# Multiple Tree-Based Online Traffic Engineering for Energy Efficient Content-Centric Networking

Ling Xu, Tomohiko Yagyu

Cloud System Research Lab
NEC Corporation, Japan
Email: lingxu@gisp.nec.co.jp, yagyu@cp.jp.nec.com

***Abstract*—Content-Centric Networking (CCN) is a new network architecture aiming to solve many fundamental problems of existing IP networks. CCN is unique in that it widely deploys caches on routers to reduce redundant data transmission. Introducing caches into routers, however, causes the networks to consume extra energy. Although great research effort has been put for improving energy efficiency in IP networks, little work has been done for CCN networks yet. We designed an online traffic engineering algorithm called Multiple Tree-based Traffic Engineering (MTTE) to fill this void. Our approach is to make traffic flow on a small portion of edges in the network and shut down underutilized edges. Data are delivered on multiple tree-topology networks that are dynamically created based on the physical network topology and the current network traffic pattern. The trees are carefully constructed so that they contain minimal edges and mitigate congestion. Simulations using network topologies of real-world autonomous systems show that by using MTTE, up to 45% of the edges can be shut down. To the best of our knowledge, MTTE is the first online green mechanism for CCN.**

***Keywords–Content-centric networking; energy efficiency; multiple trees; traffic engineering.***

## I. Introduction

Recently, *Content-Centric Networking* (CCN) has been drawing considerable attention from the industry and academic community [1]. In conventional IP networks, a great part of data transmission is redundant. CCN reduces redundant data transmission by deploying caches on all routers in the networks. A CCN network consists of hosts and routers. Each host holds contents, and each content has a unique *name*. Each name consists of a *prefix* and a *file name*. For example, the name “/Asia/Tokyo/music1.mp3” contains prefix “/Asia/Tokyo/” and file name “music1.mp3”. For host $p$ and its content $d$, $p$ registers $d$’s name on each router. This process is called *content publishing*, and $p$ is known as the *producer* of $d$. Suppose that another host, say $c$, needs $d$. $c$ sends an *Interest* $i$ to its adjacent router. The Interest is then forwarded toward $p$ according to certain forwarding policies. We refer to $c$ as a *consumer* of $d$. Having received $i$, producer $p$ packs $d$ into *content object* packets, denoted as $co(d)$, and sends $co(d)$ back to $c$. Each router $r$ along $co(d)$’s forwarding path tries to store $d$ in its caches. The next time $r$ receives an Interest for $d$, if $d$ is still in its cache, $r$ sends $co(d)$ to the consumer directly. In this paper, for ease of exposition, we do not differentiate hosts from routers. For host $h$ and its adjacent router $r$, when $h$ issues an Interest for content $d$, we simply regard $r$ as the consumer of $d$; when $h$ is a producer of $d$, we regard $r$ as $d$’s producer.

We are concerned about CCN’s energy consumption. Networks consume a huge amount of energy, and improving their energy efficiency has been a hot research topic in recent years. CCN networks generally consume more energy than conventional IP networks since in-router caches cost extra energy.

Recent studies have evaluated CCN’s energy efficiency [2][3]. However, no effective mechanism has been proposed for reducing CCN’s energy consumption.

Although many energy-saving techniques have been proposed in IP networks, they cannot be readily transplanted into CCN networks. One of the general ideas for reducing network energy is to shut down underutilized edges [4]. Routers are the main hardware components and main energy consumers in networks. A router consists of a Central Processing Unit (CPU) and a set of network interfaces. Each interface connects an adjacent router via a piece of cable. Henceforth, we use *edges* to denote interfaces. The general idea for conserving energy is to shut down CPUs and edges, and the critical challenge is to shut down edges without remarkably reducing the system performance. In IP networks, one of the most commonly used techniques for this challenge is to (1) estimate the traffic matrix (i.e., the amount of traffic that will flow between each pair of edge routers) periodically, and (2) based on the estimated traffic matrix, find the edges whose shutdown minimizes performance degradation using linear programming [5]. In CCN networks, however, all routers can cache and provide all kinds of content. The traffic patterns are more complicated than in IP networks. Hence, estimating the traffic matrix is difficult.

Our objective is to design a mechanism that reduces CCN’s energy consumption without remarkably reducing the network performance. To this end, we proposed *Multiple Tree-based Traffic Engineering* (MTTE). Previous research has found that edges in modern networks are generally underutilized [4]. Our idea is to shut down as many underutilized edges as possible. We split traffic on multiple tree-topology networks generated based on the physical network. The trees are generated in such a way that the number of edges included in the trees is minimized. Since trees generally contain fewer edges than the original physical network, energy can be conserved.

We face two challenges in our design. First, as fewer edges are used for forwarding traffic, the network is more vulnerable to traffic congestion. We assume that when a network system is heavily congested, most likely the congestion is caused by a few bottleneck edges. To reduce congestion, we need to reduce the load on the congested edges. In MTTE, the network contains a centralized server called the *controller*. The controller dynamically monitors the congestion status of the system. When the system congestion is severe, the controller generates more trees so that the congestion on the bottleneck edges can be relieved. When the maximal utilization among all edges is low, the controller merges traffic on fewer trees and shuts down more edges.

