (i.e., step 1). Thus, when a router forwards a packet received from its connected host (i.e., step 2), one of the destination router IPes can be obtained by using the hash function (i.e., step 3 and 4). Then, the source/destination router IPes are encapsulated into the packet header to forward to the destination router (i.e., step 5) which decapsulates the router IPes and forwards the packet to the destination host (i.e., step 6). Meanwhile, the mapping entry is cached by the source router for accessing the same host afterwards.

3. Load Balance Solution

In this section, we first describe the multipath routing scheme in Subsection 3.1. Then, ESB algorithm is described in Subsection 3.2.

3.1 Multipath Routing Scheme

Based on the separating mechanism mentioned above, each router can register its IPes and the neighbors' IPes which can be got by a certain protocol, e.g., the Hello protocol in OSPF, to another router which can be found in the hash ring created by the hash function. When the packets, sent from the same source host and destined to the same destination host, are received by the source router, they may be encapsulated with different IPes of the source router and its neighbors, destination router and its neighbors (which can be got from the router in which the IPes are registered). Then, the packets are forwarded along the paths indicated by these IPes. Each time the router, which is the intermediate destination indicated by one of the IPes, receives the packet, it changes the current destination IP into a new one which is contained in the packet, and then forwards it to the next intermediate destination. The process continues until the packet is received by the destination host.

Fig. 2 shows the multipath routing scheme based on the introduced separating mechanism. The two packets sent from Hs to Hd are routed on different paths that are indicated by solid and dash arrows respectively. The original protocol type in the IP header is temporarily saved between the data field and the IP header (i.e., the "**y**") by router A when the packet is forwarded from Hs to router A, and is restored by router G before being forwarded to Hd. In the forwarding process, some other specific values are

filled in the protocol field sequentially as shown in the underlined font in Fig. 2. These values can take the reserved protocol types defined in the IP standard [20], and are used by the multipath routing procedure in the packet forwarding process. Procedure 1 shows the pseudocode of the multipath routing scheme.

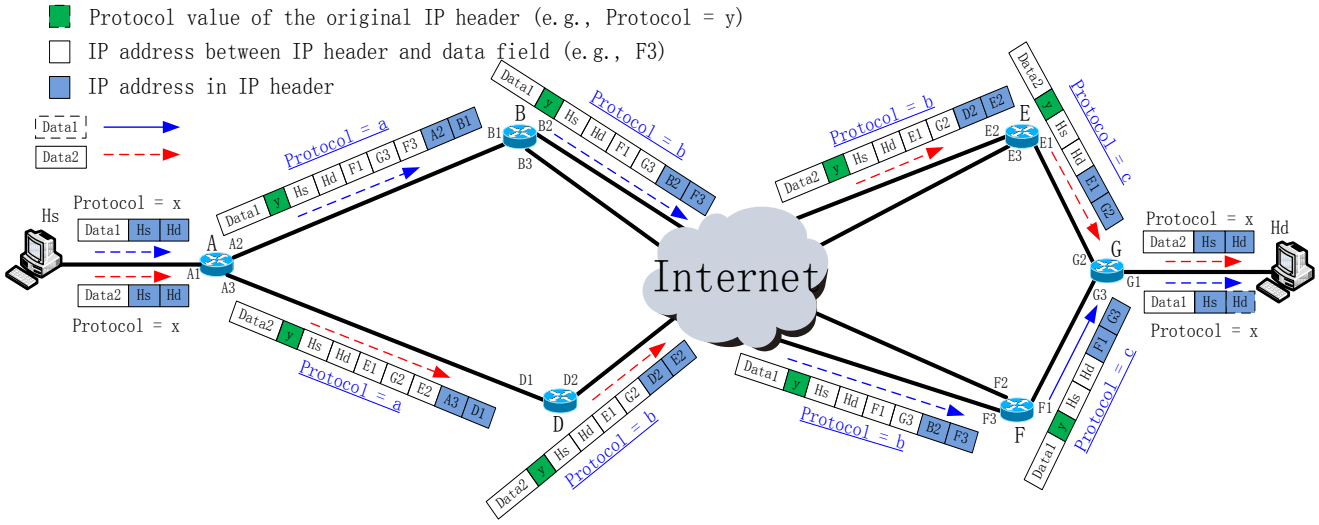


Fig. 2 Multipath routing scheme.

Procedure 1 Pseudocode of multipath routing scheme

//In this procedure, ***pkt*** is the received packet, ***pkt.hdr*** is the standard IP header, ***pkt.hdr.ptl*** is the protocol field in the IP header and the local mapping interfaces (***LMIF***) is the set of host accessible router's interface IPes, which are connected with other routers, and the router's neighbor IPes. We use ***DstLmifEnt*** to point to the <dest IP, ***LMIF***> entry which is the set of IPes of the destination host, routers and router neighbors. It is cached in the current router according to the introduced separating mechanism.

