Data Centers as Software Defined Networks: Traffic Redundancy Elimination with Wireless Cards at Routers

Yong Cui, *Member, IEEE,* Shihan Xiao, Chunpeng Liao, Ivan Stojmenovic, *Fellow, IEEE* and Minming Li, *Member, IEEE*

***Abstract*—We propose a novel architecture of data center networks (DCN), which adds wireless network card to both servers and routers. Existing traffic redundancy elimination (TRE) mechanisms reduce link loads and increase network capacity in several environments by removing strings that have appeared in earlier packets through encoding and decoding them several hops downstream. This article is the first to explore TRE mechanisms in large-scale DCNs and the first to exploit cooperative TRE among servers. Moreover, it also achieves the 'logically centralized' control over the physically distributed states in emerging software defined networks (SDN) paradigm, by sharing information among servers and routers in data centers with wireless cards. We first formulate the TREDaCeN (TRE in Data Center Networks) problem and reduce the cycle cover problem to prove that finding an optimal caching task assignment for TREDaCeN problem is NP-hard. We further describe an offline TREDaCeN algorithm which is proved to have good approximation ratio. We then discuss efficient online zero-delay and semi-distributed implementations of TREDaCeN supported by physical proximity of servers and routers, enabling status updates in a single wireless transmission, using an efficient prioritized schedule. We also address online cache replacement and consistency of information in servers and routers with and without delay. Our framework is tested on different parameters and shows superior performance in comparison to other mechanisms (imported directly to this setting). Our results show the robustness and the trade-off between the 'logically centralized' implementation and the overhead on handling inconsistency of distributed information in DCN.**

***Index Terms*—Data Center Network, Redundancy Elimination, Wireless Link, Software Defined Networks.**

I. Introduction

DATA center networks (DCNs) are the infrastructure for cloud service. Many enterprises migrate their computing needs into the cloud, increasing bandwidth requirements of DCNs. Existing data center networks use the following architectures: Fat-Tree[2], DCell[3], Bcube [4] etc. Large DCNs provide online services (web search, e-mails, online gaming) and computations such as distributed file systems, data mining etc. Alizadeh et al. [5] described DCTCP, an improved TCP protocol for wired DCNs.

Halperin et al. [6] introduced wireless links between servers to make use of the "hotspot" feature of DCN traffic and reduce the transmission time of DCN jobs. In [7–9], wireless network cards are added to each group of servers and therefore enable wireless communication among servers, and performance in wireless DCNs was further optimized.

Characteristics of DCNs traffic were studied recently in [10] [11]. Existing research does not address the traffic content of DCNs. Inspired by [12, 13], we introduce redundancy elimination problem into DCN to reduce the traffic between servers and hence improve the network performance. A large fraction of the traffic in DCNs is redundant. It originates from applications, software frameworks and underlying protocols. In web service application, some contents are more popular than others. The original data searching process is not aware of the redundancy of traffic requested from different users, thus resulting in a large fraction of duplicate traffic content carried over the network. In [14], the duplicate content of the traffic based on http (or http-opt) is up to 50%. MapReduce can produce over 30% of duplicate content [15]. Although, for the sake of robustness, security etc., some of the duplicate traffic content is essential, most is redundant and can be eliminated.

A well-known approach to eliminate traffic redundancy is to use proxy servers [16, 17] to cache popular objects at locations close to clients. Content delivery [18] and peer-to-peer [19] networks caches are working similarly. Recently, protocol-independent traffic redundancy elimination (TRE) mechanisms are proposed to remove the redundant strings across objects and traffic flows, achieving higher performance than object-level mechanisms [13, 20–22]. However, existing TRE mechanisms are not applicable to DCNs, since they are highly centralized and cause large communication overhead to gather the traffic matrix and some other information needed. Furthermore, they ignore the difference in utilities of caching the same data by different intermediate nodes, especially when the cache capacity is relatively small compared to the huge traffic volume in DCNs. In existing works, servers register the cache state of the data sent by them, but are not aware of the data sent by other servers. Thus they do not take

Manuscript received 15 December 2012; revised 31 July 2013. This work is supported by National Natural Science Foundation of China (no. 61120106008, 60911130511), National Major Basic Research Program of China (no. 2009CB320501, 2009CB320503) and partly supported by the grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. CityU 122512). The work of Ivan Stojmenovic is also supported by the Government of China for the Tsinghua 1000 Plan Distinguished Professor (2012-5) position and by NSERC Discovery grant. Part of the results have appeared in *ICDCS workshops 2012* [1].

Y. Cui, S. Xiao, and C. Liao are with the Department of Computer Science and Technology, Tsinghua University, Beijing, 100084, P.R.China (e-mail: cuiyong@tsinghua.edu.cn, xiaosh12@mails.tsinghua.edu.cn, liaocp06@mails.tsinghua.edu.cn).

I. Stojmenovic is with SEECS, University of Ottawa, Ottawa, Canada, and School of Software, Tsinghua University, Beijing, 100084, P.R.China (e-mail: ivan@site.uottawa.ca).

M. Li is with Department of Computer Science, City University of Hong Kong, Hong Kong, China (e-mail: minming.li@cityu.edu.hk).

Digital Object Identifier 10.1109/JSAC.2013.131207.

the redundancy of the data sent by different servers (**cross-source redundancy**) into account, caused by the similarity of the applications provided by the servers (e.g., file sharing). Moreover, they do not address the change of popularity of some content (e.g., web searches) with time.

To design TRE mechanisms for DCNs, the first challenge is how to handle the distributed information among servers and routers. A centralized control logic is much easier compared to the previous work done, but conflicts with the distributed environment. A similar problem arises on how to enable the network control logic to be well designed and operated on distributed control plane states in SDN (Software Defined Networks), and the 'logically centralized' thought has been applied and gains the most benefit. Inspired by this, we proposed a solution to achieve 'logically centralized' control over the distributed information among servers and routers in DCNs by importing wireless communication. Our work confirms the trade-off between the 'logically centralized' control optimality and the overhead on handling distributed information in DCN, which is also exploited in the recent work [23] in SDN.

Existing works [11, 24–26] have introduced how to overcome the wireless limitations such as the link blockage, interference and reliability when the wireless technologies are applied into DCNs. We propose to add wireless network cards not only to each router but also to each server and therefore enable wireless communication among servers and routers. Since all the devices in DCNs are concentrated in a relatively small place, a wireless transmission from a server or router will inform other servers and routers about data units memorized and corresponding delivery counts, which is used for making better decisions afterward. Hence the cross-source redundancy can be identified and addressed.

We use the term **data unit** (denoted as d) as the basic unit for applying TRE. The server splits files or packets into data units. When a user requests a data unit from a server A that does not hold it, A finds another server B that possesses it. The file content is then shared by wireless transmissions. Alternatively, a mapping of data unit identifier to servers storing it could be used. Data unit is delivered by routing from B to A over a path consisting of a series of routers. Server A may not cache the data unit (e.g., due to memory capacity limitations) and therefore the next request by another unit for the same data unit will cause a repetition of delivery from B to A with routers as interim nodes. To reduce traffic for any subsequent request, d can be memorized at one or more routers on the path. Let r^* be the last router on the path that memorizes d. Subsequent deliveries of d over the same path may be carried by routing of encoded (condensed) form d' of d from B to r^*, and then routing of d in full size from r^* to A. The utility of such delivery is defined as the difference in total consumed bandwidth (link by link) when d is transferred without and with encoding from B to r^* (it is equal to the number of hops from B to r^* multiplied by the difference $|d| - |d'|$ in packet length between original and encoded forms). If routers have no memory limitations, then the most effective router is the last one on the route, which is then able to directly deliver d to A. Otherwise the utilization is maximized by memorizing data units at the last possible router which has remaining capacity to record it.

The same data unit may be stored in several servers and be present in several server to server paths. Server B memorizes the number of requests within a certain time interval. More precisely, we assume that servers have derived the expected number of server-to-server deliveries (called **delivery counts**) of specific data units from the historical data and the profile of the applications.

The contributions of this paper can be summarized as follows. We formulate the *Traffic Redundancy Elimination in Data Center* (TREDaCeN) problem and prove that optimally assigning the caching tasks to routers is NP-hard. We introduce wireless transmitters and receivers at each router and design both offline and online solutions for utility-based caching task assignment. To the best of our knowledge, this is the first TRE mechanism for DCNs and our TREDaCeN algorithms are also the first scheme which attempts TRE using cross-source redundancy information.

The rest of the paper is organized as follows. Section II describes the related work. The problem formulation and the NP-hard proof are presented in Section III. Section IV-A describes the offline solution and the detailed mathematical analysis of its time complexity and approximate ratio are given in Section IV-B. We then propose online implementations of TREDaCeN, with and without wireless delay consideration respectively, in Section V. Our online solutions are enhanced by the time-relevant utility-based caching task assignment and cache replacement mechanisms. A priority-based broadcast mechanism and semi-decentralized decision model are also introduced when wireless delay is considered, to handle the information inconsistency and further improve the effectiveness. The simulation is described in Section VI. Finally we conclude the paper in Section VII. Preliminary version of this article appeared in [1].

II. Related work

A variety of mechanisms have been used to find and suppress duplicate data from their network transfers. The basic idea of redundancy elimination is to identify the repeating data contents that are transferred through network links, and then replace them with labels (also referred to as signatures). We review these mechanisms below.

In web caches [16], copies of documents passing through network links are stored to satisfy the subsequent search engines' requests. Web caches are based on search engines which may cache a website. Web proxies [17] are located between the client and the server to store web contents. These solutions can remove the duplicate data at the object level; they only cache whole documents, such as HTML pages and images. Similarly, Peer-to-Peer (P2P) cache also performs the redundancy elimination at the object level, which does not address cross-object duplicate data. Moreover, object-level TRE is application-specific, and does not resolve the redundancies between different applications.

