Simulation Studies of OpenFlow-Based In-Network Caching Strategies

Ting Li, Nathanael Van Vorst, Rong Rong, Jason Liu

School of Computer and Information Sciences
Florida International University
Emails: {*tli001,nvanv001,rrong001,liux*}*@cis.fiu.edu*

Keywords: In-network caching, OpenFlow, CacheFlow

Abstract

We propose an in-network caching architecture using OpenFlow to coordinate caching decisions in the network. Our scheme, called *CacheFlow*, extends the cache-and-forward concept by moving contents closer to the clients hop-by-hop using TCP for sending requests and retrieving contents. As such, CacheFlow can be incrementally implemented and deployed in the real network. In this paper, we present a simulation study of several caching policies, including a random cache policy, a statically optimal cache placement policy and a new disk placement strategy that places popular contents at the "center" of the network. Experimental results show that simple in-network caching policies can be realized using today's technology to improve network performance.

1. INTRODUCTION

The demand for network bandwidth has been growing at a relentless rate. Such an explosion is largely fueled by the constant arrival of "killer apps", such as peer-to-peer file sharing applications (e.g., BitTorent) and video-on-demand services (e.g., YouTube and Netflix), as well as the increasing popularity of network-hungry end devices, such as tablets and smart phones. Allowing efficient distribution of large volume of data and avoiding clogging of the network pathways (including both network arteries and tributaries) is a recurring theme for today and tomorrow's network design.

Caching is prevalent in today's network design. Traditional solutions for content distribution include content distribution networks (CDN) and peer-to-peer networks (P2P). The main idea behind CDN is deploying multiple servers, which are strategically located at the edge of the network, and in doing so bringing content close to the clients in order to reduce network bandwidth demand and latency. In P2P, peers get involved in redistributing and caching content, which can reduce the total transport distance and avoid congestion at specific servers. Although these approaches undoubtedly improve network performance, they have enjoyed only limited success because they can only be implemented at the peripheral of the network.

Recently there has been growing interest in designing future Internet architectures that can dramatically change the existing Internet model and thus break away from the existing network constraints. Researchers have proposed several "clean-slate" approaches, such as content-centric networking (CCN) [11] and cache-and-forward networks (CNF) [6], which promise more efficient content distribution and delivery through in-network caching. The major problem with the clean-slate approaches, however, is that they are not incrementally deployable on the current Internet, and therefore difficult to be adopted for widespread use. The issue at hand is how to *gradually* convert a best-effort packet-oriented network as of the Internet today to a service-oriented content-based network of tomorrow. Specific to content distribution, we need scalable and incrementally deployable solutions that can support in-network caching capabilities.

In this paper we propose an OpenFlow-based in-network caching architecture, which we call *CacheFlow*, for efficient content distribution. OpenFlow [14] is an open standard for a network that consists of switches managed by one or more controllers. An important feature of OpenFlow is that it provides a clean separation between data and control planes through a standardized interface. As such, OpenFlow offers a programatic approach to the gradual adoption of programmable networks. CacheFlow extends the OpenFlow concept with additional storage capabilities for caching contents in network. CacheFlow consists of: 1) *content storage switches*, which are the OpenFlow switches implemented with individual caching policies, and 2) *content controllers*, which are the OpenFlow controllers augmented with the logic for making in-network caching decisions for the content storage switches it is in charge of. In contrast to the existing in-network caching approaches, a distinct feature of the CacheFlow architecture is that it can be incrementally implemented and deployed in the existing networks. To that end, our design considers real implementation issues, such as using existing network protocols, such as the OpenFlow protocol and TCP.

Simulation has been used extensively to evaluate network performance. In particular, simulation provides a detailed operational view of the network protocols at play and their behaviors. To evaluate CacheFlow, we designed and implemented a detailed packet-level network simulator with full-fledged TCP protocols, which allows testing our in-network caching architecture using large-scale network scenarios. In this paper, we use the simulator to study the performance of

CacheFlow with three different cache placement policies: a random placement policy, a statically optimal placement policy, and a new disk placement policy, that preferably caches popular content close to the center of the network.