The second challenge is that creating new trees potentially increases transmission latency. In a native CCN network, contents are delivered on the full physical network; in MTTE, contents are delivered on sub-networks. The (shortest path) distance between each pair of consumer and producer in MTTE is essentially greater than in native CCN network. Hence, the mean length of physical forwarding paths between consumers and produces in MTTE, denoted by the *Mean Forwarding Length* (MFL), is greater. Moreover, under the original CCN protocol, when a content is forwarded, all routers along the forwarding path will try to cache this content. As the MFL increases, the same content is likely to be cached on more routers. Previous studies have shown that repeatedly caching the same content reduces caching efficiency [6]. Hence, adding more trees naively is likely to further increase the system latency. We address this challenge by reducing the diameters of the trees.

We compare the performance of MTTE and native CCN under two metrics, *Live Edge Rate* (LER) and the system latency, via simulations running on real-world autonomous system topologies. Here, *live edges* are the edges contained in trees, and other edges are called *free edges*. LER is the ratio between the average number of live edges used during the simulation and the total number of edges in the physical network. The system latency is the average time between issuing Interests and getting content objects. Simulation results reveal that MTTE can shut down up to $45\%$ of the edges while maintaining a latency comparable to CCN. Furthermore, under heavy congestion, MTTE can shut down $40\%$ of the edges and achieve a latency $90\%$ lower than CCN.

In short, we propose an online energy-saving mechanism for CCN which, to the best of our knowledge, is the first of its kind.

We detail the design of MTTE in Section III and evaluate MTTE's performance via simulation in Section IV. We highlight the relevant prior literature in this field in Section II and present our conclusion in Section V.

## II. Related Work

In conventional IP networks, *Traffic Engineering* (TE) is a kind of well-researched mechanism that adjusts routing paths of traffic for relieving congestion, balancing traffic and reducing energy consumption [4]. MTTE is one of the few TE schemes designed for CCN networks. TE mechanisms can be implemented either offline [7] or online [8]. Offline TE mechanisms need to estimate the traffic matrix. Based on the traffic matrix, the TE protocols find the routing paths that minimize the energy consumption of the whole network using linear programming. In CCN, however, due to the wide existence of caches, the traffic between each pair of edge routers are more dynamic, and the traffic matrix would be hard to predict. In contrast, online TE mechanisms monitor the real-time traffic of the network and dynamically change routing paths according to the traffic fluctuation. MTTE is, to the best of our knowledge, the first online TE mechanism for CCN networks that both reduces energy consumption and relieves congestion.

Online energy-aware TE protocols for IP and MPLS networks have been proposed recently [9][10]. Vasić et al. assume that network edges work on multiple fixed energy consumption levels, and edges can switch to lower energy consumption levels when their utilization is low [9]. Each pair of routers maintain multiple routing paths. Each router periodically adjusts the traffic partition among its routing paths so that more edges can shift to lower energy consumption levels. Coiro et al. propose EATE, a distributed energy-aware architecture [10]. EATE is created above the MPLS protocol stack. Routers in the network run a modified OPSF-TE algorithm to periodically compute the shortest paths to each other based on the hop counts and the numbers of sleeping edges along the paths. When congestion occurs, nodes create new routing paths to bypass the congested area. However, these TE schemes are designed for connection-oriented architectures. Whether they can be readily used for CCN networks is unknown.

Chanda et al. implemented an online TE scheme for *Information-Centric Networking* (ICN) for balancing traffic [2] (CCN is a specific architecture implementation of the information-centric networking philosophy). The research objective of [2] is to demonstrate that TE mechanisms can be implemented more efficiently on CCN networks than on IP networks. Xie et al. [3] implemented a TE protocol in CCN with the goal of improving the caching efficiency, but how [3] would affect CCN's energy efficiency is unclear.

Research effort has been made in assessing CCN's energy efficiency using simulation [11][12]. Many of these researches compare the energy efficiency of CCN with existing IP-based content delivery techniques such as content delivery networking and peer-to-peer networking. Song et al. [13] noticed that in modern carrier networks, a great amount of traffic is generated from the edge. [13] uses GreenTE - an existing energy-aware TE mechanism designed for IP networks [4] - for reducing energy consumption of the core network, and uses CCN for eliminating redundant traffic generated by the edge network. However, Song. et al.'s approach does not reduce the energy consumption of CCN itself.

## III. Multiple Tree-based Traffic Engineering

To reduce energy consumption, our idea is to shut down as many underutilized edges as possible. From an energy-saving viewpoint, delivering all contents on a single spanning tree would be the most favorable option. This extreme case, however, is impractical since a single tree is vulnerable to congestion and results in the high latency. Hence, we try to deliver the traffic on multiple trees to minimize edges contained in the trees and relieve congestion at the same time.