Spring et al. [13] proposed the protocol-independent technique for eliminating redundant network traffic, which identifies the redundant data based on content-based naming. Repeated bytes between packets are identified by representative fingerprints which are integers generated by a one-way function applied to a set of bytes. They designed an effective way

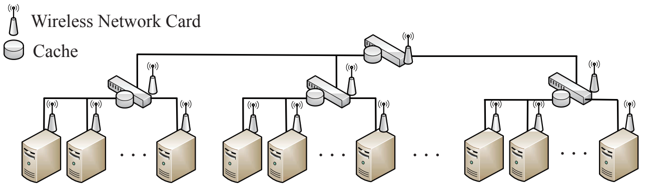


Fig. 1. Data center with wireless cards at servers and routers

to compute and select the fingerprints. Approximately 39% of the original volume of web traffic is found to be redundant and removed. Saha et al. [27] proposed CombiHeader to reduce the number of headers and hence the overhead of TRE. Halepovic et al. [28] exploited the non-uniformities in redundant traffic elimination.

TRE was exploited in specific networks. Aggarwal et al. [20] studied the end-system redundancy elimination service recently. Anand et al. [21] proposed a scheme for supporting TRE as a primitive IP-layer service on network routers and first illustrated the feasibility of caching packets in fast routers. TRE is performed considering the topology of the network. However the proposed TRE is executed hop by hop, resulting in a waste of cache resources. By considering the benefits of coordination of the router caches, a network-wide redundancy elimination (smartRE) is proposed in [22]. In smartRE, servers register the data units sent by themselves and assign the caching task based on the fingerprint, not on the utilities, therefore the different utility when caching the same data by different routers (cross-source redundancy) was not considered.

III. TREDaCeN Framework and Notations

A. TREDaCeN Framework

Our architecture (Fig. 1) further enhances the one proposed by Halperin et al. [6]. In [6], only servers (or top of rack switches) are equipped with wireless network interface cards. In our architecture, there is a cache at each router, and a wireless network interface card in each server and in each router. To address cross-source redundancy, we assume that each server learns whether and where data sent by other servers are cached. Cache information (which data units are cached by which routers) is transmitted by routers over wireless channels (**cache information broadcasting**).

Based on this architecture, both offline and online redundancy elimination mechanisms are proposed. Servers provide the source data units while routers have two roles: forwarding and decoding. The router forwards data units to the next hop without additional processing, or decodes the encoded data (specific fingerprints [13]) by matching the cached table and fetching the copy of data units it cached [21, 22]. The roles of routers are changing based on instructions from servers and their own information. Before routing, **server** checks the cache table to see whether or not data unit d is cached by some routers downstream the routing path. If so, d is routed in encoded form, and is *decoded* into original form by some specific router downstream (**decoding task assignment**). The last router on the route that has cached data unit d (decoding router, r^*) will *decode* it into original form and forward toward the destination. Other routers on the route will forward packet as received (encoded or not) to the next router on the route to B. They may cache the payload of the flow through them, if instructed by a server, and if d is not in the encoded form. This **caching task assignment** procedure is applied, considering that data units cached by different routers gain different utilities. If d is expected to be cached by the router r, r also performs **cache replacement task**, if its cache is full. Procedures of servers and routers are summarized in Fig. 2.

In the offline mechanism, we assume that servers know the information of data units already and the information does not change in a relatively long term period. During this period, data units to be sent are known in advance, and the expected delivery counts of data units are known from history. The caching task assignment is performed just once at the very beginning of the period and then keeps unchanged until the next period. Hence in our offline mechanism, we focus on the once caching task assignment.

In the online mechanism, data units to be sent are not known a priori and arrive randomly, so servers have to collect the cache information of all routers in real time, and then when a data unit arrives, make cache decisions by themselves. The algorithm should make real-time decisions on possible caching assignment and cache replacement among routers. The cache refreshments (when caching information is changed in a router) should be known by the servers to perform decoding correctly. In our online mechanism, the cache refreshments are broadcast by routers through wireless channels. Furthermore, considering the broadcasting collision challenge, they decide which router broadcasts its cache information first by *utility-based broadcast mechanism.*

B. Definitions and Notations

The set of routers is denoted by $R = \{r_1, r_2, ..., r_{|R|}\}$. The cache capacity of each router is the same and denoted by C. $S = \{s_1, s_2, ..., s_{|S|}\}$ is the set of servers, $P = \{p_1, p_2, ..., p_{|P|}\}$ is the set of routing paths between servers and $r_k \in p$ means that router r_k belongs to path p. The set of data units is $D = \{d_1, d_2 \dots d_{|D|}\}$. Data units are the same length l before encoding and length (size) l' after encoding. Each data unit may pass through any of the routing paths.

Data unit d is forwarded from source server s on path p in encoded form d' until the **decoding router** $r^*(d, p)$. The **footprint** of data unit d along path p is the total amount of consumed bandwidth resource when d is routed over p. The footprint is computed as $x{*}l + y{*}l'$, where x and y are numbers of hops on p with full length and encoded form forwarding, respectively. The reduction of footprint of d passing through p_j is $(l - l') \cdot h_{j,k}$, where $h_{j,k}$ is the number of hops from the source of p_j to decoding router $r_k = r^*(d, p_j)$. Fig. 3 illustrates the concept.

Suppose e_{ij} is the expected number of user requests for data unit d_i passing through path p_j within certain fixed time interval (the **delivery count**). The **RE utility** of one data-router-path group (d_i, r_k, p_j) is defined as the total footprint reduction of d_i when cached by r_k and routed through p_j within the same time interval, and can be computed by

$$u(i, j, k) = e_{ij} \cdot (l - l') \cdot h_{j,k}. \quad (1)$$

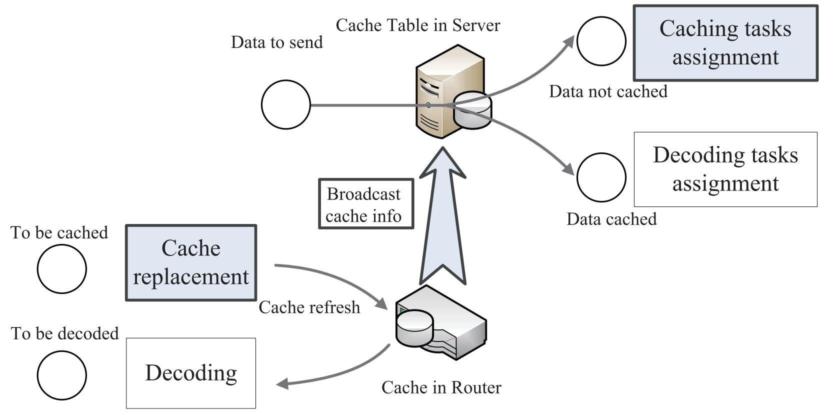


Fig. 2. Procedures of servers and routers.

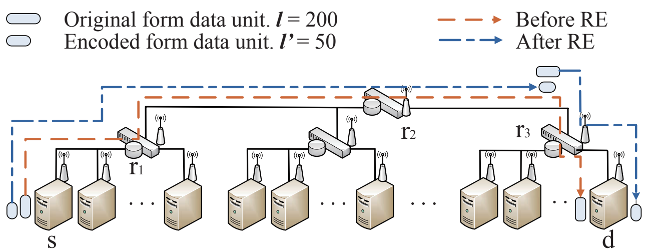


Fig. 3. Footprint reduction. (a) Before TRE, footprint is 4*200=800 bytes; (b) After TRE, footprint is 3*50+200=350 bytes. Footprint reduction is 800-350 =(200-50)*3=450 bytes.

TABLE I
NOTATIONS AND DEFINITIONS

Notations	Definitions
d, d'	Data unit and encoded data unit respectively
l, l'	Size of full data unit and encoded form data unit
r, s, p	Router r, server s and path p
$r^*(d,p)$	Decoding router of data unit d on path p
$h_{j,k}$	Number of hops from the source of p_j to r_k
$u(i,j,k)$	RE utility of d_i through p_j with decoding router r_k
$\widehat{u}(i,k)$	Cache utility of d_i cached by r_k
C, ω	The cache capacity and broadcast capacity
$E=\{e_{ij}\}$	The matrix of the delivery counts of data units
$A=\{a_{ik}\}$	The cache assignment matrix where $a_{ik}=1$ means r_k caches d_i
$u_{tb}(i,k,t_c)$	the time-based cache utility of d_i cached by r_k at time t_c
$c_t(ps,k,t)$	the replacement cost of cache position ps on r_k at time t
$p_t(i,ps,k,t)$	the replacement profit of new d_i cached by r_k at cache position ps and time t

Denote $\widehat{u}(i,k)$ as the **cache utility** of one data-router pair (d_i,r_k) where d_i is cached by r_k. It is the total *RE utility* of d_i when cached by r_k and routed over all possible paths who take r_k as their decoding router for d_i. Then we have

$$\widehat{u}(i,k)=\sum_{j=1:r_k=r^*(d_i,p_j)}^{|P|} u(i,j,k). \tag{2}$$

C. Problem Formulation

We assume that all delivery counts $E=(e_{ij})_{M*|P|}$, $M=|D|$, are known a priori. In reality, the delivery counts matrix E can be obtained from the history traffic data. Within a given time window T (one day or one week), we can record the delivery counts for different data units through different paths. We can use the delivery counts matrix recorded in the last time window, or the average value of elements in the matrices of the last multiple time windows as the estimation for the current matrix E. We focus on the caching task assignment, deciding which routers should cache which data for a long term benefit.

