DECOR: a Distributed Coordinated Resource Monitoring System

Shan-Hsiang Shen
University of Wisconsin, Madison

Aditya Akella
University of Wisconsin, Madison

***Abstract*—Network resources are often limited, so how to use them efficiently is an issue that arises in many important scenarios. Many recent proposals rely on a central controller to carefully orchestrate resources across multiple network locations. The central controller gathers network information and relative levels of usage of different resources and calculates optimized task allocation arrangements to maximize some global benefit. Examples of architectures that use this framework include coordinated sampling (CSAMP [1]) and redundancy elimination (SmartRE [2]). However, a centralized solution creates practical problems as it is susceptible to overload, and the controller is a single point of failure.**

In this paper, we present a distributed solution called DECOR that achieves global optimization based on local information that closes to centralized approaches in terms of performance. In DECOR, the responsibility of resource monitoring and information gathering is spread among multiple nodes; thus, no single point is overloaded. Allocation of tasks is also done in a similar distributed fashion. DECOR can easily scale up to large networks, and the partial network failures do not affect DECOR's functioning in other parts of the network.

DECOR can be applied to most of path-based applications. We describe in detail how to apply it to distributed SmartRE and implement it in the Click software router.

I. Introduction

Coordinated resource monitoring has become an important problem in large networks. Network device resources such as computation power, memory access time, storage size, and bandwidth are limited and expensive. Thus, resources must be efficiently managed to extract maximal benefit in various application scenarios [3], [2], [1], [4].

Many network solutions rely on coordinated resource monitoring. For example, redundancy elimination (RE) [2], [5], [6], [7] consumes computation power to encode and decode packets and needs memory access time to cache packets. Coordinated sampling, or CSAMP [1], is a flow monitoring system using small amounts of memory and processing on network elements to retrieve network-wide traffic features. The performance of RE and CSAMP strongly depend on effective network resource management.

The conventional approach to solving the coordinated resource monitoring problem is to use a central controller [2] [1] [8]. This controller queries network devices (e.g., routers or switches) for traffic and resource availability information. It then calculates a network-wide allocation of tasks (e.g., flow monitoring and caching responsibilities) to coordinate resources and extract maximal benefits for their respective applications. For example, SmartRE relies on a central controller to gather traffic redundancy profiles and arrange decoding responsibilities across a collection of network caches. Network resources can be used efficiently under the close monitoring of the controller.

978-1-4673-1298-1/12/$31.00 2012 IEEE

The centralized approaches rely on a central controller to make the decision of resource coordination. Network devices provide the overview of resources and traffic in a whole network. Thus, a central controller can make right resource coordination to maximize benefit. Finally, the decision will be delivered to each network device. Network devices follow the decision to deal with traffic.

However, there are some limitations to solutions with a central controller, as described below.

1) **Controller bottleneck:** Coordinated resource monitoring is an optimization problem and most solutions rely on linear programming. In large topologies, the number of constraints in linear programming is critical. Thus, computation time in a central controller becomes a bottleneck.
2) **Control message delay:** Before solving the linear program a central controller gathers all traffic profiles from the entire network. As some nodes are far away from the controller, the delay in gathering information may be long. In addition, this delay is variable, so the controller must either wait for control packets with the greatest delay or act on incomplete information.
3) **Single point of failure:** If a controller fails or gets disconnected from the system, the entire network's resource monitoring will go down and applications relying on it will be affected. In the case of SmartRE, encoded packets may no longer be decodable; hence the network as a whole becomes non-functional. Some studies add duplicate controllers to solve this problem; however, there is high overhead from keeping traffic status consistent in multiple controllers and replacing broken controllers.

This paper presents a distributed coordinated resource monitoring framework called DECOR. This solution uses nodes distributed throughout the entire network (these could be forwarding elements or caches themselves) to release the pressure on a single point. As a result, each node needs to solve a smaller scale optimization problem which consumes less computation time. In addition, control messages are exchanged locally, so control message delay is short and predictable. When a node responsible for coordinating resources is down,

selecting a replacement is easier because coordination workloads are light-weight and local.

Three main issues arise in the design of DECOR: (a) it must reach a globally optimal allocation which can achieve global benefit even without a global view of the entire network from a central optimizer, (b) it should make effective use of resources in nodes, and (c) it should converge quickly and be robust to link failures.

The basic idea of our algorithm starts with each path first requesting resources from its nodes, trying to optimize resource assignments to maximize benefit to itself. The benefit is defined differently in different applications. However, to achieve global optimization, each node in a topology also coordinates resource requests *received* from different paths, and it arranges resources to maximize the benefit it can provide to the network-wide traffic received. This resource allocation process starts at the exit node and is repeated node-for-node back to entry nodes. This approach runs through multiple iterations until an optimal allocation is found. Therefore, DECOR naturally expands optimization from a "local" to a "global" view without relying on a central controller.

We apply this algorithm to SmartRE [2], resulting in a system we called distributed SmartRE. We implement this in a Click software router [9], [10] with encoder, decoder, and control modules added to this router. We benchmark the modules and showed that the encoder reaches 485.6 Mbps and the decoder reaches 633.33 Mbps. The benchmark is run in the user space and includes the overhead of the Click module, so higher speed could be achieved if distributed SmartRE was implemented in hardware such as NetFPGA [11].

Evaluations using realistic ISP topologies and traffic traces suggest that distributed SmartRE works better than other RE methods such as Hop-by-hop and Edge-based. The performance of distribued SmartRE is close to SmartRE with a central controller. Even compared to the ideal case, distributed SmartRE can save half of the ideal redundancy.