We use $G$ to denote the whole network. In MTTE, $G$ contains a central server called the controller. We use $STs$ to denote the set of trees created in MTTE. Initially, the controller creates one spanning tree based on the physical network topology and adds this tree to $STs$.

### A. Congestion Detection

The controller periodically measures system congestion status and based on this, decides whether to create new trees or not. Let us briefly explain why congestion and latency occurs.

In a standard network, for each router and each edge of this router, the edge maintains a packet *queue* of a fixed length and has a fixed *capacity*. The edge's utilization is the ratio between the speed at which packets come to this edge to the edge's capacity. When utilization $> 1.0$, new incoming packets are added to the queue's tail and await forwarding. We call the edges with utilization higher than $\phi_{CE}$ the *congested edges*, where we empirically set $\phi_{CE}$ to be 0.9. When packets traverse the congested edges, the system latency is likely to grow. Each router periodically reports the utilization of its adjacent edges to the controller. The controller calculates the *Congestion Rate* (CR) as the maximal utilization of edges in the networks. When $CR > \phi_{tcc}$, where $\phi_{tcc}$ is a system parameter which we empirically set to be 0.9, the controller creates one more tree and asks routers to deliver the traffic on each tree evenly. We call this mechanism the *Tree-based Congestion Control* (TCC).

### B. *Creating a New Tree*

When the controller creates a new tree, we keep three objectives in our minds: (1) the new tree should introduce free edges into $E(STs)$ so that less traffic traverses the congested edges; (2) the new tree should not dramatically increase the live edge rate; (3) the new tree should not remarkably increase latency. Here, $E(STs)$ represents the set of live edges. Basically, the new tree is created using Kruskal's minimal spanning tree algorithm [14], while the three goals are realized by deciding which edges should be added into the new tree.

To realize goals (1) and (2), the controller assigns weights to edges so that the *uncongested edges* are chosen first, free edges are chosen later, and congested edges are chosen last. Here, uncongested edges are the live edges that have utilization lower than $\phi_{CE}$.

We realize goal (3) by reducing the diameters of the trees. By doing so, we can reduce MFL and consequently, the system latency. We need to create a spanning tree $st$ with a small diameter from the underlay physical network. Although theoretical research has been performed on creating minimal diameter trees [15], we use our own heuristic algorithm for simplicity. For each edge $e$, we compute its "edge betweenness" ($e.be$) - a widely used metric in graph theory [16]. Informally, $e.be$ represents the number of shortest paths in the entire network that traverse $e$. Imagine that routers $c$ and $p$ are a pair of consumer and producer, and $sp$ is the shortest path between $c$ and $p$ on $G$. Intuitively, if more edges with high betweennesses are added to $st$, the probability that the data transmitted on $st$ between $c$ and $p$ are delivered along the shortest path is higher. Accordingly, the diameter of $st$ will be small. Based on this observation, in MTTE, the controller selects edges with higher betweennesses first. When the controller creates the initial tree, for each edge $e$, it calculates $e.eb$, and sets $e.weight = 1/e.eb$. When subsequent trees are created, the controller makes the weights of uncongested edges directly proportional to the edges' utilization, and makes the weights of free edges inversely proportional to the edges' betweennesses. If the new tree is different from all the existing trees, the controller adds the new tree into $STs$; otherwise, the network has no more capacity for mitigating congestion, the tree creation fails and the system stays unchanged. Namely, the system will not add trees permanently.

The complete tree-creation algorithm is shown in Figure 1.

```
$E_{UE}$ = Edges in $E(STs)$ with utilization $<= \phi_{CE}$.
$E_{CE}$ = Edges in $E(STs)$ with utilization $> \phi_{CE}$.
$E_{free}$: Edges not in $E(STs)$
$U_{max}$ = maximum of edge utilization of $E_{UE}$

The controller calculates the betweenness $e.eb$ of each edge $e$.

if $|STs|$ = 0 then
  // The controller is creating the initial tree
  for all $e$ in the network do
    $e$.weight = $1/e.eb$
  end for
else
  // The controller creates a new tree for mitigating congestion
  for all $e \in E_{UE}$ do
    e.weight = e.utilization
  end for
  for all $e \in E_{free}$ do
    e.weight = $U_{max} + 1/e.eb$
  end for
  for all $e \in E_{CE}$ do
    e.weight = $U_{max} + 2$
  end for
end if

The controller generates a minimal spanning tree $st$ using Kruskal ' s algorithm on the whole network.
if $st$ is different from all the existing trees then
  return $st$
else
  return FAILED
end if
```

Figure 1. Based on current edge utilization, the controller generates a new spanning tree.