For each data unit d_i, the set of routers Υ_i that cache d_i and assigned as its decoding routers can be one of $2^{|R|}$ possible different subsets (including the empty set $\emptyset$). Then the total cache utility of d_i cached by routers in Υ_i is computed by

$$\widehat{u}^*(d_i,\Upsilon_i)=\sum_{k=1:r_k\in\Upsilon_i}^{k=|R|}\widehat{u}(i,k). \tag{3}$$

Then the total cache utility of all possible data-router pairs in this problem is

$$ut=\sum_{i=1}^{i=M}\widehat{u}^*(d_i,\Upsilon_i). \tag{4}$$

The objective of TREDaCeN is to maximize the total cache utility ut, by making decisions on caching data units by routers optimally. Here we introduce a *cache assignment matrix* $A=(a_{ik})_{M*|R|}$, where $a_{ik}=1$ means r_k caches d_i and be 0 otherwise, as the output of the solution. Let an integer constant C denote the *cache capacity* of routers (each router can only cache up to C data units). Then we have

$$\sum_{i=1}^{i=M} a_{ik}\le C; k=1,2,...,|R|. \tag{5}$$

Then the TREDaCeN problem is formulated as

$$\max ut=\sum_{i=1}^{i=M}\widehat{u}^*(d_i,\Upsilon_i) \tag{6}$$

subject to Constraints (1) and (5).

D. NP-completeness of TREDaCeN

To show the NP-completeness of TREDaCeN, we first introduce the cycle cover problem. A **cycle cover** denotes a spanning subgraph consisting of node-disjoint cycles which cover all the vertices of the graph. A **k-cycle-cover** denotes a cycle cover where each cycle's length (number of edges) is at least k. A **k-cycle** denotes a cycle with length k. A **unidirectional graph** denotes a directed graph with no directed cycles with length 2 (i.e., if (a, b) is an edge in the graph, then (b, a) is not). We use **3-UDCC** (3-UniDirectional-Cycle-Cover) to denote the problem of finding a 3-cycle-cover in a *unidirectional graph*. To show the NP-completeness of 3-UDCC, we use a similar method in [29] which is used to prove that 3-DCC (3-Directional-Cycle-Cover) problem is NP-complete. To make our discussion self contained, we restate the procedure shown in our online technical report [30]. By reducing 3-Dimensional Matching (3DM) to 3-UDCC in the proof, we generate the following lemma:

Lemma 1: 3-UDCC problem is NP-complete.

Since in a unidirectional graph, there are no 2-cycles, Lemma 1 implies the following lemma:

Lemma 2: Finding a cycle cover in a *unidirectional graph* is NP-complete. (**UDCC** problem)

Proof: See the detailed proof in technical report [30]. ■

We can transform the UDCC problem into our TREDaCeN problem by constructing a unidirectional graph corresponding to a well-designed network topology among routers and servers where each router can only cache up to one data unit. Given a kind of traffic patten of data units into the network, we can find an optimal caching assignment to routers corresponding to one feasible solution in UDCC. According to the detailed proofs shown in [30], we generate the following theorem.

Theorem 1: Finding an optimal caching task assignment for TREDaCeN problem is NP-hard.

IV. Offline TREDaCeN

In this section, we handle the offline TREDaCeN problem, which makes decisions on how to cache data units at routers with limited cache capacity so that the long term traffic redundancy elimination benefit is maximized. In this scenario, the wireless channel has sufficient bandwidth and no collisions, and each cache decision of data units is learned by servers without delay. Decisions are made iteratively, so that servers and routers use decisions made so far to make the next decision. We perform caching task assignment at the beginning of each period and does not replace already cached data units until the next period. The offline algorithm is centralized essentially, but can run in a decentralized way since all servers and routers are physically close and are able to share the same information all the time by wireless transmissions.

A. Approximate Solution

For the NP-completeness of TREDaCeN, in this section, we propose OFTRE (Offline TRE), a light-weight greedy algorithm for caching task assignment. Though the idea is simple, we theoretically prove that the approximate ratio of our algorithm is good.

The first step of OFTRE is to estimate the cache utility for each data-router pair (lines 1-8). Note that it is not possible to compute the cache utility of each data-router pair (d_i, r_k) before the decoding routers are actually assigned. However by assuming that d_i is potentially cached by all routers r_k, we can modify (2) to estimate maximal potential benefits as (line 6):

$$\widehat{u}(i,k) = \sum_{j=1:r_k \in p_j}^{|P|} u(i,j,k). \tag{7}$$

The second step is to repeatedly select to cache data unit with the largest estimated cache utility among routers with remaining capacity (line 11). Before that, we mark each data-path pair (d, p) as *'Unassigned'*, which means none of the data-path pairs has been assigned a possible decoding router (line 9). When the data-router pair is selected in (line 11), the selected router caches the selected data unit as its decoding router for all paths (referred as *affected paths*) through it,

Algorithm 1 OFTRE Offline Assignment

Input: $S, R, P, D, E, \{h_{j,k}\}, C$
// E is the matrix of the delivery counts of data units
// $\{r_i\}$ is the set of routers, $\{p_j\}$ is the set of paths
// $h_{j,k}$ the number of hops from the source of p_j to r_k
// Each router can cache C data units.
Output: $A = (a_{ik})_{M*|R|}$ // $a_{ik} = 1$ means r_k caches d_i
$r^*(d_i, p_j)_{M*|P|}$ // the allocated decoding routers
1: **for** each router r_k **do**
2: $c_k = C$ // Initialize r_k's cache capacity.
3: **for** each data unit d_i **do**
4: $a_{ik} = 0$
5: $u(i,j,k) = e_{ij} \cdot (l - l') \cdot h_{j,k}$
6: $\widehat{u}(i,k) = \sum_{j=1:r_k \in p_j}^{|P|} u(i,j,k)$
// Estimate the cache utility of (d_i, r_k).
7: **end for**
8: **end for**
9: Mark all data-path pairs (d, p) as *'Unassigned'*
10: **while** $\sum_{k=1}^{|R|} c_k \neq 0$ **do**
11: Select (d_i, r_k) which maximizes $\widehat{u}(i,k)$ with $c_k \neq 0$
12: $a_{ik} = 1$, $c_k = c_k - 1$ // r_k caches d_i
13: **for** each p_j: $r_k \in p_j$ **do**
14: **if** $e(i,j) \neq 0$ and (d_i, p_j) is *'Unassigned'* **then**
15: Mark pair (d_i, p_j) as *'Assigned'*
// (d_i, p_j) is assigned a decoding router: r_k
16: **for** each $r_h \in p_j$ **do**
17: $\widehat{u}(i,h) = \widehat{u}(i,h) - u(i,j,k)$
// Refresh the cache utility of affected routers.
18: **end for**
19: **end if**
20: **end for**
21: **if** $c_k = 0$ **then**
22: **for** each d_i **do**
23: $\widehat{u}(i,k) = 0$
24: **end for**
25: **end if**
26: **end while**
27: **for** each (d_i, p_j) **do**
28: $mk = \arg\max_k h_{j,k}$, where $r_k \in p_j$ and $a_{ik} = 1$
29: $r^*(d_i, p_j) \leftarrow r_{mk}$
// Decide the final decoding router for each (d, p)
30: **end for**

where corresponding data-path pairs should be *'Unassigned'* to avoid repeated computation. Thus its capacity decreases by one (line 12). To avoid unnecessary refresh operations for routers (referred as *affected routers*) on *affected paths*, we first perform a condition check here (line 14). And then, we mark the *affected paths* as *'Assigned'* (line 15) and refresh the *affected routers* by removing the selected data unit from their cache utilities (line 16-18). Routers whose cache capacity is reduced to 0 can no longer cache data units and their potential cache utility is set to be 0 for all data units (line 21-25). Caching ends when no additional caches are possible. It acts in a greedy way with *mark operations* step by step and is not an optimal solution. Therefore, we fix the non-optimal solution by re-assigning decoding routers (line 27-30) to get the best TRE solution after caches are decided by this greedy algorithm. The decoding router that is closest to the destination is finally selected as the sole decoding router for each data-path pair. It finally fixes possible multiple decoding routers on the same paths. One detailed example is given in [30].

The time complexity of OFTRE algorithm is bounded by $C \cdot |R| \cdot |P| \cdot h$, where h is the maximum number of hops in any path.

B. Approximation Ratio

We now analyze the approximation ratio of the OFTRE algorithm. Let $E = (e_{ij})_{M*|P|}$ be an input *delivery counts matrix* and $A = (a_{ik})_{M*|P|}$ be an output *caching assignment matrix* in the TREDaCeN problem. With different input E, we can get different output A. Therefore, we denote $\widehat{u}_t(E)$ as the total cache utility obtained by our OFTRE algorithm with input E. $\widehat{u}_o(E)$ denotes the optimal total cache utility that can be obtained by an optimal algorithm with the same input E. Additionally, the **m-leftover matrix** (matrix-based leftover matrix) of input matrix E with A (denoted by $\overline{E}_m(A)$) is generated from E by setting $e_{ij} = 0$ whenever $a_{ik} \neq 0$ with $r_k \in p_j$. In other words, it is the removal of the data units from *affected paths* after assigning a decoding router to them.

Theorem 2: Let A be the caching assignment matrix obtained by OFTRE algorithm with input E. Then $\widehat{u}_o(\overline{E}_m(A))$ is the total cache utility obtained by the optimal algorithm with another new input matrix which is the *m-leftover matrix* of A. We have

$$\widehat{u}_t(E) \geq \widehat{u}_o(\overline{E}_m(A)). \tag{8}$$

Proof: See the detailed proof in [30]. ■

Lemma 3: Let A be the caching assignment matrix obtained by OFTRE algorithm with input E. $\widehat{u}_m(i,k) = \max(\widehat{u}(i_m, k_n))$, $i_m = 1, 2, ..., M$; $k_n = 1, 2, ..., |R|$. The **p-leftover matrix** (pair-based leftover matrix) $\overline{E}_p(i,k)$ of one data-router pair (d_i, r_k) is generated from E by setting $e_{ij} = 0$, where $a_{ik} \neq 0$ and j satisfies $r_k \in p_j$. Under the scenario where each data can only be cached by one router at most, we have

$$\widehat{u}_o(\overline{E}_p(i,k)) + \widehat{u}_m(i,k) \geq \widehat{u}_o(E). \tag{9}$$

Proof: See the detailed proof in [30]. ■

The following theorem gives an approximation ratio for the OFTRE algorithm.