We also apply the DECOR framework to CSAMP [1]. Our evaluations show that DECOR can help CSAMP distribute the process of range assignments. We re-do the evaluation in the CSAMP paper and show that distributed CSAMP can work much better than other sampling mechanisms and is close to the centralized variant.

DECOR provides a solution to coordinate network resources, and avoids controller bottleneck, message delay, and a single point of failure. The contributions of this paper include description of the algorithm for distributed coordinated resource monitoring (Section III) and proof of its convergence (Section V). Other contributions are the application of the DECOR framework to distributed SmartRE (Section IV) and CSAMP and experimental validation of the effectiveness and robustness of the scaled approach (Section VI and Section VII).

II. Motivating scenarios

Our goal is to build a distributed system called DECOR to monitor and use network resources efficiently. DECOR do not rely on a central controller, so it can avoid controller bottleneck, control message delay, and single point of failure problems.

Network resources are often limited and distributed among network devices. These network resources can be assigned to some particular jobs such as traffic sampling. If these network resources cannot be coordinated well, some of them may be wasted. We take traffic sampling for example. Traffic sampling relies on network devices to collect traffic profiles. Without good coordination, multiple network devices may sample the same flow and gather the same data which is a waste of resources.

DECOR can be applied to any path-based network applications which means the network can be divided into multiple paths. A path is defined as a route from a ingress node to a egress node. . DECOR optimizes resource coordination along each path, and then extend the optimization globally. The design detail will be provided in the next section.

There are five goals DECOR needs to achieve: (a) reach global optimization, (b) avoid depletion of resources, (c) achieve a converged state, (d) scalability, and (e) robustness.

To create more benefit for applications, the system needs to achieve global optimality. However, in a distributed solution, there is no single point that gathers a global view of traffic profiles and computes a globally optimal solution. Each part of a network, called an optimization unit, instead runs a separate optimization algorithm. Thus, we need a mechanism to expand local optimization to achieve a global optimal goal. Unfortunately, the resource requirements of different optimization units may conflict. For example, one optimization unit may maximize its local benefit by using computation resources also required by another optimization unit.

Network resources are often limited. For example, each node processes a maximum number of packets per second. Depleting these resources will cause nodes to drop packets or information they cannot handle. Because our design does not have a central controller to globally monitor resource usage, it needs to ensure that network resources are not depleted in each node and link, and optimization units coordinate with each other.

As a distributed solution, the system adjusts resource arrangement to increase global benefit. If the system cannot converge, decisions oscillate ,and the system repeatedly recomputes the arrangement, wasting computation power.

With the growing scale of networks, the amount of resources that need coordination is increasing rapidly, which increases the time required for a centralized system to solve network-wide optimization problems. For example, CSAMP shows that the time to compute the optimal sampling manifest for the router-level NTT graph is 44.5 seconds [1]. Control message delay is also a problem in large networks. The number of hops from each node to a controller increases with growing network scale. DECOR scales down this optimization problem to a "path" level, where a "path" refers to a ingress-egress pair of routers in the network.

Resource monitoring is expected to be robust enough for

systems to work even when particular nodes are down or disconnected. If a system relies on a central controller, and this controller breaks, the entire network may malfunction. Some studies add duplicate central controllers to enhance robustness; however, if these central controllers keep network status, a mechanism is needed to make the status consistent which adds extra overhead. In addition, fault recovery approaches that switch functions to backup controllers impose a high performance cost (to fail over when a primary controller is broken) and/or a high replication cost. DECOR runs its algorithm locally; thus, a broken part of a network will not affect other parts.

III. Design

The basic idea of DECOR is to divide a network into smaller pieces called optimization units. We achieve local optimization in each optimization unit, and then extend to global optimization.

A network is divided into optimization units according to ingress-egress pairs, and each unit calculates optimized resource assignment locally. In this paper, we assume that flows belonging to the same ingress-egress pair follow the same hops in a network, and these ingress-egress pairs are called "paths". Paths form the optimization units in our framework.

A path is composed of network devices called nodes. Each node along a path can provide some resources; therefore, to create more benefit, these resources need to be allocated across paths. For example, in CSAMP [1], each node is assigned a hash range to decide how to use its resources to monitor traffic for a path. In DECOR, an egress node is responsible for arranging these resources of the paths it belongs to. DECOR also use hash ranges of packet headers to assign each node the fraction of traffic it is responsible for.

DECOR can be divided into two parts: (a)**hash range arrangement**: assign hash range to nodes along a path to achieve local optimization, and (b) **quota distribution**: nodes assign resource quota to paths going through them to avoid the conflict of local optimizations.

To make the right decision of resource arrangement, necessary information needs to be passed around nodes. Depending on the application, this information can include traffic features, redundancy profiles, or available resources. Control packets called HELLO packets are sent by an ingress node to gather the necessary information along a path and accepted resource request from each node in this path. HELLO packets finally are received by an egress node.

Hash range arrangement: The egress node decides on an optimized resource arrangement according to the information provided by HELLO packets. This resource arrangement decision represented by hash ranges is passed to nodes along the path, letting them know how to use their resources. Another kind of control packets called ACK is sent back into the network by the engress node to inform the whole path of the resource allocation.

Quota distribution: Each node can belong to multiple optimization units (paths), so conflict may occur when all units try to maximize their benefit and request resources from a node. To achieve global optimization, a node in conflict decides the amount of its resources each optimization unit can use according to the benefit each unit provides to the network *as a whole*. This decision is inserted into HELLO packets. The benefit can be defined differently in different application. SmartRE [2] defines benefit as the amount of traffic that can be reduced.