When a packet received by a router and the IP header has been checked:

1: **if** (***pkt.hdr.destIP*** *!=* the current interface IP) **then**
2: **if** (the interface is connected with hosts) **then**
3: **if**(*LookupCacheEntry*(***pkt.hdr.destIP***,***DstLmifEnt***)==**true**) **then**
4: *ESB*(***pkt***, ***LMIF***, ***DstLmifEnt***); //select IPes from ***LMIF*** and ***DstLmifEnt*** to be encapsulated into ***pkt***.
5: **else** //the <dest IP, ***LMIF***> entry has not been cached
6: Send a lookup packet based on the address separating mechanism and drop ***pkt***;
7: **return**;
8: **end if**
9: **end if**
10: **else** //***pkt*** destines to the interface
11: **switch** (***pkt.hdr.ptl***)
12: { **case a**: decapsulate the first IP behind the header (i.e., F3 in the data1 packet in Fig. 2) into ***pkt.hdr.destIP***, and then lookup the routing entries to get the output interface IP which is to be taken as ***pkt.hdr.srcIP***;
13: ***pkt.hdr.ptl*** = **b**; **break**;
14: **case b**: decapsulate the first IP behind the header into ***pkt.hdr.destIP***;
15: decapsulate the second IP behind the header into ***pkt.hdr.srcIP***;
16: ***pkt.hdr.ptl*** = **c**; **break**;
17: **case c**: restore the original source and destination IPes, and decapsulate the saved protocol field (i.e., "Protocol=**x**" in Fig. 2) into ***pkt.hdr.ptl***; **break**; }
18: Recalculate the other fields (i.e., checksum and etc.) in the IP header.
19: **end if**
20: Pass the packet to the existing forwarding procedure which is already modified to consider the protocol types **a**, **b** and **c** the same as regular IP packet;

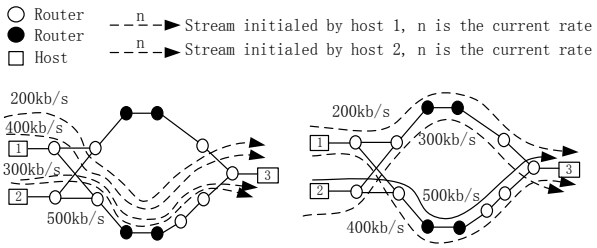
In statement "3", "*LookupCacheEntry*(***pkt.hdr.destIP***, ***DstLmifEnt***)" looks up all cached <dest IP, ***LMIF***> entries to find out the one that matches the first parameter, and then to point it with the second parameter.

Number of Paths. Here, we assume that there are averagely *k* neighbors per router and two bidirectional links between two routers at most. Thus, there are at most *2k* source router neighbor interfaces and *2k(k-1)* destination router neighbor interfaces (another two interfaces per destination router neighbor are connected with the destination router) can be selected to indicate different paths. Therefore, on average, there are $O(k^3)$ magnitude paths between each pair of routers.

3.2 Edge Stream Balance (ESB)

In Procedure 1, ESB is invoked to choose the appropriate IPes which are to be encapsulated in the packet to indicate the routers to be forwarded to (i.e., the statement "4"). The principle of ESB is: when the link utilization (LU) on the links between routers, one of which is connected with hosts is balanced (Edge Stream), the LU across the network is tended to be balanced (in the means of stream, that is, each time a new stream is initiated by host). The principle can hold for this reason: In the real network, the edge link with higher bandwidth is always connected with

the core link with higher bandwidth. This is to ensure each link can be used as transiting the traffic effectively as possible. Fig. 3 shows the single path routing scheme and the multipath routing scheme with ESB.



(a) Single Path Routing (b) ESB

Fig. 3 The schemes of the single path routing and the multipath routing with ESB. Each arrow indicates the path that a stream is forwarded along. The different colors of the nodes are used to illustrate the result of ESB compared to that of single path routing.

Assuming all links are with the same bandwidth, in Fig. 3(a), each stream initiated by either host 1 or host 2 is routed on the shortest path such that the lower link between the black nodes undertakes all 4 streams while the upper one does not carry any of them. However, in Fig. 3(b), the streams initiated by each host are separated on two paths by selecting the router interfaces which currently have the minimum LU. Therefore, the links between the black nodes are more approximately balanced. Notice that, ESB doesn't know the rate of a new initiated stream when the first packet received by the router. ESB only chooses the interfaces with minimum current LU for the stream, and then all the following packets of this stream are totally routed on the indicated path even if the LU of other interfaces may be lower. This is because the traffic can be balanced approximately in the means of stream level while keeping the time complexity of packet forwarding as lower as possible (i.e., ESB only takes more time when it is called for choosing a path for a stream for the first time). The algorithm includes two main phases. One is to balance the LU generated by the output traffic on the current router interfaces which connected with the neighbor routers, the other is to balance the LU of the input traffic on the destination router's neighbor interfaces which are not connected with the destination router. The first phase is achieved by selecting the interfaces of the minimum total current output LU among the local router interfaces and its neighbor interfaces. The second phase is achieved by selecting the interfaces of the minimum input LU in the cached <dest IP, ***LMIF***> entry that is corresponding with the destination IP of the packet. That is, the first phase records and uses the LU in the global scope while the second phase does that on each cached entry. Thus to the router that invokes ESB, the output LU on each of its link and the input LU on each link, which is not connected with the router, of all its neighbors, are balanced respectively. Algorithm 1 shows the pseudocode of ESB.