### C. *Hash-based Traffic Splitting*

We briefly discuss CCN's packet (Interests and content objects) forwarding mechanisms. In CCN, each router $r$ contains a *Forwarding Information Base* (FIB). Each FIB contains a set of entries and each entry is a mapping from one prefix to a set of network interfaces. To explain CCN's forwarding process, suppose that $r$'s FIB contains two entries $fe0$=''/Asia/'':{face 2} and $fe2$=''/Asia/Tokyo/'':{face2,face5}, and suppose that an incoming packet has a name $n$="/Asia/Tokyo/music.mp3". $r$ searches in FIB for the entry $fe$ whose prefix matches $n$'s prefix in the longest length. In this example, $fe = fe2$. $r$ has multiple forwarding strategies. According to forwarding strategies, the incoming packets will be forwarded to one or multiple faces in $fe$.faces. How routers create their FIB entries and set their forwarding strategies is not standardized yet. In this paper, we suppose that routers create FIBs in such a way that packets are forwarded along one of the shortest paths between each pair of routers.

Each time $STs$ is changed, routers update their respective FIBs so that packets can be delivered on the new $STs$. We split the hash name space of CCN names into $|STs|$ sub-name spaces, denoted by $NS[1], ...NS[|STs|]$. Packets with names whose hash values belong to $NS[i]$ will be forwarded on the i-th tree. Specifically, the controller sends both the topologies of trees and $G$ to routers. For each producer $p$ and each content

$d$ stored on $p$, $p$ publishes $d$. We suppose that $H$ is a collision-proof hash function preloaded on each router. We use $N(d)$ to denote the CCN name of $d$. Producer $p$ broadcasts $N(d)$ along the $(H(N(d))\ mod\ |STs|)$-th tree. Suppose that two routers $r1$ and $r2$ are adjacent, during the broadcast, $N(d)$ traverses $r1$ first and then $r2$, and $f$ is $r1$'s face that connects $r2$. Upon receiving $N(d)$, router $r1$ adds an entry $N(d).prefix : f$ in its FIB.

### *D. Tree Removal*

When traffic in the network decreases, the controller shrinks $STs$ and makes routers forward packets on fewer trees. That is, when $CR < \phi_{lowUtil}$, the controller removes off the last tree in $STs$ and asks routers to update their FIBs. $\phi_{lowUtil}$ is a preloaded system parameter that we empirically set to be 0.6. Routers shut down their adjacent edges that are not included in $E(STs)$.

## IV. Evaluation

This section evaluates MTTE's performance by comparing the system latency and LER between MTTE and native CCN. Our simulation is performed on ndnSIM – a simulation platform developed by UCLA for CCN-related research [17].

### *A. Performance Metrics*

The system latency is calculated in the following manner. Each consumer $r$ issues *Interest Issuing Frequency* ($IIF$) Interests for random contents per second. CCN's forwarding mechanism ensures that if $r$ issues multiple Interests for the same content $d$ before receiving the corresponding content object, finally $r$ will receive no more than one content object of $d$. Upon receiving the content object, $r$ calculates a *local latency* as the time interval since $r$ issues the first Interest for $d$, until the time $r$ receives the first content object of $d$. At any time point, we calculate the mean value of the local latencies of all consumers since the simulation starts by now as the system latency.

LER is defined as $\gamma/|E|$, where $\gamma$ is the average number of live edges used in STs during the simulation, and $E$ is the total number of edges in the physical network. We use latency(MTTE) and latency(CCN) to denote the latencies of MTTE and CCN, respectively.

### *B. Simulator Setting*

Our simulations run on the network topology of autonomous system 3257 (AS3257). This topology is provided in Rocketfuel network dataset [18], a dataset that has been used in network research [19][20]. Each node in AS3257 represents a router. We extract the largest connected component of AS3257 and use all the remaining nodes for creating trees. AS3257 contains three types of routers: *cores*, *gateways* and *leaves*. According to the definition of Rocketfuel datasets, leaves are the routers with degrees equal to or less than two, gateways are the routers directly connected to the leaves, and the remaining routers are cores. The numbers of edges and routers in AS3256 are listed in Table I. We have also run simulations on other Rocketfuel autonomous system topologies and obtained consistent performance results.

We assume that in real world CCN networks, consumers are adjacent to leaves, and producers are adjacent to both gateways and leaves. In our simulation, we assign one producer to each leaf and each gateway, and assign one consumer to each leaf. Namely, totally 132 producers and 80 consumers are generated.

TABLE I. NETWORK PARAMETERS

| Parameter | Value |
|---|---|
| Total number of edges | 420 |
| Total number of routers | 240 |
| Number of gateway routers | 52 |
| Number of leaf routers | 80 |

Each producer generates ten random prefixes, and each prefix covers ten unique file names. Therefore, a total number of (producer count) $\times 10 \times 10$ names are generated.

