Dial-Controlled Hash: Reducing Path Oscillation in Multipath Networks

Minwen Ji
Systems Research Laboratory
HP Laboratories Palo Alto
HPL-2003-98
May 8th , 2003*

path oscillation, multipath networking, TCP

In a multipath network with a naive traffic partitioning scheme, varied packet loss rates as well as varied delays on multiple paths could seriously reduce TCP bandwidth. We propose a scheme, called *dial-controlled hash* (DCH), for dynamically partitioning traffic flows across multiple paths. DCH allows for fine-grained load balance, minimizes path oscillation for each traffic flow, and does not require per-flow state maintenance or packet tagging. We evaluate DCH in two simulated networks, a parallel link topology and a corporate intranet with trace-based traffic generation. In both simulations, we compare the proposed scheme to a number of alternatives. DCH reduces path oscillation by a factor of 1.6-37 and improves TCP bandwidth by up to 66%, compared to alternatives. Our simulation results also suggest that, without a good flow partitioning scheme, there is a limit on how much one can save by using low-quality networks in combination with high-quality ones.

* Internal Accession Date Only
Approved for External Publication

Dial-Controlled Hash: Reducing Path Oscillation in Multipath Networks

Minwen Ji
Systems Research Center, HP Labs

Abstract

In a multipath network with a naive traffic partitioning scheme, varied packet loss rates as well as varied delays on multiple paths could seriously reduce TCP bandwidth. We propose a scheme, called *dial-controlled hash* (DCH), for dynamically partitioning traffic flows across multiple paths. DCH allows for fine-grained load balance, minimizes path oscillation for each traffic flow, and does not require per-flow state maintenance or packet tagging. We evaluate DCH in two simulated networks, a parallel link topology and a corporate intranet with trace-based traffic generation. In both simulations, we compare the proposed scheme to a number of alternatives. DCH reduces path oscillation by a factor of 1.6-37 and improves TCP bandwidth by up to 66%, compared to alternatives. Our simulation results also suggest that, without a good flow partitioning scheme, there is a limit on how much one can save by using low-quality networks in combination with high-quality ones.

1 Introduction

A *multipath network* in this paper refers to a network that has multiple media channels (such as wired and wireless channels), multiple routes (such as those generated by a multipath routing protocol), or redundant links between a source and a destination. Multipath networking is common in today's wide area networks, including both the public Internet and private intranets. Multipath networks are deployed for reasons of performance, economy, function or fault tolerance. In this paper, we study the *traffic partitioning* schemes for multipath networks, which select a *route* (e.g. a single-hop link or an end-to-end path) for each packet from a set of available candidates.

In current practice, network managers often have all traffic forwarded to only one of the routes (i.e. the *primary* route) or *statically* partition traffic across routes, and configure the routers for failover in case of route outages [6] [5]. The primary route strategy leaves the secondary routes idle most of the time, and hence does not have good utilization of network resources. The static partitioning can be done by manually assigning source/destination addresses to routes in the routing tables, or by mathematically hashing packet addresses to routes on the fly. In either case, packets labeled with the same source/destination addresses are always transmitted over the same path, which is a desired behavior since it eliminates path oscillation. However, static partitioning requires reconfiguration when network entities or traffic patterns change. In the interim, packets may be dropped if the traffic assigned to a route exceeds the route capacity, while other routes stay idle.

On the other hand, a *dynamic* partitioning of traffic across multiple routes can improve the utilization of network resources and handle traffic bursts gracefully by load balancing across routes. However, dynamic partitioning is not popular in current multipath networks because load balancing inevitably causes *path oscillation* to traffic flows, causing packets in the same flow to be delivered out of order or with larger delay variance. This could be a disaster to certain traffic flows, such as media streams (by reducing quality) and TCP connections (by reducing bandwidth). We quantitatively analyze the impact of path oscillation on TCP bandwidth in Section 2.

A number of techniques have been developed to reduce the negative impact of path oscillation or to reduce path oscillation itself.

A resequencing buffer at the receiver can be used to handle out-of-order delivery [2]. However, it requires additional memory for buffers and increases packet delays.

Configurations	Single	Pair	Quad	TwinL	PairL +SA	PairL	PairL-FR	SingleL
Number of links	1	2	4	2	2	2	2	1
Link loss rates	0	0, 0	0, 0, 0, 0	0, 5%	0, 5%	0, 5%	0, 5%	5%
Link latency (ms)	100	80, 120	66, 88, 112, 134	100, 100	80, 120	80, 120	80, 120	100
Fast retransmit	On	On	On	On	On	On	Off	On
Selective Ack	Off	Off	Off	Off	On	Off	Off	Off

Table 1: Configurations in the single-flow simulations. The configurations are listed from left to right in descending order of TCP sending bandwidth. "L" in the configuration names stands for "lossy link". Each link has 1 Mbps bandwidth.

A brute-force method to reduce path oscillation is to record the route assignment for each flow in a router and adjust the assignment as needed for load balancing. This method, of course, cannot scale to a large number of flows per router. It is also possible to encode the route assignment into the packet headers rather than to keep it in a router [9], but this requires all routers along the paths to agree on the same encoding scheme, which would be difficult to arrange on the public Internet.

A simple improvement to the brute-force method is to maintain the route assignment at a coarser granularity. In a commonly used hashing scheme called *bucket-based hash* (BBH), flows are grouped into *buckets* by hashing their addresses, where the number of buckets is smaller than the number of flows, but larger than the number of routes; then the router maintains the route assignment for each bucket rather than for each flow. When load is unbalanced, at least one bucket of flows needs to be moved across routes in order to rebalance the load. The question is how many buckets should there be for a given network and traffic pattern. A large number of buckets allows for fine-grained load balance but requires more memory and more computation, while a small number of buckets may not be effective in reducing path oscillation, especially if buckets are unevenly loaded. When traffic pattern changes, it might be necessary to reset the number of buckets accordingly. This is unattractive because it increases the burden on network management.

Therefore, we are motivated to investigate flow partitioning schemes for multipath networks that have the following desirable properties:

- Allow for fine-grained load balancing.
- Preserve the path for each traffic flow as long as load balancing permits.
- In case of unbalanced load, only a *minimal* amount of traffic needs to switch paths.
- Do not require per-flow state maintenance or packet tagging.

In the rest of the paper, we will study the impact of path oscillation on TCP traffic in depth (Section 2), propose a new flow partitioning scheme using *dial-controlled hash* (Section 3), and evaluate the new scheme in simulations and compare it with alternatives (Section 5).