Algorithm 1 Pseudocode of ESB

ESB(*pkt*, *LMIF*, *DstLmifEnt*)
1: *len* = *pkt.hdr.length*;
2: $t_{PktDeque}$ = (t_{now} = *GetCurrentTime*()) - $\boldsymbol{T_{LUinterval}}$; //$\boldsymbol{T_{LUinterval}}$ is used to calculate the traffic rate, and hence the LU.
3: *PktDeque*(*LMIF.ALLif.iface*,*DstLmifEnt.ALLnif.iface*,$t_{PktDeque}$); //pop up all (*len*, t_{rcv}) before $t_{PktDeque}$ from each element of *iface* or *niface* arrays of the input parameters and subtract the corresponding *len* from the *trf* field in each element. The *trf* only contains the traffic from connected hosts.
4: **if**(*GetCachePath*(*pkt*, *&IP1*, *&IP3*, *&IP2*, *&hd*, *&hs*, *&proto*, **a**, t_{now}) == **false**) **then** //If *GetCachePath*() == **false**, a new stream is initiated. Otherwise, *pkt* belongs to an existing stream of which the path has been cached in the current router. Then, the combination of the source/destination IPes and TCP or UDP ports is used as the index to get the cached router IPes. Meanwhile, "**a**" is set into *pkt.hdr.ptl* and the *len* is accumulated on the correspondingly cached interfaces (i.e., the *trf* fields) such that the paths can be chosen reasonably when the next stream is initiated.
5: *la*=*minLUindex*(*LMIF.ALLif.iface*); //obtain the minimum output LU interface, the bandwidth is employed.
6: *LMIF.ALLif.iface*[*la*].*trf* += *len*;
7: *LMIF.ALLif.iface*[*la*].*PktEnque*(*len*, t_{now}); //push in queue //accumulate the output traffic in $\boldsymbol{T_{interval}}$ on the interface.
8: *lb* = 1; //init *lb*
9: **if**(*LMIF.ALLif.iface*[*la*].*NeighborIPnum*>1) **then** //multiple interfaces of neighbors are connected to the router interface
10: *lb*=*minLUindex*(*LMIF.ALLif.iface*[*la*].*niface*); //*LMIF.ALLif.iface*[*la*].*niface* is the interface array of the neighbor routers of *la* interface.
11: **end if**
12: *LMIF.ALLif.iface*[*la*].*niface*[*lb*].*trf* += *len*;
13: *LMIF.ALLif.iface*[*la*].*niface*[*lb*].*PktEnque*(*len*, t_{now});
14: *da*=*minLUindex*(*DstLmifEnt.ALLnif.iface*);//obtain the minimum input LU interface of the destination router neighbors.
15: *DstLmifEnt.ALLnif.iface*[*da*].*trf* += *len*;
16: *DstLmifEnt.ALLnif.iface*[*da*].*PktEnque*(*len*, t_{now});
17: *db* = 1; //init *db*
18: **if**(*DstLmifEnt.ALLnif.iface*[*da*].*router.ifaceNum*>1)**then** //*DstLmifEnt.ALLnif.iface*[*da*].*router.ifaceNum* is the number of the interfaces which are connected with the destination router and belongs to the router that has *DstLmifEnt.ALLnif.iface*[*da*].
19: *DstNeighbor* = *DstLmifEnt.ALLnif.iface*[*da*].*router*;
20: *db* = *minLUindex*(*DstNeighbor.iface*); //*DstNeighbor.iface* is the interface array of the selected destination router neighbor. The interfaces are connected with the destination router.
21: **end if**
22: *DstNeighbor.iface*[*db*].*trf* += *len*;
23: *DstNeighbor.iface*[*db*].*PktEnque*(*len*, t_{now}); //Encapsulate *pkt* and modify *pkt.hdr.ptl*
24: *proto* = *pkt.hdr.ptl*;
25: *pkt.hdr.ptl* = **a**; //set it to "**a**" for neighbor receiving
26: *hs* = *pkt.hdr.srcIP*;
27: *hd* = *pkt.hdr.destIP*;
28: *pkt.hdr.srcIP* = *LMIF.ALLif.iface*[*la*].*IP*;
29: *pkt.hdr.destIP* = *LMIF.ALLif.iface*[*la*].*niface*[*lb*].*IP*;
30: *IP1* = *DstLmifEnt.ALLnif.iface*[*da*].*IP*; //destination neighbor
31: *IP2* = *DstNeighbor.iface*[*db*].*IP*; //destination neighbor
32: *IP3* = *DstNeighbor.iface*[*db*].*peerIP*; //destination router
33: *SetCachePath*(*pkt*, *IP1*, *IP3*, *IP2*, *la*, *lb*, *da*, *db*, t_{now}); //Cache the router IPes. The corresponding interface indexes are also cached to record the following traffic of the same stream on the

interfaces.
34: **end if**
35: *Encapsulate* (*pkt, IP1, IP3, IP2, hd, hs, proto*);
36: **return**;

Delete the cached paths when the timer of the corresponding streams are timeout:
//*GetCachePath*() resets timer of the stream only if the stream has existed. *SetCachePath*() resets the stream timer when it is called, that is, when the stream is initiated.
1: *DelCachePath*(*pkt*);

In the statement "32", the *peerIP* indicates the destination router interface that connected with *IP2*. Because two interfaces of a router can't be connected in the real network, the *peerIP* is unique for each *IP2*. The function "*DelCachePath*" is used to delete the path which has been indicated for a stream when the path is timeout for this stream. This is because, when a stream is released or cancelled by a host, or be even "silent" for some reason, the router which is connected with the host can't be aware. Thus, we should delete the cached stream path in the local router. When the router receives any new packet of this stream from its connected host, it takes the packet as a start of an initiating stream and assigns a new path for it.