To give a more concrete idea of how a distributed coordinated resource monitoring system works, we apply it to SmartRE and build a system called distributed SmartRE. This is described in the next section.

IV. Distributed SmartRE

SmartRE is a redundancy elimination (RE) system. Unlike other RE system, SmartRE distributes decoding responsibility among multiple downstream nodes. There is a central controller to monitor and rearrange the decoding responsibility according to traffic profiles collected by nodes in the system and the amount of resources which can be provided by the nodes. In this section, we apply DECOR to SmartRE called distributed Smart without central controller.

Despite the lack of a central controller, distributed SmartRE maximizes benefit by arranging decoding responsibility among nodes according to redundancy and traffic patterns. This section describes the basic concept, and shows an algorithm for decoding responsibility arrangement.

Data packets are incoming packets that traverse a path. These packets are encoded in ingress nodes by replacing redundancies by small shims, and then recovered by interior or egress nodes. The goal of RE is to reduce the amount of data traffic between ingress and egress nodes. This is realized by Distributed SmartRE through the use of DECOR spreads decoding responsibility across interior and egress nodes. It means shims may be recovered by different nodes.

The main goal of SmartRE is to use resources more efficiently and avoid redundant traffic. Interior and egress nodes decide which packets to decode based on the hash value of packet headers. Thus, if each Interior or egress node is assigned a hash range, decoding responsibility can be spread out appropriately.

Distributed SmartRE use DECOR to arrange the hash ranges. As described in Section III. HELLO packets gather traffic features such as the amount of traffic and number of redundancy matches collected in the ingress nodes. Interior nodes also insert the amount of available computation and memory quota resources into HELLO packets.

After receiving HELLO packets, egress nodes calculate the best assignment of hash ranges, and these hash ranges are passed to nodes along a path. By this mechanism, interior nodes know their assigned hash ranges. An ingress node has an overview of the hash range assignment and knows which encoded packets can be later decoded in some interior or egress nodes.

V. Analysis

In this section, we describe DECOR in more detail, including hash quota distribution, hash range arrangement, and proof of convergence. A path in this section is defined as an ingress-egress pair in a network. We describe the general case of DECOR and use distributed SmartRE as an example to show how it fits into a real situation.

The goal of our algorithm is to achieve the best benefit. For distributed SmartRE, this is the greatest bandwidth savings. Thus, it is critical to assign appropriate hash ranges to interior and egress nodes.

Interior nodes must determine how to distribute their allotted computation power and memory space, called quotas, among different paths. Additionally, as mentioned in Section III, egress nodes must calculate a hash range assignment for each node along a path, and interior nodes use this assignment to update their hash range.

The following section presents quota distribution and hash range assignment. In addition, a convergence issue is discussed.

A. Quota distribution

Assume that there is a new path, p. To obtain resources, path p needs to negotiate with nodes along the path by sending HELLO packets.

However, resources of the nodes are limited. For example, the maximum number of packets per second that nodes can deal with must be carefully considered. The speed of memory access also creates some limitations. In addition to resources, it is also critical to consider the benefit each path can provide is also. Thus, to avoid depleting resources, the interior node needs to decide how to distribute limited resources among paths that require them. Assuming that node r receives a HELLO packet, the following paragraphs show how r calculates quota distribution.

Interior nodes first need to figure out the unit estimated benefit that can be provided by each path per unit resource. For example, this is the amount of traffic reduced in distributed SmartRE if node r assigns each unit resource to a particular path.

Before calculating the unit estimated benefit, node r calculates the maximum possible benefit each path can provide, $B_{p,r}$. The definition of benefit depends on the application. For CSAMP, benefit is the amount of traffic a system can sample. However, in distributed SmartRE, estimated benefit can be derived from the following equation:

$$B_{p,r} = distance_{p,r} \times match_{p,q,r} \times matchlen_{p,q,r}$$

The benefit, $B_{p,r}$, is the amount of traffic in bytes that can be reduced by a particular path in node r. This depends on many factors: (1) $match_{p,q,r}$ is the number of times node r observes that the content of packets in path q matches the content of packets in path p; notice that q can be other paths or p itself; (2) $matchlen_{p,q,r}$ is the average match length, which is the traffic that can be saved from each match; and (3) $distance_{p,r}$ is the number of hops from an ingress node to node r, which is the distance encoded packets traverse.

However, different paths may require different amounts of computation power and memory access quotas. Thus, unit estimated benefit is derived from dividing maximum possible benefit by the quota needed to satisfy all requirements as in the equations below.

$$B_{M,p,r} = \frac{B_{p,r}}{N_{M,p,r}}$$
$$B_{L,p,r} = \frac{B_{p,r}}{N_{L,p,r}}$$

$B_{M,p,r}$ is the benefit per maximum memory access quota provided by path p and $B_{L,p,r}$ is the benefit per maximum computation quota provided by path p. In addition, if node r wants to serve all traffic of the path p, the path p will request both memory quota $N_{M,p,r}$ and computation quota $N_{L,p,r}$ with node r. Distributed SmartRE defines these as

$$N_{M,p,r} = v_p$$
$$N_{L,p,r} = \frac{v_p}{avgpktsize} + match_{p,q,r}$$

v_p is the volume of traffic in bytes going through this path. Because node r caches the packets that go through it, node r needs memory quota of v_p to handle the caching process. avgpktsize is the average packet size in path p, so $\frac{v_p}{avgpktsize}$ is the number of packets that are the computation power needed to cache these packets, and while $match_{p,q,r}$ is the computation power needed to scan redundancy content. Thus, the sum of these two terms is the total amount of computation power requested. In addition, because of different inter traffic redundancy patterns among nodes in a network topology, $N_{M,p,r}$ and $N_{L,p,r}$ may be different in different nodes.