Theorem 3: Let A be the caching assignment matrix obtained from OFTRE algorithm with input E, having the total cache utility $\widehat{u}_t(E)$. The optimal caching assignment matrix is $A_o = (a'_{ik})_{M*|R|}$ from an optimal algorithm with the same input E, having the total cache utility $\widehat{u}_o(E)$. Under the condition each data unit can only be cached by one router, we have

$$\widehat{u}_t(E) > 0.5 \cdot \widehat{u}_o(E). \tag{10}$$

Proof: See the detailed proof in [30]. ■

We argue that in the scenarios with large volume of traffic, most of the data units are only cached by one router. Then the condition $\widehat{u}_o(\overline{E}_p(i,k)) + \widehat{u}_m(i,k) \geq \widehat{u}_o(E)$ as described in Lemma 3 can mostly be satisfied. Thus the approximation ratio is always above 0.5. Experimental data in Fig. 4 of Section VI-A show that the total cache utility by algorithm OFTRE is much closer to the optimal result in most of the cases.

V. ONLINE TREDACEN

In OFTRE, delivery counts of data units to be sent during the period are known a priory, and no cache replacements are considered after the assignment during this period. The algorithm becomes inefficient when delivery counts dynamically change. We address here real-time caching assignment and cache replacement, responding to the real-time arriving of data units. We propose the online redundancy elimination mechanisms ZDTRE (Zero-delay Distributed TREDaCeN), DITRE (Delay Involved TREDaCeN) and SDTRE (Semi-Decentralized TREDaCeN), which exploit the fact that devices in DCNs are deployed in a relatively small area and can share real-time information.

In all algorithms, the message is received immediately after it was transmitted regardless of the transmission distance, because radio waves travel at the speed of light. However, there is delay between transmissions of two bits, and correspondingly the minimum delay in receiving messages is equal to the transmission time of update message (the message is sent as soon as the update occurred). Here the update message basically includes the change of routers' cache information and update message about the replace rate of cache information is also included in SDTRE.

All algorithms assume ideal physical layer: when there is a single ongoing transmission, it is received correctly by all the recipients. We also assume an environment with a lot of data units being sent, but wireless transmission is only used for cache information. If, following a data unit, there is no change in cache information, or no change in the matrix of delivery counts, then wireless transmission is not scheduled.

ZDTRE assumes zero delay in transmitting and receiving any new information, and no collisions, therefore all messages are correctly received by all recipients. The servers deal with both the caching task and decoding task assignment when new data units arrive, while the routers deal with the caching replacement and decoding events when receiving data units in a parallelized way. DITRE considers also zero delay in transmitting update messages, but considers transmission time and consequently the impact of collisions, where wireless messages are not received correctly by all the recipients, due to collisions which happen when transmission times of two or more messages overlap. DITRE does not add any error handler mechanisms. SDTRE differs from DITRE in adding handler for errors (detection and correction), and adds delay in transmitting due to scheduling broadcasts (by routers) with priorities. In SDTRE, the servers will utilize real-time handlers for caching and decoding task assignment. The routers, however, will not execute the instructions of servers unconditionally. Instead, they detect the inconsistent errors based on their own information and provide feedback and newest cache information to servers if necessary. The motivation for considering collisions, transmission delay and error handler, is to investigate the robustness of our concept as an example of SDN (software defined network).

1) Periodic Self-update Input: We suppose that the initial delivery counts of d_i are known from the historical data and the profile of services is changing with time. $e_{ij}(t_1, t_2)$ denotes the expected delivery counts of d_i transmitted through p_j during the time interval between t_1 and t_2, which will be real-time updated according to the pattern of arriving data units. For example, if we consider one day as a reasonable period that reflects the period of traffic pattern, then at next day, the new real-time data of delivery counts (denoted as $newE_{i,j}(t_1, t_2)$) can be added to the previously accumulated data in the table of expected delivery counts. We call this a *periodic self-update*. Based on this input, the cache utilities are estimated and caching tasks assignments are performed.

2) Utility-based Cache Replacement: We further introduce *cache replacements* in the online problem. We introduce some notations for time stamps. $t_c(i,j)$, $t_{in}(i,k)$ and $t_o(i,k)$ denote the time when d_i is sent, arrives and is removed respectively on path p_j. If not confusing, we use t_c, t_{in}, t_o instead of $t_c(i,j)$, $t_{in}(i,k)$, $t_o(i,k)$ for brevity.

Some of the data units cached by the router may have to be removed to make room for the newcomer if some long term benefit can be obtained. Thus each data unit cached by the routers has its own *living time*, the period between its cached time t_{in} and its replaced time t_o. Different *living time* of the same data unit cached in the same router can generate different cache utility. To evaluate this effect, we introduce the **time-based cache utility** for the RE utility obtained from caching data units during its *living time*. At current time t_c, the *time-based cache utility* $u_{tb}(i,k,t_c)$ of d_i cached by r_k can be computed by

$$u_{tb}(i,k,t_c) = \sum_{j:r_k \in p_j} e_{ij}(t_{in}, t_o) \cdot (l - l') \cdot h_{j,k}. \tag{11}$$

When a server calculates $u_{tb}(i,k,t_c)$ with (11) at time t_c, there are two situations to evaluate t_{in}. First if for r_k, d_i is cached already, then the server reads t_{in} from r_k's previous broadcast information. Otherwise, d_i is going to be cached there, therefore, under the zero-delay assumption, the server estimated t_{in} by the predicted caching time of d_i as

$$t_{in} = h_{j,k} * t_h + t_c \tag{12}$$

where t_h is the average time cost of one hop during routing.

Since t_o is hard to predict, we give a simple intuitive estimation of t_o by

$$t_o = t_{in} + C/\nu_k. \tag{13}$$

where C is the cache capacity of r_k and ν_k is the **replacement rate** of r_k. *Replacement rate* is the average number of replacements in the router per time unit. It is updated by the router itself according to its real-time replacement information. During cache replacement, since the replaced data unit cannot help in further traffic redundancy reduction, this cache replacement incurs the **replacement cost** and **replacement profit**.

If at time t, d_i is to be cached by router r_k with $d_{i'}$ removed at cache position ps, then the *replacement cost* of d_i cached by r_k is

$$c_t(ps,k,t) = u_{tb}(i',k,t). \tag{14}$$

For ordinary caching operations without replacement, the *replacement cost* is 0. The *replacement profit* is

$$p_t(i,ps,k,t) = u_{tb}(i,k,t) - c_t(ps,k,t). \tag{15}$$

It indicates that the RE utility we want to maximize is the benefit that we get from new d_i cached minus the *replacement cost* that we have to pay for not continuously caching old $d_{i'}$. Therefore, for one arriving data unit, if we want to get the most profits on caching, we should consider the cache position in one router that can obtain the largest *replacement profit*.

3) Assigning caching tasks: The main caching tasks assignment is performed by the servers and routers in parallel. There are events that servers should deal with, which are all independent of routers. Whenever a data unit d_i arrives at a server, the event triggers server handler. The server first performs the comparison according to the newest cache information it knows about all routers, and then forwards it to the wanted router who returns the largest replacement profit value for d_i with the best caching position $dIndex$. Whenever a router receives a data unit d_i, (delivered by one server), if d_i is encoded, router decodes it and forwards to its destination. Otherwise, router will check whether or not the position $dIndex$ lastly computed by the server is full. If it is empty, then router caches d_i there, or replaces old one by new d_i, and forwards a copy of d_i to the destination.

A. Zero-delay Distributed TREDaCeN

In ZDTRE scenario, we assume that servers and routers are in close proximity and can share real-time caching information without delay by wireless transmissions. Under this zero delay assumption, we make real-time cache decisions and replacements in parallel by all servers and routers, while neglecting the broadcast delay. Since we assume ideal physical layer in ZDTRE, where all wireless messages are received correctly by all routers and servers, and there is zero delay, all decisions and all the information in all servers and routers are consistent. The pseudo code of the ZDTRE procedure is illustrated in Algorithm 2.

B. Delay Involved TREDaCeN

We introduce DITRE algorithm to illustrate the phenomenon of **information inconsistency** and **inconsistency error** in delay-involved broadcast environment. DITRE performs the same operations as ZDTRE. While in ZDTRE zero delay provides information consistency to all servers and routers,

Algorithm 2 ZDTRE Online Assignment

Input: d_i, ν_k, e_{ij}

Output: Real-time cache assignment and cache replacement

input_self_update ($e_{i,j}$)
// triggers when one period of traffic is over
$e_{ij}(t_1, t_2) \leftarrow e_{ij}(t_1, t_2) + newE_{ij}(t_1, t_2)$

server_event_handler **(**d_i**,** p_j**)**
// triggers when a new data unit d_i arrives with its
for each router r_k on path p_j **do**
  for each cache position ps in r_k **do**
    Calculate $p_t(i, ps, k, t)$ with (11) and (15)
  end for
end for
Get position $dIndex$ in router $rIndex$ which maximizes $p_t(i, ps, k, t)$ with value $maxPt$
if r_{rIndex} caches d_i already **then**
  Encode d_i and forward to r_{rIndex} with $dIndex$
else
  Forward d_i to r_{rIndex} with $dIndex$
end if

router_receive_event **(**d_i**,** $dIndex$**)**
// triggers when router r_k receives d_i with $dIndex$
if d_i is encoded **then**
  Decode d_i and forward to its destination
else
  if position $dIndex$ in r_k is empty **then**
    Cache d_i in the $dIndex$ position
  else
    Replace old one in position $dIndex$ by new d_i
  end if
  Make a copy of d_i and forward to its destination
  Update new cache information to servers by broadcast
end if

updates in DITRE, although transmitted instantly, are received with delay or possibly not received due to collisions, leading to *information inconsistency* and inconsistent decisions (*inconsistency error*).

Here is an example to show how *information inconsistency* and *inconsistency error* among servers and routers happen in DITRE. Suppose each router has only one cache and at time t_0, each router r_k cached different data unit d_k. Suppose at time t_c, there is a router r_k who has an update of its cache information that another d_i replaces the old d_k. The router starts to broadcast and marks new d_i as 'already broadcast'. However, the broadcast of this update needs transmission time. The $delay$ is equal to the time needed to transmit the update message, and it is known by servers at time $t_c + delay$. There is a chance that the transmission is in collision with another update transmission and all servers do not receive it. Then the $delay$ can be treated as infinity for them because the broadcast is lost.