We can see that, both the multipath routing scheme and ESB algorithm only process each packet locally, while without communicating to any other router or central system beyond the address separating mechanism and routing protocol. Furthermore, ESB only chooses path by collecting and analyzing the realtime traffic that is sent from or destined to the local hosts, rather than using global traffic information or static traffic demands.

Time Complexity. In Algorithm 1, the time complexity of ESB is mainly made up of three phases. The first is "*minLUindex*" which performs the selection of the interface with minimum input or output LU. In the worst case, it sequentially traverses the interface array with random stored elements. Thus, the worst time complexity will be proportional to the array length. Assuming there are *k* neighbors per router and *n* interfaces between two routers at most, the interface number (connected with router) of a router won't be more than *k*n*. The total time complexity of the four invoking of "*minLUindex*" is

$$O(TotalminLUindex)=O(kn)_{la}+O(k)_{lb}+O(k(k-1)n)_{da}+O(n)_{db}, \quad (1)$$

where each term with subscript is the corresponding time complexity of the invoking in Algorithm 1. The largest array is *DstLifEnt.ALLnif.iface* with at most *k* routers. The length is no more than *k*(k-1)*n* (the destination router is a neighbor of each destination router neighbor). Hence, *O(TotalminLUindex)* is at most $O(k^2n)$. Generally, *n* is always far less than *k* (In general, two links between two routers at most). Thus, *O(TotalminLUindex)* won't be higher than $O(k^3)$. The second phase is "*PktDeque*" which maintains the traffic records in the most recent $\boldsymbol{T_{interval}}$ for all interfaces in the ***LMIFes***. Similarly with "*minLUindex*", "*PktDeque*" also has to sequentially traverse the four interface arrays. However, each interface in the arrays has a First In First Out (FIFO) queue which contains the traffic elements of "(*len*, t_{rcv})" sorted by time. "*PktDeque*" finds the first element that succeeds $t_{PktDeque}$ in each queue and clear all elements which prior to the element. Thus, the realtime traffic rate and LU can be obtained. Since the FIFO is sequentially sorted and can be saved in array, we can use binary search to find the successor of $t_{PktDeque}$ in each queue. The complexity is log_2N_{pkt}, where N_{pkt} is the number of traffic elements per queue. Therefore, the time complexity of "*PktDeque*" is

$$O(PktDeque)=O(TotalminLUindex)\bullet O(log_2N_{pkt}). \quad (2)$$

The third phase is made up of "*SetCachePath*" and "*GetCachePath*". Generally, the packet number of a stream is much higher than one, and "*GetCachePath*" is called each time a packet is received while "*SetCachePath*" is only called when the first packet of a stream is received. Thus, we can use "*SetCachePath*" to create a completely ordered list, and do binary search in "*GetCachePath*". Therefore, the worst time complexity of "*GetCachePath*" is

$$O(GetCachePath)=log_2\left(\left(2^{l_{IP}}\bullet 2^{l_{TCP}}\right)/2+\left(2^{l_{IP}}\bullet 2^{l_{UDP}}\right)/2\right), \quad (3)$$

where l_{IP} is the length of IP address and l_{TCP}/l_{UDP} is the length of TCP/UDP port. Such we have

$$O(GetCachePath)= l_{IP}+max(l_{TCP},l_{UDP}). \quad (4)$$

Hence, the total time complexity of ESB is

$$O(ESB)=O(TotalminLUindex)+O(PktDeque)+O(GetCachePath), \quad (5)$$

Thus, the worst *O(ESB)* is $O(k^3log_2N_{pkt}+l_{IP}+l_{TCP})$. In the existing Internet, l_{IP} is at most 128 [21], and l_{TCP} is equal to l_{UDP} which is 16 [22][23]. To our knowledge, the maximum bandwidth of a router interface today is less than 160Gbps [24]. Assuming $\boldsymbol{T_{interval}}$ is 1 second (long enough to calculate LU), due to the IP packet size is at least 20Bytes, N_{pkt} is less than 2^{30}. Thus in the worst condition, *O(ESB)* is still very small (less than $30k^3$). Compared to LP and ILP which is usually a non-polynomial (NP) or high order polynomial (P) problem [8, 25], ESB is a low order P problem thus can work well in realtime.

4. Evaluation

In this section, we perform the simulations in numerous scenarios to evaluate the performance of our approach in different network conditions. Then, we measure and analyze the unused link ratio (ULR) and the traffic distribution.