After receiving HELLO packets, the interior nodes insert available quotas into them. The available quotas include unused quotas plus the quotas occupied by paths with lower $B_{M,p,r}$ and $B_{L,p,r}$. The benefits of path p are compared with the benefits of other paths consuming some quotas, and paths that provide less benefit release some quota for path p to use in the next iteration. The iteration is defined as an exchange of a HELLO packet and an ACK packet.

The equations below are used to compute available memory access quota, $M_{p,r}$, and available computation power quota, $L_{p,r}$, which are inserted into HELLO packets.

$$M_{p,r} = \sum_{i=1}^{p-1}(x_{i,r}N_{M,p,r}) + M_u$$
$$L_{p,r} = \sum_{i=1}^{p-1}(x_{i,r}N_{L,p,r}) + L_u$$

M_u is unused memory access quota and L_u is unused computation power quota. Moreover, $\sum_{i=1}^{p-1}(x_{i,r}N_{M,p,r})$ and $\sum_{i=1}^{p-1}(x_{i,r}N_{L,p,r})$ are sums of quotas that will move to path p where path 1, 2, ..., p-1 are paths with less benefit (both of less $B_{M,p,r}$ and $B_{L,p,r}$ than path p. $x_{i,r}$ is the fraction of total assigned responsibility of path i that is handled by node r.

B. Hash range arrangement

A primary job of egress nodes is to determine the hash range for each node along a path. This process is triggered by HELLO packets that provide essential information such as

traffic volume and redundancy patterns. Using this information, an egress node can calculate the hash range with linear programming.

There are some constraints to be satisfied. The memory access quota usage cannot surpass the limit of each node. Thus, the memory access constraint on each node can be modeled as follows:

$$\forall r, x_{p,r} \times N_{M,p,r} \leq M_{p,r}$$

$M_{p,r}$ is the memory access quota assigned to path p by node r. $x_{p,r}$ specifies the fraction of traffic on path p that node r deals with. Thus, $x_{p,r} \times N_{M,p,r}$ becomes the expected memory access in node r that should not surpass the memory quota. In addition, the computation power limit should also be considered. $L_{p,r}$ is the computation power quota assigned to path p:

$$\forall r, x_{p,r} \times N_{L,p,r} \leq L_{p,r}$$

The total number of calculations should not be more than the limit.

The hash ranges can be represented by $x_{p,r}$ and the total hash range should be equal to or less than 1, which is a natural constraint shown by the following equation:

$$\sum_{i=0}^{V} x_i \leq 1$$

where V is the total number of nodes along a path. The total benefit along a path can be computed as:

$$S_p = \sum_{i=0}^{V} B_{p,i} x_{p,i}$$

For path p, linear programming will try to maximize S_p and derive $x_{p,r}$.

C. New path negotiation

When the traffic of a new path joins a network with some existing paths, the new path should negotiate with interior and egress nodes for the quotas it needs. HELLO packets and ACK packets are used for this negotiation.

When an interior or egress node receives a HELLO packet from a new path, it follows this procedure:

a) Calculates the benefit of this new path using the equation shown in SectionV-A.
b) Compares the benefit of this new path to benefits of existing paths.
c) Calculates the amount of quotas this node can provide, including computation power and memory access.
d) Assigns the new path using unused quotas as well as quota used by other paths with less benefit. The value of total available quota is inserted into HELLO packets.
e) After the nodes receive ACK packets, the node adjusts the hash range to prevent paths from depleting resources, if the available quotas are still occupied by other paths.
f) In the case where a interior node receives two successive HELLO packets and the latter one belongs to a path with higher benefit, the interior node may use ACK packets to reject resources for the earlier HELLO packet, even if the resources are accepted in the HELLO packets.

D. Proof of convergence

DECOR uses control packets to arrange and adjust responsibilities, which makes stability important.

There is no convergence issue when the first path joins the system, because no other path competes with it. This first path can thus use all resources it needs. Therefore, we discuss the convergence issue only when a new path joins a network with existing paths. The flow of proof is as follows:

We calculate the change of benefit each node can provide between iterations and define this change as $\triangle B_r$. If $\triangle B_r$ is monotonically non-decreasing, the benefit B will get closer to maximal benefit $B_{MAX,r}$.

$$B_{MAX,r} = Max_{i=1}^{P}(Min(M_r B_{M,i,r}, L_r B_{L,i,r}))$$

$B_{MAX,r}$ is the maximal possible benefit when node r give all quotas to a path that can provide maximal benefit per quota. Because both memory and computation constraints need to be satisfied, we consider the minimum benefits that memory and computation quota can create.

In DECOR, each node attempts to increase benefit by assigning quotas to different paths. Thus, each iteration increases some benefit in a node. Iteration is defined as a control packet that sets hash range in a node. According to the algorithm, each node tries to satisfy the maximum amount of quota that is requested by the path with highest benefit.

Consider an iteration where a control packet belongs to a particular path p that navigates quotas in a node r, which can be an interior or an egress node. The extra hash range it requests is

$$\triangle x_{p,r} = x_{p,r} - x_{p,r}^{'}$$

$x_{p,r}$ is the total hash range that path p asks for on node r. This hash range is represented by the fraction of traffic inside path p. In addition, $x_{p,r}$ is derived from the linear programming calculated in an egress node. $x_{p,r}^{'}$ is the hash range this path obtained in a previous iteration in this node. Thus, $\triangle x_{p,r}$ is the extra range requested in this iteration.