2 Impact of path oscillation on TCP bandwidth

2.1 Packet loss variation and out-of-order delivery

TCP, which is the protocol for the majority of traffic on the Internet and private networks, uses a *congestion window* to limit how fast the sender can inject packets into the network. The window size is adjusted in response to measured network conditions, e.g. increased by 1 when a new *acknowledgement packet* (ack) is received, and reset to 1 when a packet is inferred to have been lost and hence needs to be retransmitted [3]. In general, a network with larger capacity or less congestion will produce larger window size, which allows higher sending

bandwidth. In other words, the window size or sending bandwidth of a TCP connection is *negatively correlated* to the packet loss rate of the underlying network path. If packets in the same TCP connection are transmitted over paths of different loss rates, the resulting bandwidth may reflect the *highest* loss rate rather than take advantage of the bandwidth available in lower-loss paths.

Packets in a TCP connection are uniquely numbered in the sequence in which they are transmitted for the first time. When an above- or below-sequence packet arrives, the TCP receiver generates a *duplicate ack* with a sequence number that has been acknowledged previously. In another widely implemented retransmission mechanism, called *fast retransmit*, when the number of duplicate acks reaches a given threshold (typically 3), the sender infers that a packet was lost and retransmits it. Therefore, transmitting packets in the same TCP connection over paths of different delays may mislead the sender into shrinking the congestion window unnecessarily. A TCP extension for "selective ack" (SACK) allows more detailed feedback of which out-of-order packets are received and hence reduces unnecessary retransmissions [7]. However, SACK does not address the problem caused by loss rate variation in multipath networks.

Therefore, due to loss rate variation as well as out-of-order delivery, TCP traffic may not be able to consume the bandwidth otherwise made available by multipath networking.

2.2 Quantitative impact

In order to quantitatively motivate the design of new flow partitioning schemes, we studied the impact that path oscillation could have on TCP congestion window size and sending bandwidth. We simulated a single TCP connection between two routers with a round robin flow partitioning scheme. This simple configuration allows us to isolate the impact of path oscillation from that of other sources, such as congestion.

Detailed settings that are common in all simulations throughout this paper are described in Section 5.1.

In the simulation discussed in this section, the topology consists of two routers and the workload is a single TCP connection with 1 Mbps *nominal rate* (i.e. rate at which traffic is generated by an application). Table 1 shows the various configurations. In the configurations with multiple (2 or 4) links between the routers, each

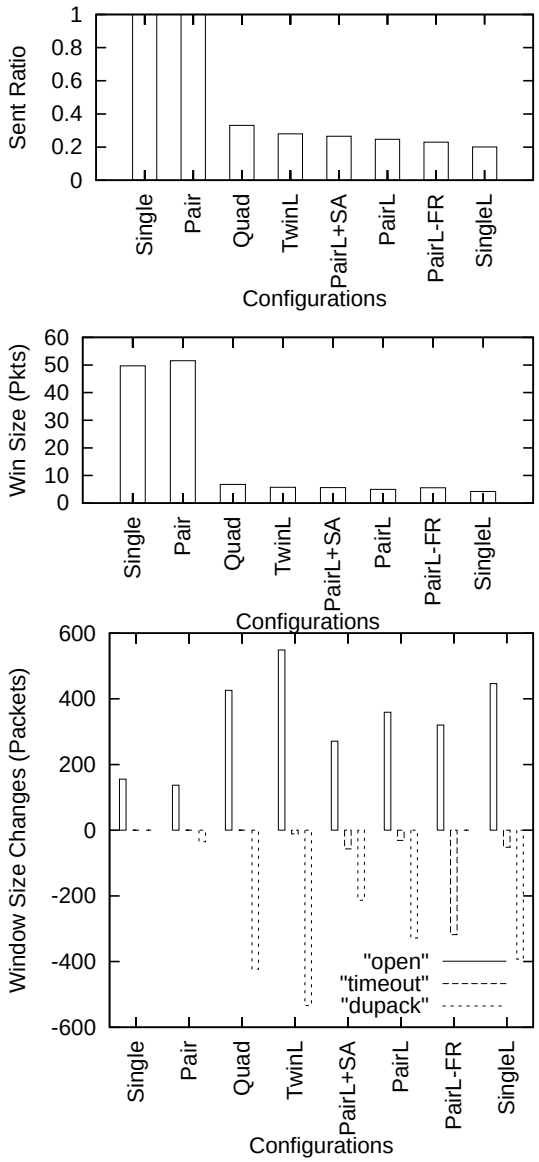


Figure 1: Results of the single-flow simulations. "Sent ratio" is the TCP sending bandwidth divided by the nominal rate of the traffic generator. "Window size" is the mean of the congestion window size in packets, where the window size is sampled whenever it is consulted to determine whether a packet should be sent. In the chart of "window size changes", the "open" numbers are the accumulated amount in packets that the congestion window size is increased after a new ack is received; the "timeout" numbers are the accumulated amount that the window size is decreased after a retransmission timeout; and the "dupack" numbers are the accumulated amount that the window size is decreased after 3 duplicate acks have arrived.

succeeding packet is forwarded to the next link from the candidate set in a round robin fashion. Unless otherwise noted, fast retransmit is turned on and selective ack is turned off in TCP.

In the real world, other factors, such as traffic pattern and over provisioning, may magnify or minimize the impact of path oscillation. Therefore, the quantitative results of this simulation reflect the impact of path oscillation alone, rather than the overall impact of multipath networking.

Figure 1 shows the results. The following are our key observations (the configurations following each bullet are the ones on which the observation was made):

- **Pair vs. Single**: Two links with varied latency but no packet loss do *not* have a negative impact on TCP bandwidth. Packets reordered in pairs (e.g. received in the sequence 2, 1, 4, 3, 6, 5, ...) will not generate enough duplicate acks to invoke fast retransmit.
- **Quad vs. Pair**: Four links with varied latency and no packet loss reduce TCP bandwidth to 33% of the nominal rate, because packets reordered in groups of four generate enough duplicate acks to cause the congestion window to shrink.
- **TwinL, PairL vs. Pair**: Two links with a lossy link and with/without varied latency reduce TCP bandwidth to 25/28%, respectively.
- **PairL+SA vs. PairL**: SACK reduces duplicate acks, but slightly increases the frequency of retransmission timeout. The improvement in TCP bandwidth or window size is insignificant.
- **PairL-FR vs. PairL**: Turning off fast retransmit prevents duplicate acks from shrinking the window, but gives rise to retransmission timeout. The window size is slightly increased, but the bandwidth is slightly decreased, because the TCP sender has to wait longer to retransmit a lost packet.
- **TwinL, PairL+SA, PairL, PairL-FR vs. SingleL**: The naive load balancing scheme (i.e. round robin) for multipath networks produces TCP bandwidth close to that of the highest-loss path, despite the available bandwidth in lower-loss paths.