In each second, each consumer issues IIF Interests with randomly selected names. The requested names are selected according to a Zipf distribution [21][22][6]: the $k$-th name is generated with a probability proportional to $1/k^{\alpha}$, where $\alpha$ is 0.7 in our simulations. Each simulation lasts 300 seconds. We evaluate the performance when the traffic is light (IIF=5) and heavy (IIF=15). The payload of each content object is 1024 bytes, the capacity of each edge is $10^6$ bps, $\phi_{CE} = \phi_{tcc} = 0.9$, and $\phi_{lowUtil} = 0.6$. For each parameter setting, we repeat the simulation ten times and measure the average results. Parameters $\phi_{tcc}$, $\phi_C$ and $\phi_{lowUtil}$ reflect the trade-off between transmission quality and energy efficiency. Generally, under the same traffic, more trees will be created and maintained when $\phi_{tcc}$ and $\phi_{lowUtil}$ are low. TCC will more aggressively choose free edges when $\phi_{CE}$ is low. In a real world CCN network, the network administrators can adjust the parameters themselves accordingly to the real needs of the system (high transmission quality or high energy efficiency).

### *C. Performance under High Traffic*

Figure 2 compares the latency between MTTE and CCN when traffic is high (IIF=15). It shows that as times passes by, latency(MTTE) decreases and latency(CCN) increases. At the time point of second 300, MTTE shuts down $40\%$ edges (Figure 3), and latency(MTTE) is $1/9$ of latency(CCN).

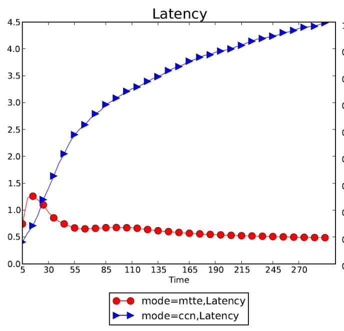

Figure 2. The comparison of the latency between MTTE and CCN when IIF=15. The horizontal and vertical axes represent the time the simulation has elapsed (in seconds) and the average latency (in seconds).

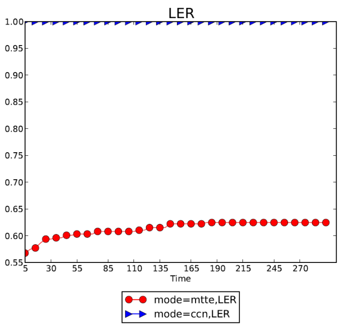

Figure 3. The comparison of LER between MTTE and CCN. The horizontal and vertical axes represent the simulation time and LER, respectively.

As argued in Section III, the system latency is mainly caused by the congestion on a few bottleneck edges (i.e., the congested edges). To validate this, we measure the mean and maximum of edge utilization over all edges (Figure 5 and Figure 6). The utilization of each edge is calculated as the ratio between the Exponentially Weighted Moving

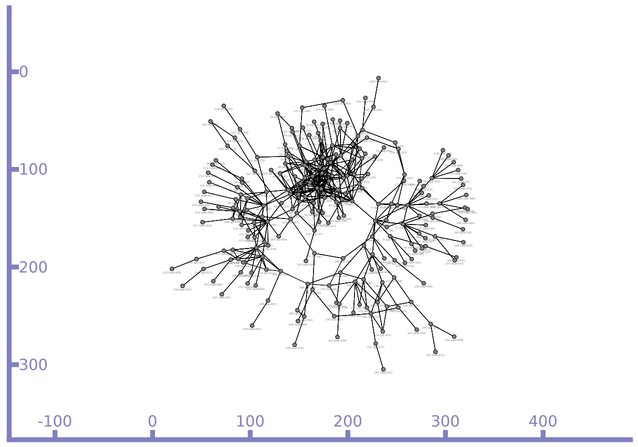
Figure 4. AS3257, the network topology used in user simulation.

Average (EWMA) of the traffic load on this edge to this edge's capacity. Theoretically, edge utilization ranges between 0.0 and 1.0. However, since the utilization is calculated in a EWMA manner, it may slightly exceed 1.0 when the network experiences congestion. In MTTE, the mean utilization is about 0.19, but CR is up to 1.0. We illustrate the topology of AS3257 in Figure 4. This figure also shows that several clusters exist in the network, where clusters are connected by a few edges. These edges correspond to the congested edges.

The reason that latency(CCN) increases with time is that the congested edges are overloaded, packet drop occurs, and consumers cannot receive the required contents. According to CCN's forwarding rules, the consumers will re-send their Interests, which makes the system even more congested and increases the system latency. The reason why latency(MTTE) decreases is that MTTE splits traffic onto multiple trees. As CR exceeds $\phi_{tcc}$ (Figure 6), new trees are generated (Figure 3). This reduces both the traffic on the congested edges and the Interest retransmission rate, which reduces the system latency accordingly.

In a CCN network, the mean edge utilization can be affected by the maximal routing capacity - the total capacity of the minimal cut of the routing paths [23], and the *Cache Hit Rate* (CHR). Generally, more traffic can be delivered and higher mean utilization can be achieved when the maximal routing capacity is high. Meanwhile, as routers deliver more traffic, the CHR increases (discussed in more detail in Section IV-D) and the mean edge utilization decreases. In MTTE, as more trees are created, the maximal routing capacity and hence the mean edge utilization increase. This trend can be observed at the early stage of the simulation (before second 30, Figure 5). As the maximal routing capacity of the overlay trees approaches the maximal capacity of the physical network, the increase in the mean utilization stops. On the other hand, routing paths in CCN and hence the maximal routing capacity do not change since the beginning of the simulation. The mean edge utilization of CCN generally decreases at the early stage of the simulation (before second 30, Figure 5), which is mainly attributed to the improve of the CHR.