The extra computation and memory quotas needed for this range are

$$M_{p,r} = \triangle x_{p,r} N_{M,p,r}$$
$$L_{p,r} = \triangle x_{p,r} N_{L,p,r}$$

$M_{p,r}$ is the memory quota needed, while $L_{p,r}$ is the computation quota for the extra hash range. The extra hash range is multiplied by $N_{M,p,r}$ and $N_{L,p,r}$, the total resources path p needs.

Because path p can get quotas that are owned by paths with lower benefit, the available quotas that can be used by path p in this node are

$$M_a = \sum_{i=1}^{p-1}(x_{i,r} N_{M,p,r}) + M_u$$
$$L_a = \sum_{i=1}^{p-1}(x_{i,r} N_{L,p,r}) + L_u$$
$$M_u = M_r - \sum_{i=1}^{P}(x_{i,r} N_{M,p,r})$$
$$L_u = L_r - \sum_{i=1}^{P}(x_{i,r} N_{L,p,r})$$

Where $\sum_{i=1}^{p-1}(x_{i,r} N_{M,p,r})$ and $\sum_{i=1}^{p-1}(x_{i,r} N_{L,p,r})$ are memory and computation quotas used by the paths with lower

benefit than path p. M_u and L_u are unused quotas, and M_r and L_r are total available quotas provided by node r. P is the total number of paths going through node r.

There two possible cases: case 1 is where available quotas can satisfy the amount that path p requests, and case 2 is where only part of the quota can be satisfied.

Case 1: The available quotas can satisfy the amount requested by path p. The quotas move from paths with benefit lower than path p benefit. The benefit change from the memory quota view is

$$\begin{aligned}\triangle B_r &= M_{p,r} \times B_{M,p} - \textstyle\sum_{i=1}^{k}(M_{i,r} \times B_{M,i}) \\ &> M_{p,r} \times B_{M,p} - \textstyle\sum_{i=1}^{k}(M_{i,r} \times B_{M,max}) \\ &= M_{p,r} \times B_{M,p} - (\textstyle\sum_{i=1}^{k} M_{i,r})B_{M,max} \\ &> (B_{M,p} - B_{M,max})M_{p,r} > 0\end{aligned}$$

Where path 1, 2, ..., k are selected to move quota to path p. $B_{M,p}$ is the benefit per unit memory quota path p can provide. $B_{M,max}$ is the maximum among $B_{M,i}$ noted as $B_{M,max} = Max_{i=1}^{k}(B_{M,i})$. Moreover, $\sum_{i=1}^{p-1} M_{i,r}$ is the total memory quotas move from path 1, 2, ..., k to path p, so $M_{p,r} = \sum_{i=1}^{p-1} M_{i,r}$. Because all per memory quota benefit among $B_{M,i}$ is smaller than $B_{M,p}$, $B_{M,p} > B_{M,max}$ implies that $(B_{M,p} - B_{M,max})M_{p,r} > 0$.

The computation quota equation is similar. It is obtained by replacing notations for memory quota by notations for computation quota. Because of space limitations, this is not shown in detail.

We conclude that in case 1, each iteration will increase some total benefit in node r.

Case 2: The available remaining quotas cannot satisfy the amount path p requests. This means that one or both of the memory and computation quotas cannot satisfy the requirement, and the short quota is the critical quota. Thus, path p depletes the available remaining critical quota:

If $M_a \times B_{M,p} < L_a \times B_{L,p}$, then memory space is a critical quota.

$$\begin{aligned}\triangle B_r &= M_a \times B_{M,p} - \textstyle\sum_{i=1}^{p-1}(M_{i,r} \times B_{M,i}) \\ &> M_a \times B_{M,p} - \textstyle\sum_{i=1}^{p-1}(M_{i,r} \times B_{M,max}) \\ &= M_a \times B_{M,p} - (\textstyle\sum_{i=1}^{p-1} M_{i,r})B_{M,max} \\ &= (B_{M,p} - B_{M,max})M_a > 0\end{aligned}$$

If $L_a \times B_{L,p} < M_a \times B_{M,p}$, then computation space is a critical quota. this equation is similar to the equation for memory quota and is obtained by replacing notations for memory quota by notations for computation quota. Because of space limitation, this is not shown in detail.

According to cases 1 and 2, if quota assignment changes, $\triangle B_r$ will be monotonically non-decreasing and benefit will never decrease with iterations. In addition, because the resources each node can provide are limited, there is an upper boundary for benefit, $B_{MAX,r}$. Thus, benefit will keep approaching the upper bound, implying that the range assignment process will converge.

E. Global optimization

Even without a central controller, global optimization, the achievement of maximum benefit, is still expected. We discuss two aspects of this issue.

1) Each optimization unit tries to maximize its local benefit. Conflicts between optimization units may happen if they request resources in the same nodes.
2) Each interior or an egress node resolves the conflict by giving quota to the path that creates the greatest benefit.

In summary, each path tries to reach local optimization, while interior and egress nodes limit the resources used by paths to achieve global optimization.

VI. Implementation

To show how DECOR works, we implement distributed SmartRE which is one of the instance of DECOR. First, we modify data packet headers and add two kinds of control packets.

A. Packet format

Packets can be categorized into data packets, HELLO packets, and ACK packets.