In summary, varied packet loss rates as well as varied delays on multiple paths could potentially have a significant impact on TCP congestion window size and sending bandwidth, even with fast retransmit and selective ack turned on. Therefore, we were convinced that there is room for improvement in the flow partitioning schemes for multipath networks.

3 Dial-controlled hash

We propose a flow partitioning scheme called *dial-controlled hash* (DCH) that allows fine grained load balancing and minimizes path oscillation in multipath networks. DCH works on a *per-hop* basis and can be deployed independently at each router. Given multiple next hops (or links) to the same destination at a router, DCH strives to assign packets of the same *flow* (defined by source/destination addresses/ports) to the same next hop, while maintaining load balance across links. As long as each router along the path preserves the next hop for each flow, the end-to-end path can be effectively preserved for the flow. No coordination among routers is necessary in the per-hop scheme.

3.1 The simple case: two links

Let us first consider the simple case where there are exactly two links to a destination at a router. For each packet addressed to the destination, the router first hashes its *flow address* (e.g. the tuple <source IP address, source port number, destination IP address, destination port number>) to an integer $hash$ in the range $[0..MaxHash)$, using a light-weight hash function such as a *universal* hash function [1]. Then the router uses a variable $pointer$ in the range $[0..MaxHash]$ to determine which link to forward the packet to. If $hash < pointer$, then the packet goes to link 1; otherwise, it goes to link 2. This is analogous to a dial labeled with the range of hash values and a pointer that divides the dial into two slices, one for each link. See Figure 2.

The value of $pointer$ is computed periodically or on demand, based on the load balancing policy in the router and the measured loads on the two links. The load condition can reflect bandwidth, delay or any other desired metric. The router keeps track of the recent load on each link. This involves maintaining a variable per link, e.g. the *exponential weighted moving average* (EWMA) of load, and updating the variable after each packet is fowarded; such bookkeeping has little space or compu-

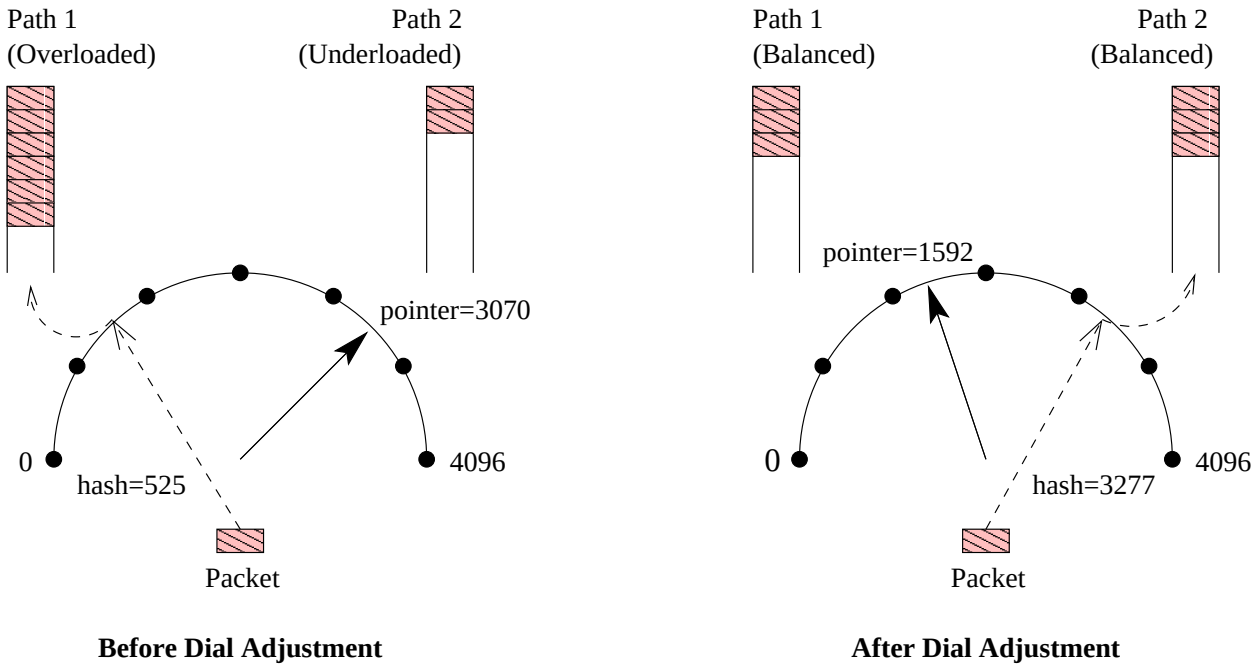


Figure 2: Dial-controlled hash for two links.

tation overhead. Given the load balancing policy in the router, the desired load change for each link can be computed. If it exceeds a certain threshold, the pointer will be adjusted to balance the load. The following is the pseudocode for adjusting the pointer, given the current link loads and the desired load changes.

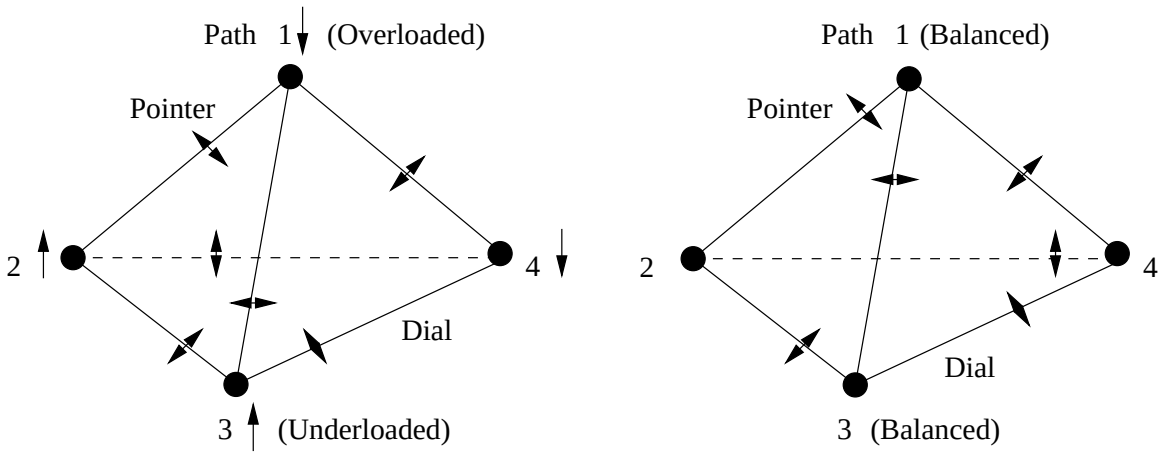
Procedure AdjustPointer(DIAL dial, METRIC load[2], METRIC dLoad[2])

```
if dLoad[1] < 0 < dLoad[2] then
   # link 1 overloaded and link 2 underloaded
   INT units = HashUnitsOnThisSide(dial.pointer, 1)
   METRIC weight = load[1]/units
   INT dUnits[2] = {dLoad[1]/weight,
   dLoad[2]/weight}
   INT delta = min(-dUnits[0], dUnits[1], units)
   dial.pointer -= delta
   dLoad[1] += delta*weight
   dLoad[2] -= delta*weight
end if
if dLoad[2] < 0 < dLoad[1] then
   # link 2 overloaded and link 1 underloaded
   # do the mirror operation
   ...
end if
# otherwise, pointer is not adjusted
```