4.1 Simulation Environment

In this paper, we simulate 432 scenarios (144 scenarios for each algorithm) in NS2 [28]. Each scenario is made up of a certain case of each dimension shown in Table 1. The ***Host Number*** has two values which are used to simulate the lower and higher total network traffic. The ***Traffic Source Type*** can either be Pareto of which the burst interval is distributed in power law (e.g., some Web traffic), or the Constant Bit Rate (CBR) of which the burst interval is constant (e.g., live video). We also attach 7 traffic sources per host on average according to ***Traffic Dist***. Each source rate is 1Mb/s and the bandwidth of each link is 7Mb/s. The ***Host Dist*** refers to the distribution of the hosts along the routers, and the ***Traffic Dist*** refers to that of the traffic sources along the hosts. All distribution types of ***Host Dist*** and ***Traffic Dist*** are used to simulate as many cases as possible in the real network.

Table 1: The Cases Of Dimensions Of The Scenarios

Dimensions	**Case**			
Synthetic Topology	randloose	randtight	powloose	powtight
Host Number	100		500	
Traffic Source type	Constant Bit Rate (CBR)		Pareto	
Host Dist	Uniform	Exponential		Pareto
Traffic Dist	Uniform	Exponential		Pareto
Scheduling Algorithm	NONE	Round Robin (RR)		ESB

Table 2: Synthetic Topology Information

Name	**Topology**	**Router #**	**Link #**	**Prob.**	**One-Nbr.**
randloose	Pure-random	100	152	0.03	Random
randtight	Pure-random	100	491	0.1	Random
powloose	Power Law	100	151	0.03	0.25
powtight	Power Law	100	497	0.1	0.25

Table 2 shows the detailed topology information in Table 1. We set 100 routers in each synthetic topology and the number of links is also listed. The **One-Nbr.** refers to the fraction of routers which only have one neighbor. The first and the second topologies are random graphs generated by GT-ITM [27] while the others are power law graphs generated by Inet-3.0 [28] with our modification on its supported ***Prob.***. The ***randloose*** and ***randtight*** are generated by the pure-random algorithm with constant connection probability (i.e., the ***Prob.***) between each pair of routers. Hence, there are more paths between the routers in ***randtight*** than in ***randloose*** according to Table 2. The ***Prob.*** of ***powloose*** and ***powtight*** are equal to that of the corresponding random graphs such that the influence of the degree distribution can be observed. Meanwhile, there are 25% routers in ***powloose*** and ***powtight*** with only one neighbor. Notice that, our approach works as single path routing while the stream is established between the hosts that connected with these routers.

Why only Synthetic Topologies and Scenarios are included? This is because we are to find out which specific topology model (other than the approximate ones, e.g., the real topology) is more appropriate for performing our approach on average of all specific traffic models. The results can be used as a reference to estimate the performance of our approach when using into practice, e.g., a real topology with predicted traffic demands.

4.2 Unused Link Ratio (ULR)

Fig. 4 shows the ULR of all scenarios. The ULR is defined as the ratio of the number of links without data traffic to the total link number in the topology. Due to round robin (RR) or ESB may select different source and destination router neighbors for the identical host pair, compared to single path routing (i.e., NONE in Fig. 4), more links can be used to transmit the traffic of the same host pair on the incompletely overlapped paths. Hence, the ULR of NONE is always larger than RR and ESB. Due to the average link number per router in ***randtight***/***powtight*** is higher than that in ***randloose***/***powloose***, the ULR in ***randtight***/***powtight*** is higher under NONE because higher ratio of links are not on the shortest path between any communication pair, and is lower under RR or ESB because the path number between each pair of routers is three order of neighbor (thus the link) number of each router on average such that higher ratio of links can be covered by the paths in ***randtight***/***powtight***. Compared ***powloose***/***powtight*** to ***randloose***/***randtight***, the ULR in ***powloose***/***powtight*** is always higher under NONE because most shortest paths between routers pass through the links that connected with the hub nodes in ***powloose/powtight*** such that the links (major part of all links) which are not connected with the hubs are much less probable to be used than all links in ***randloose***/***randtight***. The ULR in ***powloose***/***powtight*** is also higher under RR or ESB because most routers have lesser neighbors than those in ***randloose***/***randtight*** such that the path number between each pair of routers is much lower (due to the three order relationship mentioned above) than ***randloose***/***randtight*** on average, thus more links are not used. Fig. 4 also indicates that the ULR is higher when the host number is lower in the same condition. In each grid divided by the solid and dash lines, the front part of each curve is always lower than the back part. This is because the ***Host Dist*** of the front part is uniform. Although the ***Traffic Dist*** is Pareto in some cases of the front part while it is uniform in the back part, the traffic source number is 7 times of the host in each scenario (by our setting) such that all communication paths in the front part in each grid of each curve cover more links than the back ones. In Fig. 4, we can also easily find that the ***randtight*** is most appropriate for the multipath routing

scheme (the ULR reduces most sharply when using the algorithms) while both ***powloose*** and ***randloose*** have worse effects. That means, the multipath routing schem takes better effect when the number of links among routers is higher and the links distribute more uniformly on the routers. Meanwhile, the scenarios in ***randloose*** have worse effect is because the ULR of NONE is much lower than other topologies. Since RR has used practically all neighbors of the source and destination router of a stream, the ULR performance of ESB is a little lower than RR. Compared to NONE, the average reduction of ULR of all scenarios is 82.4% when using RR, and is 82.0% when using ESB.