A SmartRE shim header is inserted between a transport header and a packet payload. A shim header shows how this packet is encoded and is a clue to decode this packet. It includes multiple shims composed of path ID that shows the path that original content belongs to, hash values, offset in original packets, and original length.

The payload of a HELLO packet is divided into several fields. The fields number of bytes and number of packets are traffic patterns which is filled by ingress nodes. They are followed by some fields of the number of content match, average match length, and available resources that are filled by interior nodes.

The payload of an ACK packet provides multiple hash ranges that are calculated by egress nodes and may be modified by interior nodes.

B. Implementation environment

Distributed SmartRE was implemented on a Click software router [9] [10], which is the advantage that new modules can be easily added. Distributed SmartRE was implemented on an Intel(R) Core(TM)2 Quad CPU Q6700 computer with 3 GB of memory.

C. Click modules

We created encoding, decoding, cache, and control modules. The encoding module searches each incoming packet to find redundant content and replace it with shims. It also sends packets to the cache module. The goal of the decoding module is to recover shims included in packets to restore the original content. The decoding module also calculates the hash value of the packet headers. If the value is in some predetermined range, the decoding module sends these packets to the cache module, which buffers the packets and creates a hash table to help search the content. Finally, the control

Category	Encoding module	decoding module
Mbps	485.6	633.3
Kpackets/sec	43.4	56.7

TABLE I
PERFORMANCE ON CLICK

module periodically sends HELLO and ACK control packets and rearranges the hash ranges among the nodes.

D. Capacity test

The encoder and decoder modules implemented in the Click experience some hardware and software overhead, in addition to Click overhead. In this section, we discuss how we determined the amount of traffic the modules could handle. The volume of test traffic was 1.821 GB with 1303 K packets. The packets were fed into the modules as quickly as possible, and we measured the output traffic.

Based on table I, the encoding module can encode packets as quickly as 60.7 MBytes per second, which is about 485.6Mbps and 43.4 K packets. Because the decoding module only needs to replace shims with original content for encoded packets, it can accept faster traffic than the encoding module. Decoding was measured at 79.2 Mbytes per second (633.3Mbps) with 56.7 K packets per second.

E. Correctness issue

The cache in both encoding and decoding nodes must be consistent to correctly decode data packets.

The caches in ingress nodes are divided into buckets. Each bucket corresponds to the cache in a particular interior or egress node for a particular path. Each bucket ensures that the packets buffered in the upstream nodes are also buffered in the downstream nodes. The number and the size of buckets are adjustable and depend on the number of paths, topology, and overlap matrix.

F. Recovery from route failure

When a link is broken, a path may change its route, and some interior nodes may be replaced by other nodes. Thus, to maintain correct coding in this situation, packets cached in the buckets belonging to substituted interior nodes can no longer be used to encode incoming packets. Otherwise, encoded packets cannot be decoded.

Therefore, when an ingress node notices a route is changed, buckets are replaced. The ingress node deletes the buckets belonging to the substituted interior nodes and the packets cached in those buckets. Then, the ingress node creates buckets for the new interior nodes, and Incoming packets are cached and encoded with the new buckets.

VII. EVALUATION

To show that distributed SmartRE can also perform well in a larger topology, we evaluated it on an ISP topology. More paths feed into this topology, and they share some common interior nodes. This allows us to observe how paths compete with each other for resources provided by interior nodes.

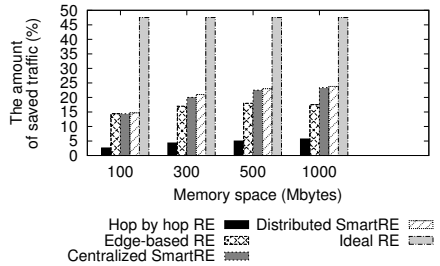

Fig. 1. The amount of saved traffic

In this section, we discuss our evaluation of the efficiency of our algorithm. We then present the convergence of the hash arrangement followed by a comparison of distributed SmartRE with other RE methods. Finally, we apply DECOR to CSAMP and evaluate the performance.

A. Distributed SmartRE

The ISP topology of Sprint was used, which included 51 nodes and 167 links. Moreover, 2GB traffic from 1569 K packets travels through the network from 15 ingress nodes to 15 egress nodes with some overlapping interior nodes. The paths compete with each other for resources within nodes. The sending period of control packets was on second.

1) Traffic reduction: Saving traffic is the goal of all RE system. Thus, we compared distributed SmartRE with hop-by-hop solution, centralized SmartRE, edge-based RE, and an ideal case.

1) Distributed SmartRE: a DECOR-based SmartRE approach. This approach uses control packet exchanges among nodes to arrange decoding responsibilities.
2) Hop-by-hop RE: nodes, including ingress, interior, and egress, encode and decode incoming packets on each hop. This approach consumes more resources.
3) Centralized SmartRE: a SmartRE approach with a centralized controller that gathers traffic and redundancy patterns and arranges decoding responsibilities among nodes.
4) Edge-based RE: uses ingress nodes to encode incoming packets and egress nodes to decode them. Interior nodes do not have any responsibilities; encoding/decoding happen in edge nodes. This approach cannot benefit by inter path redundancies.
5) The ideal case: unlimited resources in each node, including computation power, memory access, and cache size; thus, the node can figure out all possible traffic redundancies. Four cases are considered. In each, interior nodes are assigning different resources of computation power and memory space.

We vary the memory space in each node from 100MBytes to 1000 MB. Each node can access 3.52 MB of memory per secend and perform 4000 coding computations.