The rest of the paper is organized as follows. Section 2. provides background and discusses related work. Section 3. presents our in-network caching architecture. Section 4. focuses on caching strategies. Section 5. details the simulation experiments and presents the results. Section 6. concludes the paper with a discussion of future work.

2. RELATED WORK

Content distribution networks (CDN), such as Akamai [12], Google [1], and Coral [8], are designed to move a content closer to clients by distributing and caching copies of the content to servers geographically distributed across the network. Peer-to-peer (P2P) networks also provide efficient content distribution by capitalizing on the bandwidth at the end nodes [13]. P2P content distribution has been applied to large-scale video-on-demand systems, in which each user contributes a small amount of storage for content caching. It is shown that prefetching and replication strategies that can exploit various user demand characteristics are essential to the performance of such systems [9, 10].

Internet caching strategies have been studied quite extensively in the literature. For example, Chankhunthod et al. [4] outlined a hierarchical caching structure that places caches at different network levels. Povey and Harrison [17] proposed a distributed caching scheme that coordinates cooperating caches placed at the edge of the network. Tewari [21] improved the performance of distributed caching using direct cache-to-cache transfers and push caching algorithms for moving data closer to the clients. Rodriguez et al. [19] provided an analytical model for performance comparison between distributed and hierarchical caching schemes and presented a hybrid approach that combines hierarchical caching with distributed caching at every level of the hierarchy. In order to balance load and reduce backbone traffic, Paul and Fei [16] also proposed a centralized control architecture for distributed coordinated caches.

A special category of network caching is to cache data en-route in the network switching nodes. For example, Bhattacharjee et al. [2] studied several self-organizing en-route caching schemes via simulation. Tang and Chanson [20] proposed a coordinated caching scheme where caching information is passed along the routing path of a request for routers to dynamically make cache placement and replacement decisions. More recently, the cache-and-forward (CNF) architecture [5, 6] was proposed as a "clean-slate" solution using in-network en-route caching for occasionally disconnected mobile users. CNF uses a new transport service to move content in a hop-by-hop, store-and-forward manner.

Our work is inspired by the above in-network caching schemes and CNF in particular. Our work is different from CNF in that we focus more specifically on coordinated caching strategies achievable on the OpenFlow network. OpenFlow provides an open protocol to program the switches and routers [14]. Within each OpenFlow switch, there is a flow-table with an action associated with each flow entry, basically telling the switch how to process packets belonging to the flow. When a packet that does not correspond to an existing flow entry arrives at the switch, the switch will send the packet to a remote controller via a secure channel using the standardized OpenFlow protocol. The controller then determines whether to install a flow entry in the switch so that the switch can properly process the subsequent packets of the flow. In our scheme, the controller can designate an OpenFlow switch in the network to cache the content and redirect subsequent requests to access the cached content at the designated switch. Using OpenFlow, our in-network caching policies can be added incrementally and deployed using existing protocols, such as the OpenFlow protocol and TCP.

3. OPENFLOW-BASED IN-NETWORK CACHING

We provide an overview of the CacheFlow architecture in this section. CacheFlow consists of two types of nodes: the content storage switches and the content controllers. In our scheme, caching at each network is controlled by one *content controller*, which is collocated with the OpenFlow controller. The content controller coordinates the caching of content that flows through the network using *content storage switches*. Content storage switches piggyback on OpenFlow switches and cache complete copies of content. The network can be an institutional network, a regional network, or a backbone network. Consequently, the CacheFlow architecture can be used for various caching strategies, including hierarchical caching [4], distributed caching [17], or a combination of both.

We borrow the CNF concept [6] and define content as the first-class entity in the network. A content can be identified using a persistent and globally unique content identifier (CID), which is location independent. When a client asks for a content, a request is sent with the content's CID. The request will be intercepted by an OpenFlow switch, recast as a content storage switch. The content storage switch first checks whether the content has already been stored in its cache and if so, returns the content to the requesting client directly. If the content request is new, it forwards the request via a secure communication channel to the OpenFlow controller, recast as the content controller.