Suppose some data units arrive at servers during time interval $[t_c, t_c + delay)$. r_k cached new d_i, but servers think it still caches old d_k. Then we have *information inconsistency* among routers and servers. If d_k arrives at server, server believes d_k is cached by r_k, encodes d_k and forwards it to r_k. However when r_k receives encoded d_k, it cannot decode it. This is one type of inconsistency error. This type will cause both packet loss and RE utility profit loss in DITRE.

There is also another type of inconsistency error. During $[t_c, t_c + delay)$, if there is one new data unit d_n arriving at a server, and no router caches d_n then, the server calculates the best router r_k which returns the largest replacement profit for d_n. In the calculation for r_k, the server assumes r_k caches d_k, but has no knowledge that new d_i has replaced d_k. If r_k is selected, then when r_k receives d_n, it may find the current replacement profit for d_n to replace d_i is less than 0 (the previous replacement profit is calculated as d_n to replace d_k by servers), thus does not want to cache d_k. In DITRE, this means RE utility profit loss.

Because of these two types of inconsistency errors, DITRE works worse with larger $delay$. Especially when the cache capacity of routers is small, inconsistency errors will accumulate enough to result in significantly different information among what routers cache and servers know, which keeps servers making wrong cache decisions that cannot be performed by routers and thus gains little RE utility profit.

C. Semi-Decentralized TREDaCeN

DITRE deteriorates when cache capacity is smaller and delay gets larger, due to increased inconsistency errors. SDTRE has priority-based transmission scheme which imposes routers to follow certain order (round robin) in accessing wireless channels. Despite reception delays and collisions, SDTRE makes timely updates because it reacts to the inconsistency information.

1) Utility-based Refreshment Broadcast: We propose a utility-based refreshment broadcast scheme composed of fingerprint compress, refreshment and priority based broadcast techniques. Each server has a complete *cache information table* of all routers. The table is constructed by pairs of router indices and *fingerprints* of data units that they cache. The data unit's *fingerprint* denotes the combination of the data unit index and its cached position index in the router. Whenever a router caches a new data unit, it immediately transmits the change of their cache information (the fingerprint of the new data unit) to servers through wireless channels. Then all the servers will update their own *cache information table*. However, many refreshments in DCNs may happen in nearly the same time, and collisions in wireless channels may cause the loss of information.

SDTRE addresses this issue by transmitting each router's replacement information with priority. In SDTRE, routers follow the round robin order to broadcast their cache refreshment information. The total time a router (r_k) transmits its cache information once is ΔT_0. Let ω denote the **broadcast capacity**, i.e., the number of data units whose cache information can be transmitted during one unit time. Then the cache information of $\Delta T_0 \cdot \omega$ data units are transmitted by router during the router's one broadcast. When the router stops transmitting, the next router (r_{k+1}) begins its turn. To help the utility estimation in servers, when a router begins broadcasting, it broadcasts both its cache information refreshments and its current average *replacement rate*.

When r_k begins its turn, it needs to decide which cache refreshment should be broadcast first. In SDTRE, each cache refreshment is matched with a utility value, which is the *time-based cache utility* computed by r_k for its corresponding new data unit, and the one with the highest utility value is broadcast first.

If at time t_c, the cache information that d_i is cached by r_k is not known by servers yet, and the new caching does not provide profits until this information is broadcast to them. Let $t_b(i,k)$ (t_b for short) denote the time the cache information of d_i cached by r_k is known by servers (through wireless transmissions). Then different from (11) where t_{in} equals t_b, the new *time-based cache utility* of d_i in this scenario should be computed as

$$\widetilde{u}_{tb}(i,k,t_c) = \sum_{j:r_k \in p_j} e_{ij}(t_b,t_o) \cdot (l-l') \cdot h_{j,k}, \qquad (16)$$

where $t_b > t_{in}$.

When a server calculates $\widetilde{u}_{tb}(i,k,t_c)$ with (16) at time t_c, there are two situations to evaluate t_b. First if for r_k, d_i is cached already, then the server reads t_b from r_k's previous broadcast information. Otherwise, the server has to estimate t_b, the time when the information that d_i is cached by r_k will be broadcast and learned by the server in the future. Since the broadcast delay is much larger than forwarding delay in this scenerio, we focus on the broadcast delay. One round time for all routers to broadcast is $|R| * \Delta T_0$. We can estimate that d_i is broadcast at the middle of one round, i.e., $|R| * \Delta T_0/2 + t_c$. However, if the *replacement rate* of r_k is much larger than *broadcast capacity*, then it will take $\frac{\nu_k}{\omega}$ rounds for r_k to broadcast all its refreshments, where ν_k is the replacement rate of r_k. So additional $\frac{\nu_k}{\omega} * |R| * \Delta T_0/2$ should be added into the estimation. With the feedback on latest *replacement rate* of r_k, the server estimates t_b as

$$t_b = \frac{\nu_k}{\omega} * |R| * \Delta T_0/2 + |R| * \Delta T_0/2 + t_c. \qquad (17)$$

2) Semi-decentralized Decision: The intuitive scheme for the assignment decision in SDTRE is to let servers decide the best router for caching or decoding according to the cache information of all routers as in ZDTRE, based on the received updates. However, when the traffic in DCNs is busy and servers want to cache their arriving data units to the same router at nearly the same time, the scheme will cause cache assignment collisions and limited cache ability. Further, it can lead to *information inconsistency* and *inconsistency error* described in DITRE. To address this issue, we propose a two-level decision scheme (referred as *semi-decentralized* scheme, distinguished from the parallelized scheme in the zero-delay distributed solution). It exploits the proactive error detections from routers to get timely information updates.

Whenever a new data unit arrives at one server, the server will first check the cache information table and make the first-level cache decision on which router on the routed path is the best router. If the new arriving data unit has been cached on the routing path, for information inconsistency effect, SDTRE will select the furthest router that caches it on the wanted path to get fast decision for decoding and forward with encoding form to the wanted router to gain quick profit.

Otherwise, the server first estimates the replacement profit of d_i cached at r_k, and then selects r_k which maximizes replacement profit of d_i to be the best router. However, if the server still evaluates the *replacement profit* with equation (15), the estimation of *replacement cost* remains inaccurate because of the information inconsistency effect. To avoid the continuous deteriorating cases resulted by this, we use the latest average replacement cost $\overline{o}_k$ of data units cached by router r_k to do the estimation, though it causes some optimality loss. Then servers can estimate the replacement profit of d_i if it will be cached by some router r_k as

$$\widetilde{p}_t(i,ps,k,t) = \widetilde{u}_{tb}(i,k,t) - \overline{o}_k. \qquad (18)$$

Algorithm 3 SDTRE Online Assignment

Input: d_i, o_k, ν_k, ω, e_{ij}
Output: Real-time cache assignment and cache replacement
input_self_update ($e_{i,j}$)
$//$ triggers when one period of traffic is over
$e_{ij}(t_1,t_2) \leftarrow e_{ij}(t_1,t_2) + newE_{ij}(t_1,t_2)$
server_event_handler (d_i, p_j)
if d_i has been cached on path p_j **then**
  Encode d_i and forward to the farthest router caches it
else
  Same operations as *server_event_handler* in Algorithm 2 with line 4 replaced by 'Calculate $\widetilde{p}_t(i,ps,k,t)$ with equation (16) and (18)'
end if
router_receive_event (r_k, d_i)
if d_i is encoded **then**
  if r_k caches d_i already **then**
    Decode d_i and forward to destination
  else
    Inform the source server to retransmit d_i
    Record all newest cache information of r_k as cache refreshments into the broadcast queue Q_k
  end if
else
  Find the best cache position $dIndex$ who maximizes $p_t(i,ps,k,t)$ with value $maxPt$
  if $maxPt \leq 0$ **then**
    Forward d_i to its destination directly
    Record all newest cache information of r_k as cache refreshments into the broadcast queue Q_k
  else
    Cache d_i by the same operation as line 16 to 20 in Algorithm 2
    Make a copy of d_i and forward it to its destination
    Record it as a cache refreshment of r_k into Q_k
  end if
end if
priority_broadcast_event (Q_k)
$//$ triggers when Q_k is not empty
for each router r_k in a certain round **do**
  while Q_k is not empty **do**
    Pop the cache refreshment q_i that owns the highest utility value
    r_k broadcast q_i in ΔT_0
  end while
end for

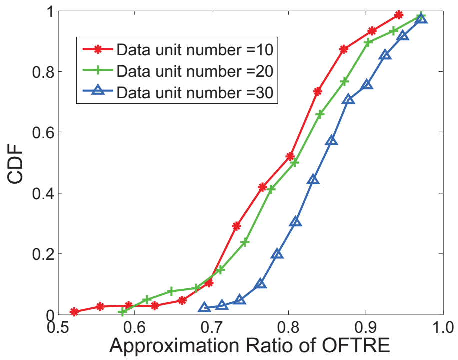


Fig. 4. The approximation ratios between OFTRE and Optimal solution.

When the data unit is forwarded to the wanted router, the router will perform the final decision on whether to cache or decode according to its own cache information.

There are two cases to handle when routers receive data units. Firstly, when the wanted router receives the encoded form of data unit d_i, then it will first immediately check whether it caches d_i. If so, then it performs decoding for d_i and forwards it to the destination. Otherwise, it recognizes inconsistency error and the server must have made wrong decision. So it will first inform the source server to retransmit d_i and also broadcast all its newest cache information to servers timely. Secondly, when the wanted router receives the complete form of data unit d_i, it performs the second-level decision on whether to agree with the previous first decision on caching according to its own newest cache information. If so, the router caches d_i in its best caching position where its replacement profit is maximized. This operation is recorded as a cache refreshment which may later be broadcast to servers for information synchronization. Otherwise, if replacement profit for d_i in any position of r_k is less than 0, then no position for caching d_i will gain RE profit. It indicates that the router detects the information inconsistency error and wrong cache decision by the source server. The router then makes its own decision on no caching and forwards to its destination directly, and also immediately records all its newest cache information as its refreshments to broadcast for information synchronization and error correction. The whole procedure is illustrated by the pseudo code in Algorithm 3.