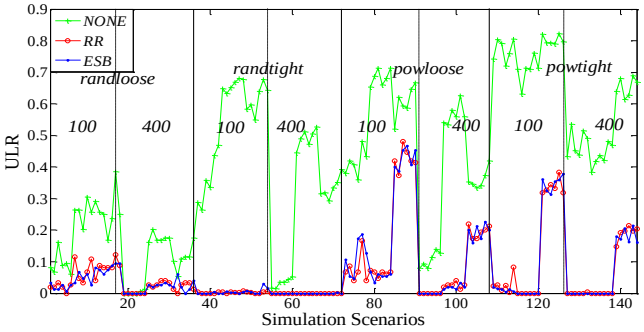


Fig. 4 ULR. The vertical solid lines separate different topologies while the dash ones separate different host number in each topology.

4.3 Traffic Distribution

Due to the total network traffic in different scenarios may not be equal (e.g., different ***Host Number***), we use the normalized link utilization (NLU) which is the ratio of LU to maximum link utilization (MLU), to present the traffic distribution of all scenarios in a same scale [17]. Figs. 5-8 show respectively the traffic ratio (TR) along NLU which of a scenario can be calculated as:

$$NLU_{link_i(j)} = \left(\frac{Trf_{link_i(j)}}{B_{link_i} \bullet T_{sim(j)}} \right) \Big/ \left(\frac{maxLUTrf_j}{B_{link_{maxLUTrf}(j)} \bullet T_{sim(j)}} \right), \quad (6)$$

where $Trf_{link_i(j)}$ and $maxLUTrf_j$ are respectively the traffic on $link_i$ and the link with MLU during the simulation time $T_{sim(j)}$ in scenario j, B_{link_i} and $B_{link_{maxLUTrf}(j)}$ are respectively the bandwidth of $link_i$ and the link with $maxLUTrf_j$. Due to the bandwidth of each link in each topology and $T_{sim(j)}$ in each scenario are both set the same in our simulation, we have

$$NLU_{link_i(j)} = Trf_{link_i(j)} \Big/ maxLUTrf_j \ . \quad (7)$$

TR is defined as the ratio of the traffic on links of a certain NLU range to the total network traffic:

$$TR_{NLU_k}(j) = \sum_{i=1}^{link\#_{NLU_k}(j)} Trf_{link_{(i,k)}(j)} \Big/ \sum_{i=1}^{n} Trf_{link_i(j)} \ , \quad (8)$$

where $Trf_{link_{(i,k)}(j)}$ is the traffic of the i^{th} link in $link\#_{NLU_k}(j)$ and n is the number of links in the topology of scenario j. Combining (6) and (8), we have

$$TR_{NLU_k}(j) = \sum_{i=1}^{link\#_{NLU_k}(j)} NLU_{link_{(i,k)}(j)} \Big/ \sum_{i=1}^{n} NLU_{link_i(j)} \ , \quad (9)$$

where $NLU_{link_{(i,k)}(j)}$ is the NLU of the i^{th} link in $link\#_{NLU_k}(j)$. That is, TR can be calculated by using NLU.

In Figs. 5-8, since each point is the mean value of TR of all scenarios under the corresponding topology and algorithm, and the standard deviation (STDEV) is small, the average traffic distribution of the scenarios can be generally represented by the corresponding curve.

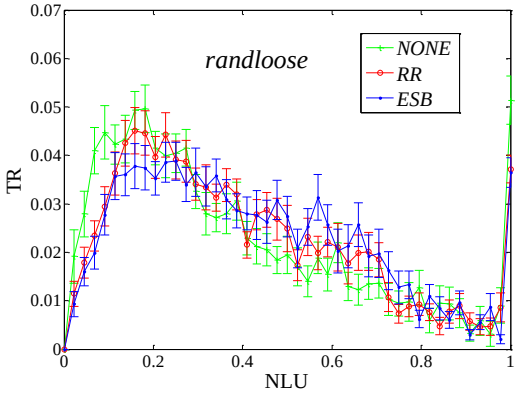


Fig. 5 Traffic Ratio.

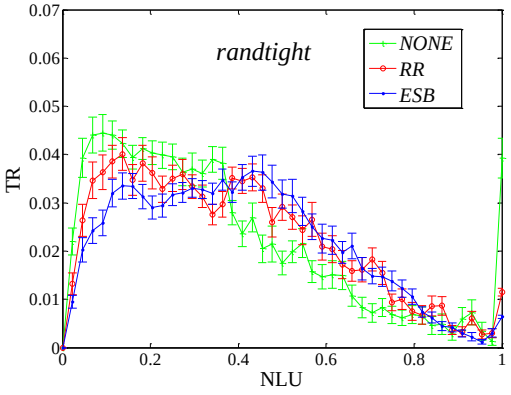


Fig. 6 Traffic Ratio.

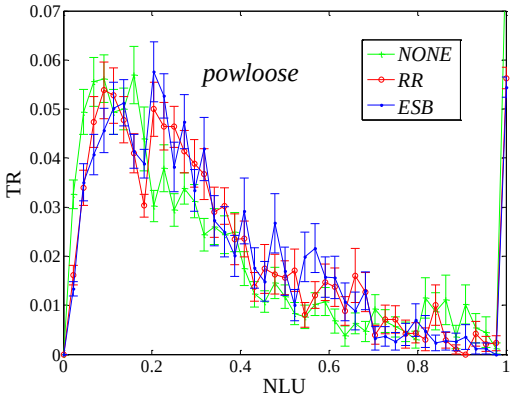


Fig. 7 Traffic Ratio.