Depending on the caching policy, the content controller must decide on whether and which content storage switch shall cache this content in the network. In case the content

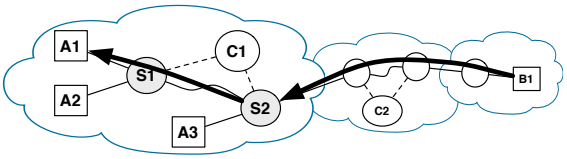

Figure 1. An example showing the CacheFlow process

has never been cached in the network, the controller needs to run a location service that looks up the location of the content using its CID. (Here, for example, we can use a mechanism similar to the content-based addressing and routing scheme proposed by Carzaniga et al. [3]). The content controller then installs a flow entry in the chosen content storage switch to act upon retrieving the content from the content provider upon receiving such a request. The controller also installs a flow entry in the original switch that first encounters the request for it to effectively forward all requests (with the same CID) to the chosen content storage switch.

Fig. 1 illustrates an example. Suppose client A_1 sends a request for content, which is located at B_1. The content request is intercepted by switch S_1, which subsequently gives it to the controller C_1. Suppose that the controller determines that switch S_2 shall cache this content according to its caching policy. C_1 then installs a flow entry in S_2 so that if a request arrives, it knows it should download the content from the content provider B_1. C_1 also installs a flow entry in S_1 so that S_1 can forward all similar request to S_2. Upon receiving the request, S_2 contacts the content provider to download and then caches the content. It then responds to client A_1 with the requested content.

Once the content has been cached in the network, subsequent requests within the network for the same content (say, from clients A_2 and A_3) will all be directed to S_2. Note that when S_2 tries to retrieve content from B_1, similar caching decisions can be made at the intermediate networks, each with its own content controller and set of content storage switches, implemented with a different caching policy. For this reason, CacheFlow can be incrementally implemented and deployed on the real Internet—one network at a time.

We use TCP as the transport protocol for reliable content delivery with congestion control. To enable content-based routing, we reserve a specific multicast IP address to be used as the destination IP address and use CID as the TCP destination port number. The OpenFlow switches can use these fields to differentiate flows in their flow-tables.

4. CACHING POLICIES

All caching decisions within a network shall be made at the CacheFlow controller. A caching policy consists of cache placement and cache eviction. Cache placement determines which CacheFlow storage switch should be selected to cache a particular content. Cache eviction determines which content needs to be flushed out from cache in order to make room for new content. We implemented three cache placement policies: random, statically optimal, and disk placement.

For baseline comparison, we implemented a random cache placement policy, which does not consider the network topology or the request pattern. The controller chooses a random storage switch with available space to store a new content. We describe the other two cache placement strategies in the subsequent sections.

For cache eviction, we use the *Least Popular First Out (LPFO)* strategy. LPFO keeps track of the number of times a content has been requested within a network. When a request arrives, if the controller finds that there is no room available to cache the content in the network, the controller will select the content having the least popularity to be evicted to make room for the new content.

4.1. Statically Optimal Placement

Dong et al. [5] formulate the problem of minimizing the average content retrieval delay as an optimization problem. The formulation uses *a priori* knowledge as input, which includes the topology of the storage nodes, the request patterns, the storage limitations, and the real-time delays between the storage nodes. The solution is the probabilities for which contents need be cached at the storage nodes. More specifically, the optimization problem has the following objective function:

$$\min \sum_{i=1}^{N} \sum_{j=1}^{F} P_{i,j} \sum_{h=1}^{H_{i,S_j}} hD_j V_{C_h^{i \to S_j},j} \prod_{k=0}^{h-1} (1 - V_{C_h^{i \to S_j},j})$$

subject to the following constraints:

$$\sum_{j=1}^{F} V_{i,j} \cdot f_j \leq R_i, \forall i = 1 \cdots N$$
$$V_{S_j,j} = 1, \forall j = 1 \ldots F$$
$$0 \leq V_{i,j} \leq 1, \forall i = 1 \cdots N, j = 1 \ldots F$$

where N is the total number of nodes (including the clients, the content storage switches, and content providers); F is the number of contents; f_i is the size of content i; $H_{a,b}$ is the number of hops from node a to node b; S_j is the provider of content j; $C_h^{a \to b}$ is the h^{th} storage switch on the routing path from node a to node b; D_j is the average delay for servicing content j over a hop (which includes the processing delay and the transmission delays of both request and content); R_i is the storage size at node i; and $P_{i,j}$ is the probability that node i requests content j. We need to solve this nonlinear optimization problem for $V_{i,j}$, which is the probability that storage switch i caches content j. In other words, $V_{i,j}$ determines the probabilistic cache placement that minimizes the average content retrieval delay.