The algorithm basically determines the number of hash units to move from the overloaded link to the underloaded link, based on the average amount of load in each unit ("weight"), the desired load change on each link ("dLoad"), and the maximum possible movement of the pointer ("units"). It also updates the remaining desired load changes for future use. (The purpose of this update will become obvious in Section 3.2.)

The parameter $MaxHash$ determines the granularity of load balancing, i.e. at least $\frac{1}{MaxHash}$ of the flows will switch links if the pointer is adjusted. Therefore, $MaxHash$ plays the same role as the number of buckets in the bucket-based hash scheme (Sections 1 and 4). However, the value of $MaxHash$ does *not* affect the amount of space required in the router for state maintenance, since the router maintains a state per dial (i.e. $pointer$), not per hash unit. The only cost for a large $MaxHash$ value is the number of bits that the hash function needs to generate for each packet.

Obviously, this DCH scheme has all the desirable properties (Section 1) for flow partitioning in dualpath networks.



Before Dial Adjustments **After Dial Adjustments**

Figure 3: Monotonic dial adjustments for multipath networks (with $n = 4$ links). In the example shown in the figure, links 1 and 4 are initially overloaded while links 2 and 3 are underloaded. After the dial adjustments, the pointers on dials $< 1, 2 >, < 1, 3 >, < 4, 2 >$ and $< 4, 3 >$ are moved closer to links 1 or 4, while the pointers on dials $< 1, 4 >$ and $< 2, 3 >$ remain in the old positions. Therefore, traffic is moved from links 1 and 4 to links 2 and 3 monotonically.

3.2 The general case: multiple links

It is, however, not straightforward to apply the dual-path version of DCH to general multipath networks. For $n(n > 2)$ links, if we simply divide the range of hashes $[0..MaxHash)$ into n segments with $n-1$ pointers, we can no longer guarantee that only a minimal amount of traffic will switch links when pointers are adjusted. It can be proved that the amount of traffic that switches links can be minimized if and only if the changes in link assignment are *monotonic*, i.e. each link either gains traffic or loses traffic, but not both. In the naive extension to DCH described above, traffic can be moved from an overloaded link to its two *neighbor links* only, by moving two pointers away from the overloaded link. If the neighbor links become overloaded as a result, other pointers need to be adjusted, resulting in a cascading effect. Therefore, more traffic than necessary will have to switch links.

We observe that, in order to make monotonic changes, we need to be able to move traffic between *any* two links. Therefore, we designed the following hashing scheme for general multipath networks. We represent each link as a *vertex* in space, draw an *edge* between every pair of vertices, and maintain a dial on each edge. For each packet, we use the high-order bits of its hash value to assign it to one of the edges, and use the low-order bits to position it on the corresponding dial. Then we use the pointer on the corresponding dial to determine which vertex (or link) to forward the packet to. Figure 3 illustrates the structure of dials.

At some routers, there might be a large difference in the capacity of different links to the same destination. If flows are evenly distributed to edges using high-order hash bits, then any single link cannot get more than $\frac{2}{n}$ of total flows even if the pointers are all pushed to the far ends from this link. This problem can be solved by *weighted* assignment of flows to edges. Let c_i be the capacity of link $i(1 \leq i \leq n)$, and $C = \sum_{i=1}^{n} c_i$. The portion of flows that are assigned to the edge between link i and j is then $\frac{c_i+c_j}{(n-1)\times C}$. The assignment can be determined once, and stored in a table with n entries for repeated lookups. This mechanism, however, is not included in the simulations reported in this paper (Section 5), because the link capacity difference in those simulations did not cause such a load balance problem.

Given the current load and desired load change on each link, we use the following algorithm to determine the monotonic pointer adjustments.

Procedure AdjustPointers(DIAL dials[n,n], METRIC load[n], METRIC dLoad[n])

for each overloaded link O in ascending order of dLoad[O] **do**
  LINK_SET neighbors = OtherEndsOfDials(dials, O)
  for each underloaded link U in neighbors in descending order of dLoad[U] **do**
    AdjustPointer(dials[O,U], {load[O],load[U]}, {dLoad[O],dLoad[U]})
  end for
end for

In the version of AdjustPointer() for multiple links, the variable "units" is calculated as the sum of units on the near side of the pointers on all dials attached to the overloaded link.

The algorithm adjusts the dial between an overloaded link and an underloaded link one at a time, and updates the remaining desired load changes accordingly. It does *not* adjust the dials between overloaded links or between underloaded links. There is a chance that some adjustment may not be successful because the pointer has already been pushed to one end. Therefore, the order of adjustment is significant. We uses a greedy strategy to maximize the opportunity to move traffic from the most loaded links to the least loaded ones, i.e. we move traffic from overloaded links in ascending order of desired load changes (which are negative), and move traffic to underloaded neighbor links in ascending order of desired load changes (which are positive).

In summary, DCH has the four desirable properties of a flow partitioning scheme (Section 1):

- It allows traffic to be moved across links for load balancing by hash unit, which can be made arbitrarily small without requiring additional space for state maintenance.
- It preserves the path for each traffic flow as long as load balancing permits.
- In case of unbalanced load, it moves traffic only from overloaded links to underloaded links; therefore, the movement is monotonic and path oscillation is minimized.
- It does not require per-flow state maintenance or packet tagging.

3.3 Fairness

A potential problem of DCH is unfairness. Flows with addresses that are hashed to values close to a bad vertex (e.g. a link with high loss rate) tend to always receive low quality of service. A simple fix to this problem is to periodically change the parameters of the hash function so that flows can be hashed to different links at different times. However, this will increase path oscillation for all flows.

We observe that the bandwidth of a TCP connection will likely be reduced when it switches from a lower-loss path to a higher-loss path, but not necessarily the other way around. We also observe that a short-lived connection may be terminated before its congestion window size grows above 1 or before it has a chance to consume the available network bandwidth. Therefore, we believe that bandwidth can be overall better utilized if TCP connections move to lower-loss paths as they age.