In Figure 2, latency (MTTE) increases before second 10 and henceforth decreases. This is because before second 10, no sufficient trees are created. Packets accumulate on the bottleneck edges, which increases the delay. After that, as more trees are created, the congestion is mitigated and the delay

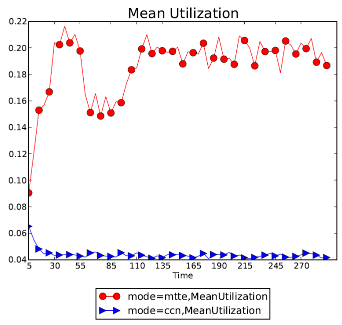
Figure 5. The comparison of the mean edge utilization (vertical axis; ranging between 0.0 and 1.0) between MTTE and CCN when IIF is 15.

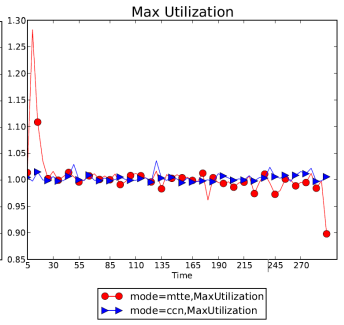
Figure 6. The comparison of the maximum of edge utilization among all edges (vertical axis) between MTTE and CCN when IIF is 15.

decreases.

## D. Performance under Low Traffic

Figure 7 compares the system latency between MTTE and CCN when traffic is low (IIF=5). Specifically, latency(MTTE) is roughly $18\%$ higher than latency(CCN), and MTTE shuts down up to $45\%$ edges (Figure 8).

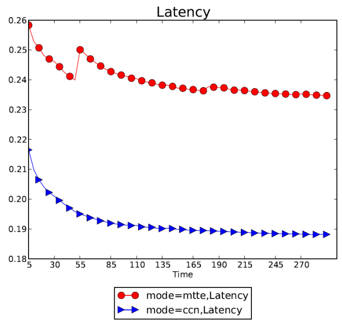
Figure 7. The comparison of the system latency (vertical axis; measured in seconds) between MTTE and CCN when the traffic is low (IIF is 5).

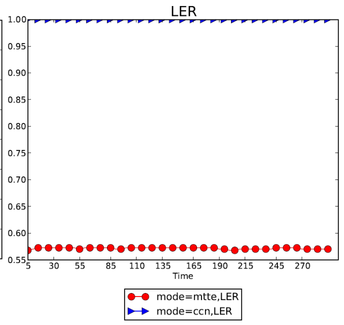
Figure 8. The comparison of LER (vertical axis) between MTTE and CCN when IIF is 5.

As the clock ticks, the system latencies of both MTTE and CCN decrease. To find out why, we measure the CHR. Suppose a router receives an Interest. If the content requested by this Interest is (not) in the router's cache, we say that the cache makes a (miss) hit. Each router records the number of hits (misses) its cache makes during the simulation as the *cache hit count* (*cache miss count*). Then, we calculate the CHR according to (1):

$$CHR = \frac{\text{mean cache hit count}}{\text{mean cache hit count} + \text{mean cache miss count}}. \quad (1)$$

As routers process more Interests, more popular contents are stored in the caches and the CHR increases (Figure 10). Accordingly, that the system latency decreases over time. Note that the CHR is calculated based on the traffic from the past five seconds. The delay is calculated based on the traffic since the simulation starts until the current time point. Hence, the converging speed of the delay is lower than the CHR, where the delay keeps slightly decreasing even when the CHR has largely turned stable at second 55.

The reason that latency(MTTE) $>$ latency(CCN) is that as stated in Section III, MFL(MTTE) is generally larger than MFL(CCN). To see this, we measure the mean hop counts

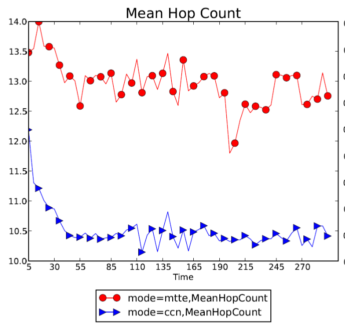

Figure 9. The comparison of the MHC (vertical axis) between MTTE and CCN when IIF is 5.

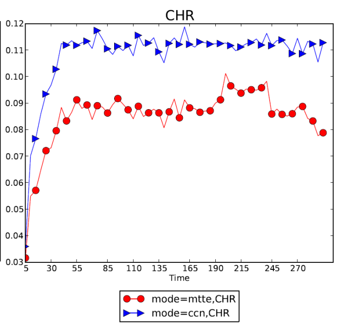

Figure 10. The comparison of the CHR (vertical axis; ranging between 0.0 and 1.0) between MTTE and CCN when IIF is 5.