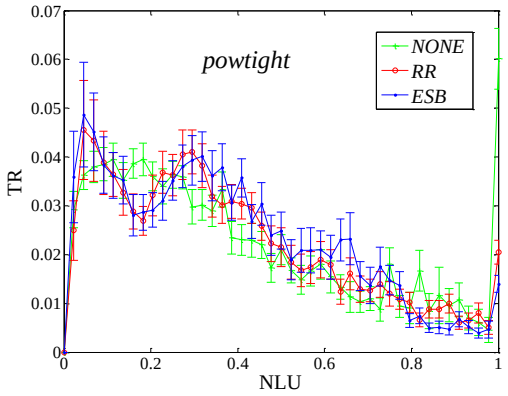


Fig. 8 Traffic Ratio.

In Figs. 5-8, compared to NONE, the TR of RR or ESB is higher when NLU nears to the middle and is lower when it nears to the two ends. This is because RR or ESB splits the traffic of the links with higher LU onto the links many of which are with lower LU, such that the LU and NLU of these links "move" towards each other. Hence, the increasing of TR in the middle part of NLU arises from balancing the LU of most links. Along with LU balancing, the increasing part of TR will be higher and move to right on NLU. That is, the LU of more links will be closer to MLU which will be smaller in each scenario accordingly. Ideally, if the TR is 1 when NLU is 1, MLU is the minimum.

In Fig. 8, we notice that the TR in the lower part of NONE is lower than RR and ESB. That is because the ULR of NONE is too high in ***powtight***. More links can only get much lower traffic in ***powtight*** than in other topologies when using RR or ESB. Therefore, RR or ESB can only make those links, which are actually "idle" in NONE, with low LU. And hence, RR or ESB make TR higher in the lower part of NLU than NONE. We use the sum of TR in the NLU range of [0, 0.1] and [0.9, 1] to present the effect of LU balance shown in Table 3. Since the LU of most links in each scenario are all still much lower than the MLU, the traffic is more balanced when the sum is lower.

Table 3: Sum of Traffic Ratio

Topology	TR (NLU in [0, 0.1] and [0.9, 1])		
	NONE	***RR***	***ESB***
randloose	24.4%	17.7%	16.1%
randtight	25.3%	17.6%	13.4%
powloose	36.5%	27.9%	25.7%
powtight	27.1%	23.4%	23.1%

In Table 3, ESB is better in each topology. This is because ESB always uses the path which has the minimum LU on the two ends currently for each stream. Compared to NONE of the 4 topologies, in the relative sense, the average reduction of the sum of TR is 23.7% when using RR, and is 31.3% when using ESB. The ***randtight*** has the best effect (reduces 30.5% in RR and 47.1% in ESB) because more paths can be used to carry the split traffic for most streams. The ***powtight*** has the worst effect (reduces 13.5% in RR and 14.6% in ESB) because the TR of RR or ESB is higher than NONE when NLU in [0, 0.1], which means more "idle" links just start to carry traffic when RR or ESB are used in ***powtight*** than in other topologies.

5. Conclusions

In this paper, we have proposed a load balance approach by separating the host and router IP addresses into two spaces. In our approach, we have proposed a scheduling algorithm, named ESB, which is used by the proposed multipath routing scheme based on the address space separation. Each router can schedule each stream that is initiated by the connected host onto the proper path to the destination host by ESB dynamically. The multiple paths between any pair of hosts can be obtained by the connected routers by using the address separating mechanism. The merit of our approach is that: it balances the network traffic dynamically while being free of traffic demand assumption and offline flow optimization. The path of each stream is selected by each router individually other than using central system based on the address separation. The time complexity of ESB is much lower than the LP and ILP which are used in flow optimization. Simulation results have shown that on average of all simulated scenarios, compared to the existing single path routing which is based on the address separating, our approach evidently reduces the ULR and in relative terms, balances the traffic across the network.

Acknowledgments

This work is supported in part by NSF of China under Grant No. 61232017 and 61102049, in part by National High Technology Research and Development

Program("863"Program) of China under Grant No. 2011AA010701 and 2011AA01A101, in part by Fundamental Research Funds for the Central Universities under Grant No. 2011JBM009 and in part by Innovation Funds for Excellent PhDs of Beijing JiaoTong University under Grant No. 2011YJS207.