We designed a fairness mechanism, called *path rotation*, based on the observations above. In a router, the n links for a destination are numbered in descending order of their long-term average loss rates. Since the long-term (e.g. daily) average loss rate of a link is reasonably stable, we rely on external information, e.g. *Service Level Agreement* (SLA) with the network provider, or simply network managers' knowledge, for sorting the links, rather than attempt to actively measure the loss rates.

We use the same hashing scheme as described in Section 3.2, except that each vertex in the dial structure no longer corresponds to a fixed link. Instead, the link $L(0 \leq L < n)$ for a vertex $V(0 \leq V < n)$ is determined by $L = (V+k)\%n$, where k is a non-negative integer that increments by 1 at a certain interval I. Whenever k increases, roughly $\frac{1}{n}$ of the flows switch from link $n-1$ (the lowest-loss link) to link 0 (the highest-loss link), while roughly $\frac{n-1}{n}$ of the flows switch from a higher-loss link to a lower-loss one.

The interval I is assigned a random value between 60 and 90 seconds every time k increments. The 60-90 range is based on the statistics that 81%/84% of TCP connections last less than 60/90 seconds [8]. This range can and should be changed for connections with different life spans. This way, most TCP connections will remain in the same path during its life time, but flows with any addresses have an equal opportunity to use any path.

When the relative loss rates of links change, the mapping from vertices to links will change, which will likely result in path changes for most flows. However, we do not expect it to have a noticeable impact in practice because changes in long-term link characteristics are infrequent.

3.4 Other issues

In some extreme conditions, the link determined by DCH may not be able to accept any packets. For example, a link can go down, or the traffic hashed to a certain edge is so bursty that the links on both ends of the edge are congested. In the cases where a link is temporarily unavailable, we simply *redirect* the packets from that link to another one that is available.

In cases where links are permanently added or removed, the dial structure needs to be updated accordingly, e.g. some dials need to be added or removed and the pointers need to be reset. Although this will cause most flows to switch paths, we do not expect it to be a big problem in practice because permanent addition or removal of links are rare.

4 Bucket-based hash

In this section, we study another hash-based scheme, the *bucket-based hash* (BBH), and compares its space and computation overhead to that of DCH. As introduced in Section 1, BBH is a fairly straightforward improvement over a brute-force scheme.

Like DCH, BBH first hashes each packet to an integer in a given range, using a universal hash function. Unlike DCH, BBH maps each hash integer to a link by first mapping it to a *bucket* and then looking up the bucket in a table. The table records the assignment of buckets to links and hence has an entry per bucket. The number of buckets m needs to be larger than the number of links n but smaller than the number of flows in order for BBH to be effective.

The per-packet computation of both DCH and BBH involves a light-weight hash computation and a lookup in a data structure, i.e. a dial structure of size $\frac{n\times(n-1)}{2}$ for DCH and a bucket table of size m for BBH.

In case of unbalanced load, BBH uses the following algorithm to monotonically move buckets across links, given the current load and desired load change on each link.

Procedure MoveBuckets(BUCKET buckets[m], METRIC load[n], METRIC dLoad[n])

```
BUCKET_SET orphans = empty
for each bucket i do
  link = buckets[i].link
  if link without bucket i is overloaded then
    # move bucket i away from link
    orphans = orphans + i
  end if
end for
for each bucket i in orphans in descending order of
load do
  link = LinkWithMostDesiredChange(dLoad)
  if link with bucket i is underloaded then
    buckets[i].link = link
  end if
end for
```

The computation cost for load rebalancing is $O(m \times \log m + m \times n)$ for BBH and $O(n^2 \times \log n)$ for DCH (Section 3.2). If m is chosen to be $O(n^2)$, then the cost for BBH is $O(n^3)$.

Assume that both DCH and BBH hash each packet to an integer in the range $[0..MaxHash)$ in the first step. DCH can maintain load balance at the granularity of $\frac{1}{MaxHash}$, at the cost of keeping a dial structure of size $\frac{n\times(n-1)}{2}$ per destination. If it uses m buckets, where $n < m \leq MaxHash$, BBH can maintain load balance at the granularity of $\frac{1}{m}$, at the cost of keeping a bucket table of size m per destination. One can improve the granularity of DCH by increasing $MaxHash$, without increasing the space requirement for DCH. However, the improvement in granularity for BBH, i.e. larger m, always comes with increased space requirement.

The difference in space requirement of DCH and BBH is magnified by the number of destinations at a router, which is often a large number in the real world. In cases where different destinations have the same set of next hops, the space requirement for both DCH and BBH can be reduced by sharing the same dial structure or bucket table among those destinations.

In summary, with $n^2 \leq m \leq MaxHash$, BBH maintains load balance at a coarser granularity than DCH, but has more computation and space overhead than DCH.

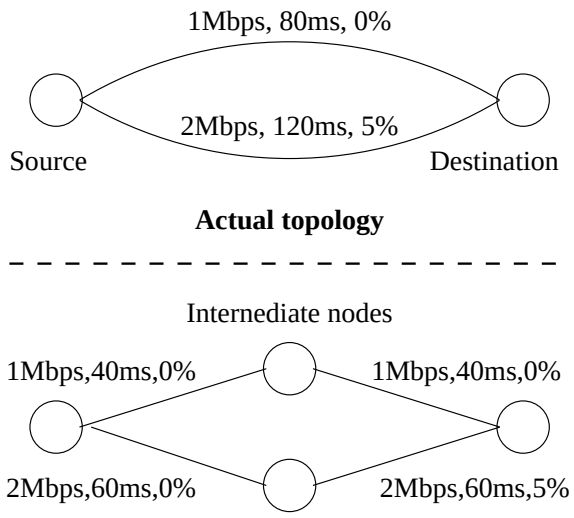


Figure 4: Simulating a parallel link with two single links and an intermediate node in *ns-2*. The labels on links are bandwidth, latency and loss rate.

5 Simulations

In this section, we present our simulation results that help answer the following questions about DCH:

1. How much does the scheme reduce in path oscillation?
2. How does the reduction in path oscillation, if any, translate to improvement in end-to-end TCP performance?
3. How does the scheme compare to alternatives, such as BBH?
4. How does the fairness mechanism (i.e. path rotation) affect the performance of the scheme?

5.1 Common setup

We used *ns-2*, a well-known network simulator, for our study. The following settings are common in all the simulations reported in this paper, including the ones in Section 2.2.

We use traffic generators with Pareto On/Off distribution and the Tahoe implementation [3] of TCP, with one traffic generator for each TCP connection. The Pareto shape parameter is 1.5. The packet size is 1000 bytes. Each link has a tail-drop queue, with maximum queuing delay equal to the link latency or the transmission time of 40 packets, whichever is larger.