(MHCs) of MTTE and CCN as indicators of the MFLs. The MHC is calculated as the average number of hops for content objects to return to consumers. The MHC is not equal to the MFL as the MHC is also affected by the CHR, but it should positively correlate to the MFL. Figure 9 shows that MHC(MTTE) can be $30\%$ greater than MHC(CCN). As the MFL increases, the same content is cached on more routers, making CHR(MTTE) decrease as well. Figure 10 shows that CHR(MTTE) can be $20\%$ lower than CHR(CCN). As the combined result of the high MFL and low CHR, latency(MTTE) is greater than latency(CCN).

### E. *Performance under Fluctuating Traffic*

In order to evaluate the performance when the network experiences fluctuating traffic, we vary the IIF so that the IIF rides a sine wave. The wave shape of the IIF is shown in Figure 11. We expect to see that (1) TCC works correctly, i.e., MTTE adds trees when congestion is heavy and removes trees when network utilization is low, and (2) MTTE keeps both the latency and the LER low.

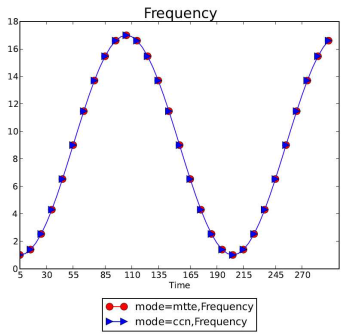

Figure 11. We make the IIF form a sine wave. The horizontal and vertical axes represent the simulation time and the IIF, respectively.

Figure 12 shows that generally, latency(MTTE) is much lower than latency(CCN). As the IIF increases, so does the congestion on bottleneck links. On one hand, MTTE dynamically creates and removes trees (Figure 14), and latency(MTTE) largely remains stable, which proves the effectiveness of TCC. On the other hand, in CCN, packet drop occurs and latency(CCN) increases as the IIF increases. As packet drop occurs, consumers re-send Interests, which makes the congestion deteriorate further. Latency(CCN) remains high even when the IIF peak is over. The peak of the IIF emerges

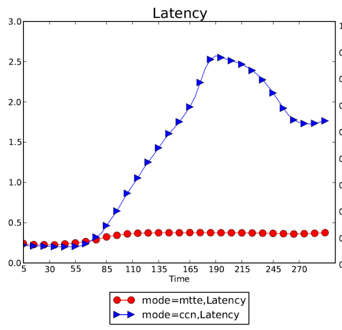

Figure 12. The comparison of the system latency (vertical axis) between MTTE and CCN when the IIF fluctuates.

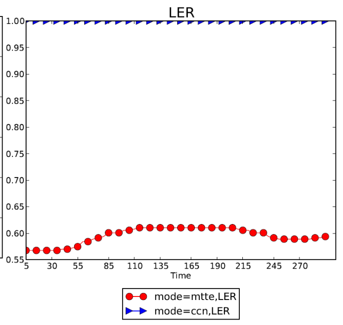

Figure 13. LER changes in MTTE and CCN as the IIF fluctuates.

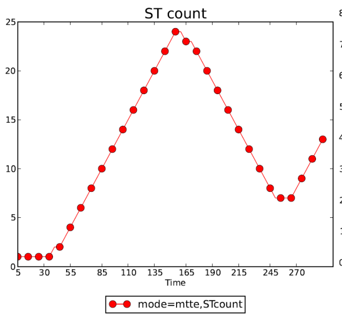

Figure 14. The change in tree count in MTTE. As the IIF fluctuates, so does the tree count.

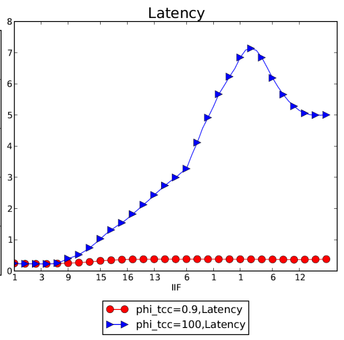

Figure 15. Comparison of latency(MTTE) (in seconds) when TCC is disabled and enabled under fluctuating traffic.

at the 100th second (Figure 11) while latency(CCN) does not start decreasing until the 190th second (Figure 12), meaning that it takes a long time for the network to completely transmit the Interests accumulated when the network was congested. All thought the simulations, MTTE shuts down up to $40\%$ of edges (Figure 13). Since trees created in MTTE are heavily overlapped, the increase in LER(MTTE) is slight even thought the traffic remarkably surges. Figure 15 compares the latency when the TCC mechanism is disabled (by setting $\phi_{tcc} = 100$ so that no tree is created) and enabled ($\phi_{tcc} = 0.9$). We can clearly see that TCC effectively reduces the latency.