Figure 1 shows the percentage of traffic each solution can save. We summed the amount of total traffic that goes through all links and the traffic saved on all links.

Topology(AS#)	PoPs	#Test Flows	#Iterations
NTT(2914)	70	34	6
Level3(3356)	63	30	5
Sprint(1239)	52	26	4
GEANT	22	10	3
Internet2	11	4	3

TABLE II
THE NUMBER OF ITERATIONS NEEDED FOR CONVERGENCE

The performance of RE is affected by cache size excepting the ideal case, so distributed SmartRE can save half to one-third of the amount of traffic that the ideal case can save. However, distributed SmartRE is 18 to 63 times better than the hop-by-hop approach, especially in cases of lower resources. Distributed SmartRE also works better than edge-based RE, because distributed SmartRE can benefit from inter-path redundancy, while edge-based RE can consider only intra-path redundancy.

Therefore, we conclude that distributed SmartRE can save lots of trafic even if it does not perform as well as ideal case, and it is also superior to the hop-by-hop approach. Edge-based RE cannot benefit from inter-path redundancy, so it reduces less traffic than distributed SmartRE, especially when the cache size is large. We also can observe that even without a central controller, distributed SmartRE can work as well as centralized SmartRE.

Moreover, distributed SmartRE is less affected by limited resources than the hop-by-hop approach, which means that it can consume less resources and still perform well.

B. Convergence

Distributed SmartRE relies on control packets to adjust hash range assignments and can reach an optimized decoding responsibility assignment after several iterations. Thus, we evaluate the number of iterations needed to reach convergence. One iteration is defined as one HELLO and ACK packets exchange for each path.

To avoid the effect of traffic variance, stable traffic is used. We measure convergence time in Rocketfuel-based [12] topologies of different scale as shown in Table II. We pick up one fourth of nodes as ingress nodes and each ingress nodes start a pair of flows that provide 4 Mbytes/s and 3000 packets/s traffic with different redundancy levels. One flow includes 84 matches/s with 282 bytes average redundancy length, and the other includes 270 matches/s with 544 bytes average redundancy length. The pair of flows show some overlap hops, so they compete with each other for resources.

We compare hash range arrangement in the current iteration to the one in previous iteration. If they are consistent, we can say it have converged. Table II shows that the number of iterations needed for convergence tends to be higher in larger ISP. However, even in the largest ISP such as NTT, the iteration is acceptable. The size of NTT is about seven times larger than Internet2, but the number of its iterations for convergence is only double. Thus, convergence time is not a critical problem in DECOR.

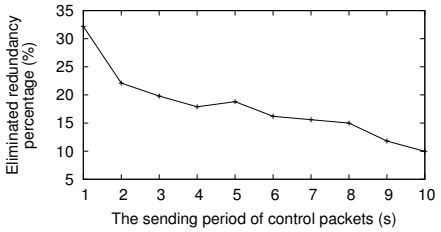

Fig. 2. Different sending rate of control packets

	NTT	Level3	Sprint	GEANT	Internet2
Flows($\times 10^6$)	51	46	37	16	8

TABLE III
CSAMP EXPERIMENT SETUP

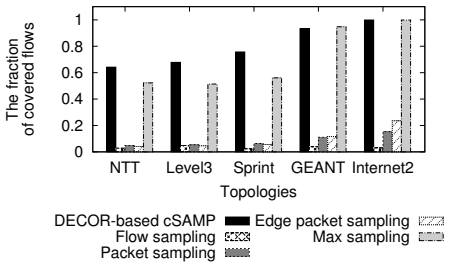

Fig. 3. The total flow coverage of different sampling methods

In Figure 2, the percentage of detected redundancy increases with the decreasing sending period of the control packets. However, a shorter sending period means more control packets, which increase traffic per unit time. Thus, adjustment of the sending period will require trade-off.

C. CSAMP

The DECOR framework can also be applied to CSAMP, as CSAMP is also a resource assignment problem. CSAMP is a traffic sampling system which tries to sample as much traffic as possible under some resource constraints. In this section, we evaluate how our algorithm works in CSAMP.

We consider five different AS topologies in PoPs level with different scales as in the CSAMP paper [1]. We use a traffic generator to create traffic, the amount of which is proportional to the scale of each topology, and the packet size is 1,000 bytes. In addition, each node can monitor 60,000 packets per second. The topologies are Rocketfuel-based [12] (Table III).

We measure the fraction of flow each sampling method can cover. We compare five sampling methods: (a) CSAMP: assigns each node different hash ranges to avoid redundant monitoring, (b) flow sampling: every node pick up one packet per 100 packets in each flow, (c) packet sampling: every node pick up one packet per 100 packets of all traffic, (d) edge packet sampling: edge nodes pick up one packet per 50 packets of all traffic, and (e) maximal flow sampling: every node samples as many packets as possible until it depletes its resources.

The performance is shown in Figure 3. CSAMP and maximal flow sampling work much better than the other

methods, as both use their resources more efficiently. Furthermore, CSAMP can arrange monitor responsibilities to avoid monitoring redundant traffic; thus, it can cover more flows than maximal flow sampling. Therefore, we can conclude that DECOR-based CSAMP is superior to other methods.