Links	1	2	3	4
Bandwidth (Mbps)	38.4	28.8	19.2	9.6
Latency (ms)	134	112	88	66
Loss rate	8%	6%	4%	2%

Table 2: Configuration of the parallel link topology.

We consider the loss rate of a link to be independent of its bandwidth, because packet loss can be caused by physical transmission errors, or by congestion on remote physical segments of a logical link. We use the error model in *ns-2* to simulate packet loss at the given packet error rates, rather than use queue drops as the sole source of packet loss.

Since *ns-2* does not support parallel links between nodes, we simulate each parallel link with two single links and an *intermediate node*, as illustrated in Figure 4. We set the queue limit in the intermediate nodes to be sufficiently high so that they never drop packets. We have checked all simulation results and verified that there were no packet drops at the intermediate nodes. Since the input bandwidth to each intermediate node is no higher than the output bandwidth at any time, there is no queuing delay at the intermediate nodes.

5.2 Specific setup

The following settings are specific to the simulations reported in this section.

We simulated DCH and alternatives in two different configurations: 1) two routers connected by four parallel links, with randomized synthetic workload, and 2) a corporate intranet that has redundant links between a subset of its core routers, with traced-based workload. The following settings are common in both configurations.

Each traffic generator in our simulations has its own nominal rate, starting and ending times. The starting and ending times are uniformly randomized in the simulation duration (with the starting time no later than the ending time). Redundant links between the same nodes are assigned varied bandwidth, latency and packet loss rates. We run the link state routing protocol in *ns-2* with

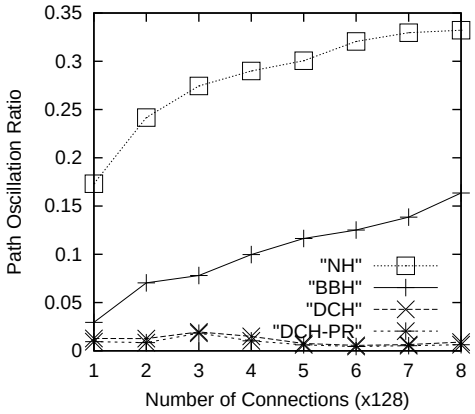


Figure 5: path oscillation ratio on the parallel link topology.

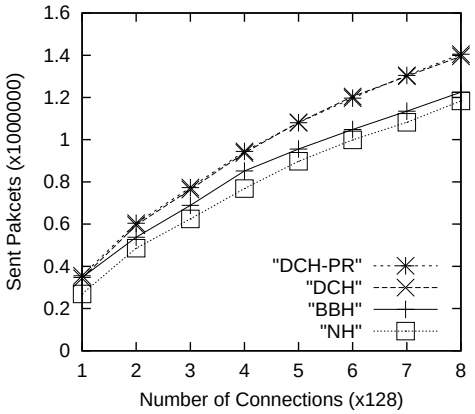


Figure 6: TCP sent packets on the parallel link topology.

a constant cost of 10 assigned to each link. The routing protocol produces multiple routes of equal cost, if available, to each destination.

We compare the following four flow partitioning schemes:

- **Dial-controlled hash (DCH)**: the scheme proposed in Section 3. For n links, it maintains $\frac{n\times(n-1)}{2}$ dials. It hashes each packet with a 16-bit universal hash function to a position on one of the dials.
- **Dial-controlled hash with path rotation (DCH-PR)**: the fairness mechanism described in Section 3.3 is added to DCH.
- **Bucket-based hash (BBH)**: the scheme discussed in Section 4. For n links, it maintains n^2 buckets so that on average n buckets can be assigned to each link. It hashes each packet with a 16-bit universal hash function to one of the buckets.
- **No hash (NH)**: a pure load balancing scheme that forwards each packet to the currently lowest-loss link that has available bandwidth.

The four schemes above are applied to the same load balancing policy, a *biased* load balancing policy. It always attempts to utilize a lower-loss link up to 95% of its capacity before it transmits any packets to a higher-loss link.

We implemented DCH, BBH and NH in a sub class of *multipath classifier* (a traffic classification module that supports multiple routes to the same destination) in *ns-2*. We added a few extensions to *ns-2* to measure end-to-end flow metrics (e.g. bandwidth, delay, delay variation, out-of-order packets, estimated round trip time, etc.) and to record TCP congestion window activities (e.g. open, close by timeout, close by duplicate acks, etc.). We present in this section the *path oscillation counts* and *sent packet counts* in TCP as the main metrics. When a packet is transmitted on a different path from the last packet in the same connection, we count it as a path oscillation. A sent packet in TCP is one that is sent to the network by a TCP agent, not one generated by the application or traffic generator.

5.3 Results on parallel links

We first run a set of simulations on a simple topology in which two routers are connected by four parallel links. Table 2 shows the parameters in configuration. Workload on this topology is a variable number of traffic generators with nominal rates between 160 Kbps and 6 Mbps.

Figure 5 shows the total path oscillation counts in all connections, divided by the total number of sent packets in TCP. Both DCH and DCH-PR have very low (<0.02) path oscillation ratio, regardless of the workload. BBH and NH have 2-18 times and 13-37 times higher path oscillation ratios than the DCH schemes, respectively.

The path oscillation ratio increases with the workload (i.e. number of connections) in both NH and BBH. With more traffic, more links are pushed to their capacity limits and cause workload to be rebalanced more frequently.

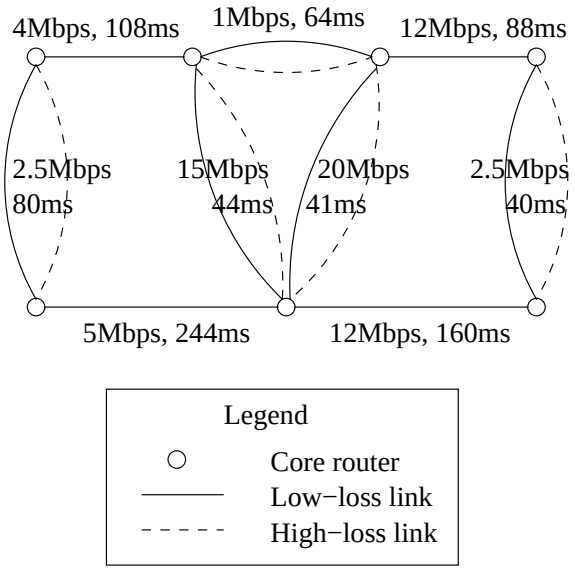


Figure 7: Topology of a corporate intranet. For each redundant link pair, the bandwidth and latency labels are the *total* bandwidth and *average* latency of the two links, respectively.