## V. Conclusion and Future Work

CCN is a promising network architecture that provides many new possibilities. In this work, we concentrated on CCN's energy efficiency, which is a barely-explored but much-needed research topic. With the core idea of shutting down expendable edges, we have proposed a novel multiple tree-based architecture called MTTE, the first online green mechanism for CCN. Through simulation, we have shown that MTTE can shut down up to $45\%$ of redundant edges, and achieve comparable, and in many cases superior, traffic transmission performance, compared to native CCN. As for future work, *we plan to improve MTTE's performance using more sophisticated tree generation algorithms, and evaluate the performance in larger-scale physical networks*. Meanwhile, we will design energy-saving mechanisms for CCN based on more accurate energy models. MTTE uses a centralized controller, which may incur scalability problems. Implementing it in a distributed manner is another future research topic.

## ACKNOWLEDGMENT

The work described in this paper was, in part, performed in the context of the FP7/NICT EU-JAPAN GreenICN project.

## REFERENCES

[1] L. Zhang et al., "Named data networking (NDN) project," Relatório Técnico NDN-0001, Xerox Palo Alto Research Center-PARC, 2010.

[2] A. Chanda, C. Westphal, and D. Raychaudhuri, "Content based traffic engineering in software defined information centric networks," Proc. IEEE NOMEN Workshop, 2013, 2013.

[3] H. Xie, G. Shi, and P. Wang, "TECC: Towards collaborative in-network caching guided by traffic engineering," in INFOCOM, 2012 Proceedings IEEE. IEEE, 2012, pp. 2546–2550.

[4] M. Zhang, C. Yi, B. Liu, and B. Zhang, "GreenTE: Power-aware traffic engineering," in Network Protocols (ICNP), 2010 18th IEEE International Conference on. IEEE, 2010, pp. 21–30.

[5] L. Chiaraviglio, M. Mellia, and F. Neri, "Reducing power consumption in backbone networks," in IEEE International Conference on Communications, 2009. IEEE, 2009, pp. 1–6.

[6] W. K. Chai, D. He, I. Psaras, and G. Pavlou, "Cache less for more in information-centric networks," in NETWORKING 2012. Springer, 2012, pp. 27–40.

[7] J. C. C. Restrepo, C. G. Gruber, and C. M. Machuca, "Energy profile aware routing," in IEEE International Conference on Communications Workshops, 2009. ICC Workshops 2009. IEEE, 2009, pp. 1–5.

[8] 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, no. 4. ACM, 2005, pp. 253–264.

[9] N. Vasić and D. Kostić, "Energy-aware traffic engineering," in Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking. ACM, 2010, pp. 169–178.

[10] A. Coiro, M. Listanti, A. Valenti, and F. Matera, "Energy-aware traffic engineering: A routing-based distributed solution for connection-oriented ip networks." Computer Networks, vol. 57, no. 9, 2013, pp. 2004–2020.

[11] N. Choi, K. Guan, D. C. Kilper, and G. Atkinson, "In-network caching effect on optimal energy consumption in content-centric networking," in 2012 IEEE International Conference on Communications (ICC). IEEE, 2012, pp. 2889–2894.

[12] U. Lee, I. Rimac, D. Kilper, and V. Hilt, "Toward energy-efficient content dissemination," Network, IEEE, vol. 25, no. 2, 2011, pp. 14–19.

[13] Y. Song, M. Liu, and Y. Wang, "Power-aware traffic engineering with named data networking," in Seventh International Conference on Mobile Ad-hoc and Sensor Networks (MSN), 2011. IEEE, 2011, pp. 289–296.

[14] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," Proceedings of the American Mathematical society, vol. 7, no. 1, 1956, pp. 48–50.

[15] R. Hassin and A. Tamir, "On the minimum diameter spanning tree problem," Information processing letters, vol. 53, no. 2, 1995, pp. 109–111.

[16] M. Girvan and M. E. Newman, "Community structure in social and biological networks," Proceedings of the National Academy of Sciences, vol. 99, no. 12, 2002, pp. 7821–7826.

[17] A. Afanasyev, I. Moiseenko, and L. Zhang, "ndnsim: Ndn simulator for ns-3," Named Data Networking (NDN) Project, Tech. Rep. NDN-0005, Rev, vol. 2, 2012.

[18] "Rocketfuel dataset," https://github.com/cawka/ndnSIM-ddos-interest-flooding/tree/master/topologies/rocketfuel_maps_cch, (retrieved: September, 2014).

[19] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP topologies with rocketfuel," ACM SIGCOMM Computer Communication Review, vol. 32, no. 4, 2002, pp. 133–145.

[20] C. Yi et al., "A case for stateful forwarding plane," Computer Communications, 2013, pp. 779–791.

[21] A. Ghodsi et al., "Information-centric networking: Seeing the forest for the trees," in Proceedings of the 10th ACM Workshop on Hot Topics in Networks. ACM, 2011, p. 1.

[22] S. K. Fayazbakhsh et al., "Less pain, most of the gain: incrementally deployable ICN," in Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM, ser. SIGCOMM '13. New York, NY, USA: ACM, 2013, pp. 147–158.

[23] G. Dantzig and D. R. Fulkerson, "On the max flow min cut theorem of networks," Linear inequalities and related systems, vol. 38, 2003, pp. 225–231.