In the CNF paper [5], the authors claim that this optimal problem can be solved online at specified time intervals. However, collecting the *a priori* information dynamically is impractical. For example, the request patterns, formulated in the form of $P_{i,j}$, which is the probability that node i wants content j, cannot be determined easily. Moreover, a network of a decent size would need a large number of constraints; therefore, the computational requirement for solving such a problem online can be prohibitive. Given the limited utility of an online solution, we opted to use a statically determined request patterns and pre-calculate the solution off-line.

In our implementation, to extract information needed by the objective function, we first have each client and each content storage switch to ping all content providers. From the results we construct an overlay network that consists of only clients, content storage switches, and content providers (as opposed to the underlying physical network). Using the topology and delay information of the overlay network we construct the equations and feed them to IPOPT, which is a large-scale nonlinear optimization problem solver [23]. Afterwards, we get $V_{i,j}$ from the solution and use them as input to the content controllers for them to determine where to place new contents probabilistically during an experiment.

4.2. Disk Placement

Since the goal of cache placement is to minimize the average delay of content retrieval, we propose a straightforward solution that heuristically places the content at storage switches close to the clients. One can think of a network like a flat disk, and the center is equidistant from all edges of the disk. If we place popular contents near the center of the network we can hopefully reduce the average distance to the clients. There are three important aspects to the algorithm. First, we must be able to find the center of the network. Second, we must estimate how far each storage switch is from the center of the network. And last, we must populate the caches of the storage switches closest to the center of the network with the most popular contents.

The controller maintains a list of switch nodes at the edge of the network it is responsible for. To estimate the diameter of the network, we measure the the number of hops between each edge node and storage switch in the network. This can be achieved by having each storage switch ping each edge node and take note of the TTLs of the echo replies. Suppose the network has E edge nodes: $e_1, e_2, \cdots, e_E$ and R storage switches $r_1, r_2, \cdots, r_R$, we use the following equation to estimate the average distance between the nodes:

$$D = \frac{\sum_{i=1}^{E} \sum_{j=1}^{R} dist(e_i, r_j)}{ER}$$

Any node which is $D/2$ hops from all edge nodes can be considered to be at the center of the network. To estimate how far a storage switch r is from the center of the network we can simply calculate the average distance of the content storage switch from all edge nodes: $C_r^2 = \sum_{i=1}^{E} (dist(r, e_i) - D/2)^2$. Smaller C_r suggests closer to the center of the network.

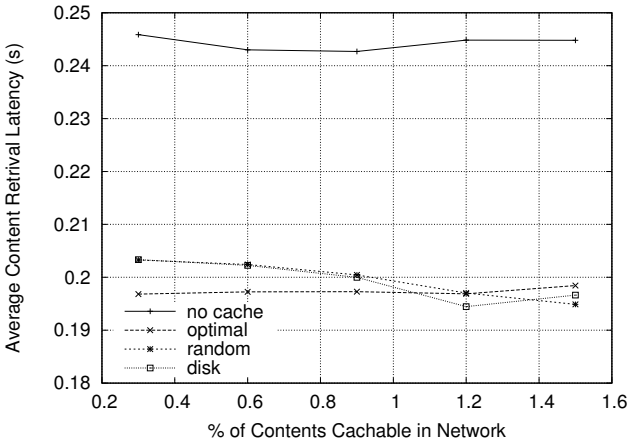

Figure 2. The effect of cache sizes on retrieval latency.

Suppose there are R storage switches with available cache space for storing the content. We sort the storage switches in increasing order of C_r and name them $r_1, r_2, \cdots, r_R$. Suppose the content has a pre-determined popularity ranking of k, where $1 \leq k \leq F$, and F is the total number of contents. We then select from the first kR/F storage switches uniformly at random. In this way, more popular contents are placed closer to the center of the network. If there is no space left for caching the content in the network, the controller will evict old content from the network. As mentioned earlier, eviction as in all implemented caching policies is LPFO, which is independent of the storage switch location.