The difference between BBH and DCH comes from the frequent *redirections* that BBH experiences. The redirection in BBH is similar to that in DCH (Section 3.4). In BBH, an entire bucket of connections (roughly $\frac{1}{16}$ of total traffic in this case) must be moved from its old link to a new one in order to rebalance the load; if the bucket to be moved contains so much traffic that it will overload the new link, BBH takes the conservative measure by leaving it with the old link. When the queuing buffer for the old link is filled, BBH temporarily redirects packets to other links without changing the bucket assignment. In DCH and DCH-PR, traffic can be moved in the granularity of hash unit (roughly $\frac{1}{4096}$ in this case); therefore, load balance can be maintained more precisely.

Figure 6 shows the total number of sent packets in each scheme. Both DCH and DCH-PR improve TCP sending bandwidth over BBH by up to 15% and over NH by up to 20%. The improvement by the DCH schemes becomes more significant as the number of connections increases.

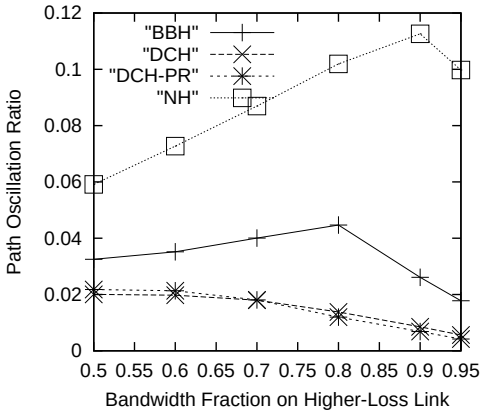


Figure 8: path oscillation ratio on the corporate intranet.

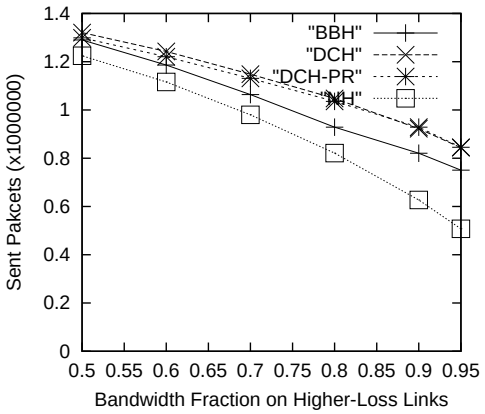


Figure 9: Sent packets on the corporate intranet.

5.4 Results on a corporate intranet

We simulated the same schemes on a corporate intranet[1] to answer a "what-if" question for the corporate network planning: what improvement on price-performance ratio can a company achieve if it adopts a cost saving strategy in which it purchases bandwidth from a cheaper but lower-quality provider (such as virtual private networks over the Internet) and uses it in addition to a more expensive and reliable network (such as leased lines and ATM links)?

Figure 7 shows the high level topology of the corporate intranet. For each redundant link pair, we vary the portion of total bandwidth that is purchased in the higher-loss network (presumably at a lower price) and measure the impact of such variation on TCP perfor-

[1]The details on the intranet topology and traffic traces in this paper are limited for confidentiality reasons.

mance. The following parameters are artificially set in the "what-if" simulations and do not necessarily reflect their actual values. The latency of the lower-loss link in a pair is set to the average latency times 0.8, while the latency of the higher-loss one is set to average times 1.2. The packet loss rate of each single or lower-loss link is set to its latency (in ms) divided by 50, while the loss rate of each higher-loss link is set to its latency divided by 12.5, i.e. roughly 4 times more than that of the other link in the same pair. Therefore, the simulation results are intended to reflect what would happen if the company adopted such a cost saving strategy, rather than what happens in the actual corporate network that the company is using.

The workload in this set of simulations is based on an hour worth of actual traffic on the intranet, reported by a network monitoring tool called "NetScout". NetScout RMON2 probes are installed on selected locations of the intranet and monitor traffic that goes through their locations. The reports we use for the simulations have an entry for each traffic flow in the following format: <source IP address - destination IP address, maximum in rate (bps), maximum out rate (bps), in bytes, out bytes, total bytes>. The terms "in" and "out" refer to the traffic directions between the source and destination.

Since the flows in the reports are classified by IP addresses, but not port numbers, we expect that each flow contains one or more TCP connections. (Statistics show that TCP accounts for the majority of traffic. Therefore, we ignore non-TCP traffic in the simulations.) In the simulations, we made a simplified approximation of the TCP traffic by creating 4 Pareto traffic generator of the same nominal rate over TCP connections for each flow in the NetScout reports. We need to set the nominal rate of each traffic generator to the rate at which the application wishes to send data. However, the rates in the NetScout reports are the actual rates, which are a result of the TCP congestion control mechanism. Therefore, we scale the recorded rates with a sufficiently large factor (2 to 32) and use the scaled rates as the nominal rates instead. We show the results with the scale factor 16 in this paper. The final workload in the simulations is 692 TCP connections with nominal rates between 1 Kbps and 10 Mbps, including 528 connections that traverse at least one redundant link pair.

Figure 8 shows the average path oscillation ratio of the TCP connections as a function of the fraction of bandwidth on higher-loss links. Figure 9 shows the total number of sent packets in TCP. In these two figures, the 164 TCP connections that do not traverse any redundant link pairs are excluded, since they are not affected by the higher-loss link bandwidth variation or by the flow partitioning schemes. They are included in the simulations, however, to generate a more realistic workload.

The path oscillation ratios of both NH and BBH first increase as a larger portion of the bandwidth is purchased on the higher-loss link, and start to decrease when the TCP sending bandwidth drops below a threshold. Below that threshold, the majority of traffic goes to the lower-loss links and leaves most higher-loss links idle. This behavior has an interesting consequence: without a good flow partitioning scheme for multi-path networks, there is an upper limit on how much a company can save by purchasing more bandwidth on cheaper, lower-quality networks; beyond that limit, the TCP performance will be nearly as low as if no bandwidth were purchased at all on the lower-quality networks.

On the other hand, with the DCH schemes, TCP performance degrades more gracefully with increased bandwidth fraction on less expensive links. Both DCH and DCH-PR reduce path oscillation ratio by a factor of 1.6-3.2 over BBH and by a factor of 3-13 over NH. The DCH schemes improve TCP sending bandwidth over BBH by up to 13% and over NH by up to 66%.

5.5 Discussions

During our simulations, we also learned some limitations about DCH and about dynamic flow partitioning schemes in general. While DCH performs better than alternatives in all of our simulations, its improvement is marginal or negligible in the following cases:

- If the network resource is over provisioned, a static partitioning scheme can work just fine.
- Static partitioning can also work well for workloads that have constant sending rate per flow.
- For applications with such a small sending rate that they do not need a TCP congestion window size larger than 1, a (biased) load balancing scheme can work reasonably well.
- If the number of flows is no greater than the number of links, hash-based schemes would not help.