References

[1] J. He and J. Rexford, "Toward Internet-Wide Multipath Routing", IEEE Network, 2008, Vol. 22, pp. 16 - 21.
[2] W. Willinger, V. Paxson, and M. S. Taqqu. "Self-similarity and heavy tails: Structural modeling of network traffic", A Practical Guide to Heavy Tails: Statistical Techniques and Applications, 1998.
[3] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of the internet topology", in Proc. ACM SIGCOMM, 1999.
[4] A. Sridharan, R. Guerin, and C. Diot, "Achieving near-optimal traffic engineering solutions for current OSPF/IS-IS networks", IEEE/ACM Trans. on Networking, Apr. 2005.
[5] B. Fortz and M. Thorup, "Internet Traffic Engineering by Optimizing OSPF Weights", Proc. IEEE INFOCOM 2000.
[6] Z. Wang, Y. Wang, and L. Zhang, "Internet traffic engineering without full mesh overlaying", Proc. IEEE INFOCOM 2001.
[7] D. Xu et. al., "Link-state routing with hop-by-hop forwarding can achieve optimal traffic engineering", IEEE Trans. on Networking, Apr. 2011.
[8] M. Antic et. al., "Two Phase Load Balanced Routing using OSPF", IEEE Jour. of Selected Area in Comm., Jan. 2010.
[9] S. Tsunoda, et. al., "Load-Balanced Shortest-Path-Based Routing Without Traffic Splitting in Hose Model", Proc. IEEE ICC 2011.
[10] E. Keller, et. al., "Rehoming edge links for better traffic engineering,", ACM SIGCOMM CCR, Mar. 2012.
[11] D. Saucez, et. al., "Interdomain Traffic Engineering in a Locator/Identifier Separation Context", Proc. INM, Oct. 2008.
[12] S. Paul, et. al., "An Identifier/Locator Split Architecture for Exploring Path Diversity through Site Multi-homing - A Hybrid Host-Network Cooperative Approach" Proc., IEEE ICC 2010.
[13] D. Farinacci, et. al., "Locator/ID separation protocol (LISP)", IETF, Internet Draft draft-ietf-lisp-16, Nov. 2011.
[14] R. Moskowitz, et. al., "Host identity protocol", RFC5201, Apr. 2008.
[15] E. Nordmark and M. Bagnulo, "Shim6: Level 3 multihoming shim protocol for IPv6", IETF, RFC 5533, Jun. 2009.
[16] C. Kim et. al., "Floodless in SEATTLE: A Scalable Ethernet Architecture for Large Enterprises", Proc. ACM SIGCOMM 2008.
[17] Sanqi Zhou, Jia Chen, Hongbin Luo and Hongke Zhang, Proceedings of 2012 World Congress on Information and Communication Technologies (WICT2012), Nov. 2012.
[18] J. Moy, "OSPF Version 2", IETF, RFC 2328, Apr. 1998.
[19] D. Eastlake and P. Jones, "US Secure Hash Algorithm 1 (SHA1)", IETF RFC3174, Sep. 2001.
[20] "Internet Protocol", IETF, RFC 791, Sep. 1981.
[21] "Internet Protocol, Version 6", IETF, RFC 2460, Dec. 1998.
[22] "Transmission Control Protocol", IETF, RFC793, Sep. 1981.
[23] "User Datagram Protocol", IETF, RFC 768, Aug. 1980.
[24] Cisco, U.S. [Online] http://www.cisco.com/en/US/prod/collateral/routers/ps5763/CRS-FP-140_DS.pdf.
[25] A. Elwalid et. al., "MATE: MPLS adaptive traffic engineering", Proc. IEEE INFOCOM, 2001.
[26] "NS2", USC/ISI, Xerox PARC, LBNL and UCB, U.S. [Online].
[27] "GT-ITM", CC, Georgia Institute of Technology, U.S. [Online].
[28] "Inet-3.0", CCES, University of Michigan, U.S. [Online].

Sanqi Zhou received the B.S. degree in electrical engineering and automation from North China Power Electric University, Beijing, China, in 2007. He received the M.S. degree in Traffic Information Engineering and Control from Beijing JiaoTong University, China, in 2009. He is pursuing the Ph.D. degree at national engineering laboratory for next generation Internet interconnection devices, Beijing Jiaotong University. His research interests include network traffic engineering, energy efficiency and the next generation Internet technology.

Jia Chen received her B.S. degree in Communication Engineering in 2005 from Beijing University of Posts and Telecommunications in China. She received Master of Research (M.R.) degree in Telecommunications in 2006, and the Ph.D degree in Electrical and Electronic Engineering 2010 from Department of Electrical and Electronic Engineering, University College London, UK. She worked in British Telecommunication (UK) for an industry fellowship position from Jan. 2009 to Apr. 2009. She joined Beijing Jiaotong University (Beijing, China) as a lecturer since July 2010. Her current research interests include architecture and protocol design and analysis for the future Internet.

Huachun Zhou received his B.S. degree from People ' s PoliceOfficer University of China in 1986, and the M.S. and Ph.D. degree from Beijing Jiaotong University of China in 1989 and 2008, respectively. He is currently a professor with the Institute of Electronic Information Engineering, Beijing Jiaotong University of China. His main research interests are in the area of mobility management, mobile and secure computing, routing protocols, network management technologies and database applications.

Hongke Zhang received his M.S. and Ph.D. degrees in Electricaland Communication Systems from the University of Electronic Science and Technology of China in 1988 and 1992, respectively. From Sep. 1992 to June 1994, he was a post-doc research associate at Beijing Jiaotong University. In July 1994, he jointed Beijing Jiaotong University, where he is a professor. He has published more than 100 research papers in the areas of communications, computer networks and information theory. He is the director of the National Engineering Laboratory for Next Generation Internet Interconnection Devices.