5. SIMULATION EXPERIMENTS

We implemented CacheFlow in a simulator and used it to study the performance of different caching policies.

5.1. The Simulator

We implemented CacheFlow in our PRIME simulator [18]. PRIME is designed to run on parallel and distributed platforms and as such capable of simulating large-scale network models. PRIME provides detailed models of 14 TCP congestion control algorithms, including RENO, BIC, and CUBIC. These TCP models have been validated carefully through extensive experiment studies [7]. PRIME also supports real-time simulation, where simulated network protocols, such as TCP, can seamlessly interact with a real network [22]. This will allow us to design future experiments involving real clients and real OpenFlow controllers and switches.

The simulator organizes the implementation of protocols and applications as "protocol sessions" on each network

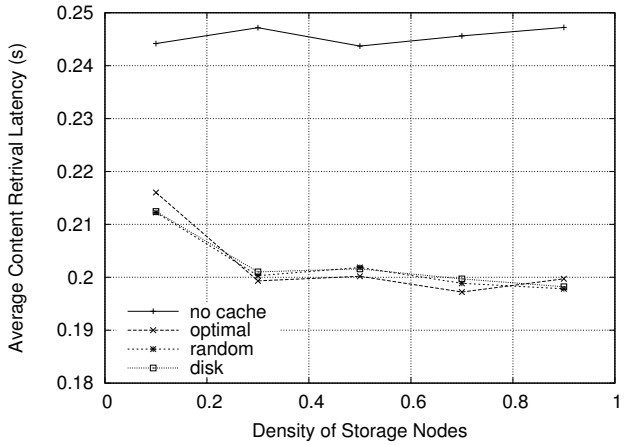

Figure 3. The effect of storage node density on retrieval latency.

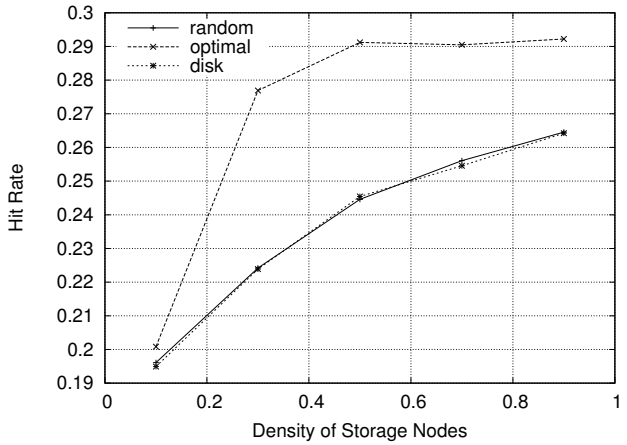

Figure 4. The effect of storage node density on cache hit rate.

node. We implemented an *application protocol session* on the client nodes to request contents according to the traffic specification and popularity of the contents. We describe the experiment scenario in the next section. The application protocol session uses TCP for sending requests and receiving contents.

On a subset of the router switches in the network we implemented a *switch protocol session* to simulate the logic of a content storage switch. After intercepting a content request, the switch protocol session first determines whether the content has been cached locally. If it is, the protocol session sends the cached content to the client via TCP. Otherwise, it forwards to the content request to the content controller. We implemented a simplified OpenFlow protocol in simulation as TCP messages between the content controller and the storage switches. If a content storage switch is selected by the controller to cache the content, upon receiving the request, the switch protocol session retrieves the content via TCP from the content provider. (The current implementation does not yet support caching at the intermediate networks.)

Each network designates a router to be the controller, and we implemented a *controller protocol session* to simulate the caching policies. We implemented the three caching policies described in the previous section. For now, each controller makes caching decisions independent from the other controllers in other networks. We will investigate coordinated multi-controller caching algorithms in the future.

5.2. Experiment Scenario

For the experiment, we use BRITE [15] to generate a network topology that consists of 10 Autonomous Systems (ASes), each with 28 routers and 18 hosts. The resulting topology consists of 10 networks for a total of 280 routers and 180 hosts. We limit the network size so that IPOPT (the nonlinear optimization problem solver) can calculate statically optimal placement probabilities within reasonable time. Our simulator can handle much larger network topologies. Given the network topology, we randomly choose one router as the content controller at each AS, and vary the number of routers selected to be the content storage switches in each AS.