- If all links are highly lossy, then none of the simulated schemes can make any difference, because TCP will always be in the slow start mode with a window size close to 1.

On the other hand, the application of DCH does not need to be restricted to flow partitioning in multipath networks. In fact, it can be applied (with certain extensions) in many contexts that require a dynamic partitioning of objects and desire minimal changes when a re-partitioning is necessary. For example, we expect it to be useful in content distribution networks and peer-to-peer information sharing systems as well.

The bandwidth difference in DCH and BBH is less than dramatic in the simulations reported in this paper. However, this should be viewed with the awareness that BBH in those simulations has roughly twice of the space requirement and three times of the computation overhead of DCH (Section 4).

6 Related work

Multipath routing, also called alternate-path routing, which generates multiple paths from a router to a destination, has gained increasing attention as traffic on the wide area networks exhibits growing and dynamically changing demand [12] [11] [10]. Multipath routing can potentially improve the utilization of network resources by load balancing. DCH, or dynamic flow partitioning schemes in general, work on a different aspect of multipath networking: given the routes generated by a multipath routing protocol, a partitioning scheme like DCH determines the next hop for each packet at each router independently. The goal of DCH is to best utilize the bandwidth made available by the multipath routing protocols.

Traffic dispersion [2] is a different way of utilizing multiple routes, often on a different network layer (e.g. ATM networks vs. IP networks). Like DCH, it makes a route selection per packet or per batch of packets. Unlike DCH, it intentionally spreads packets (of the same flow) across multiple disjoint paths for load balance or redundancy purposes. It typically uses a buffer at the receiver that resequences out-of-order packets, which requires additional buffer space and increases packet delay. In *redundant* dispersion, it is possible to reconstruct out-of-order packets using an error-correcting code, at the expense of consuming more network bandwidth.

In the LIRA network [9], each TCP connection is bound to a fixed path at establishment time and packets are tagged with the encoded hops in the fixed path at the source router. It requires per-flow state maintenance at edge routers. It binds a new flow based on the selection probability of each path, and never changes its path once a flow is bound. It makes constant adjustment to path selection probabilities in case of unbalanced load. In contrast, DCH does not maintain per-flow state, is more flexible in route selection, and makes more adaptive adjustments for load balancing.

A consistent hash function [4] is one that changes minimally as the range of hash values changes. Both consistent hash and DCH have the property of monotonic changes. However, consistent hash was designed to accommodate the dynamically changing hash range, such as the instances of content distribution servers on the Internet, while DCH is designed to accommodate the dynamically changing workload. In case of permanent addition or removal of links (which are rare in practice), the changes in DCH may not be monotonic. On the other hand, consistent hash is not directly applicable to a dynamically changing workload because it assumes that load is evenly distributed across hash units or buckets.

7 Conclusions

We quantitatively studied the impact of path oscillation on TCP performance. In addition to the well-known effects of out-of-order delivery, we observed that varied packet loss rates on multiple paths also have a significant impact on TCP bandwidth. TCP bandwidth on multiple paths with lossy links is close to that on the highest-loss path, despite the available bandwidth in lower-loss paths.

We designed a new flow partitioning scheme for multipath networks, called dial-controlled hash (DCH). DCH has the following properties: it preserves the path for each traffic flow as long as load balancing permits; in case of unbalanced load, it moves traffic from overloaded paths to underloaded paths in fine granularity and in a monotonic fashion; and it does not require per-flow state maintenance or packet tagging. We also designed a path rotation (PR) mechanism that works with DCH and ensures fairness among flows with various addresses or hash values.

We evaluated DCH with path rotation turned on and

off in two sets of simulation studies, a parallel link topology with randomized synthetic workload and a corporate intranet with trace-based workload. We compared DCH to two alternatives, a bucket-based hashing (BBH) scheme that has higher space and computation overhead than DCH, and a non-hashing, pure load balancing (NH) scheme. DCH reduces path oscillation by a factor of 1.6-18 compared to BBH and by a factor of 3-37 compared to NH. It improves TCP bandwidth over BBH by up to 15% and over NH by up to 66%. DCH reduces path oscillation more effectively than BBH because DCH is able to move traffic across paths in smaller unit. In all simulations, path rotation did not have any noticeable negative impact on TCP bandwidth.

Our simulations on the corporate intranet also have an interesting implication to how much one can save by using lower-quality networks in combination with higher-quality ones: without a good flow partitioning scheme, the TCP performance on a multipath network with a fraction of lower-quality bandwidth exceeding a certain threshold will be nearly as low as if no bandwidth were purchased at all on the lower-quality networks.

We conclude that dial-controlled hash makes it feasible to dynamically partition traffic flows in multipath networks, which requires less human intervention and achieves better resource utilization than static partitioning.

References

[1] J. L. Carter and M. N. Wegman. Universal classes of hash functions. In *Journal of Computer and System Sciences 18*, 1979.

[2] E. Gustafsson and G. Karlsson. A literature survey on traffic dispersion. *IEEE Network*, March/April 1997.

[3] V Jaccobson. Congestion avoidance and control. In *Proceedings of ACM SIGCOMM*, August 1988.

[4] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the 29th ACM Symposium on Theory of Computing*, May 1997.

[5] S. Knight, D. Weaver, D. Whipple, R. Hinden, D. Mitzel, P. Hunt, P. Higginson, M. Shand, and A. Lindem. Virtual router redundancy protocol. Technical Report RFC-2338, IETF, April 1998.

[6] T. Li, B. Cole, P. Morton, and D. Li. Cisco hot standby router protocol (hsrp). Technical Report RFC-2281, IETF, March 1998.

[7] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. Tcp selective acknowledgment options. Technical Report RFC-2018, DDN Network Information Center, Octobor 1995.

[8] V. Paxson and S. Floyd. Wide-area traffic: The failure of poisson modeling. *IEEE/ACM Transactions on Networking*, (3), June 1995.

[9] I. Stoca and H. Zhang. Lira: An approach for service differentiation in the internet. In *NOSSDAV*, 1998.

[10] X. Su and G. de Veciana. Dynamic multi-path routing: Asymptotic approximation and simulations. In *Proceedings of ACM SIGMETRICS*, June 2001.

[11] Z. Wang and J. Crowcroft. Shortest path first with emergency exits. In *Proceedings of ACM SIGCOMM*, August 1990.

[12] W. T. Zaumen and J. J. Garcia-Luna-Aceves. Loop-free multipath routing using generalized diffusing computations. In *Proceedings of IEEE INFOCOM*, March 1998.