VIII. Related work

In the recent OpenFlow proposal OpenFlow [13], [14], a central controller such as NOX [15] is used to control network switches. Yu et al. find that limited CPU resources in the controller limit OpenFlow throughput to only 50K single-packet flow/s in a topology with 10 switches [16]. Therefore, DIFANE reduces controller loads by installing rules into multiple authority switches. Authority switches process policies on partial incoming flows instead of all flows in networks; thus, the controller is only responsible for partitioning global rules into local rules for authority switches. DIFANE achieves 800K single-packet flow/s throughput. However, because traffic is dynamic, the controller still needs to compute and adjust rule partitions according to traffic and resource profiles to avoid depleting authority switche resources. With highly variable traffic, the controller still becomes a bottleneck, and represents a single failure point. We believe that even modest support within switches for the distributed coordinated resource monitoring as envisioned in DECOR could help approaches such as DIFANE scale significantly better even under highly variable traffic.

Another study, TeXCP [17], provides an online distributed traffic engineering (TE) protocol that balances load in realtime and minimizes maximum utilization in the network [18], [19]. TeXCP assign an agent residing in an ingress node for each ingress-egress pair. This agent gathers explicit feedback [20] from core routers in the network and selects an appropriate path. The system balances load throughout the network based on local information. Evaluation shows that TeXCP can reach the same utilization as traditional offline TE, but can balance load in realtime. TeXCP provides an approach for TE, but DECOR supports a more general solutions that can apply to a wider range of problems. In addition, our solution supports finer-grained engineering: TeXCP can only decide how ingress nodes deal with traffic, but our solution can also expand decision to each internal node in the network.

IX. Conclusions

Coordinated resource monitoring systems are a solution to manage the network resources used by many network applications today. Existing solutions rely on central controllers to optimize the usage of different resources, but they fail to consider that controllers can be bottlenecks that reduce scalability and introduce a single point of failure.

We present DECOR, a distributed solution that can monitor different resources even without a central controller. In contrast to current solutions, DECOR successfully spreads the load of computing resource allocation among multiple nodes; thus, there is no critical point that may become a bottleneck. We demonstrated that DECOR can achieve global optimization and work as well as current centralized solutions. In addition, we also applied DECOR to various applications such as SmartRE and cSAMP and showed that DECOR converges in practical scenarios.

References

[1] V. Sekar, M. K. Reiter, W. Willinger, H. Zhang, R. R. Kompella, and D. G. Andersen, "CSamp: a system for network-wide flow monitoring," in *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, ser. NSDI'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 233–246.

[2] A. Anand, V. Sekar, and A. Akella, "Smartre: an architecture for coordinated network-wide redundancy elimination," in *ACM SIGCOMM*, 2009.

[3] R. Egashira, A. D. Yahaya, and T. Suda, "Market-based cooperative resource allocation for overlay networks," in *Proceedings of the 28th IEEE conference on Global telecommunications*, ser. GLOBECOM'09. Piscataway, NJ, USA: IEEE Press, 2009, pp. 5943–5948.

[4] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *ACM SIGCOMM*, 2008.

[5] A. Anand, C. Muthukrishnan, A. Akella, and R. Ramachandran, "Redundancy in network traffic: Findings and implications," in *ACM SIGMETRICS*, 2009.

[6] A. Anand, A. Gupta, S. S. A. Akella, and S. Shenker, "Packet caches on routers: The implications of universal redundant traffic elimination," in *ACM SIGCOMM*, 2008.

[7] B. Aggarwal, A. Akella, A. Anand, A. Balachandran, P. Chitnis, C. Muthukrishnan, R. Ramjee, and G. Varghese, "Endre: an end-system redundancy elimination service for enterprises," in *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, ser. NSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 28–28.

[8] M. Casado, T. Garfinkel, A. Akella, M. J. Freedman, D. Boneh, N. McKeown, and S. Shenker, "Sane: a protection architecture for enterprise networks," in *Proceedings of the 15th conference on USENIX Security Symposium - Volume 15*. Berkeley, CA, USA: USENIX Association, 2006.

[9] Click software router, http://read.cs.ucla.edu/click/.

[10] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The click modular router," *ACM Transactions on Computer Systems*, vol. 18, no. 3, pp. 263–297, 2000.

[11] Netfpga website, http://www.netfpga.org/.

[12] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson, "Measuring ISP topologies with rocketfuel," *Networking, IEEE/ACM Transactions on*, vol. 12, no. 1, pp. 2–16, 2004.

[13] Openflow website, http://www.openflow.org/.

[14] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, p. 69, 2008.

[15] Nox an openflow controller, http://www.noxrepo.org/wp/.

[16] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with DIFANE," in *Proceedings of the ACM SIGCOMM 2010 conference on SIGCOMM - SIGCOMM '10*, New Delhi, India, 2010, p. 351.

[17] S. Kandula, D. Katabi, B. Davie, and A. Charny, "Walking the tightrope: responsive yet stable traffic engineering," in *ACM SIGCOMM Computer Communication Review*, vol. 35. New York, NY, USA: ACM, Aug. 2005, pp. 253–264, ACM ID: 1080122.

[18] D. Applegate, L. Breslau, and E. Cohen, "Coping with network failures: routing strategies for optimal demand oblivious restoration," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 32. New York, NY, USA: ACM, Jun. 2004, pp. 270–281, ACM ID: 1005719.

[19] D. Applegate and E. Cohen, "Making intra-domain routing robust to changing and uncertain traffic demands: understanding fundamental tradeoffs," in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, ser. SIGCOMM '03. New York, NY, USA: ACM, 2003, pp. 313–324, ACM ID: 863991.

[20] L. L. H. Andrew, S. H. Low, and B. P. Wydrowski, "Understanding XCP: equilibrium and fairness," *IEEE/ACM Transactions on Networking (TON)*, vol. 17, pp. 1697–1710, Dec. 2009, ACM ID: 1721712.