VI. Performance Evaluation

Our experiments include evaluations of offline and online solutions. We show that the approximation ratio of the offline algorithm is good, while our online solution outperforms previous proposals and some other naive proposals. The comparison between online algorithms shows the trade-off between the 'logically centralized' control optimality and the overhead on handling inconsistency distributed information in DCNs.

A. Evaluation of Offline Solution

The evaluations of the offline algorithm are performed in a small scale binary tree topology with 3 routers and 4 servers. There are 12 paths in the simulating tree.

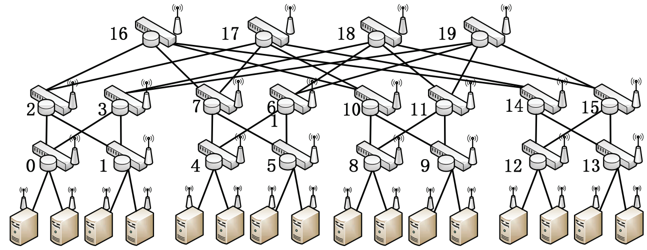


Fig. 5. The topology of online solution evaluation.

To evaluate the offline solution, we define the **approximation ratio** as the ratio between the total cache utility of all data-router pairs that are assigned by our greedy offline algorithm OFTRE (OFTRE for simplicity below) and that by an optimal algorithm. Since we proved that the problem is NP-complete, the optimal caching assignment is obtained by brute-force computing in our simulation, which traverses all possible caching assignments for the optimal total cache utility.

In the simulation, we test three cases with the number of data units 10, 20, 30 respectively. In each trial, the input *delivery counts matrix* E is generated randomly. We can plot the CDF curve of the *approximation ratio*. For example, the point $(0.9, 0.72)$ in CDF curve for data unit number 30 means that 72 percent of trials produces approximation ratio of 0.9 or below. As shown in Fig. 4, the approximation ratios of OFTRE are generally in (0.8,0.9) range which is significantly larger than 0.5 which is the lower bound given by Theorem 2. Furthermore, when the number of data units is larger, the *approximation ratio* is generally larger, and our OFTRE solution is closer to the optimal solution. It indicates that for data-intensive data centers networks, our OFTRE solution will be much closer to the optimal solution.

B. Evaluation of Online Solution

The evaluations of our online solutions are performed under the fat-tree topology (see Fig. 5) proposed by M. Al-Fares et. al. [2]. Before showing the simulation results, some notations used are given bellow.

Overall redundancy, denoted by ρ, is the total redundancy of the traffic transferred in the network. Suppose the average number of each data unit arriving over a fixed time window is m. To capture the characteristics that ρ is in the range of $[0, 1]$ and will increase when m gets larger, ρ is defined as $\rho = 1 - \frac{1}{m}$.

PathNum, the maximal number of the paths which a data unit can pass through, is denoted by ψ. In our experiment, inspired by [10], 20% of the data generate 80% traffic redundancy and the redundancy of the data units obeys the normal distribution. The time to send each of n copies of the data unit from server to server obeys the normal distribution (with the mean 800 unit times and the deviation of 200).

To compare the effectiveness of RE, we use the following metrics: 1) The **normalized RE utility** of an RE scheme is the ratio between the total RE utility of all the data units transferred and the total footprint. 2) The **cache usage rate** denotes the average times each cache used to eliminate redundant traffic per unit time. Suppose that r_k, $k = 1, 2, ..., |R|$,

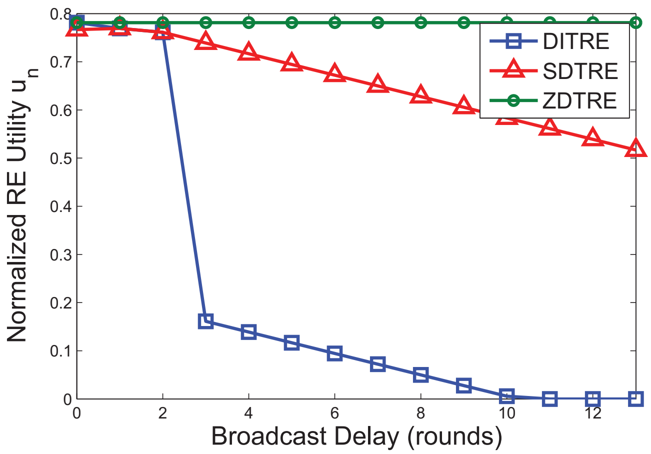


Fig. 6. Comparison of ZDTRE, DITRE and SDTRE under different broadcast delay settings.

can cache up to C data units, then the *cache usage rate* of RE can be computed by $\gamma = \frac{h}{|R|*C}/t$, where h is the total times of all the $|R| * C$ caches used to decode the encoded form into original one throughout the simulating time t.

We first compare online algorithms ZDTRE, DITRE and SDTRE under different broadcast delay settings. The broadcast delay denotes the average transmission time of one update message in a router to broadcast. We use ZDTRE as a benchmark for an upper bound of optimality to all the possible designed algorithms that act upon different delay settings. DITRE and SDTRE are 'logically centralized' versions; they are sharing distributed cache information among routers and servers with wireless medium and the decisions are made according to the global cache information. We set different broadcast delay value to test the total normalized RE utility of ZDTRE, DITRE and SDTRE. The value of delay setting is the multiplier of *broadcast rounds* (the average time for all routers to broadcast once). The average transmission time of updates in routers will be set as several rounds according to the delay setting.

As Fig. 6 shows, when broadcast delay is small, DITRE has somewhat larger RE utility profit than SDTRE, due to the less optimal strategies that SDTRE pays to overcome information inconsistency. At the beginning two algorithms gain almost the same amount of RE utility profit, and the larger the broadcast delay, the less profit they can gain. Thus increased broadcast delay causes increased inconsistency errors in information sharing. SDTRE performs better than DITRE when the broadcast delay is larger because SDTRE has some strategies to avoid, detect and handle the inconsistency errors. DITRE has no strategy to handle the inconsistency information and deteriorates quickly when certain delay is reached with the small setting of cache capacity $C = 10$ for quickly accumulating inconsistency errors. It shows the trade-off between control optimality and the overhead we pay to handle the inconsistency information.

We now focus on the evaluations of SDTRE. We first evaluate the normalized RE utility of different schemes and parameters, and then we further evaluate cache usage rate and

TABLE II
COMPARISON SCHEMES

Schemes	Task Assignment	Broadcast	Cache Replacement
SDTRE	Utility, feedback	Utility	Utility
p-DTRE	Utility	Utility	Utility
RS	Randomly	Utility	Utility
FIFO	Utility	Utility	FIFO
SmartRE	Randomly	–	FIFO
FIFB	Utility	FIFB	Utility
OPT	–	–	–

the inter-source redundancy elimination of online redundancy elimination in data centers and its broadcast mechanism. The comparison schemes are listed in Table II. p-DTRE is the scheme we previously proposed in [1]. It has utility based broadcast and cache replacement. But it has no feedback of average replacement rate and average replacement cost, and it also has no inconsistency error handlers. RS (Randomly assign Scheme) is the scheme which assigns the caching task randomly to routers. In FIFO scheme, when r_k caches new data unit, the oldest data unit cached is replaced to make room for this new data unit (if necessary), ignoring the utilities. In smartRE scheme [22], the FIFO cache replacement is applied. Each server registers the data units sent by itself and assigns the caching task based on the fingerprints, but is not aware of other servers' cache decisions. Therefore the cross-source redundancy is not addressed. In FIFB, when a router begins transmitting the caching information, the information of the oldest cache is transmitted first. OPT is the ideal caching scheme, where routers have infinite capacities and cache all data unit passing through them, and then the best router selected for each data unit is the last one on its routing path; this method serves as the benchmark.

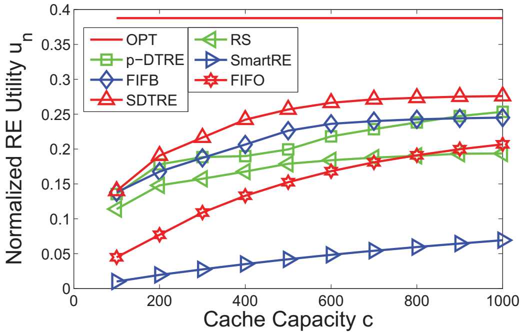
Fig. 7 shows normalized RE utility under different parameters: (a) different cache capacities with the overall redundancy 0.5 ($\rho = 0.5$), each data unit can be transmitted through at most 2 paths ($pathNum = 2$) and the broadcast capacity being 10 ($\omega = 10$); (b) different broadcast capacities with the overall redundancy 0.5 ($\rho = 0.5$), each data unit can be transmitted through at most 2 paths ($pathNum = 2$) with cache capacity 1000 ($C = 1000$); (c) different $pathNum$ with overall redundancy 0.5 ($\rho = 0.5$), the broadcast capacity 10 ($\omega = 10$), the cache capacity 1000 ($C = 1000$); (d) different overall redundancy with $pathNum = 2$, $C = 1000$, $\omega = 10$.

As shown in Fig. 7, in all these scenarios, SDTRE outperforms all other schemes. In some scenarios, the performance of SDTRE can be twice that of some other schemes.