We spread a total of 900 different contents uniformly among the hosts. We use constant content size of 140 KB. In the experiment the content requests are generated with an exponential inter-arrival time of 0.1 seconds. A host does not request a content which it already owns. The specific content requested by each host is determined according to the popularity of the contents, which can be modeled by a Mandelbrot-Zipf distribution [6]. The popularity of content i can be determined as $P_i = 1/\kappa(i+q)^{\alpha}$, where α is the skewness factor (we set $\alpha = 0.5$), q is the plateau factor (we set $q = 0$), κ is the normalizing factor: $\kappa = \sum_{i=1}^{F} 1/(i+q)^{\alpha}$, and again F is the total number of contents.

5.3. Results

We run two sets of experiments to study the performance of the optimal, disk and random caching policies. For the first experiment we enable all switches in the network to be able to store content. For each policy we measure the average content retrieval latency as we increase the amount of storage space. The results in Fig. 2 are the average of 20 runs; we do not show confidence intervals because they are extremely small. All three policies show considerable improvement over no caching. However, there is little variation between the three policies. For the second experiment, we fix the cache size of each storage switch and vary the density of storage switches in each AS. We measure the average content retrieval latency and the hit rate for all three policies. Fig. 3 shows the latency measurements and Fig. 4 shows the cache hit rate over all requests. Optimal placement shows slightly better hit ratio than the other two methods.

The similar behavior demonstrated by different placement policies is largely due to the relatively small network size. This tells us caching content in network can bring significant performance benefits; however, for small networks, the difference among cache placement strategies is inconsequential.

6. CONCLUSION

We present a CacheFlow architecture which allows for in-network caching to be incrementally implemented and deployed in today's Internet. We implement a detailed simulator, including a full-fledged TCP model, to evaluate different in-network caching scenarios. Simulation results show that caching within a network can reduce the retrieval latencies (by roughly 10%). The results concerning placement strategies were inconclusive. It suggests that the most important aspect to decreasing retrieval time is to cache the content within the network; the precise placement, however, has little effect.

To better study the performance of different caching policies we plan to use large, realistic networks. In addition, we plan to use real content access traces. We will investigate methods that allow controllers from separate ASes to coordinate caching rather than operating autonomously. We will also investigate methods for improving LPFO eviction policy by allowing the controller to move contents between storage switches. This allows the controller to place infrequently accessed content at the edge of the network while reserving the center of the network for more popular content.

ACKNOWLEDGMENT

This research is supported in part by the National Science Foundation award CNS-0836408, and a subcontract from the GENI Project Office at Raytheon BBN Technologies.

REFERENCES

[1] L. A. Barroso, J. Dean, and U. Hölzle, "Web Search for a Planet: The Google Cluster Architecture," *IEEE Micro*, vol. 23, pp. 22–28, 2003.

[2] S. Bhattacharjee, K. Calvert, and E. Zegura, "Self-organizing Wide-area Network Caches," in *INFOCOM*, vol. 2, 1998, pp. 600–608.

[3] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf, "Content-based Addressing and Routing: A General Model and its Application," University of Colorado, Tech. Rep. CU-CS-902-00, 2000.

[4] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell, "A Hierarchical Internet Object Cache," in *USENIX Technical Conference*, 1996, pp. 153–163.

[5] L. Dong, D. Zhang, Y. Zhang, and D. Raychaudhuri, "Optimized Content Caching and Request Capture in CNF Networks," in *WICON*, 2010, pp. 1–9.

[6] L. Dong, Y. Zhang, and D. Raychaudhuri, "Gateway Controlled Content Caching and Retrieval for Cache-and-forward Networks," in *GLOBECOM*, 2009, pp. 1–6.

[7] M. Erazo, Y. Li, and J. Liu, "SVEET! A Scalable Virtualized Evaluation Environment for TCP," in *TridentCom*, 2009.

[8] M. J. Freedman, E. Freudenthal, and D. Mazières, "Democratizing Content Publication with Coral," in *NSDI*, 2004, pp. 239–252.