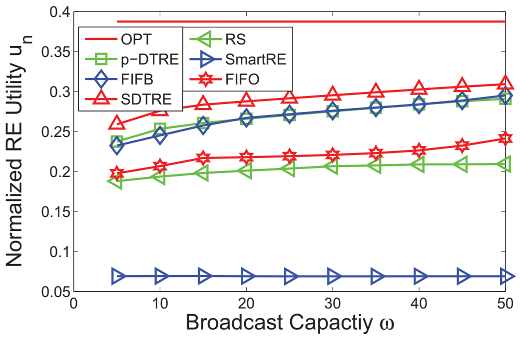
Replacement Strategy: Fig. 7(a) shows that with the reduction of the cache capacity, the performance of SDTRE decreases more slowly than that of FIFO and SmartRE schemes, meaning that the cache usage of SDTRE is more efficient when the cache capacity is small.

Fig. 8 shows the cache usage rate under different cache capacities for these schemes. The average times each cache used to eliminate redundant traffic per unit time is much larger when utility-based cache replacement strategies SDTRE, FIFB, p-DTRE, RS are deployed, especially when the cache capacity is small, compared to SmartRE and FIFO schemes.

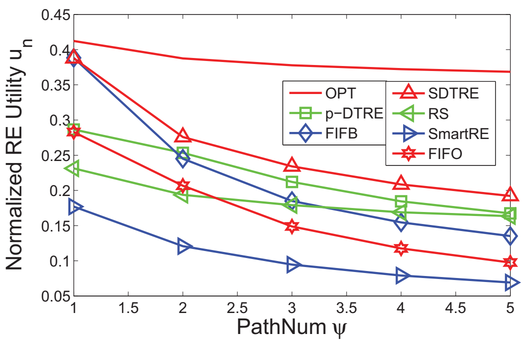
Caching task assignment: As shown in Fig. 7, the normalized RE utility of the scheme RS is always smaller than that



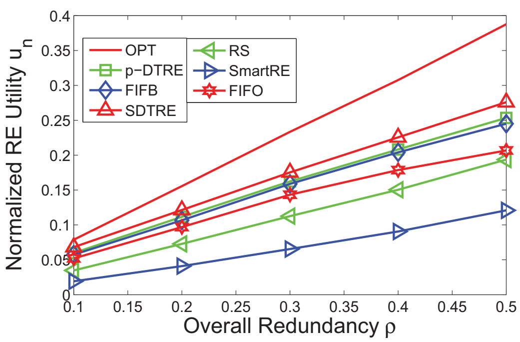
(a) Varying cache capacities



(b) Varying broadcast capacities



(c) Varying pathNum



(d) Varying redundancy

Fig. 7. Evaluation of the normalized RE utility under different schemes.

of SDTRE, between whom the only difference is the caching task assignment strategy. Then we focus on the comparison of the scheme SDTRE and RS when the overall redundancy is 0.5, the cache capacity is 100 and the broadcast bandwidth is sufficient. Comparisons under other parameter choices are similar.

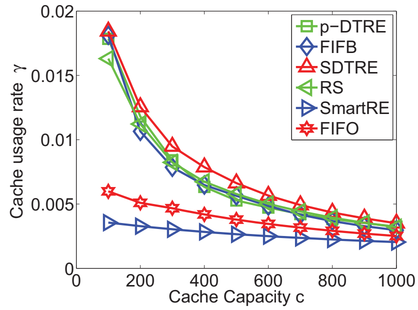


Fig. 8. Cache usage rate of different schemes' performance when $pathNum = 2$, $\omega = 10$, $\rho = 0.5$

Fig. 9(a) shows that SDTRE has much larger average RE utility of caches than RS, because a data unit is always cached in the best "position" with the maximum network-wide utility, and most of these positions are close to the destination of other copies of the data unit. Fig. 9(b) shows that the cache usage rates of SDTRE are always larger than that of RS. In SDTRE, data units are cached in the positions where two or more paths of this data unit intersect. However, they are randomly cached in RS, resulting in less efficient redundancy elimination performance.

Broadcast mechanism: As shown in Fig. 7(b), with the reduction of the broadcast capacity, the performance of SDTRE decreases more slowly than FIFB. Let h and h' be the total number of times of all the caches used to decode the encoded form data units when RE mechanism is applied, and if cache information can be known by servers without delay, respectively, then the efficiency of broadcasting is defined as $\phi = \frac{h}{h'}$.

The efficiency of broadcasting in SDTRE is always larger than that in FIFB (Fig. 10), because in SDTRE, the information of data units whose utility values are larger are broadcast first and these cached data units are more likely to perform the decoding procedure which eliminates the redundancy traffic. Furthermore, when the broadcast capacity gets smaller, the difference of the efficiency of broadcasting in SDTRE and FIFB schemes gets larger.

Routers' loads: In this section, we compare the SDTRE scheme and the p-DTRE scheme [1]. The **router's caching load** is the number of data units cached per unit time by the specific router; the **router's decoding load** is its number of data units decoded from the encoded form per unit time. Routers are divided into three groups: layer 1 routers (closest to the servers), layer 2 routers, layer 3 routers (farthest from the servers) (see Fig. 5).

Table III shows the loads of routers. The loads of layer 2 routers are lower than that of other servers, because of traffic in both directions and other routers being closer to some servers. Further, both the caching and decoding loads are lower in p-DTRE than in SDTRE. If the decoding loads are divided by the caching loads, then we can obtain the average times of each data units that are cached by routers used to perform decoding task, shown in Table III. The caching loads are positively correlated to the cache replacement rate. Data units cached by the routers in SDTRE are being replaced faster than that in p-DTRE, resulting in SDTRE's higher performance.

TABLE III
ROUTERS' LOADS

Caching Loads of Routers

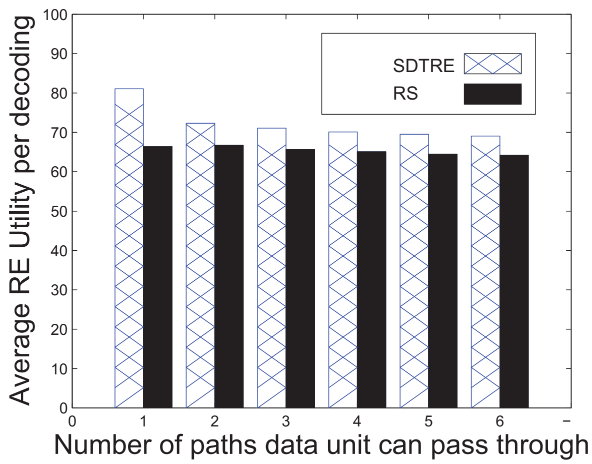
Scheme/Id	Layer 1 Routers								Layer 2 Routers								Layer 3 Routers			
	0	1	4	5	8	9	12	13	2	3	6	7	10	11	14	15	16	17	18	19
p-DTRE	2.079	1.958	2.122	1.982	2.162	1.973	3.398	3.269	0.582	0.412	0.613	0.414	0.594	0.413	0.58	0.324	2.294	2.468	2.481	2.775
SDTRE	2.362	2.197	2.399	2.253	2.436	2.234	3.701	3.609	0.716	0.52	0.756	0.527	0.727	0.526	0.786	0.392	2.762	2.8	2.794	3.159

Decoding Loads of Routers

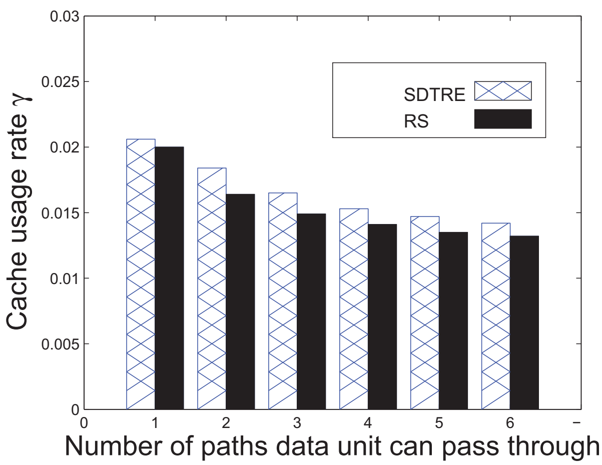
Scheme/Id	Layer 1 Routers								Layer 2 Routers								Layer 3 Routers			
	0	1	4	5	8	9	12	13	2	3	6	7	10	11	14	15	16	17	18	19
p-DTRE	3.781	3.345	3.710	3.466	3.927	3.435	5.053	4.864	1.815	1.281	1.782	1.343	1.75	1.339	1.673	1.019	5.331	5.418	5.56	4.877
SDTRE	3.990	3.546	3.93	3.682	4.078	3.648	5.923	5.649	1.871	1.391	1.892	1.449	1.869	1.443	1.866	1.145	5.732	5.75	6.107	5.124

Average times data units cached to perform decoding task

Scheme/Id	Layer 1 Routers								Layer 2 Routers								Layer 3 Routers			
	0	1	4	5	8	9	12	13	2	3	6	7	10	11	14	15	16	17	18	19
p-DTRE	1.819	1.708	1.749	1.749	1.816	1.741	1.487	1.488	3.118	3.108	2.905	3.247	2.948	3.244	2.884	3.142	2.324	2.1956	2.241	1.757
SDTRE	1.689	1.614	1.638	1.635	1.674	1.633	1.601	1.565	2.613	2.674	2.502	2.75	2.571	2.744	2.375	2.919	2.076	2.054	2.186	1.622



(a) Average RE utility per decoding



(b) Cache usage rate

Fig. 9. Both average RE utility per decoding and cache usage rate in SDTRE are larger than that in RS.

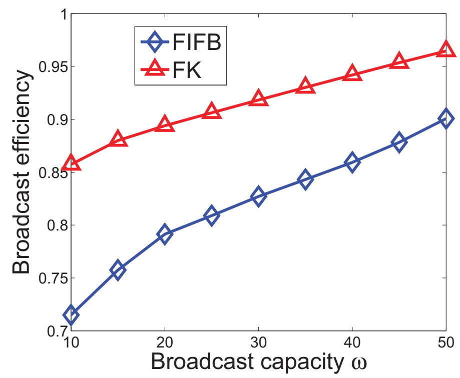


Fig. 10. The broadcast efficiency comparison: SDTRE is better than FIFB.