[9] C. Huang, J. Li, and K. W. Ross, "Can Internet Video-on-demand Be Profitable?" in *SIGCOMM*, 2007, pp. 133–144.

[10] Y. Huang, T. Z. Fu, D.-M. Chiu, J. C. Lui, and C. Huang, "Challenges, Design and Analysis of a Large-scale P2P-VoD System," in *SIGCOMM*, 2008, pp. 375–388.

[11] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking Named Content," in *CoNEXT*, 2009, pp. 1–12.

[12] A. jan Su, D. R. Choffnes, A. Kuzmanovic, and F. E. Bustamante, "Drafting Behind Akamai (Travelocity-based Detouring)," in *SIGCOMM*, 2006, pp. 435–446.

[13] T. Karagiannis, P. Rodriguez, and K. Papagiannaki, "Should Internet Service Providers Fear Peer-assisted Content Distribution?" in *IMC*, 2005, pp. 63–76.

[14] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *SIGCOMM CCR*, vol. 38, pp. 69–74, 2008.

[15] A. Medina, A. Lakhina, I. Matta, and J. Byers, "BRITE: An Approach to Universal Topology Generation," in *MASCOTS*, 2001.

[16] S. Paul and Z. Fei, "Distributed Caching with Centralized Control," in *IWCCD*, 2000, pp. 256–268.

[17] D. Povey and J. Harrison, "A Distributed Internet Cache," in *Australian Computer Science Conference*, 1997, pp. 5–7.

[18] PRIME Research Group, http://www.primessf.net/, 2011.

[19] P. Rodriguez, C. Spanner, and E. W. Biersack, "Analysis of Web Caching Architectures: Hierarchical and Distributed Caching," *IEEE/ACM TON*, vol. 9, pp. 404–418, 2001.

[20] X. Tang and S. Chanson, "Coordinated En-route Web Caching," *IEEE TOC*, vol. 51, no. 6, pp. 595–607, 2002.

[21] R. Tewari, M. Dahlin, H. M. Vin, and J. S. Kay, "Beyond Hierarchies: Design Considerations for Distributed Caching on the Internet," in *ICDCS*, 1998.

[22] N. Van Vorst, M. Erazo, and J. Liu, "PrimoGENI: Integrating Real-time Network Simulation and Emulation in GENI," in *PADS*, 2011.

[23] Wächter and L. T. Biegler, "On the Implementation of a Primal-dual Interior Point Filter Line Search Algorithm for Large-scale Nonlinear Programming," *Mathematical Programming*, vol. 106, no. 1, pp. 15–57, 2006.

AUTHOR BIOGRAPHIES

TING LI is a Ph.D. candidate and is working in the Modeling and Networking Systems Research Group at the School of Computing and Information Sciences, Florida International University. She received her B.E. degree and M.S. degree both from Nankai University in China in 2004 and 2007, respectively. Her research is in the area of network modeling and simulation, specifically network traffic modeling and network routing.

NATHANAEL VAN VORST is a Ph.D. candidate at the School of Computing and Information Sciences, Florida International University. He received a B.S. degree from Colorado State University in 2003 and an M.S. degree from the Colorado School of Mines in 2007. He is currently a member of the Modeling and Networking Systems Research Group at Florida International University. His research focuses on simulation and emulation of computer networks.

RONG RONG is a Ph.D. candidate at the School of Computing and Information Sciences, Florida International University. She received a B.E. degree from Beihang University in China in 2006 and an M.S. degree from the Renmin University in China in 2009. Her research is in the area of parallel discrete-event simulation.

JASON LIU is an Associate Professor at the School of Computing and Information Sciences, Florida International University. His research focuses on parallel simulation and high-performance modeling of computer systems and communication networks. He received a B.A. degree from Beijing University of Technology in China in 1993, an M.S. degree from College of William and Mary in 2000, and a Ph.D. degree in from Dartmouth College in 2003. He served as General Chair for MASCOTS 2010, SIMUTools 2011 and PADS 2012, and also as Program Chair for PADS 2008 and SIMUTools 2010. He is an Associate Editor for Simulation Transactions of the Society for Modeling and Simulation International, and a Steering Committee Member for PADS.