VII. Conclusion and Future Work

We present the first mechanism for traffic redundancy elimination in data center networks (DCNs). We are also the first to consider cross-source redundancy to further improve the performance of the RE mechanism utilizing broadcasting of data centers. We proposed a framework called TREDaCeN in which interior nodes can broadcast their cache information, and present both offline and online solutions. Simulation results show good performance of our proposed algorithms. Our results demonstrate that wireless communication is a viable approach to achieve 'centralized control' among servers and routers and confirm the trade-off between the 'logically centralized' control optimality and the overhead on handling inconsistency distributed information in DCN. Our proposed scenario and problem statement appears to be a good example in support of emerging software defined networks concept. Further research can address the impact of realistic physical layer, such as lognormal shadowing model, instead of ideal physical layer considered here (supported by close proximity of servers and routers) on the robustness of the system.

In our TREDaCeN framework, we suppose that new data units arrive at servers with only one desired routing path that connects its source server to its destination. However, in practical settings, e.g., in the fat-tree topology, there are multiple paths between two servers. Thus expectancy matrix could be defined between server pairs instead of path specific ones. This multi-path problem is left for future work. Next, some routing tasks of data units could be replaced by wireless broadcast, which can save a lot of traffic cost in densely connected DCN. However, this reduces the time and efficiency of sending control messages when the bandwidth is insufficient to support all the traffic. Furthermore, data units cached by routers could be migrated among routers instead of the current cache replacement scheme according to the traffic.

The extensions of this work could provide further insights into these aspects of the problem.

REFERENCES

[1] Y. Cui, C. Liao, I. Stojmenovic, D. Li, and H. Wang, "Cooperative redundancy elimination in data center networks with wireless cards at routers," in *ICDCS Workshops 2012*.

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

[3] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "Dcell: a scalable and fault-tolerant network structure for data centers," in *Proc. SIGCOMM 2008*.

[4] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "Bcube: a high performance, server-centric network architecture for modular data centers," in *Proc. SIGCOMM 2009*.

[5] M. Alizadeh, A. Greenberg, D. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (dctcp)," in *Proc. SIGCOMM 2010*.

[6] S. Kandula, J. Padhye, and P. Bahl, "Flyways to de-congest data center networks," in *Proc. HotNets 2009*.

[7] Y. Cui, H. Wang, X. Cheng, and B. Chen, "Wireless data center networking," *Wireless Communications*, vol. 18, no. 6, pp. 46–53, 2011.

[8] Y. Cui, H. Wang, and X. Cheng, "Wireless link scheduling for data center networks," in *Proc. ICUIMC 2011*.

[9] Y. Cui, H. Wang, X. Cheng, D. Li, and Y.-J. Antti, "Dynamic scheduling for wireless data center networks," *IEEE Trans. Parallel Distrib. Syst. (TPDS)*, to appear.

[10] T. Benson, A. Anand, A. Akella, and M. Zhang, "Understanding data center traffic characteristics," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 1, pp. 92–99, 2010.

[11] D. Halperin, S. Kandula, J. Padhye, P. Bahl, and D. Wetherall, "Augmenting data center networks with multi-gigabit wireless links," in *Proc. SIGCOMM 2011*.

[12] A. Anand, C. Muthukrishnan, A. Akella, and R. Ramjee, "Redundancy in network traffic: findings and implications," in *Proc. SIGMETRICS 2009*.

[13] N. Spring and D. Wetherall, "A protocol-independent technique for eliminating redundant network traffic," in *Proc. SIGCOMM 2000*.

[14] S. Ihm and V. Pai, "Towards understanding modern web traffic," in *Proc. SIGMETRICS 2011*.

[15] S. Kim, H. Han, H. Jung, H. Eom, and H. Yeom, "Improving mapreduce performance by exploiting input redundancy," *J. information science and engineering*, vol. 27, no. 3, pp. 1137–1152, 2011.

[16] A. Luotonen and K. Altis, "World-wide web proxies," *Computer Networks and ISDN systems*, vol. 27, no. 2, pp. 147–154, 1994.

[17] A. Wolman, M. Voelker, N. Sharma, N. Cardwell, A. Karlin, and H. Levy, "On the scale and performance of cooperative web proxy caching," *ACM SIGOPS Operating Systems Review*, vol. 33, no. 5, pp. 16–31, 1999.

[18] A. Vakali and G. Pallis, "Content delivery networks: Status and trends," *IEEE Internet Computing*, vol. 7, no. 6, pp. 68–74, 2003.

[19] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Proc. Middleware 2001*.

[20] 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 *Proc. NSDI 2010*.

[21] A. Anand, A. Gupta, A. Akella, S. Seshan, and S. Shenker, "Packet caches on routers: the implications of universal redundant traffic elimination," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4, pp. 219–230, 2008.

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

[23] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, "Logically centralized?: state distribution trade-offs in software defined networks," in *Proc. HotSDN 2012*.

[24] J.-Y. Shin, E. G. Sirer, H. Weatherspoon, and D. Kirovski, "On the feasibility of completely wireless datacenters," in *Proc. eighth ACM/IEEE symposium on Architectures for Networking and Communications Systems*. ACM, 2012, pp. 3–14.

[25] H. Vardhan, N. Thomas, S.-R. Ryu, B. Banerjee, and R. Prakash, "Wireless data center with millimeter wave network," in *Proc. GLOBECOM 2010*.

[26] X. Zhou, Z. Zhang, Y. Zhu, Y. Li, S. Kumar, A. Vahdat, B. Y. Zhao, and H. Zheng, "Mirror mirror on the ceiling: Flexible wireless links for data centers," in *Proc. SIGCOMM 2012*.

[27] S. Saha, A. Lukyanenko, and A. Yla-Jaaski, "Combiheader: Minimizing the number of shim headers in redundancy elimination systems," in *INFOCOM Workshops 2011*.

[28] E. Halepovic, C. Williamson, and M. Ghaderi, "Enhancing redundant network traffic elimination," *Computer Networks*, vol. 56, no. 2, pp. 795–809, 2012.

[29] M. Bläser and B. Siebert, "Computing cycle covers without short cycles," *Algorithms-ESA*, pp. 368–379, 2001.

[30] Y. Cui, S. Xiao, C. Liao, I. Stojmenovic, and M. Li, "Data centers as software defined networks: Traffic redundancy elimination with wireless cards at routers," Department of Computer Science and Technology, Tsinghua University, Tech. Rep., 2013. [Online]. Available: http://www.4over6.edu.cn/others/TREDaCeN.pdf

Yong Cui Ph.D., Professor in Tsinghua University, Council Member in China Communication Standards Association, Co-Chair of IETF IPv6 Transition WG Softwire. Having published more than 100 papers in refereed journals and conferences, he is also the winner of Best Paper Award of ACM ICUIMC 2011 and WASA 2010. Holding more than 40 patents, he is one of the authors in RFC 5747 and RFC 5565 for his proposal on IPv6 transition technologies. His major research interests include mobile wireless Internet and computer network architecture.

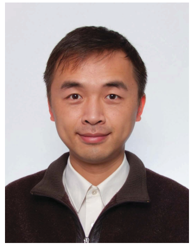
Shihan Xiao received the B.Eng. degree in Electronic and Information Engineering from Beijing University of Posts and Telecommunications, Beijing, China, in 2012.

He is currently working towards his Ph.D. degree in the Department of Computer Science and Technology, Tsinghua University, Beijing, China. His research interests are in the areas of wireless networking and mobile cloud computing.

Chunpeng Liao received the M.Eng. degree from the Department of Computer Science and Technology, Tsinghua university, Beijing, China, in 2013.

His research interests are in the areas of wireless networking and data center architectures.

Ivan Stojmenovic received his Ph.D. degree in mathematics. He is Full Professor at the University of Ottawa, Canada. He held regular and visiting positions in Serbia, Japan, USA, Canada, France, Mexico, Spain, UK (as Chair in Applied Computing at the University of Birmingham), Hong Kong, Brazil, Taiwan, China and Australia. He published over 300 different papers, and edited seven books on wireless, ad hoc, sensor and actuator networks and applied algorithms with Wiley. He is editor-in-chief of IEEE Transactions on Parallel and Distributed Systems (2010-3), and founder and editor-in-chief of three journals. He is Associate Editor-in-Chief of Tsinghua Journal of Science and Technology, steering committee member of IEEE Transactions on Emergent Topics in Computing, and editor of IEEE Network, IEEE Transactions on Cloud Computing, IEEE Transactions on Computers, ACM Wireless Networks and some other journals. Stojmenovic is on Thomson Reuters list of Highly Cited Researchers (from 2013; <300 computer scientist), has h-index 59, top h-index in Canada for mathematics and statistics, and >13000 citations. He received five best paper awards and the Fast Breaking Paper for October 2003, by Thomson ISI ESI. He received the Royal Society Research Merit Award, UK (2006), and Humboldt Research Award, Germany (2012). He is Tsinghua 1000 Plan Distinguished Professor (2012-5). He is Fellow of the IEEE (Communications Society, class 2008), and Canadian Academy of Engineering (since 2012), and Member of the Academia Europaea (The Academy of Europe), from 2012 (section: Informatics). He was IEEE CS Distinguished Visitor 2010-11 and received 2012 Distinguished Service award from IEEE ComSoc Communications Software TC. He received Excellence in Research Award of the University of Ottawa 2009. Stojmenovic chaired and/or organized >60 workshops and conferences, and served in >200 program committees. He was program co-chair at IEEE PIMRC 2008, IEEE AINA-07, IEEE MASS-04&07, founded several workshop series, and is/was Workshop Chair at IEEE ICDCS 2013, IEEE INFOCOM 2011, IEEE MASS-09, ACM Mobihoc-07&08.



Minming Li received the BE and PhD degrees from the Department of Computer Science and Technology, Tsinghua University, Beijing, China, in 2002 and 2006, respectively.

He is currently an assistant professor in the Department of Computer Science, City University of Hong Kong. His research interests include wireless ad hoc networks, algorithm design and analysis, and combinatorial optimization.